This is the full version of the extended abstract which appears in
Advances in Cryptology – Proceedings of Eurocrypt '02 (28 april – 2 may 2002, Amsterdam, Netherlands)
L. Knudsen Ed. Springer-Verlag, LNCS 2332, pages 321–336.

Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions

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. Authenticated Diffie-Hellman key exchange allows two principals communicating over a public network, and each holding public/private keys, to agree on a shared secret value. In this paper we study the natural extension of this cryptographic problem to a group of principals. We begin from existing formal security models and refine them to incorporate major missing details (e.g., strong-corruption and concurrent sessions). Within this model we define the execution of a protocol for authenticated dynamic group Diffie-Hellman and show that it is provably secure under the decisional Diffie-Hellman assumption. Our security result holds in the standard model and thus provides better security guarantees than previously published results in the random oracle model.

1 Introduction

Authenticated Diffie-Hellman key exchange allows two principals A and B communicating over a public network and each holding a pair of matching public/private keys to agree on a shared secret value. Protocols designed to deal with this problem ensure A (B resp.) that no other principals aside from B (A resp.) can learn any information about this value; the so-called authenticated key exchange with "implicit" authentication (AKE). These protocols additionally often ensure A and B that their respective partner has actually computed the shared secret value (i.e. authenticated key exchange with explicit key confirmation). A natural extension to this protocol problem would be to consider a scenario wherein a pool of principals agree on a shared secret value. We refer to this extension as authenticated group Diffie-Hellman key exchange.

Consider scientific collaborations and conferencing applications [5, 11], such as data sharing or electronic notebooks. Applications of this type usually involve users aggregated into small groups and often utilize multiple groups running in parallel. The users share responsibility for parts of tasks and need to coordinate their efforts in an environment prone to attacks. To reach this aim, the principals need to agree on a secret value to implement secure multicast channels. Key exchange schemes suited for this kind of application clearly needs to allow concurrent executions between parties.

We study the problem of authenticated group Diffie-Hellman key exchange when the group membership is dynamic – principals join and leave the group

⋆ 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-49087.

at any time – and the adversary may generate cascading changes in the membership for subsets of principals of his choice. After the initialization phase, and throughout the lifetime of the multicast group, the principals 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' should be only known to the principals in the multicast group during the period when sk' is the session key.

(2-party) Diffie-Hellman key exchange protocols also usually achieve the property of forward-secrecy [15, 16] which entails that corruption of a principal's long-term key does not threaten the security of previously established session keys. Assuming the ability to *erase* a secret, some of these protocols achieve forward-secrecy even if the corruption also releases the principal's internal state (i.e. strong-corruption [24]). In practice secret erasure is, for example, implemented by hardware devices which use physical security and tamper detection to not reveal any information [12, 22, 21, 28]. Protocols for group Diffie-Hellman key exchange need to achieve forward-secrecy even when facing strong-corruption.

Contributions. This paper is the third tier in the formal treatment of the group Diffie-Hellman key exchange using public/private key pairs. The first tier was provided for a scenario wherein the group membership is static [7] and the second, by extension of the latter for a scenario wherein the group membership is dynamic [8]. We start from the latter formal model and refine it to add important attributes. In the present paper, we model instances of players via oracles available to the adversary through queries. The queries are available to use at any time to allow model attacks involving multiple instances of players activated concurrently and simultaneously by the adversary. In order to model two modes of corruption, we consider the presence of two cryptographic devices which are made available to the adversary through queries. Hardware devices are useful to overcome software limitations however there has thus far been little formal security analysis [12, 23].

The types of crypto-devices and our notion of forward-secrecy leads us to modifications of existing protocols to obtain a protocol, we refer to it as $\mathsf{AKE1}^+$, secure against strong corruptions. Due to the very limited computational power of a smart card chip, smart card is used as an authentication token while a secure coprocessor is used to carry out the key exchange operations. We show that within our model the protocol $\mathsf{AKE1}^+$ is secure assuming the decisional Diffie-Hellman problem and the existence of a pseudo-random function family. Our security theorem does not need a random oracle assumption [4] and thus holds in the standard model. A proof in the standard model provides better security guarantees than one in an idealized model of computation [8, 7]. Furthermore we exhibit a security reduction with a much tighter bound than [8], namely we suppress the exponential factor in the size of the group. Therefore the security result is meaningful even for large groups. However the protocols are not practical for groups larger than 100 members.

The remainder of this paper is organized as follows. We first review the related work and then introduce the building blocks which we use throughout the paper. In Section 3, we present our formal model and specify through an abstract interface the standard functionalities a protocol for authenticated group Diffie-Hellman key exchange needs to implement. In Section 4, we describe the protocol $\mathsf{AKE1}^+$ by splitting it down into functions. This helps us to implement the abstract interface. Finally, in Section 5 we show that the protocol $\mathsf{AKE1}^+$ is provably secure in the standard model.

Related Work. Several papers [1, 10, 19, 14, 27] have extended the 2-party Diffie-Hellman key exchange [13] to the multi-party setting however a formal analysis has only been proposed recently. In [8, 7], we defined a formal model for the authenticated (dynamic) group Diffie-Hellman key exchange and proved secure protocols within this model. We use in both papers an ideal hash function [4], without dealing with dynamic group changes in [7], or concurrent executions of the protocol in [8].

However security can sometimes be compromised even when using a proven secure protocol: the protocol is incorrectly implemented or the model is insufficient. Cryptographic protocols assume, and do not usually explicitly state, that secrets are *definitively and reliably erased* (only the most recent secrets are kept) [12, 18]. Only recently formal models have been refined to incorporate the cryptographic action of erasing a secret, and thus protocols achieving forward-secrecy in the strong-corruption sense have been proposed [3, 24].

Protocols for group Diffie-Hellman key exchange [7] achieve the property of forward-secrecy in the strong-corruption sense assuming that "ephemeral" private keys are erased upon completion of a protocol run. However protocols for dynamic group Diffie-Hellman key exchange [8] do not, since they reuse the "ephemeral" keys to update the session key. Fortunately, these "ephemeral" keys can be embedded in some hardware cryptographic devices which are at least as good as erasing a secret [22, 21, 28].

2 Basic Building Blocks

We first introduce the pseudo-random function family and the intractability assumptions.

Message Authentication Code. A *Message Authentication Code* $\mathsf{MAC}=$ (MAC.Sgn,MAC.Vf) consists of the following two algorithms (where the key space is uniformly distributed) [2]:

- The *authentication algorithm* MAC.Sgn which, on a message m and a key K as input, outputs a tag μ. We write $\mu \leftarrow \text{MAC.Sgn}(K, m)$. The pair (m, μ) is called an authenticated message.
- The *verification algorithm* MAC.Vf which, on an authenticated message (m, μ) and a key K as input, checks whether μ is a valid tag on m with respect to K. We write $\texttt{True/False} \leftarrow \text{MAC.Vf}(K, m, \mu)$.

A (t, q, L, ϵ)-MAC-forger is a probabilistic Turing machine $\mathcal{F}$ running in time t that requests a MAC.Sgn-oracle up to q messages each of length at most

L, and outputs an authenticated message (m', μ'), without having queried the MAC.SGN-oracle on message m', with probability at least ϵ. We denote this success probability as $\mathsf{Succ}^{\mathsf{cma}}_{\mathsf{mac}}(t, q, L)$, where CMA stands for (adaptive) Chosen-Message Attack. The MAC scheme is (t, q, L, ϵ)-**CMA-secure** if there is no (t, q, L, ϵ)-MAC-forger.

Group Decisional Diffie-Hellman Assumption (G-DDH). Let $\mathbb{G} =< g >$ be a cyclic group of prime order q and n an 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:

$$\mathsf{G\text{-}DH}_\Gamma = \left\{ \left(J, g^{\prod_{j \in J} x_j}\right)_{J \in \Gamma} \mid x_1, \ldots, x_n \in_R \mathbb{Z}_q \right\}.$$

Given Γ, a (T, ϵ)-$\mathsf{G\text{-}DDH}_\Gamma$-distinguisher for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given an element X from either $\mathsf{G\text{-}DH}^\$_\Gamma$, where the tuple of $\mathsf{G\text{-}DH}_\Gamma$ is appended a random element g^r, or $\mathsf{G\text{-}DH}^\star_\Gamma$, where the tuple is appended $g^{x_1 \cdots x_n}$, outputs 0 or 1 such that:

$$\left| \Pr\left[\Delta(X) = 1 \,|\, X \in \mathsf{G\text{-}DH}^\$_\Gamma\right] - \Pr\left[\Delta(X) = 1 \,|\, X \in \mathsf{G\text{-}DH}^\star_\Gamma\right] \right| \geq \epsilon.$$

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

If $\Gamma = \mathcal{P}(I) \backslash \{I_n\}$, we say that $\mathsf{G\text{-}DH}_\Gamma$ is the **Full** *Generalized Diffie-Hellman distribution* [6, 20, 26]. Note that if $n = 2$, we get the classical DDH problem, for which we use the straightforward notation $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\cdot)$.

