The extended abstract of this paper appeared in
Advances in Cryptology – Proceedings of Asiacrypt '2001 (9 – 13 december 2001, Gold Coast, Australia)
C. Boyd Ed. Springer-Verlag, LNCS 2248, pages 290–309.

Provably Authenticated Group Diffie-Hellman Key Exchange – The Dynamic Case (Full version)

Emmanuel Bresson[1], Olivier Chevassut[2,3]*, and David Pointcheval[1]

[1] École Normale Supérieure, 75230 Paris Cedex 05, France
http://www.di.ens.fr/∼ {bresson,pointche}, {Emmanuel.Bresson,David.Pointcheval}@ens.fr.
[2] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA,
http://www.itg.lbl.gov/∼chevassu, OChevassut@lbl.gov.
[3] Université Catholique de Louvain, 31348 Louvain-la-Neuve, Belgium.

Abstract. Dynamic group Diffie-Hellman protocols for Authenticated Key Exchange (AKE) are designed to work in a scenario in which the group membership is not known in advance but where parties may join and may also leave the multicast group at any given time. While several schemes have been proposed to deal with this scenario no formal treatment for this cryptographic problem has ever been suggested. In this paper, we define a security model for this problem and use it to precisely define Authenticated Key Exchange (AKE) with "implicit" authentication as the fundamental goal, and the entity-authentication goal as well. We then define in this model the execution of a protocol modified from a dynamic group Diffie-Hellman scheme offered in the litterature and prove its security.

1 Introduction

1.1 The Group Diffie-Hellman Key Exchange

Group Diffie-Hellman schemes for Authenticated Key Exchange are designed to provide a pool of players communicating over a public network and holding long-lived secrets with a session key to be used to achieve multicast message confidentiality or multicast data integrity. In this paper, we consider the scenario in which the group membership is not known in advance – *dynamic* rather than *static* – where parties may join and leave the multicast group at any given time.

After the initialization phase, and throughout the lifetime of the multicast group, the parties need to be able to engage in a conversation after each change in the membership at the end of which the session key is updated to be sk'. The secret value sk' is only known to the party in the multicast group during the period when sk' is the session key. The adversary may generate repeated and arbitrarily ordered changes in the membership for subsets of parties of his choice.

The above scenario is a distributed application in which up to one hundred parties work together in order to get a task done where many of the parties may be sending data to the multicast group [12]. Examples of such applications include replicated server [21], audio-video conferencing [20] and collaborative tools [2].

* 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-48202.

Several papers [3, 18, 19, 28] have addressed this scenario and one of its incarnations is the system offered in [1]. However these protocols, and this existing system, are based on or use an informal approach and do not rely on proofs of security. These approaches are several years later often found to be flawed and, indeed, weaknesses have already been discovered for some protocols [23]. One way to improve the security of the protocols is to complete formal proofs and thus avoid many of the weaknesses.

1.2 The Security Notions

In the paradigm of provable security [24] 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 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 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 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 only its partners actually have possession of the distributed session key.

With these security goals in hand the security of a group Diffie-Hellman scheme can be analyzed in the standard model or in an idealized model of computation (ideal-hash model [7, 13], ideal-cipher model [5], generic model [26]). Previous security analyses in the ideal-hash model, the so-called random-oracle model [7, 13] wherein the cryptographic hash functions (like SHA or MD5) are viewed as random functions, provide satisfactorily convincing guarantees of security for numerous cryptographic schemes [8, 14, 25] although not at the same level as those in the standard model.

1.3 Contributions

This paper provides major contributions to the solution of the group Diffie-Hellman key exchange problem. We present the first formal model to help manage the complexity of definitions and proofs for the authenticated group Diffie-Hellman key exchange when the group membership is *dynamic*. This model is equipped with some notions of dynamicity in the group membership where the various types of attacks are modeled by queries to the players. This model does not yet encompass attacks involving multiple player's instances activated concurrently and simultaneously by the adversary. Also, 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.

We start with the model and definitions introduced in [10] and extend them to deal with the authenticated *dynamic* group Diffie-Hellman key exchange. We define the partnering, freshness of session key and measures of security for AKE. In this model we define the execution of a protocol, we refer to it as AKE1, modified from [3] and show that it can be proven secure under reasonable and well-defined intractability assumptions.

Our paper is organized as follows. In the remainder of this section we summarize the related work. In Section 2 we define our security model. We use it in Section 3 to define the security definitions that should be satisfied by a group Diffie-Hellman scheme. We present the AKE1 protocol in Section 4 and justify its security in the random oracle model. Finally in Section 5 we briefly deal with MA in the random oracle model.

1.4 Related Work

Many group Diffie-Hellman protocols [3, 4, 11, 15, 17, 27, 29, 30] aim to distribute a session key among the multicast group members for a scenario in which the membership is *static* and known in advance. However these protocols are not well-suited for a scenario in which members join and leave the multicast group at a relatively high rate. Fortunately, these protocols can be extended to address this latter scenario and several papers [3, 18, 19, 28] have shown how to do so. The protocol presented in [3] has been found to be flawed in [23] and the other papers assume authenticated links, or more specifially do not consider the AKE and MA goals as part of the protocols. These goals need to be addressed separately.

A first step has already been taken toward a formal treatment of the authenticated Diffie-Hellman key exchange problem in the multi-party setting. Indeed, we presented in [10] the first formal model for this problem for a scenario in which the membership is *static*. The model was derived from Bellare et al.'s model of distributed computing [5, 16]. Addressed in detail were the AKE and MA goals. For each we presented a definition, a protocol and a proof that the protocol achieves these goals.

2 The Model

In this section we formalize the group Diffie-Hellman key exchange and the adversary's capabilities. In our formalization, the players do not deviate from the protocol, the adversary is not a player and the adversary's capabilities are modeled by various queries. These queries provide the adversary a capability to initialize a multicast group via Setup-queries, add players to the multicast group via Join-queries, and remove players from the multicast group via Remove-queries.

2.1 Protocol Participants.

We fix a nonempty set $\mathcal{U}$ of players that can participate in a group Diffie-Hellman key exchange protocol P. The number n of players is polynomial in the security parameter k. Also, when we mean a specific player of $\mathcal{U}$ we use U_i while when we mean a not fixed member of $\mathcal{U}$ we use U without any index.

We also consider a nonempty subset of $\mathcal{U}$ which we call the *multicast group* $\mathcal{I}$. And in $\mathcal{I}$ a player U_{GC}, the so-called "group controller", initiates the addition of players to $\mathcal{I}$ or the removal of players from $\mathcal{I}$. U_{GC} is trusted to do only this.

2.2 Long-Lived Keys

Each player $U \in \mathcal{U}$ holds a long-lived key LL_U which is either a pair of matching public/private keys or a symmetric key. Associated to protocol P is a LL-key generator $\mathcal{G}_{LL}$ which at initialization generates LL_U and assigns it to U.

2.3 Generic Group Diffie-Hellman Schemes

A group Diffie-Hellman scheme P for $\mathcal{U}$ is defined by four algorithms: (the session key SK is known by any player in $\mathcal{I}$ but unknown to any player not in $\mathcal{I}$.)

- the *key generation algorithm* $\mathcal{G}_{LL}$ which has an input of 1^k, where k is the security parameter, provides each player in $\mathcal{U}$ with a long-lived key LL_U. $\mathcal{G}_{LL}$ is a probabilistic algorithm.
- the *setup algorithm* which has an input of a set of players $\mathcal{J}$, sets variable $\mathcal{I}$ to be $\mathcal{J}$ and provides each player U in $\mathcal{I}$ with a session key SK_U. The setup algorithm is an interactive multi-party protocol between some players of $\mathcal{U}$.
- the *remove algorithm* which has an input of a set of players $\mathcal{J}$, updates variable $\mathcal{I}$ to be $\mathcal{I} \backslash \mathcal{J}$ (the set of all players in $\mathcal{I}$ that are not in $\mathcal{J}$) and provides each player U in this updated set with an updated session key SK_U. The remove algorithm is an interactive multi-party protocol between some players of $\mathcal{U}$.
- the *join algorithm* which has an input of a set of players $\mathcal{J}$, updates variable $\mathcal{I}$ to be $\mathcal{I} \cup \mathcal{J}$ and provides each player U in this updated set with an updated session key SK_U. The join algorithm is an interactive multi-party protocol between some players of $\mathcal{U}$.

