Proceedings of the 8th ACM Conference on Computer and Communications Security. Pages 255–264.
(november 5 – 8, 2001, Philadelphia, Pennsylvania, USA)

Provably Authenticated Group Diffie-Hellman Key Exchange

Emmanuel Bresson[1], Olivier Chevassut*[2,3], David Pointcheval[1], and Jean-Jacques Quisquater[3]

[1] École Normale Supérieure – 45, rue d'Ulm – F–75230 Paris cedex 05, France
http://www.di.ens.fr/~{bresson,pointche} – {emmanuel.bresson,david.pointcheval}@ens.fr
[2] Lawrence Berkeley National Lab – 1 Cyclotron Rd, MS 50B-2239 – Berkeley, CA 94720, USA
OChevassut@lbl.gov
[3] Université Catholique de Louvain – Place du Levant, 1348 Louvain-la-Neuve
Quisquater@dice.ucl.ac.be

Abstract. Group Diffie-Hellman protocols for Authenticated Key Exchange (AKE) are designed to provide a pool of players with a shared secret key which may later be used, for example, to achieve multicast message integrity. Over the years, several schemes have been offered. However, no formal treatment for this cryptographic problem has ever been suggested. In this paper, we present a security model for this problem and use it to precisely define AKE (with "implicit" authentication) as the fundamental goal, and the entity-authentication goal as well. We then define in this model the execution of an authenticated group Diffie-Hellman scheme and prove its security.

1 Introduction

Group Diffie-Hellman schemes for Authenticated Key Exchange are designed to provide a pool of players communicating over an open network with a shared secret key which may later be used to achieve some cryptographic goals like multicast message confidentiality or multicast data integrity. Secure virtual conferencing involving up to a hundred participants is an example of such a multicast scenario [14]. In this scenario the group membership is static and known in advance: at startup the participants would like to engage in a conversation at the end of which they have established a session key. For this scenario group Diffie-Hellman schemes are attractive alternatives to methods that establish a session key between every pair of players in the multicast group or rely on a centralized key distribution center.

Over the years, several papers [2, 3, 13, 18, 20, 21, 27, 31, 32] have attempted to extend the well-known Diffie-Hellman key exchange [17] to the multi-party setting. The protocols meet a variety of performance attributes but only exhibit an informal analysis showing that they achieve the desired security goals. Some papers exhibit an *ad-hoc* analysis for the security of their schemes and some of these schemes have later been found to be flawed [21, 26]. Other papers only provide *heuristic evidence* of security without quantifying it. The remaining schemes assume authenticated links and thus do not consider the authentication as part of the protocol design.

In the paradigm of provable security [19] one identifies a concrete cryptographic problem to solve (like the group Diffie-Hellman key exchange) and defines a formal model for this problem. The model captures the capabilities of the adversary and the capabilities of the players. Within this model one defines security goals to capture what it means for a group Diffie-Hellman scheme to be secure. And, for a particular

* The second author was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-47585.

scheme one exhibits a proof of its security. The security proof aims to show that the scheme actually achieves the claimed security goals under computational security assumptions.

The fundamental security goal for a group Diffie-Hellman scheme to achieve is Authenticated Key Exchange (with "implicit" authentication) identified as AKE. In AKE, each player is assured that no other player aside from the arbitrary pool of players can learn any information about the session key. Another stronger highly desirable goal for a group Diffie-Hellman scheme to provide is Mutual Authentication (MA). In MA, each player is assured that its partners (or pool thereof) actually have possession of the distributed session key. Pragmatically, MA takes more rounds; one round of simultaneous broadcasts.

With these security goals in hand, one can analyze the security of a particular group Diffie-Hellman scheme and see how it meets the definitions. A security analysis (or proof of security) for the scheme works via reduction from the security of the scheme to the underlying "hard" problem. A reduction is a successful algorithm for the "hard" problem that uses an adversary of the scheme as a subroutine.

In this paper we assume *honest* players. Honest players do not deviate from the protocol and their instances erase any internal data when terminating. Existing two-party protocols (e.g., SSL and IPsec) make this assumption. We see the additional security goal of dealing with dishonest players (like verifiable contributory [2]) as important in some environments but less important in others [1].

This paper provides major contributions to the solution of the group Diffie-Hellman key exchange problem. We first present a formal model to help manage the complexity of definitions and proofs for the authenticated group Diffie-Hellman key exchange. A model where a process controlled by a player running on some machine is modeled as an instance of the player, the various types of attacks are modeled by queries to these instances and the security of the session key is modeled through semantic security. Moreover, in order to be correctly formalized, the intuition behind mutual authentication requires cumbersome definitions of session IDS and partner IDS which may be skipped at the first reading.

Second, we define in this model the execution of a modified known protocol [31], we refer to it as AKE1, and show that AKE1 can be proven secure under reasonable and well-defined intractability assumptions. Third, we present a generic transformation for turning an AKE protocol into a protocol that provides MA and justify its security under reasonable and well-defined intractability assumptions.

The remainder of this paper is organized as follows. The paper starts with some related work in Section 2 and cryptographic notions in Section 3. The paper continues with a description of our model of a distributed environment in Section 4 and gives the precise security definitions that should be satisfied by a group Diffie-Hellman scheme in Section 5. Section 6 presents the protocol AKE1 and justifies its security in the random oracle model. Section 7 turns AKE1 into a protocol that provides MA and justifies its security in the random oracle model.

2 Related Work

Two formal models for secure key exchange have received the most consideration. The first model initiated by Bellare and Rogaway [6, 8] modeled the two-party and three-party key distribution. This model was further extended by Blake-Wilson et al. [10, 11] to model the authenticated Diffie-Hellman key exchange. In this model,

player instances are modeled as oracles available to the adversary and attacks are modeled by oracle queries. Recently, Bellare, Pointcheval and Rogaway [5] refined this model to use session IDs as an approach to define the partnering. They also extended the model to include forward-secrecy, allow password authentication and deal with dictionary attacks. Our model is derived from [5].

The second formal model is based on the multi-party simulatability technique and was initiated by Bellare, Canetti and Krawczyk [4]. In this model Bellare et al. considered Diffie-Hellman and encryption-based key exchange. Recently Shoup [30] refined this model and showed that the two models are equivalent for two parties under specific conditions. However no such treatment has been provided for the group setting yet.

The work of Ateniese et al. [2] is of particular interest since it identifies the fundamental and additional desirable security goals of authenticated group Diffie-Hellman key exchange. The authors offer provably secure authenticated protocols and sketch informal proofs that their protocols achieve these goals. Unfortunately these protocols have later been found to be flawed [26].

Other related papers are [23, 24]. Although they do not tackle the exact same problem and do not achieve the same goal, they still seem relevant enough to mention.

3 Background

We use the following cryptographic notions throughout the paper.

3.1 Concrete Security

In this paper we develop proofs in the framework of concrete provable security. We provide an exact analysis of the security of the schemes rather than asymptotic ones. That is, we explicitly quantify the reduction from the security of a scheme to the security of the underlying "hard" problem(s) on which it is based. This allows us to know exactly how much security is maintained by the reduction and thus to determine the strength of the reduction.

In order to quantify the reductions, we define the advantage $\mathsf{Adv}^{ake}(\mathcal{A})$ that a computationally bounded adversary $\mathcal{A}$ will defeat the **AKE** security goal of a protcol. The advantage is twice the probability that $\mathcal{A}$ will defeat the AKE security goal of the protcol minus one[1].

In order to quantify the reduction, we also consider the probability $\mathsf{Succ}^{ma}(\mathcal{A})$ that a computationally bounded adversary $\mathcal{A}$ will defeat the **MA** security goal of a protocol[2].