Lemma 1. *The DDH assumption implies the G-DDH assumption.*

Proof. Steiner, Tsudik and Waidner proved it in [26]. □

Multi Decisional Diffie-Hellman Assumption (M-DDH). We introduce a new decisional assumption, based on the Diffie-Hellman assumption. Let us define the *Multi Diffie-Hellman* M-DH and the *Random Multi Diffie-Hellman* $\mathsf{M\text{-}DH}^\$$ distributions of size n as:

$$\mathsf{M\text{-}DH}_n = \{(\{g^{x_i}\}_{1 \leq i \leq n}, \{g^{x_i x_j}\}_{1 \leq i < j \leq n}) \,|\, x_1, \ldots, x_n \in_R \mathbb{Z}_q\}$$
$$\mathsf{M\text{-}DH}^\$_n = \{(\{g^{x_i}\}_{1 \leq i \leq n}, \{g^{r_{j,k}}\}_{1 \leq j < k \leq n}) \,|\, x_i, r_{j,k} \in_R \mathbb{Z}_q, \forall i, 1 \leq j < k \leq n\}.$$

A (T, ϵ)-$\mathsf{M\text{-}DDH}_n$-distinguisher for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given an element X of either $\mathsf{M\text{-}DH}_n$ or $\mathsf{M\text{-}DH}^\$_n$ outputs 0 or 1 such that:

$$\left| \Pr[\Delta(X) = 1 \,|\, X \in \mathsf{M\text{-}DH}_n] - \Pr[\Delta(X) = 1 \,|\, X \in \mathsf{M\text{-}DH}^\$_n] \right| \geq \epsilon.$$

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

Lemma 2. *For any group $\mathbb{G}$ and any integer n, the M-DDH$_n$ problem can be reduced to the DDH problem and we have: $\mathsf{Adv}^{\mathsf{mddh}_n}_{\mathbb{G}}(T) \leq n^2 \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(T)$.*

Proof. It follows from an easy hybrid argument [20]. □

3 Model

In this section, we model instances of players via oracles available to the adversary through queries. These oracle 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. By making these queries available to the adversary at any time we provide him an ability to generate concurrent membership changes. We also take into account hardware devices and model their interaction with the adversary via queries.

Players. We fix a nonempty set $\mathcal{U}$ of N players that can participate in a group Diffie-Hellman key exchange protocol P. A player $U_i \in \mathcal{U}$ can have many *instances* called oracles involved in distinct concurrent executions of P. We denote instance t of player U_i as Π_i^t with $t \in \mathbb{N}$. Also, when we mean a not fixed member of $\mathcal{U}$ we use U without any index and denote an instance of U as Π_U^t with $t \in \mathbb{N}$.

For each concurrent execution of P, we consider a nonempty subset $\mathcal{I}$ of $\mathcal{U}$ called the *multicast group*. And in $\mathcal{I}$, the group controller $\mathsf{GC}(\mathcal{I})$ initiates the addition of players to the multicast group or the removal of players from the multicast group. The group controller is trusted to do only this.

In a multicast group $\mathcal{I}$ of size n, we denote by $\mathcal{I}_i$, for $i = 1, \ldots, n$, the index of the player related to the i-th instance involved in this group. This i-th instance is furthermore denoted by $\Pi(\mathcal{I}, i)$. Therefore, for any index $i \in \{1, \ldots, n\}$, $\Pi(\mathcal{I}, i) = \Pi_{\mathcal{I}_i}^t \in \mathcal{I}$ for some t.

Each player U holds a long-lived key LL_U which is a pair of matching public/private keys. LL_U is specific to U not to one of its instances.

Abstract Interface. We define the basic structure of a group Diffie-Hellman protocol. A group Diffie-Hellman scheme GDH consists of four algorithms:

- The *key generation algorithm* GDH.KeyGen(1^ℓ) is a probabilistic algorithm which on input of a security parameter 1^ℓ, provides each player in $\mathcal{U}$ with a long-lived key LL_U. The structure of LL_U depends on the particular scheme.

The three other algorithms are interactive multi-party protocols between players in $\mathcal{U}$, which provide each principal in the new multicast group with a new session key SK.

- The *setup algorithm* GDH.Setup($\mathcal{J}$), on input of a set of instances of players $\mathcal{J}$, creates a new multicast group $\mathcal{I}$, and sets it to $\mathcal{J}$.
- The *remove algorithm* GDH.Remove($\mathcal{I}, \mathcal{J}$) creates a new multicast group $\mathcal{I}$ and sets it to $\mathcal{I} \backslash \mathcal{J}$.
- The *join algorithm* GDH.Join($\mathcal{I}, \mathcal{J}$) creates a new multicast group $\mathcal{I}$, and sets it to $\mathcal{I} \cup \mathcal{J}$.

An execution of P consists of running the GDH.KeyGen algorithm once, and then many concurrent executions of the three other algorithms. We will also use the term *operation* to mean one of the algorithms: GDH.Setup, GDH.Remove or GDH.Join.

Security Model. The security definitions for P take place in the following game. In this game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$, the adversary $\mathcal{A}$ plays against the players in order to defeat the security of P. The game is initialized by providing coin tosses to GDH.KeyGen$(\cdot)$, $\mathcal{A}$, any oracle Π_U^t; and GDH.KeyGen(1^ℓ) is run to set up players' LL-key. A bit b is as well flipped to be later used in the Test-query (see below). The adversary $\mathcal{A}$ is then given access to the oracles and interacts with them via the queries described below. We now explain the capabilities that each kind of query captures:

Instance Oracle Queries. We define the oracle queries as the interactions between $\mathcal{A}$ and the oracles only. These queries model the attacks an adversary could mount through the network.

- $\mathsf{Send}(\Pi_U^t, m)$: This query models $\mathcal{A}$ sending messages to instance oracles. $\mathcal{A}$ gets back from his query the response which Π_U^t would have generated in processing message m according to P.
- $\mathsf{Setup}(\mathcal{J})$, $\mathsf{Remove}(\mathcal{I}, \mathcal{J})$, $\mathsf{Join}(\mathcal{I}, \mathcal{J})$: These queries model adversary $\mathcal{A}$ initiating one of the operations GDH.Setup, GDH.Remove or GDH.Join. Adversary $\mathcal{A}$ gets back the flow initiating the execution of the corresponding operation.
- $\mathsf{Reveal}(\Pi_U^t)$: This query models the attacks resulting in the loss of session key computed by oracle Π_U^t; it is only available to $\mathcal{A}$ if oracle Π_U^t has computed its session key $\mathrm{SK}_{\Pi_U^t}$. $\mathcal{A}$ gets back $\mathrm{SK}_{\Pi_U^t}$ which is otherwise hidden. When considering the *strong-corruption model* (see Section 5), this query also reveals the flows that have been exchanged between the oracle and the secure coprocessor.
- $\mathsf{Test}(\Pi_U^t)$: This query models the semantic security of the session key $\mathrm{SK}_{\Pi_U^t}$. It is asked only once in the game, and is only available if oracle Π_U^t is Fresh (see below). If $b = 0$, a random ℓ-bit string is returned; if $b = 1$, the session key is returned. We use this query to define $\mathcal{A}$'s advantage.

Secure Coprocessor Queries. The adversary $\mathcal{A}$ interacts with the secure coprocessors by making the following two queries.

- $\mathsf{Send}_c(\Pi_U^t, m)$: This query models $\mathcal{A}$ *directly* sending and receiving messages to the secure coprocessor. $\mathcal{A}$ gets back from his query the response which the secure coprocessor would have generated in processing message m. The adversary could directly interact with the secure coprocessor in a variety of ways: for instance, the adversary may have broken into a computer without being detected (e.g., bogus softwares, trojan horses and viruses).
- $\mathsf{Corrupt}_c(\Pi_U^t)$: This query models $\mathcal{A}$ having access to the private memory of the device. $\mathcal{A}$ gets back the internal data stored on the secure coprocessor. This query can be seen as an attack wherein $\mathcal{A}$ gets physical access to a secure coprocessor and bypasses the tamper detection mechanism [29]. This query is only available to the adversary when considering the *strong-corruption model* (see Section 5). The $\mathsf{Corrupt}_c$-query also reveals the flows the secure coprocessor and the smart card have exchanged.

Smart Card Queries. The adversary $\mathcal{A}$ interacts with the smart cards by making the two following queries.

- $\mathsf{Send}_s(U, m)$: This query models $\mathcal{A}$ sending messages to the smart card and receiving messages from the smart card.
- $\mathsf{Corrupt}_s(U)$: This query models the attacks in which the adversary gets access to the smart card and gets back the player's LL-key. This query models attacks like differential power analysis or other attacks by which the adversary bypasses the tamper detection mechanisms of the smart card [29].