An execution of P consists of running the *key generation* algorithm once, and then many times the *setup, remove* and *join* algorithms. We will also use the term *operation* to mean one of the algorithms: *setup, remove* or *join*.

Session IDS. We define the session IDS (SIDS) for player U_i in an execution of protocol P as $\mathrm{SIDS}(U_i) = \{\mathrm{SID}_{ij} : j \in ID\}$ where SID_{ij} is the concatenation of all flows that U_i exchanges with player U_j in executing an operation. Therefore, U_i setsSK_{U_i} to 0 and $\mathrm{SIDS}(U_i)$ and $\emptyset$ before executing an operation. (SIDS is publicly available.)

Accepting and Terminating. A player U accepts when it has enough information to compute a session key SK_U. At any time a player U who is in "expecting state" can accept and it accepts at most once in executing an operation. As soon as U accepts in executing an operation, SK and SIDS are defined.

Now once having accepted U has not yet terminated this execution. Player U may want to get confirmation that its partners in this execution have actually computed SK or that they are really the ones it wants to share a session key with. As soon as U gets this confirmation message, it terminates the execution of this operation - it will not send out any more messages and remains in a "stand by" state until the next operation.

2.4 Security Model

Queries. The adversary $\mathcal{A}$ interacts with the players U by making various queries. There are seven types of queries. The Setup, Join and Remove queries may at first seem useless since, using Send queries, the adversary already has the ability to initiate a *setup*, a *remove* or a *join* operation. Yet these queries are essential for properly dealing with the dynamic case. To deal with sequential membership changes, these three queries are only available if all the players in $\mathcal{U}$ have terminated. We now explain the capability that each kind of query captures.

- Setup($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *setup* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated and are thus in a "stand by" state.. $\mathcal{A}$ gets back from the first player U in $\mathcal{J}$ the flow initiating the *setup* execution. Other players are aware of the *setup* and move to an "expecting state" but do not reply any message.
- Remove($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *remove* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated. $\mathcal{A}$ gets back from the group controller U_{GC} the flow initiating the *remove* execution. Other players are aware of the *remove* operation but do not reply. They move from a "stand by" state to an "expecting state".
- Join($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *join* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated. $\mathcal{A}$ gets back from the group controller U_{GC} the flow initiating the *join* execution. Other players are aware of the *join* operation but do not reply. They move from a "stand by" state to an "expecting state".
- Send(U, m): This query models adversary $\mathcal{A}$ sending a message to a player. The adversary $\mathcal{A}$ gets back from his query the response which player U would have generated in processing message m (this could be the empty string if the message is uncorrect or unexpected). If player U has not yet terminated and the execution of protocol P leads to accepting, variable $\mathrm{SIDS}(U)$ is updated as explained above.
- Reveal(U): This query models the attacks resulting in the misuse of the session key, which may then be revealed. The query is only available to adversary $\mathcal{A}$ if player U has accepted. The Reveal-query unconditionally forces player U to release SK_U which is otherwise hidden to the adversary.
- Corrupt(U): This query models the attacks resulting in the player U's LL-key been revealed. $\mathcal{A}$ gets back LL_U but does not get any internal data of U executing P.

– $\mathsf{Test}(U)$: This query models the semantic security of the session key SK, namely the following game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ between adversary $\mathcal{A}$ and the players U involved in an execution of the protocol P. The Test-query is only available if U is **Fresh** (see Section 3). In the game $\mathcal{A}$ asks any of the above queries however it can only ask a Test-query once. Then, one flips a coin b and returns sk_U 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 $b = b'$.

Executing the Game. Choose a protocol P with a session-key space $\mathbf{SK}$, and an adversary $\mathcal{A}$. The security definitions take place in the context of making $\mathcal{A}$ play $\mathbf{Game}^{ake}(\mathcal{A}, P)$. P determines how players behave in response to messages from the environment. $\mathcal{A}$ sends these messages: she controls all communications between players; she can repeatedly initiate in a non-concurrent way but in arbitrary order sequential changes in the membership for subsets of players of her choice; she can at any time force a player U to divulge SK or more seriously LL_U. This game is initialized by providing coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all U, and running $\mathcal{G}_{LL}(1^k)$ to set LL_U. Then

1. Initialize any U with $\mathrm{SIDS} \leftarrow \textsc{null}, \mathrm{PIDS} \leftarrow \textsc{null}, \mathrm{SK} \leftarrow \textsc{null}$,
2. Initialize adversary $\mathcal{A}$ with 1^k and access to all U,
3. Run adversary $\mathcal{A}$ and answer queries made by $\mathcal{A}$ as defined above.

3 The Definitions

In this section we present the definitions that should be satisfied by a group Diffie-Hellman scheme. We define the partnering from the session IDS and use it to define security measurements that an adversary will defeat the security goals. We also recall that a function $\varepsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that for all $k > k_c$, $\varepsilon(k) < k^{-c}$.

3.1 Partnering using SIDS

The partnering captures the intuitive notion that the players with which U_i has exchanged messages in executing an operation, are the players with which U_i believes it has established a session key. Another simple way to understand the notion of partnering is that U_j is a partner of U_i in the execution of an operation, if U_j and U_i have directly exchanged messages or there exists some sequence of players that have directly exchanged messages from U_j to U_i.

In an execution of P, or in $\mathbf{Game}^{ake}(\mathcal{A}, P)$, we say that players U_i and U_j are **directly partnered** if both players accept and $\mathrm{SIDS}(U_i) \cap \mathrm{SIDS}(U_j) \neq \emptyset$ holds. We denote the direct partnering as $U_i \leftrightarrow U_j$.

We also say that players U_i and U_j are **partnered** if both players accept and if, in the graph $G_{SIDS} = (V, E)$ where $V = \{U_i \ : \ i = 1, \ldots, |\mathcal{I}|\}$ and $E = \{(U_i, U_j) \ : \ U_i \leftrightarrow U_j\}$ the following holds:

$$\exists k > 1, \prec U_1, U_2, \ldots, U_k \succ \ \text{with } U_1 = U_i,\ U_k = U_j,\ U_{i-1} \leftrightarrow U_i.$$

We denote this partnering as $U_i \leftrightsquigarrow U_j$.

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 $E' = \{(U_i, U_j) \ : \ U_i \leftrightsquigarrow U_j\}$, and then define the partner IDS for oracle U_i as:

$$\mathrm{PIDS}(U_i) = \{U_j \ : \ U_i \leftrightsquigarrow U_j\}$$

3.2 Freshness

A player U is **Fresh**, in the current operation execution, (or holds a **Fresh** SK) if the following two conditions are satisfied. First, nobody in $\mathcal{U}$ has ever been asked for a Corrupt-query from the beginning of the game. Second, in the current operation execution, U has accepted and neither U nor its partners PIDS(U) have been asked for a Reveal-query.

Let's also recall that forward-secrecy entails that loss of a LL-key does not compromise the semantic security of previously-distributed session keys.

3.3 Security Notions

AKE Security. In an execution of P, we say an adversary $\mathcal{A}$ *wins* if she asks a single Test-query to a **Fresh** player U 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. (The advantage is twice the probability that $\mathcal{A}$ will defeat the AKE security goal of the protocol minus one[1].) Protocol P is an **$\mathcal{A}$-secure AKE** if $\mathsf{Adv}_P^{ake}(\mathcal{A})$ is negligible.

MA Security. In an execution of P, we say adversary $\mathcal{A}$ violates mutual authentication (MA) if there exists an operation execution wherein a player U terminates holding SIDS(U), PIDS(U) and $|\mathrm{PIDS}(U)| \neq |\mathcal{I}| - 1$. We denote the **ma success** as $\mathsf{Succ}_P^{ma}(\mathcal{A})$ 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, we consider a new game, we denote $\mathbf{Game}^{ma}(\mathcal{A}, P)$, wherein the adversary exactly plays the same way as in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ with the same player accesses but with a different goal: to violate the mutual authentication.

Secure Signature Schemes. A signature scheme is defined by the following [25]:

- 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.

[1] $\mathcal{A}$ can trivially defeat AKE with probability 1/2, multiplying by two and substracting one rescales the probability.

– 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 $\mathsf{Succ}^{cma}_{\Sigma}(\mathcal{A})$.

