Provably-Secure Authenticated Group Diffie-Hellman Key Exchange

Emmanuel Bresson, Olivier Chevassut, and David Pointcheval

[1] DCSSI Crypto Laboratory, 75700 Paris 07 SP, France
Emmanuel.Bresson@polytechnique.org – http://crypto.bresson.org.
[2] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
http://www.dsd.lbl.gov/∼chevassu – OChevassut@lbl.gov.
[3] CNRS – École normale supérieure, 75230 Paris Cedex 05, France
http://www.di.ens.fr/users/pointche – David.Pointcheval@ens.fr

Abstract: Authenticated key exchange protocols allow two participants A and B, communicating over a public network and each holding an authentication means, to exchange a shared secret value. Methods designed to deal with this cryptographic problem ensure A (resp. B) that no other participants aside from B (resp. A) can learn any information about the agreed value, and often also ensure A and B that their respective partner has actually computed this value. A natural extension to this cryptographic method is to consider a pool of participants exchanging a shared secret value and to provide a formal treatment for it. Starting from the famous 2-party Diffie-Hellman (DH) key exchange protocol, and from its authenticated variants, security experts have extended it to the multi-party setting for over a decade and completed a formal analysis in the framework of modern cryptography in the past few years. The present paper synthesizes this body of work on the provably-secure authenticated group DH key exchange.

The present paper revisits and combines the full versions of the following four papers:

- E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater.
  Provably authenticated group Diffie-Hellman key exchange.
  In P. Samarati, editor, *Proc. of ACM CCS '01*, pages 255–264. ACM Press, November 2001.
- E. Bresson, O. Chevassut, and D. Pointcheval.
  Provably authenticated group Diffie-Hellman key exchange – the dynamic case.
  In C. Boyd, editor, *Proc. of Asiacrypt '01*, volume 2248 of *LNCS*, pages 290–309. Springer-Verlag, December 2001.
- E. Bresson, O. Chevassut, and D. Pointcheval.
  Dynamic group Diffie-Hellman key exchange under standard assumptions.
  In L. Knudsen, editor, *Proc. of Eurocrypt '02*, volume 2332 of *LNCS*, pages 321–336. Springer-Verlag, May 2002.
- E. Bresson, O. Chevassut, and D. Pointcheval.
  The Group Diffie-Hellman Problems.
  In H. Heys and K. Nyberg, editors, *Proc. of Selected Areas in Cryptography '02*, volume 2595 of *LNCS*, pages 325–338. Springer-Verlag, August 2002.

1 Introduction

1.1 Motivation

The idea of modern cryptography is to identify cryptographic problems that need to be solved and to provide a rigorous treatment for them. An essential problem in distributed computing (e.g, scientific and conferencing applications, Grid applications [15, 41]) is the ability to establish a security context within which messages sent over the wire are encrypted and authenticated. A cryptographic means to do that is to have the distributed system's components exchange a secret value and to use this value to compute the keying material for a symmetric cipher and a Message Authentication Code [8, 55]. The keying material is set as the output of a key-derivation function that maps the secret value to the (bit-string) keys of the symmetric algorithms. All being considered, the critical step in the establishment of this security context clearly remains the mechanism for exchanging the secret value. This step is often carried via a DH key exchange [38] or, in the group scenario through its possible generalizations (see, e.g., [67, 46, 66, 30]).

The Diffie-Hellman (DH) key exchange, as well as some generalizations, were initially designed to protect against a *passive* adversary that only eavesdrops on messages. However, when it comes to implement these schemes in a distributed system's security architecture a much stronger adversary must be taken into account. Hackers have a great deal of control over our Internet communications. They can relay, schedule, inject, and alter our messages, or even try to impersonate us via man-in-the-middle attacks. One way to prevent these *active* attacks is to add authentication services to the group key exchange protocol. In spite of the apparent simplicity of adding authentication services to a group key exchange, it is a task fraught with many complications. Many authenticated key exchange protocols were later found to be flawed and in some cases the flaws even took years before being discovered (see for instance [16, 39, 55, 59] and the discussion in section 1.3 below). One way to avoid many of the flaws is to provide a formal treatment in the framework of modern cryptography.

Active attacks are even easier to mount and more destructive as middleware technologies enable the exchange of data among a large number of components which form a multicast group [3, 17, 14, 70]. These technologies provide *asynchronous* and *reliable* communication channels to coordinate the distributed application's components spread on the Internet. Each component shares responsibility for parts of a task and coordinates its efforts with the other components. In this environment prone to faults (e.g. faults can result from host failures, network failures, network congestion, CPU load, or malice) creating a security context —within which messages are protected— is challenging as application's components join and leave the multicast group [1, 2, 62]. Accommodating this dynamic membership means updating the secret value after each change in the membership of the multicast group. This step is often carried via a *dynamic* group DH key exchange [26, 27, 68].

1.2 Contribution

The first contribution of the paper is to provide cryptographic experts with a *provable-security* framework to assess the security of authenticated group key exchange protocols. The framework captures the adversary's capabilities and defines the security requirements to satisfy. It is the result of three successive papers. In the first one [29], we have captured the characteristics of an authenticated key exchange which allows a pool of participants, communicating over a public network and each holding a pair of public/private keys, to agree on a session key — these participants do not share any secrets before hand. In real life, however, the membership of the group is not built once and for all but is built incrementally as the network topology evolves [1, 2, 62]. Participants can indeed join/leave the pool at any time or the pool itself can be split into disjoint components due to network faults or malice. Thus, in the second paper [26], we have equipped our framework with this notion of dynamicity in the membership. This is done by enhancing the framework with additional, atomic[1] operations which enable the group to grow or decrease: an authenticated dynamic group key exchange allows an existing pool of participants to update the value of their session key after each change in the membership so that this value is only known to the members of the newly formed pool. We note that re-running the protocol from scratch is always possible, and hence the goal of such operations is to provide an efficient means to *update* the existing session key into a new one. Finally, in [27], we have captured the ability to initiate parallel executions of a dynamic group key exchange; *concurrency* is an important feature to consider when a key exchange is meant for practical use. An authenticated group key exchange provides a set of participants with an interactive protocol to exchange a session key and, therefore form a secure group. In real life, however, the participants may be part of several pools at the same time and, therefore, may need to run multiple key exchanges in parallel. Later on these participants may close one session while keeping the others opened. As this simplistic scenario shows concurrency introduces technical difficulties in the security analysis since an adversary could inject data extracted from one execution into another one to defeat the security of this later key exchange. Concurrent executions are more realistic than sequential ones and must be included in a provable-security framework for authenticated dynamic group key exchange.

In addition to the formal security model, the second contribution of this work is to provide engineers with a *generic* authenticated group DH key-exchange construction which once instantiated leads to the schemes of Bresson *et al.* [29, 26, 27]. The construction is described in terms of modules that perform the key-exchange and the authentication operations. The modules can be instantiated via processes [29, 26] or hardware devices [27] that use tamper detection to not reveal any information. Embedding the critical cryptographic material in some hardware cryptographic devices is at least as good as erasing secrets [57, 58, 71]; cryptographers assume and usually do not explicitly state that secrets are definitively and reliably erased (only the most recent secrets are kept) [37, 48]. In our security model as described in [27], we have captured the adversary's ability to gain access to the internal memory of participants and incorporated in the framework the action of erasing a secret. The generic authenticated group DH key-exchange construction achieves in a provably-secure and practical way the security requirements specified in the framework. Provable-security is reached

[1] We do not deal with the cases where participants decide to halt *during* an execution of the protocol itself. Our Join and Remove operations are simply formal tools to describe evolutions of a group, step by step —by one or several members at once— and assuming each of these steps is done using the appropriate algorithm. Premature halting during execution of such an algorithm is not considered here (more precisely, it is not considered further than what the adversary can basically do: block messages and turn into infinite time-out).

by constructing a reduction showing that, in our formal framework, the scheme achieves the afore mentioned security requirements under reasonable intractability assumptions.