When $\mathcal{A}$ terminates, it outputs a bit b'. We say that $\mathcal{A}$ *wins* the AKE game (see in Section 5) if $b = b'$. Since $\mathcal{A}$ can trivially win with probability 1/2, we define $\mathcal{A}$'s advantage by $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2 \times \Pr[b = b'] - 1$.

4 An Authenticated Group Diffie-Hellman Scheme

In this section, we describe the protocol $\mathsf{AKE1}^+$ by splitting it into functions that help us to implement the GDH abstract interface. These functions specify how certain cryptographic transformations have to be performed and abstract out the details of the devices (software or hardware) that will carry out the transformations. In the following we identify the multicast group to the set of indices of players (instances of players) in it. We use a security parameter ℓ and, to make the description easier see a player U_i not involved in the multicast group as if his private exponent x_i were equal to 1.

4.1 Overview

The protocol $\mathsf{AKE1}^+$ consists of the $\mathsf{Setup1}^+$, $\mathsf{Remove1}^+$ and $\mathsf{Join1}^+$ algorithms. As illustrated in Figures 1, 2 and 3, in $\mathsf{AKE1}^+$ the players are arranged in a ring and the instance with the highest-index in the multicast group $\mathcal{I}$ is the group controller $\mathsf{GC}(\mathcal{I})$: $\mathsf{GC}(\mathcal{I}) = \Pi(\mathcal{I}, n) = \Pi^t_{\mathcal{I}_n}$ for some t. This is also a protocol wherein each instance saves the set of values it receives in the down-flow of $\mathsf{Setup1}^+$, $\mathsf{Remove1}^+$ and $\mathsf{Join1}^+$[1].

The session-key space $\mathbf{SK}$ associated with the protocol $\mathsf{AKE1}^+$ is $\{0,1\}^\ell$ equipped with a uniform distribution. The arithmetic is in a group $\mathbb{G} =< g >$ of prime order q in which the DDH assumption holds. The key generation algorithm $\text{GDH.KeyGen}(1^\ell)$ outputs ElGamal-like LL-keys $LL_i = (s_i, g^{s_i})$.

4.2 Authentication Functions

The authentication mechanism supports the following functions:

- $\text{Auth\_Key\_Derive}(i, j)$. This function derives a secret value K_{ij} between U_i and U_j. In our protocol, $K_{ij} = F_1(g^{s_i s_j})$, where the map F_1 is specified in Section 4.4. (K_{ij} is never exposed.)

[1] In the subsequent removal of players from the multicast group any oracle Π could be selected as the group controller GC and so will need these values to execute $\mathsf{Remove1}^+$.

– $\textsc{Auth\_Sig}(i, j, m)$. This function invokes $\text{MAC.}\textsc{Sgn}(K_{ij}, m)$ to obtain tag μ, which is returned.
– $\textsc{Auth\_Ver}(i, j, m, \mu)$. This function invokes $\text{MAC.}\textsc{Vf}(K_{ij}, m, \mu)$ to check if (m, μ) is correct w.r.t. key K_{ij}. The boolean answer is returned.

The two latter functions should of course be called after initializing K_{ij} via $\textsc{Auth\_Key\_Derive}(\cdot)$.

4.3 Key-Exchange Functions

The key-exchange mechanism supports the following functions:

– $\textsc{Gdh\_Picks}(i)$. This function generates a new private exponent $x_i \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$. Recall that x_i is never exposed.
– $\textsc{Gdh\_Picks}^\star(i)$. This function invokes $\textsc{Gdh\_Picks}(i)$ to generate x_i but do not delete the previous private exponent x_i'. x_i' is only deleted when explicitly asked for by the instance.
– $\textsc{Gdh\_Up}(i, j, k, \mathsf{Fl}, \mu)$. First, if $j > 0$, the authenticity of tag μ on message Fl is checked with $\textsc{Auth\_Ver}(j, i, \mathsf{Fl}, \mu)$. Second, Fl is decoded as a set of intermediate values $(\mathcal{I}, Y, Z)$ where $\mathcal{I}$ is the multicast group and

$$Y = \bigcup_{m \neq i} \{Z^{1/x_m}\} \text{ with } Z = g^{x_t}.$$

The values in Y are raised to the power of x_i and then concatenated with Z to obtain these intermediate values

$$Y' = \bigcup \{Z'^{1/x_m}\}, \text{ where } Z' = Z^{x_i} = g^{x_t}.$$

Third, $\mathsf{Fl}' = (\mathcal{I}, Y', Z')$ is authenticated, by invoking $\textsc{Auth\_Sig}(i, k, \mathsf{Fl}')$ to obtain tag μ'. The flow (Fl', μ') is returned.
– $\textsc{Gdh\_Down}(i, j, \mathsf{Fl}, \mu)$. First, the authenticity of (Fl, μ) is checked, by invoking $\textsc{Auth\_Ver}(j, i, \mathsf{Fl}, \mu)$. Then the flow Fl' is computed as in $\textsc{Gdh\_Up}$, from $\mathsf{Fl} = (\mathcal{I}, Y, Z)$ but without the last element Z' (i.e. $\mathsf{Fl}' = (\mathcal{I}, Y')$). Finally, the flow Fl' is appended tags $\mu_1, \ldots, \mu_n$ by invoking $\textsc{Auth\_Sig}(i, k, \mathsf{Fl}')$, where k ranges in $\mathcal{I}$. The tuple $(\mathsf{Fl}', \mu_1, \ldots, \mu_n)$ is returned.
– $\textsc{Gdh\_Up\_Again}(i, k, \mathsf{Fl} = (\mathcal{I}, Y'))$. From Y' and the previous random x_i', one can recover the associated Z'. In this tuple (Y', Z'), one replaces the occurrences of the old random x_i' by the new one x_i (by raising some elements to the power x_i/x_i') to obtain Fl'. The latter is authenticated by computing via $\textsc{Auth\_Sig}(i, k, \mathsf{Fl}')$ the tag μ. The flow (Fl', μ') is returned. From now the old random x_i' is no longer needed and, thus, can be erased.
– $\textsc{Gdh\_Down\_Again}(i, \mathsf{Fl} = (\mathcal{I}, Y'))$. In Y', one replaces the occurrences of the old random x_i' by the new one x_i, to obtain Fl'. This flow is appended tags $\mu_1, \ldots, \mu_n$ by invoking $\textsc{Auth\_Sig}(i, k, \mathsf{Fl}')$, where k ranges in $\mathcal{I}$. The tuple $(\mathsf{Fl}', \mu_1, \ldots, \mu_n)$ is returned. From now the old random x_i' is no longer needed and, thus, can be erased.

- $\textsc{Gdh\_Key}(i, j, \mathsf{Fl}, \mu)$ produces the session key sk. First, the authenticity of (Fl, μ) is checked with $\textsc{Auth\_Ver}(j, i, \mathsf{Fl}, \mu)$. Second, the value $\alpha = g^{\prod_{j \in \mathcal{I}} x_j}$ is computed from the private exponent x_i, and the corresponding value in Fl. Third, sk is defined to be $F_2(\mathcal{I} \| \mathsf{Fl} \| \alpha)$, where the map $F_2(\cdot)$ is defined below.

4.4 Key Derivation Functions

The key derivation functions F_1 and F_2 are implemented via the so-called "entropy-smoothing" property. We use the left-over-hash lemma to obtain (almost) uniformly distributed values over $\{0,1\}^\ell$.

Lemma 3 (Left-Over-Hash Lemma [17]). *Let $\mathcal{D}_s : \{0,1\}^s$ be a probabilistic space with entropy at least σ. Let e be an integer and $\ell = \sigma - 2e$. Let $h : \{0,1\}^k \times \{0,1\}^s \to \{0,1\}^\ell$ be a universal hash function. Let $r \in_{\mathcal{U}} \{0,1\}^k$, $x \in_{\mathcal{D}_s} \{0,1\}^s$ and $y \in_{\mathcal{U}} \{0,1\}^\ell$. Then the statistical distance δ is:*

$$\delta(h_r(x) \| r, y \| r) \leq 2^{-(e+1)}.$$

Any universal hash function can be used in the above lemma, provided that y is uniformly distributed over $\{0,1\}^\ell$. However, in the security analysis, we need an additional property from h. This property states that the distribution $\{h_r(\alpha)\}_\alpha$ is *computationally undistinguishable* from the uniform one, for any r. Indeed, we need there is no "bad" parameter r, since such a parameter may be chosen by the adversary.

The map $F_1(\cdot)$ is implemented as follows through *public certified random strings*. In a Public-Key Infrastructure (PKI), each player U_i is given $N-1$ random strings $\{r_{ij}\}_{j \neq i}$ each of length k when registering his identity with a Certification Authority (CA). Recall that $N = |\mathcal{U}|$. The random string $r_{ij} = r_{ji}$ is used by U_i and U_j to derivate from input value x a symmetric-key $K_{ij} = F_1(x) = h_{r_{ij}}(x)$.