3.4 Diffie-Hellman Problems

Computational Diffie-Hellman Assumption (CDH). Let $\mathbb{G}$ be a cyclic group $< g >$ of prime order q and x_1, x_2 chosen at random in $\mathbb{Z}_q$. A (T, ε)-CDH-attacker in $\mathbb{G}$ 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 ε. We denote this probability by $\mathsf{Succ}^{cdh}_{\mathbb{G}}(\Delta)$. The CDH problem is (T, ε)-**intractable** if there is no (T, ε)-attacker in $\mathbb{G}$.

Group Computational Diffie-Hellman Assumption (G-CDH). Let $\mathbb{G}$ be a cyclic group $< g >$ of prime order q and a polynomial-bounded integer n. 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:

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

If $\Gamma = \mathcal{P}(I) \backslash \{I_n\}$, we say that $G\text{-}CDH_{\Gamma}$ is the **Full** *Generalized Diffie-Hellman distribution* [9, 22, 29].

Given Γ, a (T, ε)-G-CDH$_{\Gamma}$-attacker in $\mathbb{G}$ 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 in $\mathbb{G}$.

Random Self-Reducibility of CDH and G-CDH. In a prime-order group $\mathbb{G}$, the CDH and G-CDH are random self-reducible problems [22]. Informally, this property means that solving the problem on any original instance $\mathcal{D}$ can be reduced to solving the problem on a random instance $\mathcal{D}'$. This requires an efficient way to generate the random instances $\mathcal{D}'$ from the original instance $\mathcal{D}$ and an efficient way to compute the solution to the problem on $\mathcal{D}'$ from the solution to the problem on $\mathcal{D}$.

Certainly the most common is the additive random self-reducibility of the CDH and G-CDH problems. We examplify this property for the G-CDH problem. Given, for example, an instance $\mathcal{D} = (g^a, g^b, g^c, g^{ab}, g^{bc}, g^{ac})$ for any a, b, c it is possible to generate a random instance

$$\mathcal{D}' = (g^{(a+\alpha)}, g^{(b+\beta)}, g^{(c+\gamma)}, g^{(a+\alpha).(b+\beta)}, g^{(b+\beta).(c+\gamma)}, g^{(a+\alpha).(c+\gamma)})$$

where α, β and γ are random numbers in $\mathbb{Z}_q$; however the cost of such a computation may be high. And given the solution $z = g^{(a+\alpha).(b+\beta).(c+\gamma)}$ to the instance $\mathcal{D}'$ it is possible to recover the solution g^{abc} to the random instance $\mathcal{D}$ (i.e. $g^{abc} = z(g^{ab})^{-\gamma}(g^{ac})^{-\beta}(g^{bc})^{-\alpha}(g^{a})^{-\beta\gamma}(g^{b})^{-\alpha\gamma}(g^{c})^{-\alpha\beta}g^{-\alpha\beta\gamma}$). It is, in effect, easy to see that such a reduction works only if $\mathcal{D}$ is the **Full** Generalized DH distribution and that its cost increases exponentially with the size of $\mathcal{D}$.

The other one is the multiplicative random self-reducibility of the CDH and G-CDH problems. The property holds if $\mathbb{G}$ is a prime-order cyclic group. We examplify this property for the G-CDH problem. Given, for example, an instance $\mathcal{D} = (g^a, g^b, g^{ab}, g^{ac})$ for any a, b, c it is easy to generate a random instance $\mathcal{D}' = (g^{a\alpha}, g^{b\beta}, g^{ab\alpha\beta}, g^{ac\alpha\gamma})$ where α, β and γ are random numbers in $\mathbb{Z}_q^*$. And given the solution $g^{a\alpha b\beta c\gamma}$ to the instance $\mathcal{D}'$ it is easy to see that the solution g^{abc} to the random instance $\mathcal{D}$ can be efficiently computed (i.e. $g^{abc} = \left(g^{a\alpha b\beta c\gamma}\right)^{(\alpha\beta\gamma)^{-1}}$). Such a reduction is efficient and only requires a linear number of modular exponentiations.

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;
- $q_s, q_r, q_c, Q_S, Q_R, Q_J$ numbers of Send, Reveal, Corrupt, Setup, Remove and Join queries the 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.

4 A Secure Authenticated Group Diffie-Hellman Scheme

In the following theorem and proof we assume the random oracle model [6] and denote $\mathcal{H}$ a hash function 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. The arithmetic is in a finite cyclic group $\mathbb{G} =< g >$ of order a k-bit prime number q and the operation is denoted multiplicatively. This group could be a prime subgroup of $\mathbb{Z}_p^*$, or it could be an (hyper)-elliptic curve based group.

4.1 Description

The AKE1 protocol consists of the SETUP1, REMOVE1 and JOIN1 algorithms. As illustrated by an AKE1 execution in Figures 1, 2 and 3 (an execution with more steps is depicted in Figure 6), this is a protocol wherein the players are arranged in a ring, and wherein each player saves the set of values it receives in the down-flow of SETUP1, REMOVE1, JOIN1. In effect, in the subsequent removal of players from $\mathcal{I}$ any player U could be selected as U_{GC} and so will need these values to execute REMOVE1.

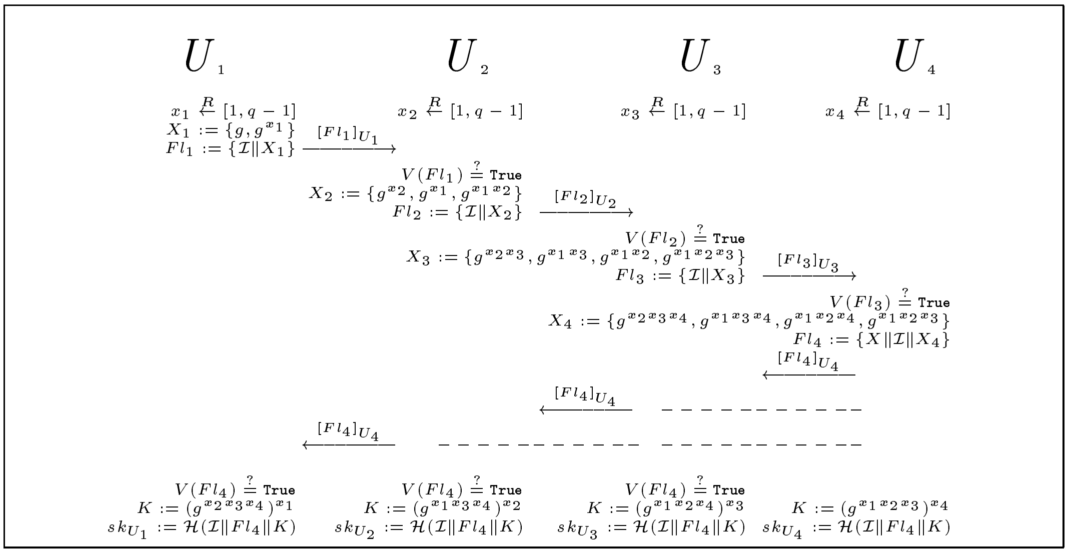

Fig. 1. Algorithm SETUP1. An example of an honest execution with 4 players: $\mathcal{J} = \{U_1, U_2, U_3, U_4\}$. The multicast group is $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_4 \| g^{x_1 x_2 x_3 x_4})$. The partner IDS for U_1 is $pids_{U_1} = \{U_2, U_3, U_4\}$, for U_2 is $pids_{U_2} = \{U_1, U_3, U_4\}$, for U_3 is $pids_{U_3} = \{U_1, U_2, U_4\}$ and for U_4 is $pids_{U_4} = \{U_1, U_3, U_4\}$.

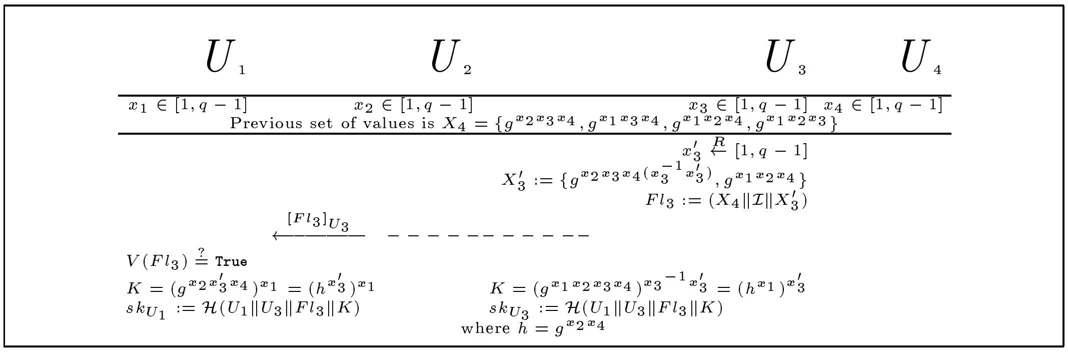

Fig. 2. Algorithm REMOVE1. An example of an honest execution with 4 players: $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$, $\mathcal{J} = \{U_2, U_4\}$. The new multicast group is $\mathcal{I} = \{U_1, U_3\}$, $U_{GC} = U_3$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_3 \| g^{x_1 x_2 x'_3 x_4})$, the partner IDS for U_1 is $pids_{U_1} = \{U_3\}$, for U_3 is $pids_{U_3} = \{U_1\}$.