1.3 Related Work

General issues A comprehensive treatment of *"Protocols for Authentication and Key Establishment"* can be found in Boyd and Mathuria's book [23]. In previous papers [21, 22], Boyd gave an overview of key agreement issues; his work provides a high-level classification of 2-party and multi-party key agreement protocols, and a discussion of their security, depending on which class of function is used to combine the nonces of each party. More bibliography can be found in the Handbook of Applied Cryptography [55]. It is important to distinguish two kinds of scenario: in the first one, *key distribution* (also known as *key transport*), the key is chosen by a single party and provided to the participants. In the case of *key agreement* (also referred as *key exchange*), all users participate in determining the key value. In the present paper, we concentrate exclusively on (group) key agreement.

Security models for group key agreement In the framework of modern cryptography one finds a formal model and security definitions for the task of exchanging a secret value —the so-called session key—. Bellare and Rogaway proposed a formal model wherein the instances of a player are modeled via *oracles*, the capabilities of the adversary are modeled via *queries* to these oracles, and the secrecy of the session key is modeled via the notion of semantic security [42]. This model was originally used to analyze the security of methods for key distribution [11]. In [13], they consider a three-party scenario, in the on-line TTP (*trusted third party*) setting, in which an incoercible server is available to the parties; it has been later extended to the public-key setting by Blake-Wilson *et al.* [19, 18] and a specific adaptation was done few years ago by Bellare *et al.* in the password-based key exchange setting [10]. Another kind of security models is based on the multi-party simulatability technique, and was initiated by Bellare, Canetti and Krawczyk [9]; further refinements were proposed by Canetti and Krawczyk: in [34], they make use of the indistinguishability approach as proposed in [13] to propose the notion of secure channels; then in [35] they developed the property of universal composability (UC) of such channels. In 1999, Shoup [64] provided a technical modification of the original work by Bellare, Canetti and Krawczyk, in particular he took into account several corruption models in order to encompass the forward-secrecy property (which states that knowing long-term keys does not help in compromising previously established session keys). Our treatment of the authenticated group key-exchange is derived from the first kind of approach [11, 10]. We provided the first formal security models and proven secure protocols in our series of papers [29, 26, 27].

Previous work on Group Diffie-Hellman There have been several protocols aiming to generalize the DH key exchange [38] to the multi-party setting.

These were tackled by Ingemarsson *et al.* [46], Diffie *et al.* [66], Burmester and Desmedt [30], and Steiner *et al.* [67]. The use of "multiple-decker" exponents in the protocol of Diffie *et al.* makes it difficult to reduce the security of the protocol to the standard DH problem and, therefore, its security is heuristic. In 1996, Steiner *et al.* proposed a natural extension to DH, named the group DH key exchange [67] which in 2001 we enhanced with authentication services and proved it secure [29]. This authentication enhancement and the formal model for its analysis are at the core of the present work. We note that the works by Ateniese *et al.* [4, 5] also aim at adding authentication services to the schemes by Steiner *et al.*, however the security proof was only informal.

Previous to the work by Steiner *et al.*, Diffie *et al.* [39] presented the STS (*Station-to-Station*) protocol, but this protocol does not cover concurrent executions. Also, the well-known protocol by Burmester and Desmedt [30] is a very elegant protocol, which interestingly achieves a constant-round complexity. However, as shown by Just and Vaudenay [49], it does not achieve key authentication.

Dynamicity for group key agreement The notion of *dynamicity* in the group membership was pioneered by Steer *et al.* [66]. Adding members to the group is easy, but removing them is not. Steiner *et al.* [68] modified their original method for group DH key exchange [67, 4] to easily add and remove members from the group. In addition, Ateniese *et al.* [4, 5] identified additional, useful security notions for a group key exchange (such as Perfect Forward-Secrecy, Contributory, Key Confirmation) and informally show how to enhance [68] with authentication. The present paper describes our contribution, based on their works, in order to achieve provable security in *dynamic* groups [26, 27].

Other researchers have proposed methods for dynamic group DH key exchange. Perrig extends the work of one-way function trees (OFT, originally introduced by McGrew and Sherman [54]) to design a tree-based key agreement scheme for peer groups [60]. However, this work lacked the facilities for handling group partitions and merges. Further refinements by Kim *et al.* [51, 52] addressed these issues but do not specify a rigorous security model for a formal proof.

Protocols' complexity The schemes we analyze in this paper are directly derived from those by Steiner *et al.* and, thus, have linear complexity. For this reason, it is not reasonable to use them at a large or even medium scale. However, we emphasize that the main contribution of this work remains the formal model for provable security, and we insist that many recently proposed schemes for group key exchange have been analyzed using our model (see, e.g., [50, 24]).

The round complexity of a key agreement protocol becomes critical at a large scale. The paper by Becker and Wille [7] also gave 1 single round as an optimal lower complexity bound for multi-party key agreement. Joux used pairings to design a one-pass 3-party Diffie-Hellman key exchange [47], but generalizing his construction with multi-linear forms seems to be hard [40]. In 2003, Boyd and Nieto came up with a round-optimal protocol [24], however, their solution does not provide forward-secrecy.

Secret-sharing techniques also give advantage to design methods for group key exchange. Li *et al.* [53] proposed the first key-exchange method based on secret-sharing; by using polynomial-secret-sharing tools Tzeng [69] proposed a fault-tolerant protocol with constant-round complexity but in which the message-complexity per user is proportional to the number of users. Later Cachin and Strobl [31] provide a formal analysis of an (optimal) fault-tolerant scheme, in the framework of asynchronous reactive systems (such as [32, 61]). On the other hand, Bresson and Catalano [25] designed a scheme with both message-efficiency and constant round complexity, but without fault-tolerance.

Using cryptographic hardware protections We note that the use of cryptographic hardware devices for session key *distribution* was already explore by Rubin and Shoup [63]. Even though a cryptographic method is proved secure, security can sometimes be compromised when the method is incorrectly implemented. Cryptographers assume (and usually do not explicitly state) that secrets are *definitively and reliably erased* (only the most recent secrets are kept) [37, 48]. In our 2002 paper [27] we incorporate the cryptographic action of erasing a secret. This allows us to consider forward-secrecy issues: in the strong-corruption model as defined by Bellare *et al.* [10], in which the corruption of a player reveals his internal state (including "ephemeral" data), one can prevent attacking the session key before or after the lifetime of these data. When dealing with the weak-corruption model, in which corruption reveals only the long-term key, we achieve Perfect Forward Secrecy: knowledge of a long-live key is useless for obtaining *any* past session key. Our model assumes these critical data are embedded in some hardware cryptographic devices which are at least as good as erasing a secret [57, 58, 71]. In other words, we offer a technological choice: either the previously used data are tamper-protected or they are securely erasable.

1.4 Organization of the Paper

The remainder of the paper is organized as follows. In Section 2, we introduce the group DH assumptions and show how these assumptions relate to the DH assumptions. In Section 3, we present our provable-security framework and abstract out the functionalities of the authenticated group DH key exchange. In Section 4, we describe the AKE1 method for authenticated group DH key exchange. In Section 5 we show that it is provably secure in the standard model under the classical decisional Diffie-Hellman assumption. We finally conclude the paper.

2 Computational Problems

We first present the notion of group DH distribution and use it to define the computational and decisional group DH assumptions. Our adversary is time-constrained which means that all the success probabilities and advantages — $\mathsf{Succ}(t, \ldots))$ and $\mathsf{Adv}(t, \ldots)$ respectively— represent the maximal probabilities over all the adversaries running in time t.