3.2 The Ideal Hash Model

In the ideal hash model, also called the "random oracle model" [7], the cryptographic hash functions (like SHA or MD5) are viewed as random functions with the appropriate range. Security proofs in this model identify the hash functions as oracles which produce a truly random value for each new query and identical answers if the same query is asked twice. Later, in practice, the random functions are instantiated using

[1] To defeat AKE security means for $\mathcal{A}$ distinguishing the session key from a random value. Hence, $\mathcal{A}$ can trivially defeat AKE with probability 1/2, multiplying by two and substracting one rescales the probability.

[2] To defeat the MA security for $\mathcal{A}$ means impersonating a player.

specific functions derived from standard cryptographic hash functions like SHA or MD5.

Analysis in this idealized model has been quite successful in ensuring security guarantees of numerous cryptographic schemes provided that the hash function has no weakness. Security proofs in this model are superior to those provided by *ad hoc* protocol designs although they do not, of course, provide the same security guarantees as those in the standard model.

3.3 The Group Diffie-Hellman Problems

The Group Diffie-Hellman schemes have traditionally been designed based on different intractability assumptions. The schemes of [13, 18] are based on heuristic assumptions that are not known to be reducible to a well-known "hard" problem. The schemes of [20, 22, 27] are based on assumptions that are reducible to a well-known "hard" problem.

In a cyclic prime-order group $\langle g \rangle$, the "standard" assumptions that have been used so far are:

1. The Decisional Diffie-Hellman (DDH) assumption. Under this assumption, distinguishing g^{ab} from a random value when given g^a and g^b is computationally hard.
2. The Group Decisional Diffie-Hellman (G-DDH) assumption. One considers the elements $g^{\prod x_i}$ for some subsets of indices i (either all these subsets, except $\{1, \ldots, n\}$, or only a part of them) and tries to distinguish $g^{x_1 \cdots x_n}$ from a random value.

In the ideal hash function model, one usually uses the CDH and G-CDH assumptions:

1. The Computational Diffie-Hellman (CDH) assumption. This assumption claims that given two elements g^a, g^b , it is computationally hard to compute g^{ab}.
2. The Group Computational Diffie-Hellman (G-CDH) assumption. In the G-CDH problem, one considers the elements $g^{\prod x_i}$ for some subsets of indices i (either all these subsets, except $\{1, .., n\}$, or only a part of them) and tries to compute $g^{x_1 \cdots x_n}$. G-CDH is believed to be a "hard" problem.

The G-DDH problem appears to have first surfaced in the cryptographic literature in the paper of Steiner et al. [31] which also proves that the DDH assumption implies the G-DDH assumption. Since then, the G-DDH has been used in several other cryptographic settings [12, 25].

The G-CDH assumption is a potentially weaker intractability assumption than G-DDH. It is also believed that the CDH assumption implies the G-CDH assumption but it has not yet been proved. The G-CDH has however, when considered modulo a composite number, been related to factoring [9].

4 Model

In our model, the adversary $\mathcal{A}$, which is not a player in our formalization, is given enormous capabilities. It controls all communications between player instances and can at any time ask an instance to release a session key or a long-lived key. In the rest of this section we formalize the protocol and the adversary's capabilities.

4.1 Protocol Participants

We fix a nonempty set ID of n players that want (and are supposed) to participate in a group Diffie-Hellman protocol P. The number n of players is polynomial in the security parameter k.

A player $U_i \in ID$ can have many *instances* called oracles, involved in distinct concurrent executions of P. We denote instance s of player U_i as Π_i^s with $s \in \mathbb{N}$. Also, when we mean a not fixed member of ID we use U without any index and so denote an instance of U as Π_U^s with $s \in \mathbb{N}$.

4.2 Long-Lived Keys

Each player $U \in ID$ holds a long-lived key LL_U which is either a pair of matching public/private keys or a symmetric key. LL_U is specific to U not to one of its instances. Associated to protocol P is a LL-key generator $\mathcal{G}_{LL}$ which at initialization generates LL_U and assigns it to U.

4.3 Session IDS

We define the session IDS (SIDS) for oracle Π_i^s in an execution of protocol P as $\mathrm{SIDS}(\Pi_i^s) = \{\mathrm{SID}_{ij} : j \in ID\}$ where SID_{ij} is the concatenation of all flows that oracle Pi_i^s exchanges with oracle Π_j^t (possibly by the intermediate of $\mathcal{A}$) in an execution of P. We emphasize that SIDS is public – it does not depend on the session key – and, thus, is available to the adversary $\mathcal{A}$; $\mathcal{A}$ can just listen on the wire and construct it. We will use SIDs to properly define partnering through the notion of partners IDs (PIDs).

4.4 Accepting and Terminating

An oracle Π_U^s accepts when it has enough information to compute a session key SK. At any time an oracle Π_U^s can accept and it accepts at most once. As soon as oracle Π_U^s accepts, SK and SIDS are defined. Now once having accepted Π_U^s has not yet terminated. Π_U^s may want to get confirmation that its partners have actually computed SK or that its partners are really the ones it wants to share a session key with. As soon as Π_U^s gets this confirmation message, it terminates – it will not send out any more messages.

4.5 Oracle Queries

The adversary $\mathcal{A}$ has an endless supply of oracles Π_U^s and makes various queries to them. Each query models a capability of the adversary. The four queries and their responses are listed below:

- $\mathsf{Send}(\Pi_U^s, m)$: This query models adversary $\mathcal{A}$ sending messages to instances of players. The adversary $\mathcal{A}$ gets back from his query the response which oracle Π_U^s would have generated in processing message m. If oracle Π_U^s has not yet terminated and the execution of protocol P leads to accepting, variables SIDS are updated. A query of the form $\mathsf{Send}(\Pi_U^s,$ "`start`") initiates an execution of P.
- $\mathsf{Reveal}(\Pi_U^s)$: This query models the attacks resulting in the session key being revealed. The Reveal query is only available to adversary $\mathcal{A}$ if oracle Π_U^s has accepted. The Reveal-query unconditionally forces Π_U^s to release SK which otherwise is hidden to the adversary.

- $\mathsf{Corrupt}(U)$: This query models the attacks resulting in the player U's LL-key been revealed. Adversary $\mathcal{A}$ gets back LL_U but does not get the internal data of any instances of U executing P.
- $\mathsf{Test}(\Pi_U^s)$: This query models the semantic security of the session key SK, namely the following game, denoted by $\mathbf{Game}^{ake}(\mathcal{A}, P)$, between adversary $\mathcal{A}$ and the oracles Π_U^s involved in the executions of P. During the game, $\mathcal{A}$ can ask any of the above queries, and once, asks a Test-query. Then, one flips a coin b and returns SK if $b = 1$ or a random string if $b = 0$. At the end of the game, adversary $\mathcal{A}$ outputs a bit b' and *wins* the game if $b = b'$. The Test-query is asked only once and is only available if Π_U^s is **Fresh** (see section 5).

4.6 Executing the Protocol in the Presence of an Adversary

Choose a protocol P with a session-key space **SK**, and an adversary $\mathcal{A}$. The security definitions take place in the context of making $\mathcal{A}$ play the above game $\mathbf{Game}^{ake}(\mathcal{A}, P)$. P determines how Π_U^s behaves in response to messages from the environment. $\mathcal{A}$ sends these messages: it controls all communications between instances; it can at any time force an oracle Π_U^s to divulge SK or more seriously LL_U; it can initiate simultaneous executions of P. This game is initialized by providing coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all Π_U^s, and running $\mathcal{G}_{LL}(1^k)$ to set LL_U. Then

1. Initialize any Π_U^s to SIDS $\leftarrow$ NULL, PIDS $\leftarrow$ NULL, SK $\leftarrow$ NULL.
2. Initialize adversary $\mathcal{A}$ with 1^k and access to any Π_U^s,
3. Run adversary $\mathcal{A}$ and answer oracle queries as defined above.