Unlike [3], this is a protocol wherein the player with the highest-index in $\mathcal{I}$ is the group controller, the flows are signed using the long-lived key LL_U, the names of the players are in the protocol flows, and the session key SK is $sk = \mathcal{H}(\mathcal{I} \| Fl_{max(\mathcal{I})} \| g^{x_1 \ldots x_{\max(\mathcal{I})}})$; $Fl_{max(\mathcal{I})}$ is the down-flow, SIDS and PIDS are appropriately defined. The notion of index models "pre-existing" relationships among players: for example, it may capture different levels of reliability (i.e. the higher the index is, the more reliable the player). This is also a protocol, unlike [3], where the set of values from the down-flow is included in the flows of REMOVE1 and JOIN1, which avoids replay attacks.

Algorithm SETUP1. The algorithm consists of two stages: up-flow and down-flow. The multicast group $\mathcal{I}$ is set to $\mathcal{J}$. As illustrated by the example in Figure 1,

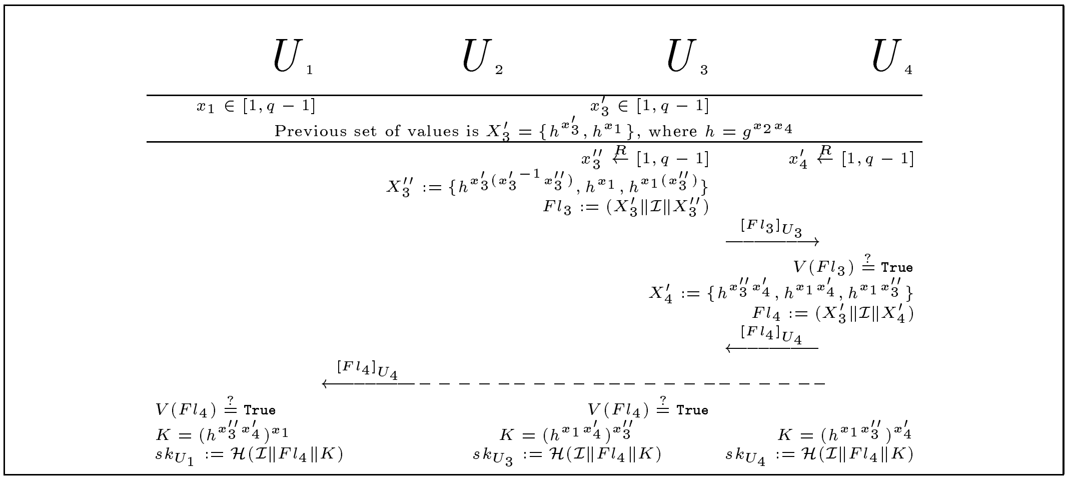

Fig. 3. Algorithm JOIN1. An example of an honest execution with 4 players: $\mathcal{I} = \{U_1, U_3\}$, $\mathcal{J} = \{U_4\}$ and $U_{GC} = U_3$. The new multicast group is $\mathcal{I} = \{U_1, U_3, U_4\}$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_4 \| g^{x_1 x_2 x''_3 (x_4 x'_4)})$ The partner IDS for U_1 is $pids_{U_1} = \{U_3, U_4\}$, for U_3 is $pids_{U_3} = \{U_1, U_4\}$ and for U_4 is $pids_{U_4} = \{U_1, U_3\}$.

in the up-flow the player U_i receives a set (Y, Z) of intermediate values, with

$$Y = \bigcup_{0<m<i} \{Z^{1/x_m}\} \text{ and } Z, \text{ where } Z = g^{\prod_{0<t<i} x_t}.$$

Player U_i chooses at random a private value x_i, raises the values in Y to the power of x_i and then concatenates with Z to obtain his intermediate values

$$Y' = \bigcup_{0<m\leq i} \{Z'^{1/x_m}\}, \text{ where } Z' = Z^{x_i} = g^{\prod_{0<t\leq i} x_t}.$$

Player U_i then forwards the values (Y', Z') to the next player in the ring. The down-flow takes place when $U_{max(\mathcal{I})}$ receives the last up-flow. At that point $U_{max(\mathcal{I})}$ performs the same steps as a player in the up-flow but broadcasts the set of intermediate values Y' only. In effect, the value Z' computed by $U_{max(\mathcal{I})}$ will lead to the session key sk, since $Z' = g^{\prod_{0<t\leq n} x_t}$. Players in $\mathcal{I}$ compute sk and accept.

Algorithm REMOVE1. This algorithm consists of a down-flow only. The multicast group $\mathcal{I}$ is first set to $\mathcal{I} \backslash \mathcal{J}$. As illustrated in Figure 2, the group controller U_{GC} (i.e. player with the highest-index in $\mathcal{I} \backslash \mathcal{J}$) generates a random value x'_{GC} and removes from the saved previous broadcast the values destinated to the players in $\mathcal{J}$. U_{GC} then raises all the remaining values in which x_{GC} appeared to the power of $(x^{-1}_{GC}.x'_{GC})$ and broadcasts the result. (x_{GC} is U_{GC}'s previous secret value.) Players in $\mathcal{I}$ compute sk and accept. Players in $\mathcal{J}$ erase any internal data. U_{GC} erases x_{GC} and x^{-1}_{GC} while internally saving x'_{GC}.

Algorithm JOIN1. This algorithm consists of two stages: up-flow and down-flow. As illustrated in Figure 3, the group controller U_{GC} (i.e. player with the highest-index in $\mathcal{I}$) generates a random value x'_{GC}, raises the values from the saved previous broadcast in which x_{GC} appears to the power of $(x_{GC}^{-1}.x'_{GC})$ and obtains a set of values Y'. (x_{GC} is U_{GC}'s previous secret exponent.) U_{GC} also computes the value Z' by raising the last value in Y' to x'_{GC}. U_i then forwards the values (Y', Z') to the first joining player in $\mathcal{J}$. From that point JOIN1 will work as the SETUP1 algorithm. Upon receiving the brodcast flow players in $\mathcal{I} \cup \mathcal{J}$ erase previous session keys, compute sk and accept. The multicast group $\mathcal{I}$ is then set to $\mathcal{I} \cup \mathcal{J}$.

4.2 Security Result

Theorem 1. *Let P be the AKE1 protocol, $\boldsymbol{SK}$ be the session-key space and $\mathcal{G}$ be the associated LL-key generator. Let $\mathcal{A}$ be an adversary against the AKE security of P within a time bound T, on a multicast group of size s among the n players in $\mathcal{U}$, after $Q = Q_S + Q_J + Q_R$ interactions with the parties, q_s send-queries and q_h hash-queries. Then we have:*

$$\mathsf{Adv}_P^{ake}(T, Q, q_s, q_h) \leq 2Q \cdot \binom{n}{s} \cdot s \cdot q_h \cdot \mathsf{Succ}_{\mathbb{G}}^{gcdh_{\Gamma_s}}(T') + 2n \cdot \mathsf{Succ}_{\Sigma}^{cma}(T', Q + q_s)$$

where $T' \leq T + (Q + q_s)nT_{exp}(k)$; $T_{exp}(k)$ is the time of computation required for an exponentiation modulo a k-bit number and Γ_s corresponds to the elements the adversary $\mathcal{A}$ can possibly view:

$$\Gamma_s = \bigcup_{2 \leq j \leq s-2} \{\{i \,|\, 1 \leq i \leq j, i \neq l\} \,|\, 1 \leq l \leq j\} \\ \bigcup \{\{i \,|\, 1 \leq i \leq s, i \neq k, l\} \,|\, 1 \leq k, l \leq s\}\,.$$

Let us just highlight 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 gotten her advantage by forging a signature with respect to some player's long-lived public key. 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 instance of G-CDH received as input.