2.1 The Group Diffie-Hellman Distribution

Given $\mathbb{G} = \langle g \rangle$ a cyclic group of prime order q, n an integer, I_n the set $\{1, \ldots, n\}$, $\mathcal{P}(I_n)$ the set of all subsets of I_n, and Γ a subset of $\mathcal{P}(I_n)$ such that $I_n \notin \Gamma$, the *Group Diffie-Hellman distribution* relative to Γ is defined as follows (with the convention that $\prod_\emptyset x_i = 1$):

$$\mathsf{GDH}_\Gamma = \{\mathcal{V}iew_\Gamma(x_1, \ldots, x_n) \,|\, x_1, \ldots, x_n \in_R \mathbb{Z}_q\},$$
$$\text{where} \quad \mathcal{V}iew_\Gamma(x_1, \ldots, x_n) = \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle|\, J \in \Gamma \right\}.$$

When there is no risk of confusion, we will simply describe the tuple $\mathcal{V}iew$ as a collection of group elements (rather than a collection of pairs). Since this distribution is a function of the parameters n and Γ it could be instantiated with any of the following special forms:

- *The Diffie-Hellman distribution:* $n = 2$ and $\Gamma = \{\{1\}, \{2\}\}$.
- *The basic trigon* (see Figure 1): Γ has the following triangular structure $\mathcal{T}_n$ (which is involved in the security of the group DH method [29]):

$$\mathcal{T}_n = \bigcup_{1 \leq j \leq n} \bigcup_{1 \leq k \leq j} \{\, \{i \,|\, 1 \leq i \leq j, i \neq k\} \,\}$$

$j = 1$					$\{\}$	S_1
$j = 2$				$\{1\}$	$\{2\}$	S_2
$j = 3$			$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	S_3
$j = 4$ $(= n-1)$		$\{1,2,3\}$	$\{1,2,4\}$	$\{1,3,4\}$	$\{2,3,4\}$	S_4
$j = 5$ $(= n)$	$\{1,2,3,4\}$	$\{1,2,3,5\}$	$\{1,2,4,5\}$	$\{1,3,4,5\}$	$\{2,3,4,5\}$	S_5

basic trigon

Fig. 1. GDH-Distribution for the Basic Trigon (Example when $n = 5$ and $\Gamma = \mathcal{T}_5$)

- *The extended trigon* (see Figure 2): Γ has the following structure $\mathcal{E}_n$ (which is involved in the security of the dynamic group DH methods [26, 27]): it is similar to the above $\mathcal{T}_n$ structure but with an extended $n-1$-th line.

$$\mathcal{E}_n = \bigcup_{1 \leq j \leq n-2} \bigcup_{1 \leq k \leq j} \{\, \{i \,|\, 1 \leq i \leq j, i \neq k\} \,\}$$
$$\cup \bigcup_{1 \leq k < l \leq n} \{\, \{i \,|\, 1 \leq i \leq n, i \neq k, l\} \,\} \cup \bigcup_{1 \leq k \leq n} \{\, \{i \,|\, 1 \leq i \leq n, i \neq k\} \,\}$$

- *The Generalized group Diffie-Hellman distribution:* $\Gamma = \mathcal{P}(I_n) \backslash \{I_n\}$ is *all* the proper subsets of $\{1, \ldots, n\}$ [20, 56, 67].

				{}							S_1
			{1}	{2}							S_2
		{1,2}	{1,3}	{2,3}							S_3
	{1,2,3}	{1,2,4}	{1,3,4}	{2,3,4}	{1,2,5}	{1,3,5}	{1,4,5}	{2,3,5}	{2,4,5}	{3,4,5}	S_4
{1,2,3,4}	{1,2,3,5}	{1,2,4,5}	{1,3,4,5}	{2,3,4,5}							S_5
basic trigon					extension						

Fig. 2. GDH-Distribution for the Extended Trigon (Example when $n = 5$ and $\Gamma = \mathcal{E}_5$)

2.2 The Group Diffie-Hellman Problem

Given an integer n and a structure Γ, a (t, ϵ)-*Group Computational Diffie-Hellman attacker* ($\mathbf{G\text{-}CDH}_\Gamma$-attacker for short) for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time t that given a tuple from GDH_Γ, outputs $g^{x_1 \cdots x_n}$ with probability greater than ϵ:

$$\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_\Gamma}(\Delta) \stackrel{\mathsf{def}}{=} \Pr_{x_i}\left[\Delta(\mathcal{V}iew_\Gamma(x_1, \ldots, x_n)) = g^{x_1 \cdots x_n}\right] \geq \epsilon.$$

The $\mathbf{G\text{-}CDH}_\Gamma$ problem is (t, ϵ)-**intractable** if there is no (t, ϵ)-$\mathbf{G\text{-}CDH}_\Gamma$-attacker for $\mathbb{G}$. The $\mathbf{G\text{-}CDH}_\Gamma$-assumption states this is the case for all polynomial t and non-negligible ϵ, for a family $\mathbf{\Gamma} = \{\Gamma_n\}_n$. If $n = 2$, we get the well-known *Computational Diffie-Hellman* problem, for which we use the straightforward notation $\mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}(\cdot)$.

2.3 The Group Decisional Diffie-Hellman Problem

The decisional problem consists, informally, to distinguish between $g^{x_1 \cdots x_n}$ and a random power g^r. To that goal, we either add to the tuple $\mathcal{V}iew(x_i)$ the "right" value or a random one, obtaining two kinds of tuples $\mathcal{V}iew^{\$}$ and $\mathcal{V}iew^{\star}$. Thus it leads to two additional distributions from the GDH-distribution:

$$\begin{aligned}\mathsf{GDH}_\Gamma^{\star} &= \{\mathcal{V}iew_\Gamma^{\star}(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in_R \mathbb{Z}_q\},\\ \mathsf{GDH}_\Gamma^{\$} &= \left\{\mathcal{V}iew_\Gamma^{\$}(x_1, \ldots, x_n, r) \mid x_1, \ldots, x_n, r \in_R \mathbb{Z}_q\right\},\end{aligned}$$

where

$$\begin{aligned}\mathcal{V}iew_\Gamma^{\star}(x_1, \ldots, x_n) &= \mathcal{V}iew_\Gamma(x_1, \ldots, x_n) \cup \{(I_n, g^{x_1 \cdots x_n})\}\\ \mathcal{V}iew_\Gamma^{\$}(x_1, \ldots, x_n, r) &= \mathcal{V}iew_\Gamma(x_1, \ldots, x_n) \cup \{(I_n, g^r)\}\end{aligned}$$

Given an integer n and a structure Γ, a (t, ϵ)-*Group Decisional Diffie-Hellman distinguisher* ($\mathbf{G\text{-}DDH}_\Gamma$-distinguisher for short) for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time t that given an element X from either $\mathsf{GDH}_\Gamma^{\$}$ or $\mathsf{GDH}_\Gamma^{\star}$ outputs 0 or 1 such that:

$$\begin{aligned}&\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_\Gamma}(\Delta) \stackrel{\mathsf{def}}{=}\\ &\left|\Pr_{x_i}\left[\Delta\big(\mathcal{V}iew_\Gamma^{\star}(x_1, \ldots, x_n)\big) = 1\right] - \Pr_{x_i, r}\left[\Delta\big(\mathcal{V}iew_\Gamma^{\$}(x_1, \ldots, x_n, r)\big) = 1\right]\right| \geq \epsilon\end{aligned}$$

The $\mathbf{G\text{-}DDH}_\Gamma$-problem is (t, ϵ)-**intractable** if there is no (t, ϵ)-$\mathbf{G\text{-}DDH}_\Gamma$-distinguisher for $\mathbb{G}$. The **G-DDH**-assumption states this is the case for all polynomial t and non-negligible ϵ, for a family $\boldsymbol{\Gamma} = \{\Gamma_n\}_n$. If $n = 2$, we get the well-known *Decisional Diffie-Hellman* problem, for which we use the straightforward notation $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\cdot)$.