4.7 Discussion

The group Diffie-Hellman-like protocols [2, 3, 13, 18, 20, 27, 31] are generally specified using the broadcast communication primitive; the broadcast primitive allows a player to send messages to an arbitrary pool of players in a single round. However such a communication convention is irrelevant to our notions of security; for example, one can always turn a broadcast-based protocol P into a protocol P' which sends only one message in each round and which still meets our definitions of security as long as P does.

The group Diffie-Hellman-like protocols also employ a different connectivity graph (e.g, ring or tree) to route messages among players. The connectivity graph allows the protocols to meet specific performance attributes. However the way the messages are routed among players does not impact our security definitions; one can always turn a protocol P into a protocol P' that differs only in its message routing.

5 Definitions of Security

In this section we present the definitions that should be satisfied by a group Diffie-Hellman scheme and what *breaking* a group Diffie-Hellman scheme means. We uniquely define the partnering from the session IDS and, thus, it is publicly available to the adversary[3]. We present each definition in a systematic way: we give an intuition and then formalize it.

[3] In the definition of partnering, we do not require that the session key SK computed by partnered oracles be the same since it can easily be proven that the probability that **partnered** oracles come up with different SK is negligible (see Section 7.4).

Recall that forward-secrecy entails that loss of a LL-key does not compromise the semantic security of previously-distributed session keys. For the purpose of this paper, we only consider a *weak corruption* model, in which the adversary obtains only the long-lived key and not any *internal data* (i.e. random bits used by a process). Let's also recall that a function $\epsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that for all $k > k_c$, $\epsilon(k) < k^{-c}$.

5.1 Partnering using SIDS

The partnering definition captures the intuitive notion that the players with which oracle Π_i^s has exchanged messages are the players with which Π_i^s believes it has established a session key. Another simple way to understand the notion of partnering is that an instance t of a player U_j is a partner of oracle Π_i^s if Π_j^t and Π_i^s have directly exchanged messages or there exists some sequence of oracles that have directly exchanged messages from Π_j^t to Π_i^s.

After many executions of P, or in $\mathbf{Game}^{ake}(\mathcal{A}, P)$, we say that oracles Π_i^s and Π_j^t are **directly partnered** if both oracles accept and $\mathrm{SIDS}(\Pi_i^s) \cap \mathrm{SIDS}(\Pi_j^t) \neq \emptyset$ holds. We denote the direct partnering as $\Pi_i^s \leftrightarrow \Pi_j^t$.

We also say that oracles Π_i^s and Π_j^t are **partnered** if both oracles accept and if, in the graph $G_{SIDS} = (V, E)$ where $V = \{\Pi_U^s \ : \ U \in ID, \quad i = 1, \ldots, n\}$ and $E = \{(\Pi_i^s, \Pi_j^t) \ : \ \Pi_i^s \leftrightarrow \Pi_j^t\}$ the following holds:

$$\exists k > 1, \prec \Pi_1^{s_1}, \Pi_2^{s_2}, \ldots, \Pi_k^{s_k} \succ$$

with :

$$\Pi_1^{s_1} = \Pi_i^s, \ \Pi_k^{s_k} = \Pi_j^t, \ \Pi_{i-1}^{s_{i-1}} \leftrightarrow \Pi_i^{s_i}.$$

We denote this partnering as $\Pi_i^s \rightsquigarrow \Pi_j^t$.

We complete in polynomial time (in $|V|$) the graph G_{SIDS} to obtain the graph of partnering : $G_{PIDS} = (V', E')$, where $V' = V$ and $E' = \{(\Pi_i^s, \Pi_j^t) \ : \ \Pi_i^s \rightsquigarrow \Pi_j^t\}$ (see [15] for graph algorithms), and then define the partner IDS for oracle Π_i^s as:

$$\mathrm{PIDS}(\Pi_i^s) = \{\Pi_j^t \ : \ \Pi_i^s \rightsquigarrow \Pi_j^t\}$$

Although the above definitions may appear quite artificial, we emphasize that the authentication goals need to be defined from essentially public criteria (in other words, from the partnering notion). Claiming "players are mutually authenticated iff they hold the same SK" would lead to unpractical definitions. The mutual authentication is essentialy a public, verifiable notion.

5.2 Freshness

The freshness definition captures the intuitive notion that a session key SK is defined **Fresh** if no oracle is corrupted at that moment, and it remains **Fresh** if no Reveal-query is asked later to the oracle or one of its partners. More precisely, an oracle Π_U^s is **Fresh** (or holds a **Fresh** SK) if the following four conditions hold: First, Π_U^s has accepted. Second, nobody has been asked for a Corrupt-query before Π_U^s accepts. Third, Π_U^s has not been asked for a Reveal-query. Fourth, the partners of Π_U^s, $\mathrm{PIDS}(\Pi_U^s)$ have not been asked for a Reveal-query.

5.3 AKE Security

In an execution of P, we say an adversary $\mathcal{A}$ (computationally bounded) *wins* if she asks a single Test-query to a **Fresh** oracle and correctly guesses the bit b used in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$. We denote the **ake advantage** as $\mathsf{Adv}_P^{ake}(\mathcal{A})$; the advantage is taken over all bit tosses. Protocol P is an $\mathcal{A}$**-secure AKE** if $\mathsf{Adv}_P^{ake}(\mathcal{A})$ is negligible.

5.4 Authentication Security

This definition of authentication captures the intuitive notion that it should be hard for a computationally bounded adversary $\mathcal{A}$ to impersonate a player U through one of its instances Π_U^s.

In an execution of P, we say adversary $\mathcal{A}$ violates player-to-players authentication (PPsA) for oracle Π_U^s if Π_U^s terminates holding $\mathrm{SIDS}(\Pi_U^s)$, $\mathrm{PIDS}(\Pi_U^s)$ and $|\mathrm{PIDS}(\Pi_U^s)| \neq n-1$. We denote the **ppsa probability** as $\mathsf{Succ}_P^{ppsa}(\mathcal{A})$ and say protocol P is an $\mathcal{A}$**-secure PPsA** if $\mathsf{Succ}_P^{ppsa}(\mathcal{A})$ is negligible.

In an execution of P, we say adversary $\mathcal{A}$ violates mutual authentication (MA) if $\mathcal{A}$ violates PPsA authentication for at least one oracle Π_U^s. We name the probability of such an event the **ma success** $\mathsf{Succ}_P^{ma}(\mathcal{A})$ and say protocol P is an $\mathcal{A}$**-secure MA** if $\mathsf{Succ}_P^{ma}(\mathcal{A})$ is negligible.

Therefore to deal with mutual authentication (or player-to-players authentication in a similar way), we consider a new game $\mathbf{Game}^{ma}(\mathcal{A}, P)$ in which the adversary exactly plays the same way as in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ with the same oracle accesses but with a different goal: to violate the mutual authentication. In this new game, the adversary is not really interested in the Test-query, in the sense that it can terminate whenever he wants. However, we leave this query available for simplicity.

5.5 Secure Signature Schemes

A signature scheme is defined by the following [28]:

- Key generation algorithm $\mathcal{G}$. On input 1^k with security parameter k, the algorithm $\mathcal{G}$ produces a pair (K_p, K_s) of matching public and secret keys. Algorithm $\mathcal{G}$ is probabilistic.
- Signing algorithm Σ. Given a message m and (K_p, K_s), Σ produces a signature σ. Algorithm Σ might be probabilistic.
- Verification algorithm V. Given a signature σ, a message m and K_p, V tests whether σ is a valid signature of m with respect to K_s. In general, algorithm V is not probabilistic.