To work (2) requires two things. We first "guess" the moment of the Test-query which means that we have to "guess": the number of operations that will occur before the adversary makes the Test-query and the membership of the multicast group when the adversary makes the Test-query. Second, based on this guess we "embed" the instance of G-CDH into the protocol. We generate many random instances from the original instance of G-CDH using the (multiplicative) random self-reducibility property of the G-CDH problem[2]. Indeed, the group Diffie-Hellman secret key relative to these random instances can efficiently be computed from the group Diffie-Hellman secret relative to the original instance.

[2] The multiplicative random self-reducibility will lead to a far more efficient reduction than the additive one would do.

The specific structure of Γ_s (see figure 4 for Γ_4) makes the simulation perfectly indistinguishable from the adversary point of view if our guesses are all correct.

$j = 0$							
$j = 1$				$\{1\}$			
$j = 2$			$\{1\}$	$\{2\}$			
$j = 3\ (= s - 1)$		$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,4\}$	$\{2,4\}$	$\{3,4\}$
$j = 4\ (= s)$	$\{1,2,3\}$	$\{1,2,4\}$	$\{1,3,4\}$	$\{2,3,4\}$			
	basic trigon				extension		

Fig. 4. Extended Trigon for Γ_4

But then, because of the random oracle $\mathcal{H}$, to have any information about the session key the adversary wants to test, she has to have asked for $\mathcal{H}(\mathcal{I}\|Fl_{last}\|K)$, where K is the value we are looking for. Therefore, if the adversary has some advantage in breaking the AKE security, this value K can be found in the list of the queries asked to $\mathcal{H}$. The details of the simulation can be found in appendix A.

4.3 AKE1 in Practice

We want our results to be practical. This means that when system designers choose a scheme they will take into account its security but also its efficiency in terms of computation, communication, ease of integration and so on. However, if provable security is achieved at the cost of a loss of efficiency, system designers will often prefer the heuristic schemes.

AKE1 is to date the first group Diffie-Hellman scheme to exhibit a proof that it achieves a strong notion of security. It is secure in the random oracle model under the G-CDH assumption. It thus provides stronger security guarantees than other schemes [3, 11, 17] while being more efficient than [3]. However security proofs for existing schemes or slight variants may show up.

On the integration front, the question that may be raised is what happens when several groups merge to form a larger group. A scenario that occurs in practice when a network failure partitions the multicast group in several disjoints sub-groups which will later need to merge when the network is be repaired [1]. The most efficient way in terms of computation and communication is to add players from the smaller sub-groups into the largest of the merging sub-groups. That is, U_{GC} is chosen as the player with the highest-index in the largest merging sub-group and the players from the smaller sub-groups are added via the JOIN1 algorithm.

5 Mutual Authentication

The well-known approach [5] for turning an AKE protocol into a protocol that provides mutual authentication (MA) is to use the shared session key to construct a simple "authenticator" for the other parties. We have described in [10] the transformation for turning an AKE group Diffie-Hellman scheme into a protocol providing MA and justified its security in the random-oracle model. We turn an AKE *dynamic* group Diffie-Hellman scheme into a protocol providing MA by simply applying the transformation MA described in [10] to the setup, join and remove algorithms respectively.

6 Conclusion and Further Research

This paper provides the first formal treatment of the authenticated group Diffie-Hellman key exchange problem in a scenario in which the membership is *dynamic* rather than static. Addressed 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 random oracle model that the protocol meets its goals.

The model introduced in this paper captures attacks that are realistic threats in practice. However the model does not yet capture "more serious" attacks: indeed, it does not recognize multiple player's instances the adversary may activate in concurrent and simultaneous sessions. A typical research topic is to enhance our model to capture these attacks and to investigate in this more stringent setting the security of the protocols presented in this paper. We are currently extending our model to encompass these attacks.

The security reduction presented for AKE1 in this paper does not inject much of the security of the group computational Diffie-Hellman problem and signature scheme: actually, the reduction is exponential in s. This leads one to use a larger security parameter or to limit the maximum size of the group. Another research direction is to find a security proof that would achieve a better security bound. We believe it is possible and are currently working on it.

Acknowledgements

The authors thank Deborah Agarwal and Jean-Jacques Quisquater for many insightful discussions and comments on an early draft of this paper. The authors also thank the anonymous referees for their useful comments.

References

1. D. A. Agarwal, O. Chevassut, M.R. Thompson, and G. Tsudik. An Integrated Solution for Secure Group Communication in Wide-Area Networks. In *Proc. of 6th IEEE Symposium on Computers and Communications*, 2001.
2. D. A. Agarwal, S. R. Sachs, and W. E. Johnston. The Reality of Collaboratories. *Computer Physics Communications*, 10(issue 1-3):pages 270–299, coverdate May 1998.
3. G. Ateniese, M. Steiner, and G. Tsudik. New Multiparty Authentication Services and Key Agreement Protocols. *IEEE Journal of Selected Areas in Communications*, April 2000.
4. K. Becker and U. Wille. Communication Complexity of Group Key Distribution. In *5th ACM Conference on Computer and Communications Security*, pages 1–6, November 1998.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In B. Preneel, editor, *Proc. of Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

6. M. Bellare and P. Rogaway. Entity Authentification and Key Distribution. In D.R. Stinson, editor, *Proc. of Crypto '93*, Lecture Notes in Computer Science. Springer-Verlag, 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. The Exact Security of Digital Signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Proc of Eurocrypt'96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.
9. D. Boneh. The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.
10. E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably Group Diffie-Hellman Key Exchange. In *Proc. of 8th ACM Conference on Computer and Communications Security*, Nov 2001.
11. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In A. De Santis, editor, *Proc of Eurocrypt' 94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1995.
12. 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.
13. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc of. Symposium on the Theory of Computing (SOC)*. ACM, March 1998.
14. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure under the RSA Assumption. In *Proc of. Crypto'01*, August 2001.
15. 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.
16. M. Jakobsson and D. Pointcheval. Mutual Authentication for Low-Power Mobile Devices. In *Proc. of Financial Cryptography '2001*, 2001.
17. M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. In *Proc. of ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 36–49. Springer-Verlag, 1996.
18. Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Group. In *Proc. of ACM Conference on Computer and Communications Security (CCS-7)*, November 2000.
19. Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proc. of International Federation for Information Processing (IFIP SEC 2001)*, June 2001.
20. S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *ACM Multimedia '95*, pages 511–522, November 1995.
21. L.E. Moser, P.M. Melliar-Smith, and P. Narasimhan. Consistent Object Replication in the Eternal System. *Theory and Practice of Object Systems*, 4(2):pages 81–92, 1998.
22. M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In *Proc. of 38th IEEE FOCS Symposium*, pages 458–467, 1997.
23. O. Pereira and J. J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *14-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 2001.
24. D. Pointcheval. Secure Designs for Public-Key Cryptography based on the Discrete Logarithm. *To appear in Discrete Applied Mathematics*, Elsevier Science, 2001.
25. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *J. of Cryptology*, 13(3):361–396, 2000.
26. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In W. Fumy, editor, *Proc. of Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.
27. D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A Secure Audio Teleconference System. In S. Goldwasser, editor, *Proc. of Crypto' 88*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. Springer-Verlag, 1988.
28. M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. In *IEEE Transactions on Parallel and Distributed Systems*, August 2000.
29. M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Groups. In *ACM CCS'96*, March 1996.
30. Wen-Guey Tzeng. A Practical and Secure Fault-Tolerant Conference-Key Agreement Protocol. In *Proc. of PKC2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

A Proof of Theorem 1

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$_{\Gamma_s}$-attacker Δ.

A.1 Forger $\mathcal{F}$

Let's assume that $\mathcal{A}$ breaks the protocol P by forging, with probability greater than ν, a signature with respect to some player's (public) LL-key (Of course before $\mathcal{A}$ corrupts U). 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, produced by $\mathcal{G}_{LL}(1^k)$.

$\mathcal{F}$ receives as input K_p and access to a signing oracle. $\mathcal{F}$ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$ and all U_i. $\mathcal{F}$ picks at random $i_0 \in [1, n]$ and runs $\mathcal{G}_{LL}(1^k)$ to set the players' LL-keys. However for player i_0, $\mathcal{F}$ sets LL_{i_0} 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 $\emptyset$.