2.4 The Random Self-Reducibility Property

The Diffie-Hellman problems have the nice property of random self-reducibility. Certainly the most common is the additive random self-reducibility, which works as follows. Given, for example, a $\mathbf{G\text{-}CDH}_\Gamma$-instance with $\Gamma = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}\}$, $\mathcal{V}iew = \mathcal{V}iew_\Gamma(x_1, x_2, x_3) = (g^{x_1}, g^{x_2}, g^{x_3}, g^{x_1x_2}, g^{x_2x_3}, g^{x_1x_3})$ for any x_1, x_2, x_3 it is possible to generate a random instance

$$\begin{aligned}\mathcal{V}iew' &= \mathcal{V}iew_\Gamma(x_1 + r_1, x_2 + r_2, x_3 + r_3)\\ &= (g^{(x_1+r_1)}, g^{(x_2+r_2)}, g^{(x_3+r_3)},\\ &\quad g^{(x_1+r_1)(x_2+r_2)}, g^{(x_2+r_2)(x_3+r_3)}, g^{(x_1+r_1)(x_3+r_3)})\end{aligned}$$

where r_1, r_2 and r_3 are random numbers in $\mathbb{Z}_q$, whose solution may help us to solve $\mathcal{V}iew$. Indeed, given the solution $z = g^{(x_1+r_1)\cdot(x_2+r_2)\cdot(x_3+r_3)}$ to the instance $\mathcal{V}iew'$ it is possible to recover the solution $g^{x_1x_2x_3}$ to the random instance $\mathcal{V}iew$:

$$\begin{aligned}g^{x_1x_2x_3} = z \cdot (g^{x_1x_2})^{-r_3} \cdot (g^{x_1x_3})^{-r_2} \cdot (g^{x_2x_3})^{-r_1} \cdot (g^{x_1})^{-r_2r_3}\\ \cdot (g^{x_2})^{-r_1r_3} \cdot (g^{x_3})^{-r_1r_2} \cdot g^{-r_1r_2r_3}.\end{aligned}$$

However the cost of such a computation may be high; furthermore it is easily seen that such a reduction works for the *Generalized* DH-distribution Γ only and thus its cost increases exponentially with the size of $\mathcal{V}iew$.

On the other hand, the multiplicative random self-reducibility works for any form of the GDH-problems in a prime order cyclic group. Given, for example, a $\mathbf{G\text{-}CDH}_\Gamma$-instance with $\Gamma = \{\{1\}, \{2\}, \{1, 2\}, \{1, 3\}\}$, $\mathcal{V}iew = \mathcal{V}iew_\Gamma(x_1, x_2, x_3) = (g^{x_1}, g^{x_2}, g^{x_1x_2}, g^{x_1x_3})$ for any x_1, x_2, x_3 it is easy to generate a random instance

$$\mathcal{V}iew' = \mathcal{V}iew_\Gamma(x_1r_1, x_2r_2, x_3r_3) = (g^{x_1r_1}, g^{x_2r_2}, g^{x_1r_1\cdot x_2r_2}, g^{x_1r_1\cdot x_3r_3})$$

where r_1, r_2 and r_3 are random numbers in $\mathbb{Z}_q^*$. And given the solution K' to the instance $\mathcal{V}iew'$, we directly get the solution $K = {K'}^\delta$, where $\delta = (r_1r_2r_3)^{-1} \bmod q$, to the instance $\mathcal{V}iew$. Such a reduction is efficient and only requires a linear number of modular exponentiations, but is restricted to prime order groups. The latter restriction is not so strong since these groups are anyway the usual ones, where the Diffie-Hellman problems are the most difficult to solve.

2.5 Relations among the Diffie-Hellman Problems

In our paper [28], we state several relations between all these problems.

Theorem 1 (– Intractability of GDDH). *The intractability of the* Group Decisional Diffie-Hellman problem *is implied by the intractability of the* Decisional Diffie-Hellman problem. *If Γ is either the basic trigon or the extended trigon, then we have:*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_\Gamma}(t) \leq (2n-3)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ with } t' \leq t + n^3 t_{\mathbb{G}},$$

where $t_{\mathbb{G}}$ is the time needed for an exponentiation in $\mathbb{G}$.

Theorem 2 (– Intractability of GCDH). *The intractability of the* Group Computational Diffie-Hellman problem *is implied by the intractability of the* Computational Diffie-Hellman problem *and the intractability of the* Decisional Diffie-Hellman problem. *If Γ is either the basic trigon or the extended trigon, then we have:*

$$\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_\Gamma}(t) \leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}(t') + (n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ where } t' \leq t + n^3 t_{\mathbb{G}}.$$

The proofs of these two theorems are provided in appendix and were originally published in [28]. More precisely, in the later paper, we have identified formal criteria allowing us to define "good structures" Γ for which the hybrid reduction above can actually be performed. The basic and extended trigons do satisfy these criteria, and are thus considered as appropriate for the reduction theorem.

3 Model

In this section, we describe our formal model, which is, again, derived from that by Bellare and Rogaway [11, 13]. The formalism models instances of players via oracles available to the adversary through queries.

3.1 Players

We fix a nonempty set $\mathcal{U}$ of N players that can participate in a group key exchange protocol P. A player $U_i \in \mathcal{U}$ can have many *instances*; we denote instance t of player U_i as Π_i^t with $t \in \mathbb{N}$. A given instance can be involved in at most one execution of P. And for each concurrent execution of P, we consider the nonempty set $\mathcal{I}$, called the *multicast group*, made of players instances involved in that execution. We emphasize that each set $\mathcal{I}$ is related to one unique execution of the protocol[2]. Finally in a multicast group $\mathcal{I}$ of size n, we denote by $\mathcal{I}_1, \ldots, \mathcal{I}_n$, the indices of players involved in this group; this allows to translate numbering of players into numbering of instances involved in a given group.

As in previous works, there is in $\mathcal{I}$ a *group controller* $\mathsf{GC}(\mathcal{I})$ who 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 our protocols,

[2] That is, if players U_1 and U_2 are running two concurrent executions of P, the first one involving instance Π_1^t of U_1 and instance $\Pi_2^{t'}$ of U_2, the second one involving instance Π_1^s of U_1 and instance $\Pi_2^{s'}$ of U_2, then there will be two multicast groups to deal with: $\mathcal{I} = \{\Pi_1^t, \Pi_2^{t'}\}$ and $\mathcal{I}' = \{\Pi_1^s, \Pi_2^{s'}\}$.

the group controller is (essentially) the player instance with the highest index in $\mathcal{U}$ (see details in section 4).

To properly deal with security issues, and for the sake of modularity, we will distinguish two kinds of module each instance is given access to. First, there is a secure co-processor (the *Key Exchange Module* — KEM) which performs (in a tamper-resistant fashion) the cryptographic computations. Second, there is an authentication device (the *Authentication Module* — AM) such as a smart card, which due to its lower computational power, is only in charge of authentication mechanism.

3.2 Abstract Interface

We define the basic structure of a dynamic group key exchange protocol. A dynamic group key exchange scheme GKE consists of four algorithms:

– The *key generation algorithm* $\mathsf{GKE}.\textsc{KGen}(1^\ell)$ 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 authentication 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* $\mathsf{GKE}.\textsc{Setup}(\mathcal{J})$, 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* $\mathsf{GKE}.\textsc{Remove}(\mathcal{I}, \mathcal{J})$ creates a new multicast group and sets it to $\mathcal{I} \backslash \mathcal{J}$.
– The *join algorithm* $\mathsf{GKE}.\textsc{Join}(\mathcal{I}, \mathcal{J})$ creates a new multicast group and sets it to $\mathcal{I} \cup \mathcal{J}$.