The signature scheme is (t, ϵ)-**CMA-secure** if there is no adversary $\mathcal{A}$ which can get a probability greater than ϵ in mounting an existential forgery under an adaptively chosen-message attack (CMA) within time t. We denote this probability ϵ as $\mathsf{Succ}_\Sigma^{cma}(\mathcal{A})$.

5.6 Decisional and Computational Diffie-Hellman Assumptions

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and x_1, x_2, r chosen at random in $\mathbb{Z}_p$. A (T, ϵ)-DDH-distinguisher for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given any triplet (g^{x_1}, g^{x_2}, g^r) outputs "`True`" or "`False`" such that:

$$\Big| \Pr[\Delta(g^{x_1}, g^{x_2}, g^{x_1 x_2}) = \text{“}\texttt{True}\text{”}] \; - \; \Pr[\Delta(g^{x_1}, g^{x_2}, g^r) = \text{“}\texttt{True}\text{”}] \Big| \geq \epsilon$$

We denote this difference of probabilities as $\mathsf{Adv}^{ddh}_{\mathbb{G}}(\Delta)$. The DDH problem is (T,ϵ)-**intractable** if there is no (T,ϵ)-DDH-distinguisher for $\mathbb{G}$.

A (T,ϵ)-CDH-attacker for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given (g^{x_1}, g^{x_2}), outputs $g^{x_1 x_2}$ with probability at least $\epsilon = \mathsf{Succ}^{cdh}_{\mathbb{G}}(\Delta)$. The CDH problem is (T,ϵ)-**intractable** if there is no (T,ϵ)-attacker for $\mathbb{G}$.

5.7 Group Computational Diffie-Hellman Assumption (G-CDH)

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and n be a polynomially-bounded integer. Let I_n be $\{1,\ldots,n\}$, $\mathcal{P}(I_n)$ be the set of all subsets of I_n and Γ be a subset of $\mathcal{P}(I_n)$ such that $I_n \notin \Gamma$.

We define the *Group Diffie-Hellman distribution* relative to Γ as:

$$G\text{-}CDH_{\Gamma} = \Big\{ \bigcup_{J\in\Gamma} (J, g^{\prod_{j\in J} x_j}) \mid (x_1,\ldots,x_n) \in_R \mathbb{Z}_p^n \Big\}$$

If $\Gamma = \mathcal{P}(I)\backslash\{I_n\}$, we say that $G\text{-}CDH_{\Gamma}$ is the **Full** *Generalized Diffie-Hellman distribution* [12, 25, 31].

Given Γ, a (T,ϵ) G-CDH$_{\Gamma}$-attacker for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given $G\text{-}CDH_{\Gamma}$ outputs $g^{x_1\cdots x_n}$ with probability at least ϵ. We denote this probability by $\mathsf{Succ}^{gcdh}_{\mathbb{G}}(\Delta)$. The G-CDH$_{\Gamma}$ problem is (T,ϵ)-**intractable** if there is no (T,ϵ)-G-CDH$_{\Gamma}$-attacker for $\mathbb{G}$.

In the same way, we can define a G-DDH$_{\Gamma}$ distinguisher as a probabilistic Turing machine that given G-CDH$_{\Gamma}$ and either $g^{x_1\cdots x_n}$ or a random value, can distinguish the two situations with non-negligible probability.

5.8 Adversary's Resources.

The security is formulated as a function of the amount of resources the adversary $\mathcal{A}$ expends. The resources are:

- t time of computing;
- q_{se}, q_{re}, q_{co} number of Send, Reveal and $\mathsf{Corrupt}$ queries adversary $\mathcal{A}$ respectively makes.

By notation $\mathsf{Adv}(t,\ldots)$ or $\mathsf{Succ}(t,\ldots)$, we mean the maximum values of $\mathsf{Adv}(\mathcal{A})$ or $\mathsf{Succ}(\mathcal{A})$ respectively, over all adversaries $\mathcal{A}$ that expend at most the specified amount of resources.

6 A Secure Authenticated Group Diffie-Hellman Scheme

We first introduce the protocol AKE1 and then prove it is a secure AKE scheme in the ideal hash model. Then at the end of this section we comment on the security theorem and the proof.

6.1 Preliminaries

In the following we assume the ideal hash function model. We use a hash function $\mathcal{H}$ from $\{0,1\}^*$ to $\{0,1\}^{\ell}$ where ℓ is a security parameter. The session-key space **SK** associated to this protocol is $\{0,1\}^{\ell}$ equipped with a uniform distribution. In this model,

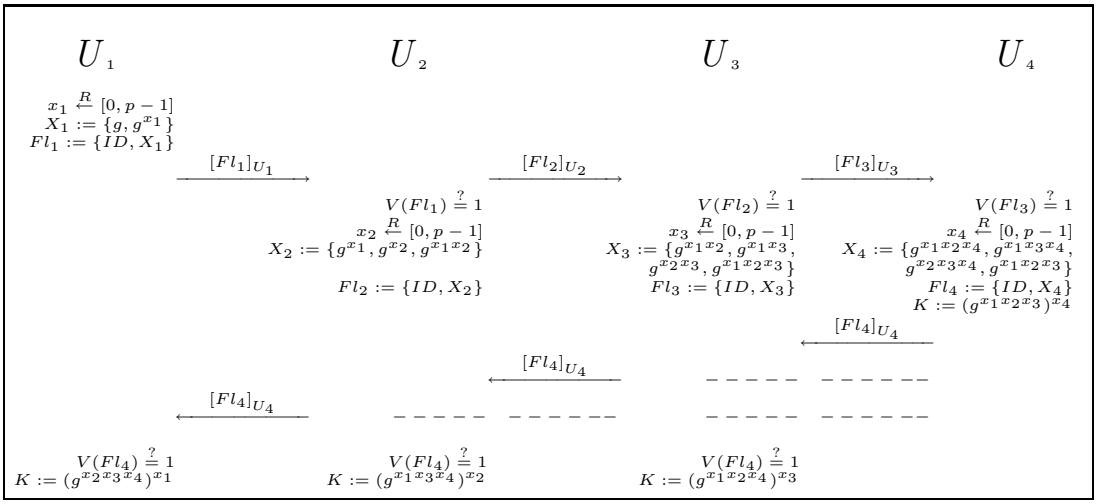

Fig. 1. Protocol AKE1. An example of a honest execution with 4 players: $ID = \{U_1, U_2, U_3, U_4\}$. The shared session key SK is $sk = \mathcal{H}(U_1, U_2, U_3, U_4, Fl_4, g^{x_1 x_2 x_3 x_4})$.

a new query, namely Hash-query is available to adversary $\mathcal{A}$; the adversary can submit an arbitraly long bit string and obtain the value of $\mathcal{H}(m)$.

Arithmetic is in a finite cyclic group $\mathbb{G} =< g >$ of order a k-bit prime number q. This group could be a prime subgroup of $\mathbb{Z}_p^*$, or it could be an (hyper)-elliptic curve group. We denote the operation multiplicatively.

6.2 Description of AKE1

This is a protocol in which the players $ID = \{U_i : 1 \leq i \leq n\}$ are arranged in a ring, the name of the players are in the protocol flows, the flows are signed using the long-lived key LL_U, the session key SK is $sk = \mathcal{H}(ID, Fl_n, g^{x_1 \cdots x_n})$, where Fl_n is the downflow; SIDS and PIDS are appropriately defined.

As illustrated by the example on Figure 1, the protocol consists of two stages: up-flow and down-flow. In the up-flow the player raises the received intermediate values to the power of its private input and forwards the result to the next player in the ring. The down-flow takes place when U_n receives the last up-flow and computes sk. U_n raises the intermediate values it has received to the power of its private key and broadcasts the result (i.e. Fl_n) which allows the other players to construct sk, granted their private data.