The Send-queries, Setup-queries, Join-queries and Remove-queries are answered in a straightforward way, except if the query is made to player U_{i_0}. In this latter case the answers go through the signing oracle, and $\mathcal{F}$ stores in $\mathcal{K}$ the oracle query and the oracle reply. The Reveal-queries and Test-query are answered in a straightforward way as well. Eventually, the Corrupt-query is also answered in a straightforward way, except if the query is made to player U_{i_0}. In this latter case since $\mathcal{F}$ does not know the LL-key K_s for player i_0, $\mathcal{F}$ stops and outputs "`Fail`". But anyway, no signature forgery occurred before, and so, such an execution can be used with the other reduction. The Hash-query is simulated as depicted in Figure 5.

$\mathcal{F}$ succeeds 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 $(\sigma, m) \notin \mathcal{K}$. In this case $\mathcal{F}$ 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 by the probability of "correctly guessing" the value of i_0: $\mathsf{Succ}_{\Sigma}^{cma}(\mathcal{F}) \geq \nu / n$.

The running time of $\mathcal{F}$ is the running time of $\mathcal{A}$ added to the time to process the Send, Setup, Join and Remove-queries. This is essentially the time for at most n modular exponentiations. This leads to the given formula for T.

A.2 G-CDH$_{\Gamma_s}$-attacker Δ

Let's assume that $\mathcal{A}$ breaks the protocol P without producing a forgery. Here, with probability smaller than ν, the (valid) flows signed using LL_U come from player U and not from $\mathcal{A}$ (Of course before $\mathcal{A}$ corrupts U). The replay attacks involving the flows of JOIN1 and REMOVE1 do not also need to be considered since the values from the previous broadcast are included in these flows. One may then worry about replay attacks against SETUP1, however SETUP1 has already been proved to be secure for concurrent executions by Bresson et al. [10].

We now construct from $\mathcal{A}$ a (T', ε')-G-CDH$_{\Gamma_s}$-attacker Δ that receives as input an instance $\mathcal{D}$ of G-CDH$_{\Gamma_s}$ with random size s and outputs the Diffie-Hellman secret value (i.e $g^{x_1 \ldots x_s}$) relative to this instance. More precisely, a G-CDH$_{\Gamma_s}$ with size $s \in [1, n]$ and Γ_s of the form

$$\Gamma_s = \bigcup_{2 \leq j \leq s-2} \{\{i \,|\, 1 \leq i \leq j, i \neq l\} \,|\, 1 \leq l \leq j\} \\ \bigcup \{\{i \,|\, 1 \leq i \leq s, i \neq k, l\} \,|\, 1 \leq k, l \leq s\}\,.$$

This in turn leads to an instance $\mathcal{D} = (S_1, S_2, \ldots, S_{s-2}, S_{s-1}, S_s)$ wherein: S_j, for $2 \leq j \leq s-2$ and $j = s$, is the set of all the $j-1$-tuples one can build from $\{1, \ldots, j\}$; but S_{s-1} is the set of all $s-2$ tuples one can build from $\{1, \ldots, s\}$.

The aim of the simulation is to have all the elements of S_s, embedded into the protocol when the adversary $\mathcal{A}$ asks the Test-query. In this case, $\mathcal{A}$ will not be able to get any information about the value sk of the session key without having previously queried the random hash oracle $\mathcal{H}$ on the Diffie-Hellman secret value $g^{x_1 \ldots x_s}$. Thus, to break the security of P the adversary $\mathcal{A}$ would have to have asked a query of the form $\mathcal{H}(\mathcal{I}, Fl_{last}, g^{x_1 \ldots x_s})$ which as a consequence will be in the list of queries asked to $\mathcal{H}$.

To reach this aim Δ has to guess several values: $c_0, \mathcal{I}_0$ and i_0. We now describe what these values are used for and we will return to the formal simulation later on.

Δ first picks at random in $[1, Q]$ the number of operations c_0 that will occur before $\mathcal{A}$ asks the Test-query and embeds the elements of S_s into the operation that will occur at c_0. However Δ can not embed all the elements of S_s at c_0 since, contrary to SETUP1, in JOIN1 and REMOVE1 the players are not all added to the group at c_0. Δ rather embeds the elements from S_1 to S_s in the order the players are added to the group but only for the players that will belong to the group at c_0. Thus, Δ also chooses at random s index-values u_1 through u_s in $[1, n]$ that it hopes will make up the group membership at c_0.

Δ also needs to cope with protocol executions wherein the players u_i, $1 \leq i \leq s$, are repeatedely added and removed from the group in order to have several times before reaching c_0 the group membership be $\mathcal{I}_0$. If, in effect, Δ embeds all the elements of S_s into the protocol execution the first time the group membership is $\mathcal{I}_0$, Δ is neither able to compute the Diffie-Hellman secret value involved nor the session key value sk needed to answer to the Reveal-query.

To be able to answer, Δ does not in fact embed S_s into the broadcast flow of the operation which updates the group membership to be $\mathcal{I}_0$ but embeds truly random values. Δ guesses the player u_{i_0} from $\mathcal{I}_0$ who will embed S_s into the broadcast flow of the operation that occurs at c_0[3] but generates a truly random exponent and uses it to embed truly random values for the operations that occur before c_0 and after c_0. The index i_0 is set as follows. If the c_0-th operation is JOIN1 then i_0 is the last joining player's index, otherwise i_0 is the group controller's index $\max(\mathcal{I}_0)$.

[3] Δ may also embed a self-reduced element generated from S_s into the broadcast flow.

We now show that the above simulation and the random self-reducibility of G-CDH allows Δ to answer all the queries until $\mathcal{A}$ asks the Test-query at c_0. Since Δ embeds elements of S_i when a player u_i from $\mathcal{I}_0$ (except u_{i_0}) is added to the group and Δ does not remove it when u_i leaves, each protocol flow consists of a random self-reduction on one line (line 0, i.e. S_0 down to line $s-1$, i.e. $_{s-1}$) of the basic trigon. The trigon is illustrated on Figure 4. Thus, Δ can derivate the value sk of the session key from one of the values in the line below (line 1, i.e. S_1 up to line s, i.e. S_s).

However, Δ also needs to be able to answer to all queries after c_0 and more specifically the Reveal-queries. To this aim, Δ has to un-embed the element S_s from the protocol and do it in the operation that occurs at $c_0 + 1$. However depending on the operation that occurs at $c_0 + 1$, Δ may not be able to do it for player u_{i_0}. This is the reason why the line S_{s-1} has to contain all the possible $(s-2)$-tuples: extension of the basic trigon illustrated on Figure 4. For the operations that will occur after $c_0 + 1$, Δ uses truly random exponents for all the players including those in $\mathcal{I}_0$. Thus, after $c_0 + 1$ all the protocol flows involve elements in S_{s-1} and S_s only.

Formally, the simulator works as follows. Δ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all U_i and runs $\mathcal{G}_{LL}(1^k)$ to set the player's LL-keys. Δ sets an operation counter c to 0, and two variable $\mathcal{K}$ and $\mathcal{T}$ to $\emptyset$. Δ will use variable $\mathcal{K}$ to store (all) the random exponents involved in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ and variable $\mathcal{T}$ to store which exponents of instance $\mathcal{D}$ have been injected in the game so far. Then, Δ starts running $\mathcal{A}$ as a subroutine and answers the queries as depicted in Figure 5.

When $\mathcal{A}$ makes a Send-query to some player U_i, if this player is not in $\mathcal{I}_0$ then Δ proceeds as in the real protocol P using a random exponent. Otherwise Δ proceeds with the (multiplicative) random self-reducibility property using the elements from the instance $\mathcal{D}$ in the order wherein players join the multicast group, players u_{i_0} excepted since it uses a random exponent. To properly deal with self-reducibility, Δ uses variable $\mathcal{T}$ to reconstruct well-formatted (blinded) flows from $\mathcal{D}$.

When $\mathcal{A}$ makes a query of the form $\mathsf{Send}(U_{i_0}, *)$, Δ answers using random exponents before c_0, but for operation c_0, injects the last element from the instance $\mathcal{D}$.

This way, after the joining operation of the j-th player from $\mathcal{I}_0$, U_{i_0} excepted, the broadcast flow involves a random self-reduction of the j-th line in the basic trigon (see figure 4), the up-flows involve elements in the $j-1$-th line, and the session key one element from the $j+1$-th line. Thus, before operation c_0, Δ is able to answer the Join and Remove-queries and knows all the session keys needed to answer the Reveal-queries.