An execution of P consists of running the $\mathsf{GKE}.\textsc{KGen}$ 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: $\mathsf{GKE}.\textsc{Setup}$, $\mathsf{GKE}.\textsc{Remove}$ or $\mathsf{GKE}.\textsc{Join}$.

Whenever a membership operation is performed on a multicast group $\mathcal{I}$, we are going to create a new instance for each player in the resulting multicast group, say $\mathcal{J}$; in other words, the multicast group $\mathcal{I}$ continues to live (with its own session key), while the new multicast group $\mathcal{J}$ is being constructed. Players instances in $\mathcal{I}$ continue to execute their own processes (e.g., answering the queries asked by the adversary), and newly created instances run independent processes in $\mathcal{J}$. We emphasize that the multicast group creation is a monotone process: once created, a group continues to live until the end of the game. In particular, if a player joins a group $\mathcal{I}$ (therefore creating a group $\mathcal{J}$) and then leaves the group $\mathcal{J}$, the resulting multicast group is not $\mathcal{I}$, but a newly created one $\mathcal{I}'$ (even if its membership is identical to $\mathcal{I}$ from the player point of view, they are made of different instances).

3.3 Security Model

The adversary $\mathcal{A}$ is given access to the oracles and interacts with them via the queries described below. We explain the capabilities that each kind of query captures:

These oracles provide the adversary with the ability 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 it with the ability to generate concurrent changes in the membership. We also take into account hardware devices and model their interactions with the adversary via specific queries.

Queries to Players Instances 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 its query the response which Π_U^t would have generated in processing message m according to P.
- $\mathsf{Setup}(\mathcal{J})$, $\mathsf{Remove}(\mathcal{I}, \mathcal{J})$, or $\mathsf{Join}(\mathcal{I}, \mathcal{J})$: These queries model the adversary $\mathcal{A}$ initiating one of the operations $\mathsf{GKE}.\textsc{Setup}$, $\mathsf{GKE}.\textsc{Remove}$ or $\mathsf{GKE}.\textsc{Join}$. Adversary $\mathcal{A}$ gets back the flow initiating the execution of the corresponding operation.
  Note that combined with Send-queries, these 3 operation queries are enough to model both passive and active attacks. While they only send back the flow initiating the actual operation, the answer can be forwarded to the appropriate player, which answer is also forwarded, etc. This way, passive attacks can be modeled. Of course, the adversary can alter the message before forwarding it, which models active attacks.
- $\mathsf{Reveal}(\Pi_U^t)$: This query models the attacks resulting in the loss of the 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 sk_U^t (Π_U^t has set its flag accept to true). $\mathcal{A}$ gets back sk_U^t which is otherwise hidden.

Corruption Capabilities The adversary $\mathcal{A}$ can bypass the tamper detection mechanisms [72], through physical or side-channel attacks. Such capabilities are modeled via the following two queries:

- $\mathsf{Corrupt}_{\mathsf{am}}(U)$: This query models $\mathcal{A}$ corrupting the authentication module (the smart card). $\mathcal{A}$ gets back the player's LL-key.
- $\mathsf{Corrupt}_{\mathsf{kem}}(\Pi_U^t)$: This query models $\mathcal{A}$ corrupting the key exchange module (the secure co-processor). $\mathcal{A}$ gets back the private memory of the *instance*. This query is only available in the *strong-corruption model* (see below).

3.4 Security Notions

The main security requirement for a secure group key exchange method to achieve is "implicit" authentication. In Authenticated Key Exchange (**AKE** for brevity), each party is assured that no other party aside from the intended pool of players can learn any information about the session key. An additional security notion is "explicit" authentication or key confirmation often both referred to as Mutual Authentication (**MA** for brevity). MA should not be mistaken for the liveness property which provides guarantees on the delivery of messages [6, 36]. MA ensures each player that his partners (or pool thereof) have actually computed the shared session key.

In the following, we only focus on the AKE notion only, since this is the most important one. Furthermore, classical techniques are known to enhance it with MA, such as additional key confirmation rounds.

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

More formally, let us first denote by $\mathsf{SIDS}(\Pi)$ the set of all the *significant* flows sent and received by Π before acceptance (flag accept set to true). By significant flows, we mean flows with high entropy and thus specific to the execution of the actual protocol. A first round of nonces is often used to introduce high entropy and to avoid to make players, from different executions, to be partners.

In an execution of P, we say that two instances Π and Π' are **directly partnered** if both instances accept and $\mathsf{SIDS}(\Pi) \cap \mathsf{SIDS}(\Pi') \neq \emptyset$ holds. We denote the direct partnering as $\Pi \leftrightarrow \Pi'$.

We also say that instances Π and Π' are **partnered** if they both accept and if, in the graph $G_{\mathsf{SIDS}} = (V, E)$ where $V = \{\Pi_U^t \ : \ U \in \mathcal{U}, t \in \mathbb{N}\}$ and $E = \{(\Pi_U^t, \Pi_{U'}^{t'}) \ : \ \Pi_U^t \leftrightarrow \Pi_{U'}^{t'}\}$ the following holds:

$$\exists k > 1, (\Pi_1, \Pi_2, \ldots, \Pi_k) \text{ with } \begin{cases} \Pi_1 = \Pi, \\ \Pi_{i-1} \leftrightarrow \Pi_i \text{ for } i = 1, \ldots, k \\ \Pi_k = \Pi'. \end{cases}$$

We denote this partnering as $\Pi \leftrightsquigarrow \Pi'$.

We complete in polynomial time (in $|V|$) the graph G_{SIDS} to obtain the graph of partnering: $G_{\mathsf{PIDS}} = (V', E')$, where $V' = V$ and $E' = \{(\Pi_U^t, \Pi_{U'}^{t'}) \ : \ \Pi_U^t \leftrightsquigarrow \Pi_{U'}^{t'}\}$, and then define the partner identities for oracle Π as:

$$\mathsf{PIDS}(\Pi) = \{\Pi' \ : \ \Pi' \leftrightsquigarrow \Pi\}.$$

Semantic Security **The Test-query.** This query, that we denote $\mathsf{Test}(\Pi_U^t)$, models the semantic security of the session key sk_U^t. It is asked only once in the following AKE attack game, and is meaningful only if oracle Π_U^t is still Fresh at the end of the game (which informally means that the session key is not trivially known to the adversary, and it will be defined more formally below). The query is answered according to a private (*i.e.*, out of $\mathcal{A}$'s view) bit b. If $b = 0$, a random

ℓ-bit string is returned; if $b = 1$, the session key sk_U^t is returned. We use this query to define $\mathcal{A}$'s advantage.

AKE Security. The security definition for P takes place in the following game, denoted $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. The game is initialized by providing coin tosses to $\mathcal{A}$, $\mathsf{GKE}.\mathrm{KGen}(\cdot)$ and any oracle Π_U^t and by running $\mathsf{GKE}.\mathrm{KGen}(1^\ell)$ to set up players' LL-keys. A bit b is as well flipped to be later used in the Test-query. Then, the adversary starts interactions with the players instances: he can ask Send, Setup, Join, Remove, Reveal queries, as well as, depending on the considered corruption model, Corrupt-queries; in addition, $\mathcal{A}$ can ask at most one Test-query, but at any time of its choice. When $\mathcal{A}$ terminates it outputs a bit b'. We say that $\mathcal{A}$ *wins* the AKE game if $b = b'$ and the "Test-ed" instance is still Fresh (see below). Note, $\mathcal{A}$ can trivially win with probability $1/2$, and thus we define $\mathcal{A}$'s advantage by $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2 \times \Pr[b = b'] - 1$. Protocol P is an (t, ϵ)-**secure AKE** protocol if $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A})$ is lower than ϵ for all adversary $\mathcal{A}$ running in time t.