The map $F_2(\cdot)$ is implemented as follows. First, $\textsc{Gdh\_Down}(\cdot)$ is enhanced in such a way that it also generates a random value $r_\alpha \in \{0,1\}^k$, which is included in the subsequent broadcast. Then, player U_i derives from input value x a session key $sk = F_2(x) = h_{r_\alpha}(x)$.

One may note that in both cases, the random values are used only once, which gives almost uniformly and independently distributed values, according to the lemma 3.

4.5 Scheme

We correctly deal with concurrent sessions running in an adversary-controlled network by creating a new instance for each player in a multicast group. We in effect create an instance of a player via the algorithm $\mathsf{Setup1}^+$ and then create new instances of this player through the algorithms $\mathsf{Join1}^+$ and $\mathsf{Remove1}^+$.

Setup1$^+$($\mathcal{I}$): This algorithm consists of two stages, up-flow and down-flow (see Figure 1). On the up-flow oracle $\Pi(\mathcal{I}, i)$ invokes $\text{GDH\_PICKS}(\mathcal{I}_i)$ to generate its private exponent $x_{\mathcal{I}_i}$ and then invokes $\text{GDH\_UP}(\mathcal{I}_i, \mathcal{I}_{i-1}, \mathcal{I}_{i+1}, \mathsf{Fl}_{i-1}, \mu_{i-1,i})$ to obtain both flow Fl_i and tag $\mu_{i,i+1}$ (by convention, $\mathcal{I}_0 = 0$, $\mathsf{Fl}_0 = \mathcal{I}\|g$ and $\mu_{0,i} = \emptyset$). Then, $\Pi(\mathcal{I}, i)$ forwards $(\mathsf{Fl}_i, \mu_{i,i+1})$ to the next oracle in the ring. The down-flow takes place when $\mathsf{GC}(\mathcal{I})$ receives the last up-flow. Upon receiving this flow, $\mathsf{GC}(\mathcal{I})$ invokes $\text{GDH\_PICKS}(\mathcal{I}_n)$ and $\text{GDH\_DOWN}(\mathcal{I}_n, \mathcal{I}_{n-1}, \mathsf{Fl}_{n-1}, \mu_{n-1,n})$ to compute both Fl_n and the tags $\mu_1, \ldots, \mu_n$. $\mathsf{GC}(\mathcal{I})$ broadcasts $(\mathsf{Fl}_n, \mu_1, \ldots, \mu_n)$. Finally, each oracle $\Pi(\mathcal{I}, i)$ invokes $\text{GDH\_KEY}(\mathcal{I}_i, \mathcal{I}_n, \mathsf{Fl}_n, \mu_i)$ and gets back the session key $\text{SK}_{\Pi(\mathcal{I},i)}$.

Remove1$^+$($\mathcal{I}, \mathcal{J}$): This algorithm consists of a down-flow only (see Figure 2). The group controller $\mathsf{GC}(\mathcal{I})$ of the new set $\mathcal{I} = \mathcal{I} \backslash \mathcal{J}$ invokes $\text{GDH\_PICKS}^\star(\mathcal{I}_n)$ to get a *new* private exponent and then $\text{GDH\_DOWN\_AGAIN}(\mathcal{I}_n, \mathsf{Fl}')$ where Fl' is the saved previous broadcast. $\mathsf{GC}(\mathcal{I})$ obtains a new set of intermediate values from which it deletes the elements related to the removed players (in the set $\mathcal{J}$) and updates the multicast group. This produces the new broadcast flow Fl_n. Upon receiving the down-flow, $\Pi(\mathcal{I}, i)$ invokes $\text{GDH\_KEY}(\mathcal{I}_i, \mathcal{I}_n, \mathsf{Fl}_n, \mu_i)$ and gets back the session key $\text{SK}_{\Pi(\mathcal{I},i)}$. Here, is the reason why an oracle must store its private exponent and only erase its internal data when it leaves the group.

Join1$^+$($\mathcal{I}, \mathcal{J}$): This algorithm consists of two stages, up-flow and down-flow (see Figure 3). On the up-flow the group controller $\mathsf{GC}(\mathcal{I})$ invokes $\text{GDH\_PICKS}^\star(\mathcal{I}_n)$, and then $\text{GDH\_UP\_AGAIN}(\mathcal{I}_n, j, \mathsf{Fl}')$ where Fl', j are respectively the saved previous broadcast and the index of the first joining player. One updates $\mathcal{I}$, and forwards the result to the first joining player. From that point in the execution, the protocol works as the algorithm $\mathsf{Setup1}^+$, where the group controller is the highest index player in $\mathcal{J}$.

4.6 Practical Considerations

When implementors choose a protocol, they take into account its security but also its ease of integration. For a minimal disruption to a current security infrastructure, it is possible to modify $\mathsf{AKE1}^+$ so that it does not use *public certified random strings*. In this variant, the key derivation functions are both seen as ideal functions (i.e. the output of $F_1(\cdot)$ and $F_2(\cdot)$ are uniformly distributed over $\{0,1\}^\ell$) and are instantiated using specific functions derivated from cryptographic hash functions like SHA-1 or MD5. The analogue of Theorem 4 in the random oracle model can then easily be proven from the security proof of $\mathsf{AKE1}^+$.

5 Analysis of Security

In this section, we assert that the protocol $\mathsf{AKE1}^+$ securely distributes a session key. We refine the notion of forward-secrecy to take into account two modes of corruption and use it to define two notions of security. We show that when

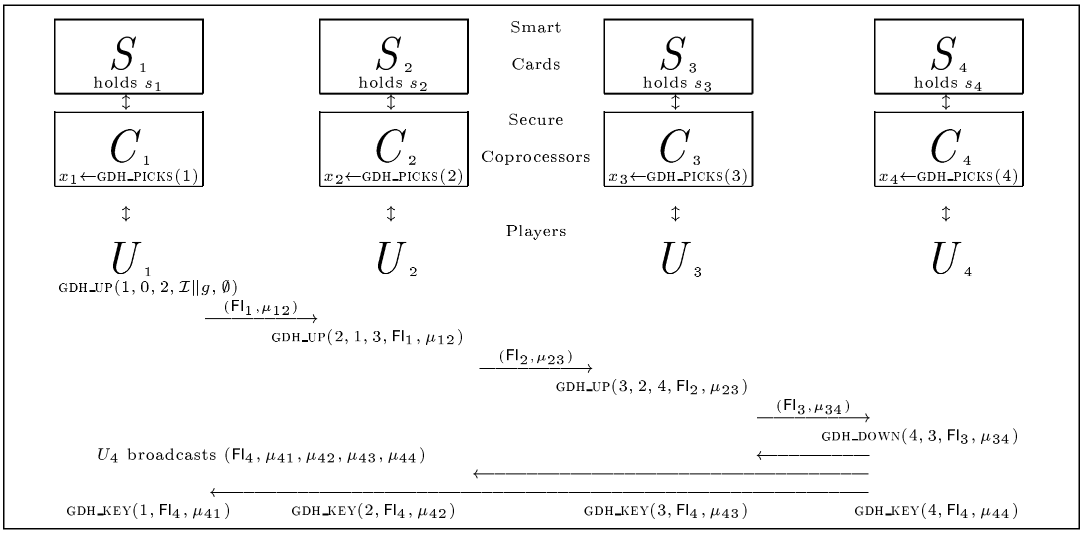

Fig. 1. Algorithm $\mathsf{Setup1}^+$. A practical example with 4 players $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$.

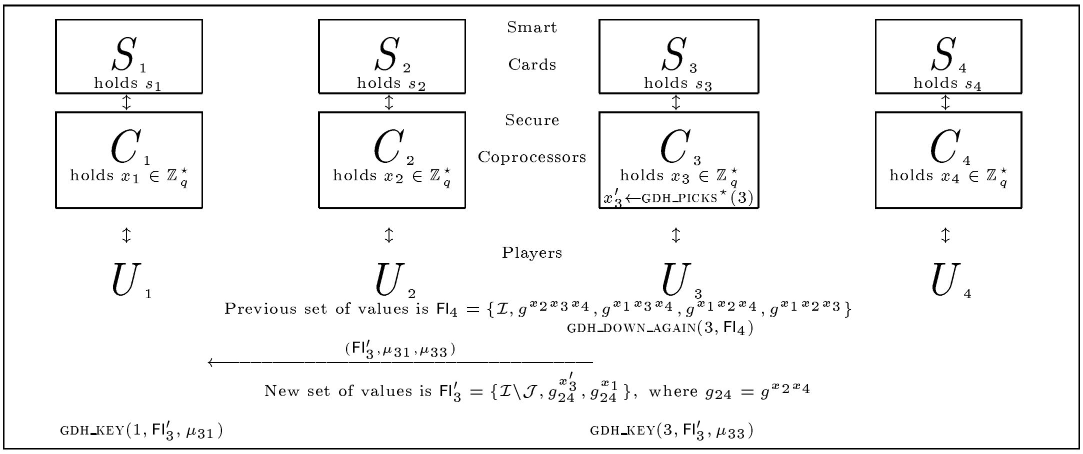

Fig. 2. Algorithm $\mathsf{Remove1}^+$. A practical example with 4 players: $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$ and $\mathcal{J} = \{U_2, U_4\}$. The new multicast group is $\mathcal{I} = \{U_1, U_3\}$ and $\mathsf{GC} = U_3$.

considering the weak-corruption mode the protocol $\mathsf{AKE1}^+$ is secure under standard assumptions. This proof can in turn be adapted to cope with the strong-corruption mode.