6.3 Security Theorem

Let P be the AKE1 protocol, $\mathcal{G}_{LL}$ be the associated LL-key generator. One can state the following security result:

Theorem 1. *Let $\mathcal{A}$ be an adversary against the AKE security of protocol P within a time bound t, after q_{se} interactions with the parties and q_h hash queries. Then we have:*

$$\mathsf{Adv}_P^{ake}(t, q_{se}, q_h) \leq 2q_h q_{se}^n \cdot \mathsf{Succ}_{\mathbb{G}}^{gcdh_\Gamma}(t') + n \cdot \mathsf{Succ}_{\Sigma}^{cma}(t'')$$

where $t' \leq t + q_{se} n T_{exp}(k)$ and $t'' \leq t + q_{se} n T_{exp}(k)$; $T_{exp}(k)$ is the time of computation required for an exponentiation modulo a k-bit number and Γ corresponds to the

elements adversary $\mathcal{A}$ can possibly view:

$$\Gamma = \bigcup_{1 \leq j \leq n} \{\{i \,|\, 1 \leq i \leq j, i \neq l\} \,|\, 1 \leq l \leq j\}$$

Before describing the details of the proof let us first provide the main ideas. We consider an adversary $\mathcal{A}$ attacking the protocol P and then "breaking" the AKE security. $\mathcal{A}$ would have carried out her attack in different ways: (1) she may have got her advantage by changing the content of the flows, hence forging a signature with respect to some player's long-lived public key (otherwise, the player would have rejected). We will then use $\mathcal{A}$ to build a forger by "guessing" for which player $\mathcal{A}$ will produce her forgery. (2) she may have broken the scheme without altering the content of the flows. We will use it to solve an instance of the G-CDH problem, by "guessing" the moment at which $\mathcal{A}$ will make the Test-query and by injecting into the game the elements from the G-CDH instance received as input.

6.4 Security Proof

Proof. Let $\mathcal{A}$ be an adversary that can get an advantage ϵ in breaking the AKE security of protocol P within time t. We construct from it a (t'', ϵ'')-forger $\mathcal{F}$ and a (t', ϵ')-G-CDH$_\Gamma$-attacker Δ .

Forger $\mathcal{F}$. Let's assume that $\mathcal{A}$ breaks the protocol P because she forges a signature with respect to some player's (public) LL-key and she is able to do it with probability greater than ν. We construct from it a (t'', ϵ'')-forger $\mathcal{F}$ which outputs a forgery (σ, m) with respect to a given (public) LL-key K_p (Of course K_p was produced by $\mathcal{G}_{LL}(1^k)$).

$\mathcal{F}$ receives as input K_p and access to a (public) signing oracle. $\mathcal{F}$ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$ and all Π_U^s. $\mathcal{F}$ picks at random $i \in [1, n]$ and runs $\mathcal{G}_{LL}(1^k)$ to set the players' LL-keys. However for player i, $\mathcal{F}$ sets LL_i to K_p. $\mathcal{F}$ then starts running $\mathcal{A}$ as a subroutine and answers the oracle queries made by $\mathcal{A}$ as explained below. $\mathcal{F}$ also uses a variable $\mathcal{K}$, initially set to $\emptyset$.

When $\mathcal{A}$ makes a Send-query, $\mathcal{F}$ answers in a straightforward way, using LL-keys to sign the flows, except if the query is of the form $\mathsf{Send}(\Pi_i^s, *)$ $(\forall s \in \mathbb{N})$. In this latter case the answer goes through the signing oracle, and $\mathcal{F}$ stores in $\mathcal{K}$ the request to the signing oracle and the signing oracle response. When $\mathcal{A}$ makes a Reveal-query or a Test-query, $\mathcal{F}$ answers in a straightforward way. When $\mathcal{A}$ makes a Corrupt-query, $\mathcal{F}$ answers in a straightforward way except if the query is of the form $\mathsf{Corrupt}(\Pi_i^s)$ $(\forall s \in \mathbb{N})$. In this latter case, since $\mathcal{F}$ does not know the LL-key K_s for player i, $\mathcal{F}$ stops and outputs "`Fail`". But anyway, no signature forgery occurred before, and so, such an execution can be used with the other reduction. When $\mathcal{A}$ makes a Hash-query, $\mathcal{F}$ answers the query as depicted on Figure 2.

If $\mathcal{A}$ has made a query of the form $\mathsf{Send}(*, (\sigma, m))$ where σ is a valid signature on m with respect to K_p and $(\sigma, m) \notin \mathcal{K}$, then $\mathcal{F}$ halts and outputs (σ, m) as a forgery. Otherwise the process stops when $\mathcal{A}$ terminates and $\mathcal{F}$ outputs "`Fail`".

The probability that $\mathcal{F}$ outputs a forgery is the probability that $\mathcal{A}$ produces a valid flow by itself multiplied to the probability to "correctly guess" the value of i:

$$\mathsf{Succ}_{\Sigma}^{cma}(\mathcal{F}) \geq \frac{\nu}{n}$$

The runnning time of $\mathcal{F}$ is the running time of $\mathcal{A}$ added to the time to process the Send-queries. This is essentially a constant value. This gives the formula for t:

$$t'' \leq t + q_{se} n T_{exp}(k)$$

G-CDH$_\Gamma$-attacker Δ. Let's assume that $\mathcal{A}$ gets its advantage without producing a forgery. (Here with probability greater than ν the valid flows signed with LL_U come from oracle U before U gets corrupted and not from $\mathcal{A}$.) We construct from $\mathcal{A}$ a (t', ϵ')-G-CDH$_\Gamma$-attacker Δ which receives as input an instance of $G\text{-}CDH_\Gamma$ and outputs the group Diffie-Hellman secret value relative to this instance.

Δ receives as input an instance $\mathcal{D} = (\{\{1\}, g^{x_1}), (\{2\}, g^{x_2}), \ldots, Fl_n)$ of the G-CDH$_\Gamma$ problem, where Fl_n are the terms corresponding to subsets of indices of cardinality $n-1$ (with the same structure as in the broadcast). Δ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all Π^s_U, and runs $\mathcal{G}_{LL}(1^k)$ to set the players' LL-keys. Δ picks at random n values u_1 through u_n in $[1, q_{se}]^n$. Then Δ starts running $\mathcal{A}$ as a subroutine and answers the oracle queries made by $\mathcal{A}$ as explained below. Δ uses a set of counters c_i through c_n, initially set to zero.

When $\mathcal{A}$ makes a Send-query to some instance of player U_i, then Δ increments c_i and proceeds as in protocol P using a random value. However if $c_i = u_i$ and m is the flow corresponding to the instance $\mathcal{D}$, Δ answers using the elements from the instance $\mathcal{D}$. When $\mathcal{A}$ makes a Corrupt-query, Δ answers in a straightforward way. When $\mathcal{A}$ makes a Hash-query, $\mathcal{F}$ answers the query as depicted on Figure 2. When $\mathcal{A}$ makes a Reveal-query, Δ answers in straightforward way. However, if the session key has to be constructed from the instance $\mathcal{D}$, Δ halts and outputs "`Fail`". When $\mathcal{A}$ makes the Test-query, Δ answers with a random string.

We emphasize that, since Δ knows all the keys except for one execution of P (i.e. the execution involving $\mathcal{D}$ in all flows), this simulation is perfectly indistinguishable from an execution of the real protocol P.