Freshness As already introduced, the *freshness* formalizes the fact that the session key is not *obviously known* by the adversary through basic means. On top of this, and because the corruption capabilities of an adversary can make him learn the session key trivially, the definition is relevant to the notion of forward-secrecy: forward-secrecy entails that the corruption of a player does not compromise the *previously* established session keys. However while a corruption may have exposed the long-term key of a player it may have also exposed the player's internal data[3] (for instance, an ephemeral, private GDH exponent). We hence define several flavors of freshness, depending on which corruptions are allowed.

1. **scenario without any Corrupt-query** We say that an oracle Π_U^t is Fresh, in the current execution, (or holds a Fresh sk) if (1) Π_U^t has accepted, and (2) neither him nor his partners has been asked for a Reveal-query.
2. **standard corruption model** Here the adversary has the ability to make $\mathsf{Corrupt}_{\mathsf{am}}$-queries only. We use this model when dealing with *(perfect) forward secrecy*, which we refer to as fs. We say that an oracle Π_U^t is fs-Fresh, in the current execution, if: (1) Π_U^t has accepted, (2) neither Π_U^t nor his partners has been asked for a Reveal-query, and (3) no $\mathsf{Corrupt}_{\mathsf{am}}$-query has been made (to U or his partners) by $\mathcal{A}$ before Π_U^t accepted (no $\mathsf{Corrupt}_{\mathsf{kem}}$-query is allowed at all).
3. **strong-corruption model** Here the adversary has the ability to make both $\mathsf{Corrupt}_{\mathsf{am}}$ and $\mathsf{Corrupt}_{\mathsf{kem}}$-queries. We use this model when considering *strong forward secrecy*, which we refer to as sfs. We say that an oracle Π_U^t is sfs-Fresh, in the current execution, if: (1) Π_U^t has accepted, (2) neither Π_U^t nor his partners have been asked for a Reveal-query, and (3) neither $\mathsf{Corrupt}_{\mathsf{am}}$-

[3] Remind that the freshness notion is relative to an instance, not to a player. And in a concurrent setting, each Join/Remove operation results in creating a new multicast group with a new session key; that later, however, is typically updated from the previous one using these internal data, and hence the corruption should distinguish whether these data are revealed or not.

query has been made (to U or his partners) by $\mathcal{A}$ before Π_U^t accepted nor a $\mathsf{Corrupt_{kem}}$-query has been made to Π_U^t by $\mathcal{A}$.

At an intuitive level, the standard corruption model is to be used when ephemeral data are protected in a tamper-resistant device: the adversary cannot see them. On the other hand, considering the strong corruption model allows to deal with scenarios in which the adversary can obtain ephemeral data, however if we want to limit the damages of such leakage of information, we need to assume that these data are securely erased once they are not useful anymore.

Remark 3. In the definition of freshness, one can note that only Reveal-queries can later change the status of freshness of a key (or an instance): any Corrupt-query does not change anything when the key is agreed on. However, if the adversary asks a Reveal-query to the instance, or any of his partners, the instance is not fresh anymore. This is the reason why it is essential for the partnering to be a public relation so that the adversary is aware of altering the freshness when asking a Reveal-query.

4 Authenticated Group Diffie-Hellman Protocol

We describe the authenticated group DH protocol, which was formerly presented in our previous paper [27][4] under the name AKE1, by splitting it into functions that help us to implement the GKE abstract interface. These functions specify in a modular way how cryptographic transformations are performed, and abstract out the details the transformations. In the following we identify the multicast group to the set of indices (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 AKE1 consists of the Setup, Remove and Join algorithms. As illustrated in Figures 4, 5 and 6, in 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 $\mathsf{GC}(\mathcal{I})$: $\mathsf{GC}(\mathcal{I}) = \Pi_{\mathcal{I}_n}^t$ for some t, if n is the size of the multicast group. When some players ask to join the group the Group Controller initiates the protocol with the joining players; when some players are leaving, the remaining player with the highest index is the new Group Controller and performs the broadcast to update the group key. Even if it is not said precisely in the description of the algorithms, each instance saves the set of values it receives in the down-flow broadcast of Setup, Remove and Join: 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 Remove (that is, to generate a new broadcast from the saved one).

[4] The initial formal model in [29, 26] enabled us to propose a first protocol named AKE1. This was later refined in [27].

The session-key space **SK** associated with the protocol AKE1 is $\{0,1\}^\ell$ equipped with a uniform distribution. The arithmetic is in a group $\mathbb{G} = \langle g \rangle$ of prime order q in which the **DDH** assumption holds.

4.2 Authentication Module

The *Authentication Mechanism* Auth supports the following functions:

- $\mathsf{Auth}.\textsc{KGen}(1^k, i, j)$. This function, from the given security parameter 1^k, generates a pair of keys, which is either a pair of matching public/secret keys $(\mathrm{PK}_i, \mathrm{SK}_i)$ for player U_i, or a symmetric key $K_{ij} = \mathrm{PK}_{ij} = \mathrm{SK}_{ij}$ between a sender U_i and a receiver U_j. The secret keys are never exposed.
- $\mathsf{Auth}.\textsc{Sign}(i, j, m)$. This function authenticates a message m between a sender U_i and a receiver U_j, by using the authentication key SK_{ij}. It returns an authenticated data that is denoted $\mu = [m]_{ij}$.
- $\mathsf{Auth}.\textsc{Ver}(i, j, m, \mu)$. This function checks whether μ is an authenticator on message m from a sender U_i to a receiver U_j with respect to the verification key PK_{ij}. The boolean answer is returned.

The two latter functions should of course be called after initializing the keys via $\mathsf{Auth}.\textsc{KGen}(\cdot)$. Then we define the notion of signing oracle. An $\mathsf{Auth}.\textsc{Sign}$-oracle for messages authentication is an oracle that takes as input two indices i and j and a message m, and returns an authenticator data $\mu = [m]_{ij}$ using the authentication key generated by $\mathsf{Auth}.\textsc{KGen}(1^k, i, j)$.

Definition 4 (– Chosen Message, Existential Unforgeability). A (t, q, ϵ)-Auth-forger $\mathcal{F}$ is a probabilistic Turing machine running within time t that requests an $\mathsf{Auth}.\textsc{Sign}$-oracle up to q messages (and for any pair of indices), and outputs (m, μ, i, j) where m is a message authenticated by $\mu = [m]_{i,j}$, and without having queried the $\mathsf{Auth}.\textsc{Sign}$-oracle on message m, with the corresponding entities (i, j), with probability at least ϵ. We denote this success probability as $\mathsf{Succ}^{\mathsf{cma}}_{\mathsf{auth}}(t, q)$, where CMA stands for (adaptive) Chosen-Message Attack. The Auth scheme is (t, q, ϵ)-**CMA-secure** if there is no (t, q, ϵ)-Auth-forger.

Any appropriate signature scheme $\textsc{Sign}$ or message authentication code MAC can be used.

4.3 Key Derivation

Informally, a *Key Derivation Function* (KDF for short) is defined as follows:

- A function KDF, that given a string x sampled from an arbitrary distribution, together with a uniformly distributed randomizer, outputs a string of a fixed length.

Clearly, in the random oracle model [12], a hash function is a perfect key derivation function, however it does not provide the same level of security as a proof in the standard model [33]. In the standard model, KDF has to be implemented with more sophisticated tools, such as the left-over-hash lemma [44] with authenticated randomness, or a deterministic randomness extractor, to obtain (almost) uniformly distributed values over $\{0,1\}^\ell$.