5.1 Security Notions

Forward-Secrecy. The notion of forward-secrecy entails that the corruption of a (static) LL-key used for authentication does not compromise the semantic security of previously established session keys. However while a corruption may have exposed the static key of a player it may have also exposed the player's internal data. That is either the LL-key or the ephemeral key (private exponent) used for session key establishment is exposed, or both. This in turn leads us to define two modes of corruption: the weak-corruption model and the strong-corruption model.

In the weak-corruption model, a corruption only reveals the LL-key of player U. That is, the adversary has the ability to make $\mathsf{Corrupt}_s$ queries. We then talk

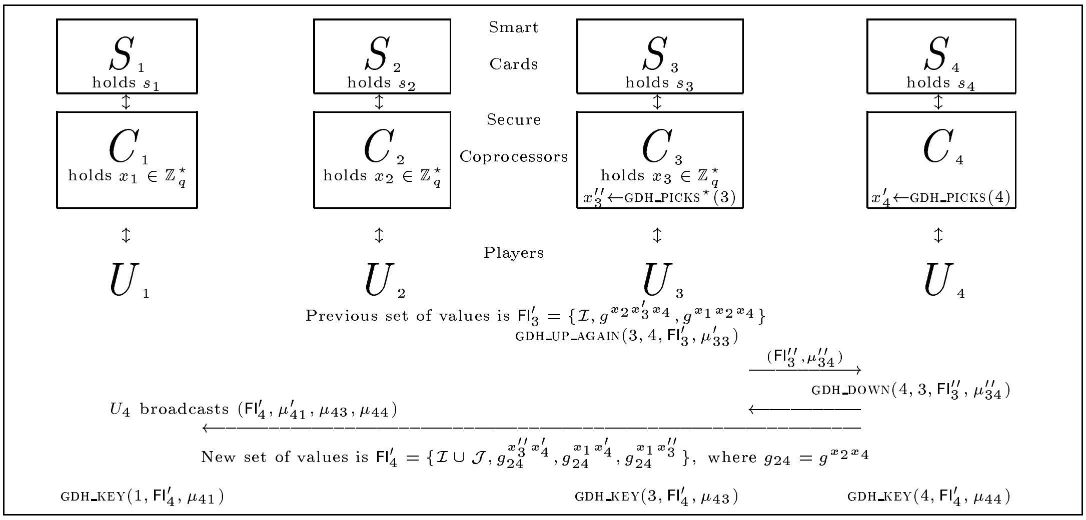

Fig. 3. Algorithm $\mathsf{Join1}^+$. A practical example with 4 players: $\mathcal{I} = \{U_1, U_3\}$, $\mathcal{J} = \{U_4\}$ and $\mathsf{GC} = U_3$. The new multicast group is $\mathcal{I} = \{U_1, U_3, U_4\}$.

about *weak-forward secrecy* and refer to it as wfs. In the strong-corruption model, a corruption will reveal the LL-key of U and additionally all internal data that his instances did not explicitly erase. That is, the adversary has the ability to make $\mathsf{Corrupt}_s$ and $\mathsf{Corrupt}_c$ queries. We then talk about *strong-forward secrecy* and refer to it as fs.

Freshness. As it turns out from the definition of forward-secrecy two flavors of freshness show up. An oracle Π_U^t is wfs-Fresh, in the current execution, (or holds a wfs-Fresh SK) if the following conditions hold. First, no $\mathsf{Corrupt}_s$ query has been made by the adversary since the beginning of the game. Second, in the execution of the current operation, U has accepted and neither U nor his partners has been asked for a Reveal-query.

An oracle Π_U^t is fs-Fresh, in the current execution, (or holds a fs-Fresh SK) if the following conditions hold. First, neither a $\mathsf{Corrupt}_s$-query nor a $\mathsf{Corrupt}_c$-query has been made by the adversary since the beginning of the game. Second, in the execution of the current operation, U has accepted and neither U nor his partners have been asked for a Reveal-query.

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 $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. We denote the AKE advantage as $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A})$. Protocol P is an **$\mathcal{A}$-secure AKE** if $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A})$ is negligible.

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

5.2 Security Theorem

A theorem asserting the security of some protocol measures how much computation and interactions helps the adversary. One sees that $\mathsf{AKE1}^+$ is a secure AKE protocol provided that the adversary does not solve the group decisional Diffie-Hellman problem G-DDH, does not solve the multi-decisional Diffie-Hellman problem M-DDH, or forges a Message Authentication Code MAC. These terms can be made negligible by appropriate choice of parameters for the group $\mathbb{G}$. The other terms can also be made "negligible" by an appropriate instantiation of the key derivation functions.

Theorem 4. *Let $\mathcal{A}$ be an adversary against protocol P, running in time T, allowed to make at most Q queries to any instance oracle. The adversary is also restricted to not ask $\mathsf{Corrupt}_c$-queries. Let n be the number of players involved in the operations which lead to the group on which $\mathcal{A}$ makes the Test-query. Then we have:*

$$\begin{aligned} \mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}, q_{se}) \leq 2nQ \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(T') + 2\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T) \\ +n(n-1) \cdot \mathsf{Succ}_{\mathsf{mac}}^{\mathsf{cma}}(T) + n(n-1) \cdot \delta_1 + 2nQ \cdot \delta_2 \end{aligned}$$

where δ_i denotes the distance between the output of $F_i(\cdot)$ and the uniform distribution over $\{0,1\}^\ell$, $T' \leq T + QnT_{exp}(k)$, where $T_{exp}(k)$ is the time of computation required for an exponentiation modulo a k-bit number, and Γ_n corresponds to the elements adversary $\mathcal{A}$ can possibly view:

$$\begin{aligned} \Gamma_n = \bigcup_{2 \leq j \leq n-2} \{\{i \,|\, 1 \leq i \leq j, i \neq l\} \;|\; 1 \leq l \leq j\} \\ \bigcup \{\{i \,|\, 1 \leq i \leq n, i \neq k, l\} \;|\; 1 \leq k, l \leq n\}. \end{aligned}$$

Proof. The formal proof of the theorem is omitted due to lack of space and can be found in the Appendix 4. We do, however, provide a sketch of the proof here.

Let the notation $\mathbf{G}_0$ refer to $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. Let b and b' be defined as in Section 3 and S_0 be the event that $b = b'$. We incrementally define a sequence of games starting at $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We define in the execution of $\mathbf{G}_{i-1}$ and $\mathbf{G}_i$ a certain "bad" event E_i and show that as long as E_i does not occur the two games are identical [25]. The difficulty is in choosing the "bad" event. We then show that the advantage of $\mathcal{A}$ in breaking the **AKE** security of P can be bounded by the probability that the "bad" events happen. We now define the games $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3, \mathbf{G}_4, \mathbf{G}_5$. Let S_i be the event $b = b'$ in game $\mathbf{G}_i$.

Game $\mathbf{G_1}$ is the same as game $\mathbf{G}_0$ except we abort if a MAC forgery occurs before any Corrupt-query. We define the MAC forgery event by Forge. We then show: $|\Pr[\mathsf{S}_0] - \Pr[\mathsf{S}_1]| \leq \Pr[\mathsf{Forge}]$.

Game $\mathbf{G_2}$ is the same as game $\mathbf{G}_1$ except that we add the following rule: we choose at random an index i_0 in $[1, n]$ and an integer c_0 in $[1, Q]$. If the Test-query does not occur at the c_0-th operation, or if the very last broadcast flow before the Test-query is not operated by player i_0, the simulator outputs "`Fail`" and sets b' randomly. Let E_2 be the event that these guesses are not correct. We show: $\Pr[\mathsf{S}_2] = \Pr[\mathsf{E}_2]/2 + \Pr[\mathsf{S}_1](1 - \Pr[\mathsf{E}_2])$, .