The probability that Δ correctly "guesses" on which session key $\mathcal{A}$ will make the Test-query is the probability that Δ correctly "guesses" the values u_1 through u_n. That is:

$$\delta = \prod_n \frac{1}{q_{se}} = \frac{1}{q_{se}^n}$$

In this case, Δ is actually able to answer to all Reveal-queries, since Reveal-query must be asked to a **Fresh** oracle, holding a key different from the Test-ed one, and thus, known to Δ.

Then, when $\mathcal{A}$ terminates outputting a bit b', Δ looks in the $\mathcal{H}$-list to see if some queries of the form $\mathsf{Hash}(U_1, \ldots, U_n, Fl_n, *)$ have been asked. If so, Δ chooses at random one of them, halts and outputs the remaining part "$*$" of the query.

Let Ask be the event that $\mathcal{A}$ makes a Hash-query on $(U_1, ..., U_n, Fl_n, g^{x_1 \cdots x_n})$. The advantage of $\mathcal{A}$ in breaking the AKE security without forging a signature, conditioned by the fact that we correctly guessed all u_i's, is:

$$\begin{aligned}
\frac{\epsilon - \nu}{q_{se}^n} &\leq \mathsf{Adv}_P^{ake}(\mathcal{A}) = 2\Pr[b = b'] - 1 \\
&= 2\Pr[b = b' | \neg\mathsf{Ask}] \Pr[\neg\mathsf{Ask}] \quad + \\
&\qquad 2\Pr[b = b' | \mathsf{Ask}] \Pr[\mathsf{Ask}] \; - \; 1 \\
&\leq 2\Pr[b = b' | \neg\mathsf{Ask}] - 1 + 2\Pr[\mathsf{Ask}] = 2\Pr[\mathsf{Ask}]
\end{aligned}$$

In the random oracle model, $2\Pr[b = b'|\neg\mathsf{Ask}] - 1 = 0$, since $\mathcal{A}$ can not gain any advantage on a random value without asking for it.

The success probability of Δ is the probability that $\mathcal{A}$ asks the correct value to the hash oracle multiplied by the probability that Δ correctly chooses among the possible Hash-queries:

$$\mathsf{Succ}_{\mathbb{G}}^{gcdh_\Gamma}(\Delta) \geq \frac{\Pr[\mathsf{Ask}]}{q_h} \geq \frac{\epsilon - \nu}{2q_{se}^n} \times \frac{1}{q_h}$$

The runnning time of Δ is the running time of $\mathcal{A}$ added to the time to process the Send-queries. This is essentially n modular exponentiation computation per Send-query. Then

$$t' \leq t + q_{se} n T_{exp}(k)$$

Hash function $\mathcal{H}$	
$\xrightarrow{\text{query } m}$	If $m \notin \mathcal{H}$-list, then $r \overset{R}{\leftarrow} \in \{0,1\}^\ell$, and $\mathcal{H}$-list $\leftarrow \mathcal{H}$-list$\|(m, r)$.
$\xleftarrow{\mathcal{H}(m)}$	Otherwise, r is taken from $\mathcal{H}$-list.

$\mathcal{H}$-list		
List	Members	Meaning
$\mathcal{H}$-list	(m, r)	$\mathcal{H}(m) = r$; Hash query has been made on m

Fig. 2. Hash-oracle simulation.

6.5 Result Analysis

The quality of the reduction measures how much security of the G-CDH and the signature scheme is injected into AKE1. We view q_{se} as an upper bound on the number of queries we are willing to allow (e.g., $q_{se} = 2^{30}$ and $q_h = 2^{60}$) and n as the number of participants involved in the execution of AKE1 (e.g., current scientific collaborations involve up to 20 participants). Moreover, because of the network latency and computation cost, the practicability of AKE1 becomes an issue with groups larger than 40 members operating in a wide-area environment [1].

We may then ask how the security proof is meaningful in practice. First, one has to be clear that such a proof of security is much better than no proof at all and that AKE1 is the first AKE scheme to have a proof of security. Second, several techniques can be used to carry out a proof which achieves a better (or tighter) security reduction.

In effect the reduction can be improved using a technique of Shoup [29]. Shoup's technique runs two attackers, similar to the one above, in parallel on two different instances obtained by random self-reducibility [25], and a common value will appear in the $\mathcal{H}$-list of the attackers with overwhelming probability and thus leads to the right solution for G-CDH.

The reduction can also be improved if the security of AKE1 is based on the G-DDH assumption. The idea is to use a technique similar to the one used by Coron [16] and to use the random self-reducibility of G-DDH$_\Gamma$ to generate many instances $\mathcal{D}'$ from

$\mathcal{D}$ such that all the $\mathcal{D}'$ lie in the same distribution as $\mathcal{D}$, either $G\text{-}DDH_\Gamma$ or R_Γ. Such instances are randomly used. But then, the resulting session key will be unknown. Therefore, the reduction will work if all the Reveal-queries are asked for known session keys, but the Test-query is asked to one involving an instance $\mathcal{D}'$. By correctly tuning the probability of using a $\mathcal{D}'$ instance or not, one can slightly improve the efficiency of the reduction[4]. Moreover, if the session key is simply fixed as $g^{x_1 \cdots x_n}$ the proof can be carried out in the standard model.

7 Adding Authentication

In this section we sketch generic transformations for turning an AKE protocol P into a protocol P' that provides player-to-players authentication (PPsA) and mutual authentication (MA). Then, we prove in the ideal hash model that the transformation provides a secure MA scheme and comment on the security theorem.

It may be argued that PPsA and MA are not absolutely necessary, can be achieved by a variety of means (e.g, encryption could begin on some carefully chosen known data) or even that MA does give real security guarantees in practice. However, the task of a cryptographic protocol designer is to make no assumptions about how system designers will use the session key and provide application developers with protocols requiring only a minimal degree of cryptographic awarness.

7.1 Approach

The well-known approach uses the shared session key to construct a simple "authenticator" for the other parties. However, one has to be careful in the details and this is a common "error" in the design of authentication protocols. Actually the protocols offered by Ateniese et al. [2] are seen insecure under our definitions since the "authenticator" is computed as the hash of the session key sk and sk is the same as the final session key SK. The adversary learns some information about the session key sk – the hash of sk – and can use it to distinguish SK from a session key selected at random from session-key space **SK**. Therefore these protocols sacrifice the security goal that a protocol establishes a semantically secure session key.

7.2 Description of the Transformations

The transformation AddPPsA (adding player-to-players authentication) for player U consists of adding to protocol P one more round in such a way that the partners of U are convinced they share sk with U. As an example, on figure 3 player U_n sends out $\mathcal{H}(sk, n)$.

More formally the transformation AddPPsA works as follows. Suppose that in protocol P player U_n has accepted holding $sk_{U_n}, sid_{U_n}, pid_{U_n}$ and has terminated. In protocol $P' = \text{AddPPs}(P)$, U_n sends out one additional flow $auth_{U_n} = \mathcal{H}(sk_{U_n}, n)$, accepts holding $sk'_{U_n} = \mathcal{H}(sk_{U_n}, 0)$, $sid'_{U_n} = sid_{U_n}$, $pid'_{U_n} = pid_{U_n}$, and then terminates. Suppose now that in P the partner U_i $(i \neq n)$ of U_n has accepted holding $sk_{U_i}, sid_{U_i}, pid_{U_i}$ and has terminated. In protocol P', U_i receives one additional flow $auth_{U_n}$ and checks if $auth_{U_n} = \mathcal{H}(sk_{U_i}, n)$. If so, then U_i accepts holding $sk'_{U_i} = \mathcal{H}(sk_{U_i}, 0), sid'_{U_i} = sid_{U_i}, pid'_{U_i} = pid_{U_i}$, and then terminates. Otherwise, U_i rejects.