4.4 Key-Exchange Module

The *Key-Exchange Mechanism* supports the following functions. They are essentially performed in the secure co-processor, out of which the ephemeral Diffie-Hellman exponent should not leak; most of them, however, invoke the Authentication Mechanism functions, which means that communication between these two devices are assumed. The content of these communications is subject to attacks when considering the strong corruption model.

The following functions help to build the trigon of successive flows that will be sent in the protocol, as shown in Figure 3. One may use these functions to pick a private exponent (GDH_PICKS$(\cdot)$ and GDH_PICKS$^\star(\cdot)$), to go through the lines of the trigon (GDH_UP$(\cdot)$), to return the values needed to compute the key (GDH_DOWN$(\cdot)$), to restart going through the lines (GDH_UP_AGAIN$(\cdot)$), to return needed values again (GDH_DOWN_AGAIN$(\cdot)$), and to compute the key itself (GDH_KEY$(\cdot)$).

- GDH_PICKS(i). This function generates a new private exponent $x_i \stackrel{R}{\leftarrow} \mathbb{Z}_q^\star$. It also erases any previous exponent x_i'. However, note that x_i is never exposed.
- GDH_PICKS$^\star(i)$. This function invokes GDH_PICKS(i) to generate x_i but does not delete the previous private exponent x_i'. The latter exponent x_i' is only deleted when explicitly asked for by the instance.
- GDH_UP$(i, j, k, \mathsf{Fl}, \mu)$. This function forwards the successive values in the group by performing the following steps.
  1. if $j > 0$, the authenticity of tag μ on message Fl is checked with $\mathsf{Auth}.\textsc{Ver}(j, i, \mathsf{Fl}, \mu)$; if the verification fails, the protocol stops.
  2. Fl is parsed as a set of intermediate values $(\mathcal{I}, Y, Z)$ where $\mathcal{I}$ is the multicast group and
  $$Y = \bigcup_{k=1,\ldots,i-1} \{Z^{1/x_k}\} \text{ with } Z = g^{\prod_{k=1}^{i-1} x_k}.$$
  Then the values in Y are raised to the power of x_i and then concatenated with Z to obtain these intermediate values
  $$Y' = \bigcup_{k=1,\ldots,i-1} \{Z^{x_i/x_k}\} \cup \{Z\} = \bigcup_{k=1,\ldots,i} \{Z^{x_i/x_k}\} = \bigcup_{k=1,\ldots,i} \{Z'^{1/x_k}\},$$
  $$\text{where } Z' = Z^{x_i} = g^{\prod_{k=1}^{i} x_k}.$$
  3. $\mathsf{Fl}' = (\mathcal{I}, Y', Z')$ is authenticated, by invoking $\mathsf{Auth}.\textsc{Sign}(i, k, \mathsf{Fl}')$ to obtain tag μ'. The flow (Fl', μ') is returned.
- GDH_DOWN(i, j, Fl, μ). This function prepares the set of values to be broadcasted by performing the following steps.
  1. the authenticity of (Fl, μ) is checked, by invoking $\mathsf{Auth}.\textsc{Ver}(j, i, \mathsf{Fl}, \mu)$; if the verification fails, the protocol stops.
  2. the flow Fl' is computed as in GDH_UP, from $\mathsf{Fl} = (\mathcal{I}, Y, Z)$ but without the last element Z' (*i.e.*, $\mathsf{Fl}' = (\mathcal{I}, Y')$).
  3. the flow Fl' is appended tags $\mu_1, \ldots, \mu_n$ by invoking $\mathsf{Auth}.\textsc{Sign}(i, k, \mathsf{Fl}')$, where k ranges in $\mathcal{I}$. The tuple $(\mathsf{Fl}', \mu_1, \ldots, \mu_n)$ is returned.

$j = 1$				g	g^{x_1}	GDH_UP
$j = 2$			g^{x_2}	g^{x_1}	$g^{x_1 x_2}$	
$j = 3$ $(= n - 1)$		$g^{x_2 x_3}$	$g^{x_1 x_3}$	$g^{x_1 x_2}$	$g^{x_1 x_2 x_3}$	
$j = 4$ $(= n)$	$g^{x_2 x_3 x_4}$	$g^{x_1 x_3 x_4}$	$g^{x_1 x_2 x_4}$	$g^{x_1 x_2 x_3}$	$g^{x_1 x_2 x_3 x_4}$	GDH_DOWN
	$Y \Longrightarrow Y'$				$Z \Rightarrow Z'$	

Fig. 3. Successive flows, when $n = 4$

- GDH_UP_AGAIN$(i, k, \mathsf{Fl} = (\mathcal{I}, Y'))$. This function restarts the process by refreshing the i-th line as follows. From Y' and the previous random x_i', one can recover the associated Z' (by raising the last component of Y' to the power of x_i'). 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 a tag μ' via $\mathsf{Auth}.\text{SIGN}(i, k, \mathsf{Fl}')$. The pair (Fl', μ') is returned. From now the old random x_i' is no longer needed and, thus, can be *erased*.
- GDH_DOWN_AGAIN$(i, \mathsf{Fl} = (\mathcal{I}, Y'))$. This function refreshes the set of values to be broadcasted as follows. 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 $\mathsf{Auth}.\text{SIGN}(i, k, \mathsf{Fl}')$, 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*.
- GDH_KEY(i, j, Fl, μ) produces the session key sk. First, the authenticity of the flow (Fl, μ) is checked with $\mathsf{Auth}.\text{VER}(j, i, \mathsf{Fl}, \mu)$. 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 $\mathsf{KDF}(\mathcal{I} \| \mathsf{Fl} \| \alpha)$.

Intuitively, the basic protocol runs as follows. Each successive player will use GDH_PICKS to get its own private exponent, GDH_UP to embed it in the received values and to forward the new values to the next player; this process starts from an empty set of values. The last player then will use GDH_DOWN to broadcast the sufficient information such that each player can compute the key, using GDH_KEY. When one (or more) player(s) want(s) to join the group, the last player in the current group refreshes its private exponent with GDH_PICKS* and restarts sending successive values via GDH_UP_AGAIN. The joining players will use GDH_UP to embed their contribution until the last joining player. The latter will broadcast, as previously, a set of values using GDH_DOWN. When one (or more) player(s) want(s) to leave the group, the highest-index remaining player refreshes its private exponent with GDH_PICKS* and will use GDH_DOWN_AGAIN to generate a new broadcast such that the other remaining players can compute the key (via GDH_KEY) and then set the accept flag to true. A more formal description is given below.

4.5 Setup($\mathcal{I}$)

This algorithm consists of two stages: the up-flow and the down-flow (see Figures 3 and 4). Remind that $\mathcal{I}_i$ denotes the index (in $\mathcal{U}$) of the i-th oracle instance involved in $\mathcal{I}$. Let n be the number of instances in $\mathcal{I}$.