Another technical difficulty may show up if the adversary $\mathcal{A}$ does not output the bit b' right away after asking the Test-query and keeps playing the game for more rounds. Indeed, the session key is derivated from the G-CDH one is looking for. And forthcoming session keys would as well. Therefore, Δ would be unable to answer Reveal-queries. Δ has to reduce the number of exponents taken from the instance $\mathcal{D}$: basically, we go down in the basic trigon while player join the group. Until having involved $s-1$ exponents from instance $\mathcal{D}$. At the very last

Setup($\mathcal{J}$)	Reset $\mathcal{T}$ to 0 Increment c Update $\mathcal{I} \leftarrow \mathcal{J}$ $u \leftarrow \min(\mathcal{J})$ • $c < c_0 : u \in \mathcal{I}_0, u \neq i_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ • $c = c_0 : \mathcal{J} \neq \mathcal{I}_0 \Rightarrow$ output "`Fail`" $\mathcal{J} = \mathcal{I}_0, u = i_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ `Else` proceed as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$
Join($\mathcal{J}$)	Increment c $u \leftarrow \max(\mathcal{I})$ Update $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{J}$ • $c < c_0 : u \in \mathcal{I}_0, u \neq i_0$ simulate using RSR according to $\mathcal{T}$ • $c = c_0 : \mathcal{I} \neq \mathcal{I}_0 \vee \max(\mathcal{J}) \neq i_0 \Rightarrow$ output "`Fail`" $\mathcal{I} = \mathcal{I}_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ `Else` proceed as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$
Remove($\mathcal{J}$)	Increment c Update $\mathcal{I} \leftarrow \mathcal{I} \backslash \mathcal{J}$ $u \leftarrow \max(\mathcal{I})$ • $c < c_0 : u \in \mathcal{I}_0, u \neq i_0$ simulate using RSR according to $\mathcal{T}$ • $c = c_0 : \mathcal{I} \neq \mathcal{I}_0 \Rightarrow$ output "`Fail`" $\mathcal{I} = \mathcal{I}_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ `Else` proceed as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$
Send(U_i, m)	• $c < c_0 : i \in \mathcal{I}_0, i \neq i_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ • $c = c_0 : i \in \mathcal{I}_0 \Rightarrow$ simulate using RSR according to $\mathcal{T}$ `Else` proceed as in P using $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$
Reveal(U_i)	`If` U_i has accepted `Then` `If` $c = c_0$ `Then` output "`Fail`" `Else` return sk_{U_i}.
Corrupt(U_i)	return LL_{U_i}.
Test (U_i)	`If` U_i has accepted `Then` `If` $c = c_0$ `Then` return a random ℓ-bit string `Else` output "`Fail`".
Hash(m)	$\mathcal{H}(m) = r$; Hash-query has been made and the answer is r. If $m \notin \mathcal{H}$-list, then r is a chosen random value in the corresponding range, and $\mathcal{H}$-list $\leftarrow \mathcal{H}$-list$\|(m, r)$. Otherwise, r is taken from $\mathcal{H}$-list.

Fig. 5. Game$^{ake}(\mathcal{A}, P)$. The multicast group is $\mathcal{I}$. The Test-query is "guessed" to be made: after c_0 operations, the multicast group is $\mathcal{I}_0$, and the last joining player is U_{i_0}.

broadcast before the Test-query, we inject the last exponent (x_s, s being the size of our G-CDH instance). Then, just after the last broadcast (i.e., just after the Test-query), the group controller removes his own exponent, in order to come back into the trigon.

Unfortunately, this group controller is not necessarily U_{i_0}, and thus we do not go back into the basic trigon, but anyway with only $s-1$ exponents involved. Therefore, the future session keys will be derivated from the s-th line, but the broadcasts may involve any element in the extended $s-1$-th line, and the up-flows may also involve any element in the extended $(s-1)$-th line.

When $\mathcal{A}$ makes a Setup, Join or Remove-query, Δ increments c and proceeds in a similar way as for the Send-query. That is, Δ uses a random exponent for players that do not belong to $\mathcal{I}_0$ and proceeds with the random self-reducibility for players that belong to $\mathcal{I}_0$ but again only at c_0 for U_{i_0}. Each time a Setup-query occurs, variable $\mathcal{T}$ is reset to 0.

Δ stops and outputs "`Fail`" if some of his guesses turn out to be wrong. When $\mathcal{A}$ makes a Reveal-query, Δ proceeds as in the real protocol P except at $c = c_0$ where Δ stops and outputs "`Fail`": the guess on c_0 was wrong. Δ answers the Corrupt-queries in a straightforward way, since he knows the long-term keys. Finally when $\mathcal{A}$ makes a Test-query Δ stops and outputs "`Fail`" if $c \neq c_0$. Otherwise Δ returns a random string of length ℓ.

We now show that given the group Diffie-Hellman secret value relative to the instance $\mathcal{D}'$, involved in tested session key, Δ can easily compute the group Diffie-Hellman secret value relative to the instance $\mathcal{D}$. We emphasize that there may be more than s players in the multicast group before c_0. For the players that do not belong to $\mathcal{I}_0$, Δ had chosen random exponents and so will be able to "unblind" the self-reduced instance $\mathcal{D}'$ even if those exponents still appear in the session key (One may have already noticed that a leaving player leaves its secret exponent in the subsequent session keys). For the players in $\mathcal{I}_0$, Δ had used the instance $\mathcal{D}$ with blinding exponents and, thus, Δ is also able to unblind the G-CDH$_{\Gamma_s}$ instance. Assuming the Δ's guesses are correct, the elements from $\mathcal{D}$ involving the s-th exponent are only used in the final broadcast (just before the Test-query). This latter session key thus involves the solution to a blinded version $\mathcal{D}'$ of $\mathcal{D}$, and Δ knows how to unblind the solution, possibly found among the queries asked to the random oracle $\mathcal{H}$.

Indeed, if one assumes that the adversary $\mathcal{A}$ has made a Test-query and has terminated outputing a bit b' at some point. Δ then looks in the $\mathcal{H}$-list to see if queries of the form $\mathsf{Hash}(\mathcal{I}_0 \| Fl_0 \| *)$ have been asked (Fl_0 is the flow broadcasted in the execution of the c_0-th operation). If so, Δ chooses at random one of them and then looks in variable $\mathcal{K}$ (thanks to the flow Fl_0) for the corresponding random exponents Δ had used with the random self-reducibility to blind. Δ then unblinds' the remaining part "$*$" of the Hash-query and outputs it.

The probability that Δ correctly "guesses" the moment of the Test-query is the probability that $\mathcal{A}$ makes its Test-query after c_0 operations (proba $\geqslant 1/Q$) multiplied by the probability that at c_0 the multicast group is $\mathcal{I}_0$ (proba $\geqslant 1/\binom{n}{s}$). The probability Δ correctly "guesses" the index i_0 player is at least $1/s$. Also the solution is correctly extracted from the $\mathcal{H}$-list with probability $1/q_h$, since one just picks one candidate at random. Otherwise, one could output all the possible unblinded candidates and use the by now classical reduction from [26].

$$\mathsf{Succ}_{\mathbb{G}}^{gcdh_\Gamma}(\Delta) \geq \frac{\Pr[\mathsf{AskH}]}{q_h} \times \frac{1}{Q \cdot \binom{n}{s} \cdot s}.$$

The running time of Δ is the running time of $\mathcal{A}$ added to the time to process the Send-queries, Setup-queries, Remove-queries and Join-queries. This is essentially n modular exponentiation computations per Send-query, Setup-query, Remove-query or Join-query.