Game $\mathbf{G_3}$ is the same as game $\mathbf{G}_2$ except that we modify the way the queries made by $\mathcal{A}$ are answered; the simulator's input is $\mathcal{D}$, a $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$ element, with $g^{x_1 \ldots x_n}$. During the attack, based on the two values i_0 and c_0, the simulator injects terms from the instance such that the Test-ed key is derived from the $\mathsf{G\text{-}DH}$-secret value relative to that instance. The simulator appears in Appendix A. We then show that: $\Pr[\mathsf{S}_2] = \Pr[\mathsf{S}_3]$.

Game $\mathbf{G_4}$ is the same as game $\mathbf{G}_3$ except that the simulator is given as input an element $\mathcal{D}$ from $\mathsf{G\text{-}DH}^{\$}_{\Gamma_n}$, with g^r. And in case $b = 1$, the value random value g^r is used to answer the Test-query. The, the difference between $\mathbf{G}_3$ and $\mathbf{G}_4$ is upper-bounded by the computational distance between the two distributions $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$ and $\mathsf{G\text{-}DH}^{\$}_{\Gamma_n}$: $|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_4]| \leq \mathsf{Adv}^{\mathsf{gddh}_{\Gamma_n}}_{\mathbb{G}}(T')$, where T' takes into account the running time of the adversary, and the random self-reducibility operations, and thus $T' \leq T + QnT_{exp}(k)$.

Game $\mathbf{G_5}$ is the same as $\mathbf{G}_4$, except that the Test-query is answered with a completely random value, independent of b. It is then straightforward that $\Pr[\mathsf{S}_5] = 1/2$. Let δ_2 be the distance between the output of $F_2(\cdot)$ and the uniform distribution, we have: $|\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]| \leq \delta_2$.

The theorem then follows from the above equations. They indeed lead to

$$\Pr[\mathsf{S}_0] \leq \Pr[\mathsf{Forge}] + \Pr[\mathsf{S}_1] \leq \Pr[\mathsf{Forge}] + nQ\left(\mathsf{Adv}^{\mathsf{gddh}_{\Gamma_n}}_{\mathbb{G}}(T') + \delta_2\right) + \frac{1}{2}.$$

□

Remark. When considering strong-corruptions we have to answer to all the $\mathsf{Corrupt}_c$-queries made by the adversary along the games but we can only do so if we know the private exponents involved in the games. To reach this aim, we can no longer benefit from the self-random reducibility property of G-DDH and have to "guess" the moment at which the adversary will ask the Test-query. Unfortunately, reductions carried out in such a way add an exponential factor in the size of the multicast group [7, 8].

6 Conclusion

This paper represents the third tier in the treatment of the group Diffie-Hellman key exchange using public/private keys. The first tier was provided for a scenario wherein the group membership is static [7] and the second, by extension of the latter to support membership changes [8]. This paper adds important attributes (strong-corruption, concurrent executions of the protocol, tighter reduction, standard model) to the group Diffie-Hellman key exchange.

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. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM CCS '98*, pp. 17–26. 1998.
2. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proc. of Crypto '96*, LNCS 1109, pp. 1–15. Springer, 1996.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proc. of Eurocrypt '00*, LNCS 1807, pp. 139–155. Springer, 2000.
4. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM CCS '93*, pp. 62–73. 1993.
5. K. P. Birman. A review experience with reliable multicast. *Software – Practice and Experience*, 29(9):741–774, 1999.
6. D. Boneh. The decision Diffie-Hellman problem. In *Proc. of ANTS III*, LNCS 1423, pp. 48–63. Springer, 1998.
7. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *ACM CCS '01*, pp. 255–264. 2001.
8. E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In *Proc. of Asiacrypt '01*, LNCS 2248, pp. 290–309. Springer, 2001.
9. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Proc. of Eurocrypt '02*, LNCS. Springer, 2002. Full version of this paper available at `http://www.di.ens.fr/~pointche`.
10. M. Burmester and Y. G. Desmedt. A secure and efficient conference key distribution system. In *Proc. of Eurocrypt '94*, LNCS 950, pp. 275–286. Springer, 1995.
11. G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):1–43, 2001.
12. G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. How to forget a secret. In *Proc. of STACS '99*, LNCS 1563, pp. 500–509. Springer, 1999.
13. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
14. W. Diffie, D. Steer, L. Strawczynski, and M. Wiener. A secure audio teleconference system. In *Proc. of Crypto '88*, LNCS 403, pp. 520–528. Springer, 1988.
15. W. Diffie, P. van Oorschot, and W. Wiener. Authentication and authenticated key exchange. In *Designs, Codes and Cryptography*, vol. 2(2), pp. 107–125, 1992.
16. C. G. Gunter. An identity-based key exchange protocol. In *Proc. of Eurocrypt '89*, LNCS 434, pp. 29–37. Springer, 1989.
17. J. Hstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28(4):1364–1396, 1999.
18. M. Joye and J.-J. Quisquater. On the importance of securing your bins: The garbage-man-in-the-middle attack. In *ACM CCS'97*, pp. 135–141. 1997.
19. M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Proc. of Asiacrypt '96*, LNCS 1163, pp. 36–49. Springer, 1996.
20. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS '97*, pp. 458–467. IEEE, 1997.
21. NIST. *FIPS 140-1: Security Requirements for Cryptographic Modules.* U. S. National Institute of Standards and Technology, 1994.
22. E. R. Palmer, S. W. Smith, and S. Weingart. Using a high-performance, programmable secure coprocessor. In *Financial Crypto '98*, LNCS 1465, pp. 73–89. Springer, 1998.
23. A. Rubin and V. Shoup. Session-key disribution using smart cards. In *Proc. of Eurocrypt '96*, LNCS 1070, pp. 321–331. Springer, 1996.
24. V. Shoup. On formal models for secure key exchange. Technical Report RZ 3120, IBM Zrich Research Lab, 1999.
25. V. Shoup. OAEP reconsidered. In J. Kilian, editor, *Proc. of Cryto' 01*, volume 2139 of *LNCS*, pages 239–259. Springer-Verlag, 2001.
26. M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *ACM CCS '96*, pp. 31–37. 1996.
27. W. G. Tzeng. A practical and secure fault-tolerant conference-key agreement protocol. In *Proc. of PKC '00*, LNCS 1751, pp. 1–13. Springer, 2000.
28. K. Vedder and F. Weikmann. Smart cards requirements, properties, and applications. In *State of the Art in Applied Cryptography*, LNCS 1528. Springer, 1997.
29. S. H. Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Proc. of CHES '00*, LNCS 1965, pp. 302–317. Springer, 2000.

A Proof of theorem 4

Proof. Let $\mathcal{A}$ be an adversary that can get an advantage ϵ in breaking the AKE security of protocol P within time t, assuming n players have been involved in the protocol. By *player involved in a group*, we mean a player who has joined the group at least once since its setup.

In the following we define a sequence of games $\mathbf{G}_0, \ldots \mathbf{G}_5$ and also several events. We denote the event $b = b'$ in the game $\mathbf{G}_i$ by S_i and also define a "bad" event E_i. We will then show that as long as E_i does not occur then the two games $\mathbf{G}_{i-1}$ and $\mathbf{G}_i$ are identical.

The queries made by $\mathcal{A}$ are answered by a simulator Δ. Δ maintains for each concurrent execution of P two variables $\mathcal{T}$ and $\mathcal{L}_0$. In $\mathcal{L}_0$ it keeps the set of the first n players which have been involved in the group so far. In $\mathcal{T}$ it keeps the order of arrival of the players in $\mathcal{L}_0$: i.e. to know which elements of the GDH-trigon have to be used for each player in Game $\mathbf{G}_3$ (see Figure 4). These variables are reset whenever a $\mathsf{Setup1}^+$ occurs.

Game $\mathbf{G_0}$. This game $\mathbf{G}_0$ is the real attack $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. We set At the beginning of this game we set the bit b at random.

Game $\mathbf{G_1}$. The game $\mathbf{G}_1$ is identical to $\mathbf{G}_0$ except that we abort if a MAC forgery occurs before any Corrupt-query. We define such an event by Forge. Using a well-know lemma we get:

$$|\Pr[\mathsf{S}_0] - \Pr[\mathsf{S}_1]| \leq \Pr[\mathsf{Forge}]. \tag{1}$$

Lemma 5. *Let δ_1 be the distance between the output of the map F_1 and the uniform distribution. Then, we have:*

$$\Pr[\mathit{Forge}] \leq \mathit{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T) + \frac{n(n-1)}{2}\mathit{Succ}_{\mathsf{mac}}^{\mathsf{cma}}(T) + \frac{n(n-1)}{2}\delta_1. \tag{2}$$

Proof. The proof of this lemma appears in appendix B.