[4] However, such a proof gets complicated when one adds in the concern of forward-secrecy. Instead the ideas in the proof of Section 6.4 can easily be extended to show that AKE1 guarantees forward-secrecy.

The transformation AddMA (add mutual authentication) is analogous to AddPPsA. It consists of adding to protocol P one more round of simultaneous broadcasts. More precisely, all the players U_i send out $\mathcal{H}(sk, i)$ and they all check the received values.

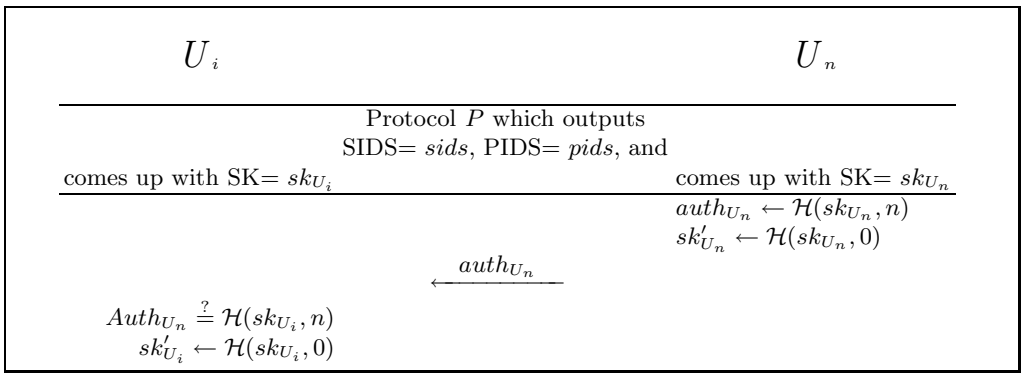

Fig. 3. Transformation P' = AddPPsA(P). The shared session key SK is $sk' = \mathcal{H}(sk, 0)$, SIDS and PIDS are unchanged.

7.3 Security Theorem

Let P be an AKE protocol, **SK** be the session-key space and $\mathcal{G}$ be the associated LL-key generator. One can state the following security result about P'=AddMA(P):

Theorem 2. *Let $\mathcal{A}$ be an adversary against the security of protocol P' within a time bound t, after q_{se} interactions with the parties and q_h hash queries. Then we have:*

$$\mathsf{Adv}^{ake}_{P'}(t, q_{se}, q_h) \leq \mathsf{Adv}^{ake}_{P}(t, q_{se}, q_h) + \frac{q_h}{2^\ell}$$
$$\mathsf{Succ}^{ma}_{P'}(t, q_{se}, q_h) \leq \mathsf{Adv}^{ake}_{P}(t', q_{se}, q_h) + \frac{nq_h}{2^\ell}$$

where $t' \leq t + (q_{se} + q_h)\mathcal{O}(1)$.

Before describing the details of the proof let us first provide the main ideas. We first show that the transformation AddMA preserves the AKE security (session key indistinguishability) of protocol P. We then show that impersonating a player in MA rounds implies for $\mathcal{A}$ to "fake" the authentication value $Auth_i$. Since this value goes through the hash function, it implies that $\mathcal{A}$ has computed the session key value sk and, thus, made the Hash-query.

7.4 Security Proof

Proof. Let $\mathcal{A}$ be an adversary that can get an advantage $\mathsf{Adv}^{ake}_{P'}(t, q_{se}, q_h)$ in breaking the AKE security of protocol P'=AddMA(P) within time t or can succeed with probability $\mathsf{Succ}^{ma}_{P'}(t, q_{se}, q_h)$ in breaking the MA security of protocol P'. We construct from it an attacker $\mathcal{B}$ that gets an advantage $\mathsf{Adv}^{ake}_{P}(t', q_{se}, q_h)$ in breaking the AKE security of protocol P within time t'.

Disrupt Partnering We are not concerned with partnered oracles coming up with different session keys, since our definition of partnering implies the oracles have exchanged *exactly* the same flows.

We also note that the probability that two instances of a given player come to be partnered is negligible; in fact, it would mean they have chosen the same random value in the protocols, which occurs with probability $\mathcal{O}(\frac{q_{se}^2}{2^k})$.

AKE break We construct from $\mathcal{A}$ an adversary $\mathcal{B}$ that gets an advantage ϵ' in breaking the AKE security of P within time t'.

$\mathcal{B}$ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all Π_U^s and starts running the game $\mathbf{Game}^{ake}(\mathcal{A}, P')$. $\mathcal{B}$ answers the queries made by $\mathcal{A}$ as follow.

The oracle queries made by $\mathcal{A}$ to $\mathcal{B}$ are relayed by $\mathcal{B}$ and the answers are subsequently returned to $\mathcal{A}$. However $\mathcal{B}$'s answers to Reveal and Test-queries go through the Hash-oracle to be padded with "0" before being returned to $\mathcal{A}$. The Hash-queries are answered as usual Figure 2.

In the ideal hash model, in which $\mathcal{H}$ is seen as a random function, $\mathcal{A}$ can not get any advantage in correctly guessing the bit involved in the Test-query without having made a query of the form $\mathcal{H}(sk, 0)$. So $\Pr[\mathcal{A} \text{ asks } (sk, 0)] \geq \mathsf{Adv}_P^{ake}(\mathcal{A}) \geq \epsilon$.

At some point $\mathcal{A}$ makes a Test-query to oracle Π_U^s, $\mathcal{B}$ gets value τ and relays $\mathcal{H}(\tau, 0)$ to $\mathcal{A}$. $\mathcal{B}$ then looks for τ in the $\mathcal{H}$-list: $\mathcal{B}$ outputs 1 if $(\tau, 0)$ is in the $\mathcal{H}$-list of queries made by $\mathcal{A}$, otherwise $\mathcal{B}$ flips a coin and outputs the coin value.

The advantage of $\mathcal{B}$ to win $\mathbf{Game}^{ake}(\mathcal{B}, P)$ is the probability that $\mathcal{A}$ made of query of the form $\mathcal{H}(sk, 0)$ minus the probability that $\mathcal{A}$ made such query by "pure chance":

$$\mathsf{Adv}_P^{ake}(\mathcal{B}) = \Pr\left[\mathcal{A} \text{ asks } (sk, 0)\right] - \frac{q_h}{2^\ell} \geq \mathsf{Adv}_{P'}^{ake}(\mathcal{A}) - \frac{q_h}{2^\ell}$$

The runnning time of $\mathcal{B}$ is the running time of $\mathcal{A}$ added to the time to process the Send-queries and Hash-queries:

$$t' \leq t + (q_{se} + q_h)\mathcal{O}(1)$$

MA break We construct from $\mathcal{A}$ an adversary $\mathcal{B}$ which gets advantage ϵ' in breaking the AKE security of P within time t'.

$\mathcal{B}$ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all Π_U^s, and starts running the game $\mathbf{Game}^{ma}(\mathcal{A}, P')$. $\mathcal{B}$ answers the oracle queries made by $\mathcal{A}$ as follows.

The oracle queries made by $\mathcal{A}$ to $\mathcal{B}$ are relayed by $\mathcal{B}$ and the answers are subsequently returned to $\mathcal{A}$. However $\mathcal{B}$'s answers to Reveal and Test-queries go through the Hash-oracle to be padded with "0" before being returned to $\mathcal{A}$. The Hash-queries are answered as usual Figure 2.

In the ideal hash model, in which $\mathcal{H}$ is seen as a random function, $\mathcal{A}$ can not get any advantage in impersonating some oracle $\Pi_i^{s_i}$ without having made a query of the form $\mathcal{H}(sk, i)$.