Finally, we have:

$$\Pr[b = b'] = \Pr[b = b' | \mathsf{Forge}] + \Pr[b = b' | \neg\mathsf{Forge}]$$

$$
\begin{aligned}
&\leq \nu + \Pr[b = b' | \neg \mathsf{Forge} \wedge \mathsf{AskH}] \Pr[\neg \mathsf{Forge} \wedge \mathsf{AskH}] \\
&\quad + \qquad \Pr[b = b' | \neg \mathsf{Forge} \wedge \neg \mathsf{AskH}] \Pr[\neg \mathsf{Forge} \wedge \neg \mathsf{AskH}] \\
&\leq \nu + \Pr[\mathsf{AskH}] + \frac{1}{2}
\end{aligned}
$$

The result then follows from the definition $\varepsilon = 2 \Pr[b = b'] - 1$:

$$
\varepsilon \leq 2n \cdot \mathsf{Succ}_{\Sigma}^{cma}(\mathcal{F}) + 2Q \cdot \binom{n}{s} \cdot s \cdot q_h \cdot \mathsf{Succ}_{\mathbb{G}}^{gcdh_{\Gamma_s}}(\Delta)
$$

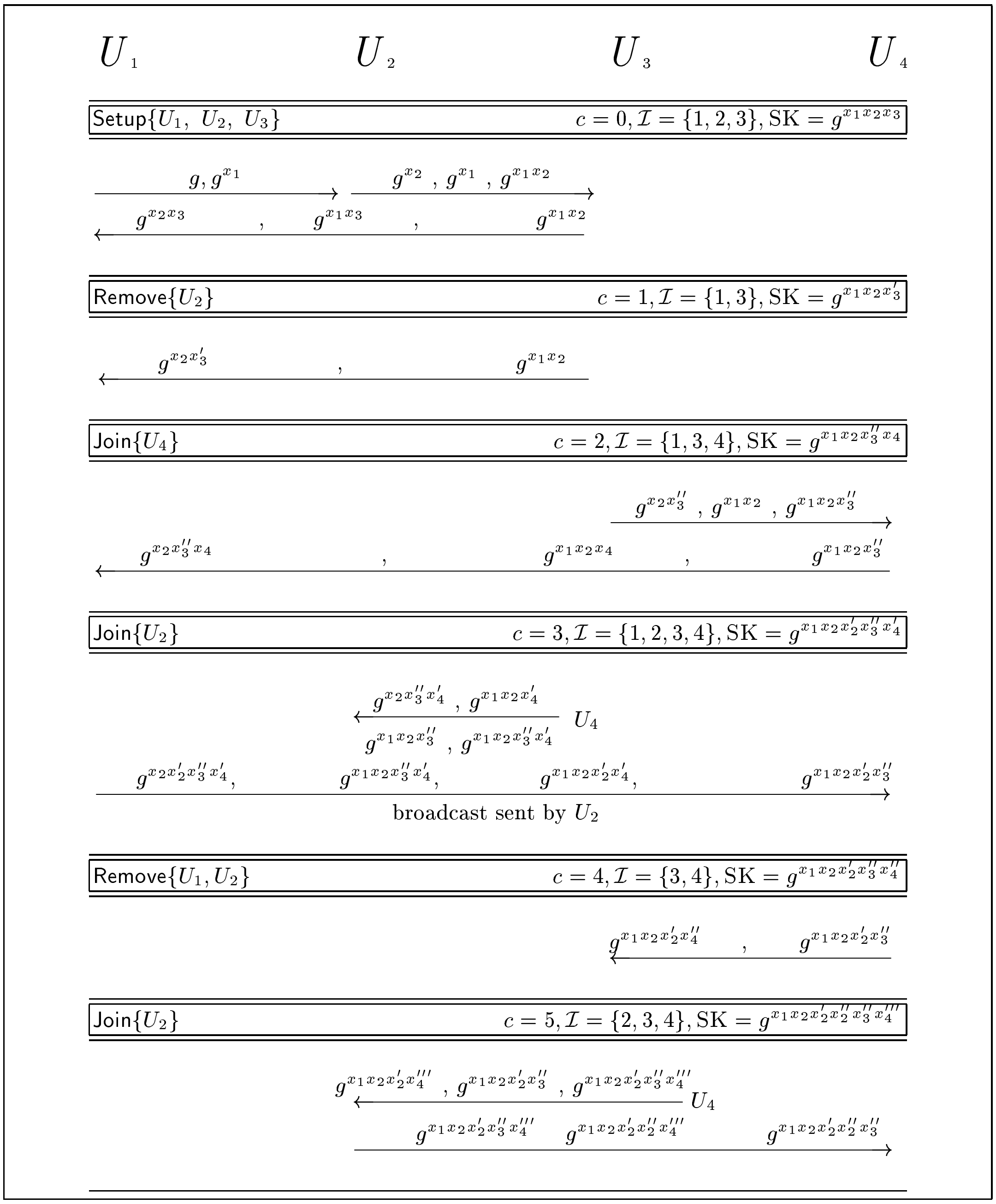

Fig. 6. An example of an execution of the real protocol P=AKE1

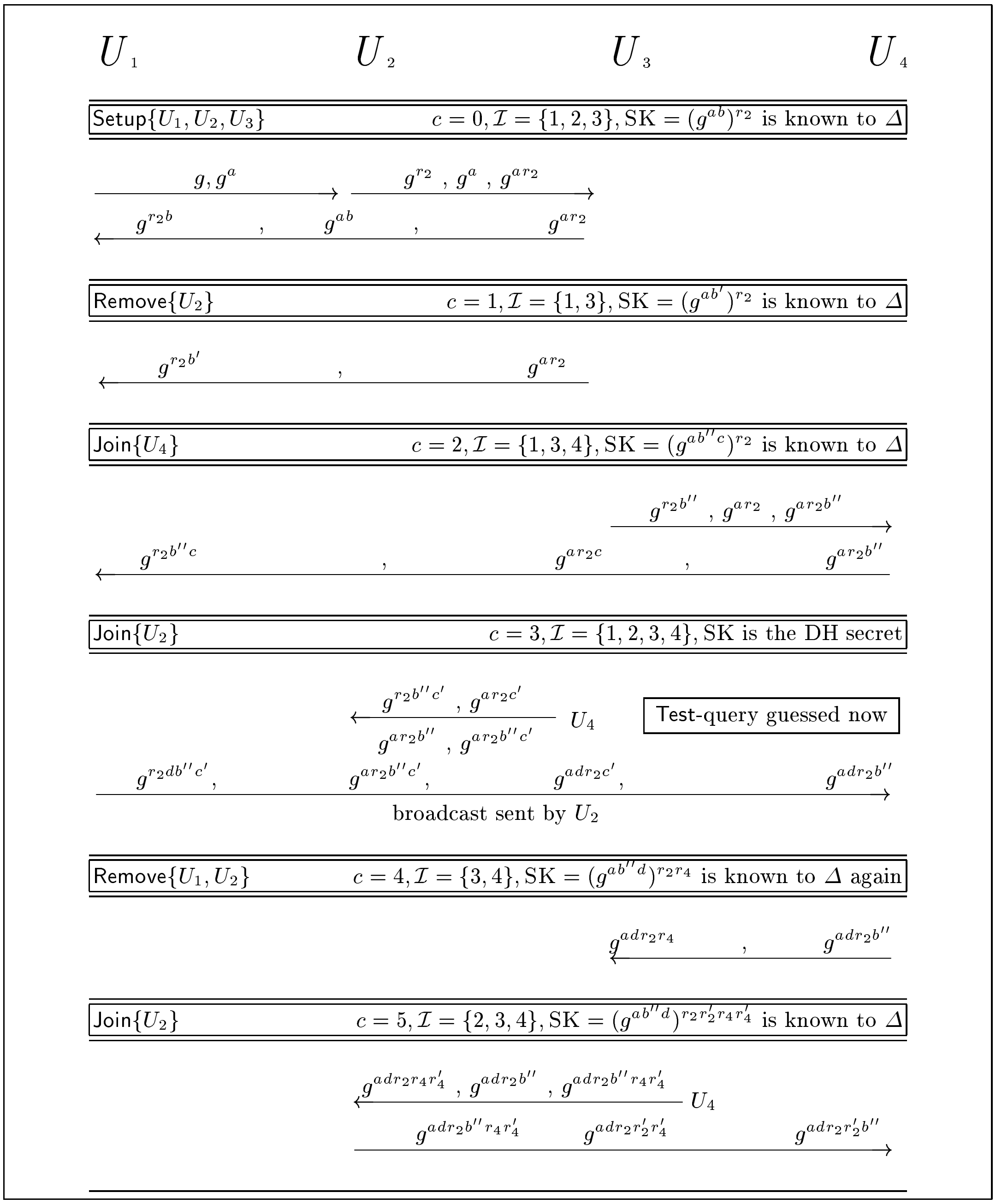

Fig. 7. An example of an execution of the protocol P=AKE1 with the adversary. We represent the simulation by Δ according to the following "guesses": $c_0 = 3, s = 4, \mathcal{I}_0 = \{1, 2, 3, 4\}, i_0 = 2$. We denote by b, b', b'' etc. some blinding exponents used in the self-reduction of G-CDH (think b'' as being $b\beta''$, e.g.). Also note that when rejoining the group at steps $c = 3$ and $c = 5$, U_2 does not "remove" its random exponent.