Game $\mathbf{G_2}$. Game $\mathbf{G}_2$ is the same as game $\mathbf{G}_1$ except that we add the following rule: we choose at random two values i_0 in $[1, n]$ and c_0 in $[1, Q]$. c_0 is a guess for the number of operations that will occur before $\mathcal{A}$ asks the Test-query and i_0 is a guess for the player who will send the very last broadcast flow before the Test-query. If the c_0-th operation is $\mathsf{Join1}^+$ or $\mathsf{Setup1}^+$, then i_0 is the last joining player's index, otherwise i_0 is the group controller's index (hoped to be $\max(\mathcal{L}_0)$). If the Test-query does not occur at the c_0-th operation, or if the very last broadcast flow before the Test-query is not operated by player i_0, the simulator outputs "`Fail`" and sets b' randomly. Let E_2 be the event that these guesses are not correct. Then we have:

$$\begin{aligned}\Pr[\mathsf{S}_2] &= \Pr[\mathsf{S}_2 \wedge \mathsf{E}_2] + \Pr[\mathsf{S}_2 \wedge \neg\mathsf{E}_2] = \Pr[\mathsf{S}_2 \mid \mathsf{E}_2] \Pr[\mathsf{E}_2] + \Pr[\mathsf{S}_2 \mid \neg\mathsf{E}_2] \Pr[\neg\mathsf{E}_2] \\ &= \frac{1}{2} \Pr[\mathsf{E}_2] + \Pr[\mathsf{S}_1] \left(1 - \Pr[\mathsf{E}_2]\right), \end{aligned} \tag{3}$$

where $\Pr[\mathsf{E}_2] = 1 - 1/nQ$. Note that we use the fact that E_2 and S_1 are independent.

Game $\mathbf{G_3}$. Game $\mathbf{G}_3$ is the same as game $\mathbf{G}_2$ except that we slightly modify the way the queries made by $\mathcal{A}$ are answered. Δ receives as input an instance $\mathcal{D}$ of size n from $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$, with its solution $g^{x_1 \cdots x_n}$:

$$\Gamma_n = \bigcup_{2 \le j \le n-2} \{\{i \,|\, 1 \le i \le j, i \ne l\} \,|\, 1 \le l \le j\}$$
$$\bigcup \{\{i \,|\, 1 \le i \le n, i \ne k, l\} \,|\, 1 \le k, l \le n\}.$$

This in turn leads to an instance $\mathcal{D} = (S_1, \dots, S_{n-2}, S_{n-1}, S_n) \cup \{g^x_1 \dots x_n\}$ wherein: S_j, for $2 \le j \le n-2$ and $j = n$, is the set of all the $j-1$-tuples one can build from $\{1, \dots, j\}$; but S_{n-1} is the set of all $n-2$ tuples one can build from $\{1, \dots, n\}$ (see Figure 4).

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

Fig. 4. Extended Trigon for Γ_4

Based on the two values i_0 and c_0, the simulator injects in the game many random instances, generated by (multiplicative) random self-reduction, from $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$ such that the Test-ed key is the $\mathsf{G\text{-}DH}$ secret value $g^{x_1 \cdots x_n}$ relative to $\mathcal{D}$. That is all the elements of S_n will be embedded into the protocol at c_0 when the adversary $\mathcal{A}$ asks the Test-query.

Δ cannot embed all the elements of S_n at c_0 since the players are not all added to the group at c_0. The strategy of Δ is as follows: embed the successive elements of instance $\mathcal{D}$ in the protocol flows in the order wherein the players join the group, until $n-1$ players have been involved and except for player i_0; just before the Test-query, embed the last elements of instance $\mathcal{D}$ via the broadcast operated (hopefully) by i_0; and after the Test-query, returns to line S_{n-1} with session keys in S_n.

This strategy allows Δ to deal with situations where n players are involved in the group *before* c_0, and are added and removed repeatedly. If, in effect, Δ embeds all the elements of S_n into the protocol execution the first time the size of $\mathcal{L}_0$ is n, Δ is not able to compute the session key value sk needed to answer to the Reveal-query. Before c_0, Δ uses truly random values instead of instance $\mathcal{D}$ for player U_{i_0} or if there are already $n-1$ players involved in the group. Note that Δ embeds elements of S_i when a new player U_i (except U_{i_0}) is added to the group $\mathcal{I}$ for the first time and Δ does not remove it when U_i leaves. This way, after the joining operation of the j-th player from $\mathcal{L}_0$, 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. To correctly deal with self-reducibility, Δ make use of variable $\mathcal{T}$ to reconstruct well-formatted (blinded) flows from $\mathcal{D}$.

When the c_0-th operation occurs, the last broadcast flow is operated by U_{i_0} who embeds line S_n of the trigon. If follows that the corresponding session key (which is the Test-ed key) is the $\mathsf{G\text{-}CDH}_{\Gamma_n}$ value $g^{x_1 \cdots x_n}$ relative to $\mathcal{D}$, blinded by self-reducibility. Δ then answers the Test-query as in the real protocol, according to the value of bit b.

However, Δ also needs to be able to answer to all queries *after* c_0 and more specifically the Reveal-queries (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). To this aim, Δ has to *un-embed* the element S_n from the protocol (in order to reduce the number of exponents taken from the instance $\mathcal{D}$) and it does it in the operation that occurs at $c_0 + 1$. However depending on which player performs that later operation, Δ may not be able to do it without going "out" of the basic trigon (but anyway with only $n-1$ exponents involved). This is the reason why the line S_{n-1} has to contain all the possible $(n-2)$-tuples: extension of the basic trigon illustrated on Figure 4. For the operations that will occur after $c_0 + 1$, Δ uses (random) blinding exponents for all the players including those in $\mathcal{L}_0$, keeping all the x_i but one in the flows[2]. Therefore, the future session keys will be derivated from the n-th line, but the broadcasts may involve any element in the *extended* $n-1$-th line.

The simulation is therefore undistinguishable from the game $\mathbf{G}_2$:

$$\Pr[\mathsf{S}_2] = \Pr[\mathsf{S}_3]. \tag{4}$$

Game $\mathbf{G_4}$. Game $\mathbf{G}_4$ is the same as game $\mathbf{G}_3$ except that the simulator is given as input an instance $\mathcal{D}$ from $\mathsf{G\text{-}DH}^{\$}_{\Gamma_n}$, with g^r as "candidate" solution. And in case $b = 1$, the value g^r is used to answer the Test-query. Then, the difference between $\mathbf{G}_3$ and $\mathbf{G}_4$ is upperbounded by the computational distance between the two distributions $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$ and $\mathsf{G\text{-}DH}^{\$}_{\Gamma_n}$, with $g^{x_1 \cdots x_n}$ and g^r respectively:

$$|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_4]| \leq \mathsf{Adv}^{\mathsf{gddh}_{\Gamma_n}}_{\mathbb{G}}(T'). \tag{5}$$

The running time of simulator in games $\mathbf{G}_2$ and $\mathbf{G}_3$ is essentially the same as in the previous game, except that each query may implies computation of up to n exponentiation needed for self-reducibility: $T' \leq T + nQT_{exp}(k)$, where $T_{exp}(k)$ is the time needed to perform an exponentiation modulo a k-bit number.

Game $\mathbf{G_5}$. Game $\mathbf{G}_5$ is the same as $\mathbf{G}_4$, except that the Test-query is answered with a completely random value, independently of b. It is then straightforward that $\Pr[\mathsf{S}_5] = 1/2$. Let δ_2 be the distance between the output of $F_2(.)$ and the uniform distribution, we have:

$$|\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]| \leq \delta_2. \tag{6}$$

[2] Another solution would have been to guess which player performs the operation at $c_0 + 1$. With this second guess j_0, the extension of the trigon would have contained all the $n-2$ tuples but those containing both i_0 and j_0.

Putting all together Equations (1), (2), (3), (4), (5), (6), we get

$$\begin{aligned}\Pr[\mathsf{S}_0] &= \Pr[\mathsf{S}_0 \wedge \mathsf{Forge}] + \Pr[\mathsf{S}_0 \wedge \neg\mathsf{Forge}] \\ &\leq \Pr[\mathsf{Forge}] + \Pr[\mathsf{S}_1] = \Pr[\mathsf{Forge}] + nQ\left(\Pr[\mathsf{S}_2] - \frac{1}{2}(1 - 1/nQ)\right) \\ &\leq \Pr[\mathsf{Forge}] + nQ\left(\Pr[\mathsf{S}_2] - \frac{1}{2}\right) + \frac{1}{2} \\ &\leq \Pr[\mathsf{Forge}] + nQ\left(\Pr[\mathsf{S}_5] + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(T) + \delta_2 - \frac{1}{2}\right) + \frac{1}{2} \\ &\leq \Pr[\mathsf{Forge}] + nQ\left(\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(T) + \delta_2\right) + \frac{1}{2}.\end{aligned}$$

The theorem then follows from lemma 5. □

B Proof of lemma 5

Proof. Our goal here is to upper bound the probability of the "bad" event Forge. Forge is the event the adversary $\mathcal{A}$ outputs during the attack a MAC forgery *before* corrupting a player. To reach this aim we evaluate the probability of Forge in a sequence of games $\mathbf{G}'_0, \ldots, \mathbf{G}'_4$. We formally refer to Forge'_i as the event Forge in game $\mathbf{G}'_i$.