At some point $\mathcal{B}$ makes a Test-query to oracle Π_U^s and gets value τ. Later $\mathcal{A}$ terminates and $\mathcal{B}$ looks for τ in $\mathcal{H}$-list: $\mathcal{B}$ outputs 1 if $(\tau, *)$ is in $\mathcal{H}$-list, otherwise $\mathcal{B}$ flips a coin and outputs the coin value. (τ, i) is in $\mathcal{H}$-list if $\mathcal{A}$ violates PPsA for oracle $\Pi_i^{s_i}$ except with probability $q_h \cdot n \cdot \frac{1}{2^\ell}$.

The advantage of $\mathcal{B}$ to win $\mathbf{Game}^{ake}(\mathcal{B}, P)$ is the probability that $\mathcal{A}$ makes a query of the form $\mathcal{H}(sk, i)$:

$$\mathsf{Adv}_P^{ake}(\mathcal{B}) = \Pr\left[\mathcal{A} \text{ asks } (sk, i)\right] \geq \epsilon - \frac{nq_h}{2^\ell}$$

The runnning time of $\mathcal{B}$ is the running time of $\mathcal{A}$ added to the time to process the Send-queries and Hash-queries:

$$t' \leq t + (q_{se} + q_h)\mathcal{O}(1)$$

7.5 Result Analysis

The quality of the reduction measures how much security of the AKE security strength of protocol P is injected into protocol P'. We see that the reduction injects much of the security strength of protocol P into P'. In effect we can see it since $\mathsf{Adv}^{ake}_{P'}(t, q_{se}, q_h)$ ($\mathsf{Succ}^{ma}_{P'}(t, q_{se}, q_h)$ respectively) is inside an additive factor of $\mathsf{Adv}^{ake}_{P}(t, q_{se}, q_h)$ ($\mathsf{Adv}^{ake}_{P}(t', q_{se}, q_h)$ respectively) and this additive factor decreases exponentially with ℓ.

8 Conclusion

In this paper we presented a model for the group Diffie-Hellman key exchange problem derived from the model of Bellare et al. [5]. Some specific features of our approach that were introduced to deal with the Diffie-Hellman key exchange in the multi-party setting are: defining the notion of session IDS to be a set of session ID, defining the notion of partnering to be a graph of partner ID. Addressed in detail in this paper were two security goals of the group Diffie-Hellman key exchange: the authenticated key exchange and the mutual authentication. For each we presented a definition, a protocol and a security proof in the ideal hash model that the protocol meets its goals. This paper provided the first formal treatment of the authenticated group Diffie-Hellman key exchange problem.

The model and definitions introduced in this paper may seem limited at first sight. Our final goal is a model to help manage the complexity of definitions and proofs in the following broader scenario. A scenario in which the group membership is dynamic rather than static: after the initialization phase, and throughout the lifetime of the multicast group, the parties would like to engage in a conversation after each change in the membership at the end of which the session key, sk, is updated to sk'. The new session key is known to nobody but the parties in the multicast group. We are currently extending our model to encompass this larger scenario.

Acknowledgements

The authors thank Deborah Agarwal for many insightful comments on an early draft of this paper and the anonymous referees for their many useful comments.

References

1. D. A. Agarwal, O. Chevassut, M. Thompson, and G. Tsudik. An Integrated Solution for Secure Group Communication in Wide-Area Networks. In *Proc. of 6th Symposium on Computers and Communications*. IEEE Press, July 2001.
2. G. Ateniese, M. Steiner, and G. Tsudik. New Multiparty Authentication Services and Key Agreement Protocols. *Journal of Selected Areas in Communications*, 18(4):1–13, IEEE, 2000.
3. K. Becker and U. Wille. Communication Complexity of Group Key Distribution. In *Proc. of ACM CCS '98*, pages 1–6. ACM Press, 1998.

4. M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proc. of STOC '98*. ACM Press, 1998.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Proc. of Eurocrypt '00*, vol. 1807 of *LNCS*, pages 139–155. Springer, 2000.
6. M. Bellare and P. Rogaway. Entity Authentification and Key Distribution. In *Proc. of Crypto '93*, vol. 773 of *LNCS*. Springer, 1993.
7. M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *Proc of ACM CCS '93*. ACM Press, 1993.
8. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proc. of STOC '95*, ACM Press 1995.
9. E. Biham, D. Boneh, and O. Reingold. Breaking Generalized Diffie-Hellman Modulo a Composite is no Easier than Factoring. In *Information Processing Letters (IPL)*, vol. 70, pages 83–87, 1999.
10. S. Blake-Wilson, D. Johnson, and A. Menezes. Key Agreement Protocols and their Security Analysis. In *Proc. of 6th IMA International Conference on Crypotography and Coding*, vol. 1355 of *LNCS*, pages 30–45. Springer, 1997.
11. S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. In *Proc. of the 5th Annual Workshop on Selected Areas in Cryptography (SAC '98)*, vol. 1556 of *LNCS*, pages 339–361. Springer, 1998.
12. D. Boneh. The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, vol. 1423 of *LNCS*, pages 48–63. Springer, 1998.
13. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Proc of Eurocrypt' 94*, vol. 950 of *LNCS*, pages 275–286. Springer, 1995.
14. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Issues in Multicast Security: A Taxonomy and Efficient Constructions. In *Proc. of INFOCOM '99*, March 1999.
15. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science, 1990.
16. J.-S. Coron. On the Exact Security of Full-Domain-Hash. In *Proc. of Crypto' 2000*, vol. 1880 of *LNCS*, pages 229–235. Springer, August 2000.
17. W. Diffie and M. Hellman. New Directions In Cryptography. In *IEEE Transactions on Information Theory*, vol. IT-22(6), pages 644–654, November 1976.
18. W. Diffie, D. Steer, L. Strawczynski, and M. Wiener. A Secure Audio Teleconference System. In *Proc. of Crypto' 88*, vol. 403 of *LNCS*, pages 520–528. Springer, 1988.
19. S. Goldwasser and S. Micali. Probabilisitic Encryption. *Journal of Computer and System Sciences*, 28():270–299, 1984.
20. I. Ingemarsson, D. Tang, and C. Wong. A Conference Key Distribution System. In *IEEE Transactions on Information Theory*, volume 28(5), pages 714–720, September 1982.
21. M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. In *Proc. of ASIACRYPT'96*, vol. 1163 of *LNCS*, pages 36–49. Springer, 1996.
22. Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM CCS '00*, pages 235–244. ACM Press, November 2000.
23. A. Mayer and M. Yung. Secure Protocol Transformation via "Expansion" from Two-Party to Multi-Party. In *ACM CCS '99*, pages 83–92. ACM Press, November 1999.
24. C. Meadows. Extending Formal Cryptographic Protocol Analysis Techniques for Group Protocols and Low-Level Cryptographic Primitives. In *Workshop on Issues in the Theory of Security (WITS '00)*, 2000,.
25. M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In *Proc. of FOCS '97*, pages 458–467, IEEE Press, 1997.
26. O. Pereira and J. J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *14-th IEEE Computer Security Foundations Workshop*. IEEE Press, June 2001.
27. A. Perrig. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *International Workshop on Cryptographic Techniques and E-Commerce CrypTEC '99*, 1999.
28. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *J. of Cryptology*, 13(3):361–396, 2000.
29. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Proc. of Eurocrypt '97*, vol. 1233 of *LNCS*, pages 256–266. Springer, 1997.
30. V. Shoup. On Formal Models for Secure Key Exchange. Technical report, IBM Zrich Research Lab, 1999.
31. M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Groups. In *Proc. of ACM CCS '96*, ACM Press 1996.
32. W.-G. Tzeng. A Practical and Secure Fault-Tolerant Conference-Key Agreement Protocol. In *Proc. of PKC '2000*, LNCS. Springer, February 2000.