Game $\mathbf{G}'_0$: The game $\mathbf{G}'_0$ is defined as being the real attack against our protocol: $\mathbf{G}'_0 = \mathbf{G}_0$.

Game $\mathbf{G}'_1$. The game $\mathbf{G}'_1$ is identical to $\mathbf{G}'_0$, except that each MAC key K_{ij} is computed as $F_1(g^{r_{ij}})$, where r_{ij} is a random, instead of $F_1(g^{x_i x_j})$. It follows that the difference between the two games is upper-bounded by the computational distance $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T)$:

$$|\Pr[\mathsf{Forge}'_0] - \Pr[\mathsf{Forge}'_1]| \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T).$$

Game $\mathbf{G}'_2$. Game $\mathbf{G}'_2$ is identical to $\mathbf{G}'_1$ except that instead of chosen each MAC key K_{ij} as the output of key derivation map F_1 we choose them at random according to the uniform distribution. Thus, the difference between the two games is upperbounded by a function in the distance δ_1 of the output of F_1 from the uniform distribution.

More precisely, we use a classical "hybrid distribution" technique and define an (ordered) sequences of auxiliary games ${\mathbf{G}'_2}^{ij}$ $(1 \leq i < j \leq n)$. Given $1 \leq i < j \leq n$, game ${\mathbf{G}'_2}^{ij}$ is identical to $\mathbf{G}'_1$ except that all MAC keys K_{kl} for $(k < i)$ or $(k = i, l \leq j)$ are replaced by a uniformly chosen random key. Then ${\mathbf{G}'_2}^{11} = \mathbf{G}'_1$ whereas ${\mathbf{G}'_2}^{n-1,n} = \mathbf{G}'_2$. There are $n(n-1)/2$ such games and the only difference between two "consecutive" auxiliary games is upperbounded by δ_1. It then follows that:

$$|\Pr[\mathsf{Forge}'_1] - \Pr[\mathsf{Forge}'_2]| \leq \frac{n(n-1)}{2}\delta_1.$$

In case the map F_1 were a random oracle we would have had the distance δ_1 equal to 0. If the map F_1 is based on a universal hash function and left-over hash lemma (see lemma 3), we have $\delta_1 \leq 2^{-(e+1)}$. Recall that the latter hash functions use as input of a random value and this value is either certified or sent as part of the protocol flows.

Game $\mathbf{G}'_3$. The game $\mathbf{G}'_3$ is identical to $\mathbf{G}'_2$, except that the simulator chooses at random two indices a and b, $a < b$, in $[1, n]$ and aborts if no MAC forgery w.r.t. K_{ab} occurs before a $\mathsf{Corrupt}$-query. The probability of correctly guessing a and b is $\frac{2}{n(n-1)}$. It follows that:

$$\Pr[\mathsf{Forge}'_3] = \frac{2}{n(n-1)} \Pr[\mathsf{Forge}'_2].$$

Game $\mathbf{G}'_4$. The game $\mathbf{G}'_4$ is identical to $\mathbf{G}'_3$, except that the simulator is given access to a MAC.$\textsc{Sgn}$-oracle and will use it to authenticate the flows between players a and b. All other MAC keys are known, uniformly distributed values. If the MAC scheme uses uniformly distributed keys, the two games are identical and $\Pr[\mathsf{Forge}'_4] = \Pr[\mathsf{Forge}'_3]$. By construction the probability of Forge'_4 is exactly the probability of breaking the security of the MAC scheme:

$$\Pr[\mathsf{Forge}'_4] = \mathsf{Succ}^{\mathsf{cma}}_{\mathsf{mac}}(T).$$

Finally, we easily get:

$$\Pr[\mathsf{Forge}] = \Pr[\mathsf{Forge}'_0] \leq \mathsf{Adv}^{\mathsf{mddh}_n}_{\mathbb{G}}(T) + \frac{n(n-1)}{2}\delta_1 + \frac{n(n-1)}{2}\mathsf{Succ}^{\mathsf{cma}}_{\mathsf{mac}}(T).$$

$\square$

Setup($\mathcal{J}$)	Initialize new variables $\mathcal{T}$ and $\mathcal{L}_0$ to $\emptyset$ Increment c Initialize new multicast group $\mathcal{I}' \leftarrow \mathcal{J}$ $u \leftarrow \min(\mathcal{J})$ • $c < c_0$: Update $\mathcal{L}_0$ up to cardinality $n-1$ with $\mathcal{J}$, except i_0 $u \neq i_0 \Rightarrow$ simulate the answer using RSR according to $\mathcal{T}$ $u = i_0 \Rightarrow$ do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow • $c = c_0$: $\#(\mathcal{L}_0) \neq n \Rightarrow$ output "`Fail`" $\#(\mathcal{J}) = n \Rightarrow \mathcal{L}_0 \leftarrow \mathcal{J}$ then simulate the answer using RSR according to $\mathcal{T}$ • $c > c_0$: do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow
Join($\mathcal{I}, \mathcal{J}$)	Increment c $u \leftarrow \max(\mathcal{I})$ Initialize a new multicast group $\mathcal{I}' \leftarrow \mathcal{I} \cup \mathcal{J}$ Update $\mathcal{L}_0$ up to cardinality $n-1$ with $\mathcal{J}$, except i_0 • $c < c_0$: $u \in \mathcal{L}_0 \Rightarrow$ simulate the answer using RSR according to $\mathcal{T}$ $u \notin \mathcal{L}_0 \Rightarrow$ do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow • $c = c_0$: $\mathcal{L}_0 \leftarrow \mathcal{L}_0 \cup \{i_0\}$ $(\max(\mathcal{J}) \neq i_0) \vee (\mathcal{I}' \nsubseteq \mathcal{L}_0) \vee (\#(\mathcal{L}_0) \neq n) \Rightarrow$ output "`Fail`" simulate the answer using RSR according to $\mathcal{T}$ • $c = c_0 + 1$: $\mathcal{L}_0 \leftarrow \mathcal{L}_0 \backslash \{u\}$ • $c > c_0$: $u \in \mathcal{L}_0 \Rightarrow$simulate the answer using RSR according to $\mathcal{T}$ $u \notin \mathcal{L}_0 \Rightarrow$do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow
Remove($\mathcal{I}, \mathcal{J}$)	Increment c Initialize a new multicast group $\mathcal{I}' \leftarrow \mathcal{I} \backslash \mathcal{J}$ $u \leftarrow \max(\mathcal{I}')$ • $c < c_0$: $u \in \mathcal{L}_0$ simulate the answer using RSR according to $\mathcal{T}$ $u \notin \mathcal{L}_0 \Rightarrow$ do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow • $c = c_0$: $\mathcal{L}_0 \leftarrow \mathcal{L}_0 \cup \{i_0\}$ $(u \neq i_0) \vee (\mathcal{I} \nsubseteq \mathcal{L}_0) \vee (\#(\mathcal{L}_0) \neq n) \Rightarrow$ output "`Fail`" simulate the answer using RSR according to $\mathcal{T}$ • $c = c_0 + 1$: $\mathcal{L}_0 \leftarrow \mathcal{L}_0 \backslash \{u\}$ • $c > c_0$: $u \in \mathcal{L}_0 \Rightarrow$simulate the answer using RSR according to $\mathcal{T}$ $u \notin \mathcal{L}_0 \Rightarrow$do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow
Send(Π_i^t, m)	• $c \neq c_0$: $i \in \mathcal{L}_0 \Rightarrow$ simulate the answer using RSR according to $\mathcal{T}$ $i \notin \mathcal{L}_0 \Rightarrow$do as in P using $r_u \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$ to generate the flow • $c = c_0$: simulate the answer using RSR according to $\mathcal{T}$
Reveal(Π_i^t)	`If` U_i has accepted `Then` `If` $c = c_0$ `Then` output "`Fail`" `Else` return $sk_{\Pi_i^t}$.
Corrupt(U_i)	return LL_{U_i}.
Test (U_i)	`If` U_i has accepted `Then` `If` $c = c_0$ `Then` return $F_2(g^{r\rho})$, where ρ is an adequate blinding exponent. `Else` output "`Fail`".

Fig. 5. Game $\mathbf{G}_4$. The multicast group is $\mathcal{I}$. The Test-query is "guessed" to be made: after c_0 operations, the multicast group is $\mathcal{L}_0$, and the last joining player is U_{i_0}. In the variable $\mathcal{T}$, Δ store which exponents of instance $\mathcal{D}$ have been injected in the game so far. RSR holds for *random self-reducibility.*