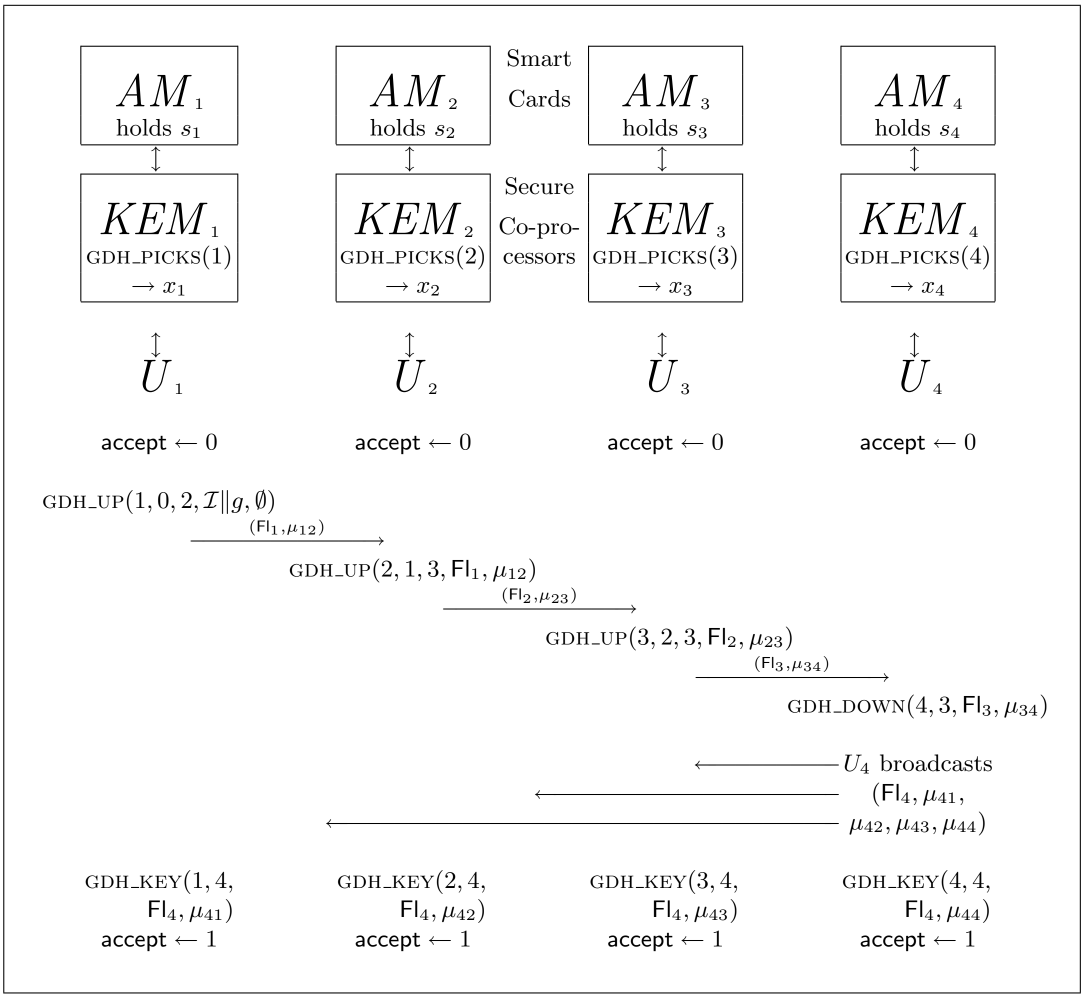

Fig. 4. Algorithm Setup. A practical example with 4 players $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$.

One starts with the convention $\mathcal{I}_0 = 0$, $\mathsf{Fl}_0 = (\mathcal{I}, \{g\})$ and $\mu_{0,i} = \emptyset$. Then, on the up-flow, each oracle $\Pi^t_{\mathcal{I}_i}$ for $i = 1, \ldots, n$ invokes GDH_PICKS$(\mathcal{I}_i)$ to generate its private exponent $x_{\mathcal{I}_i}$ and then (only if $i \leq n-1$) invokes 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 $\mu_{i,i+1}$. Then, $\Pi^t_{\mathcal{I}_i}$ 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 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 $\Pi^t_{\mathcal{I}_i}$ invokes GDH_KEY$(\mathcal{I}_i, \mathcal{I}_n, \mathsf{Fl}_n, \mu_i)$ and gets back the session key $\mathsf{sk}^t_{\mathcal{I}_i}$ (and accepts the session).

To illustrate this, assume U_2, U_4 and U_6 do run the algorithm. We have then $\mathcal{I}_1 = 2, \mathcal{I}_2 = 4, \mathcal{I}_3 = 6$; we slightly abuse the notation and denote for all of them the session by t so that $\mathcal{I} = \{\Pi^t_2, \Pi^t_4, \Pi^t_6\}$. The protocol starts by having

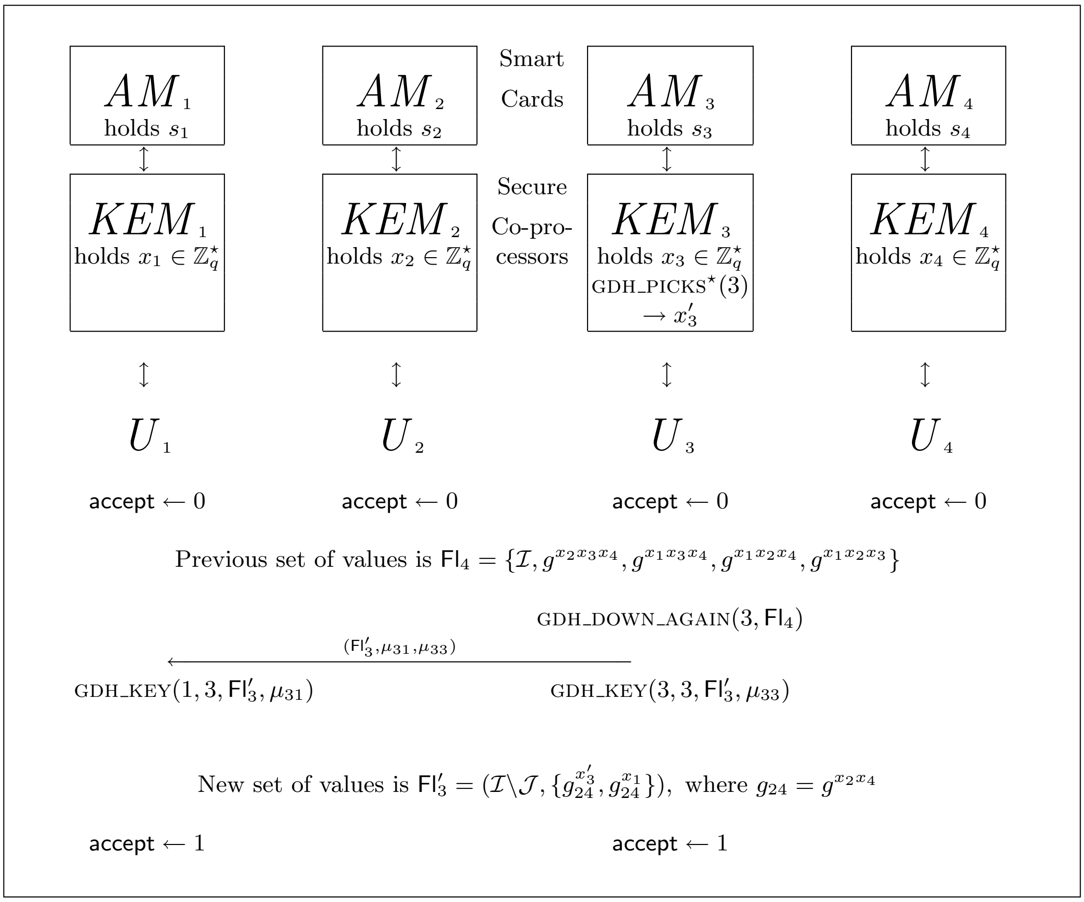

Fig. 5. Algorithm Remove. A practical example with 4 players: $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$ and $\mathcal{J} = \{U_2, U_4\}$. The new multicast group is $\mathcal{I} = \{U_1, U_3\}$ and $\mathsf{GC} = U_3$.

Π_2^t choosing x_2 and, from $(2, 0, 4, (\mathcal{I}, \{g\}), \emptyset)$, generates $\mathsf{Fl}_1 = (\mathcal{I}, \{g, g^{x_2}\})$ together with an authenticator μ_{12}. The second player instance Π_4^t chooses x_4 and generates from $(4, 2, 6, \mathsf{Fl}_1, \mu_{12})$ the values $\mathsf{Fl}_2 = (\mathcal{I}, \{g^{x_4}, g^{x_2}, g^{x_2x_4}\})$ and μ_{23}. Finally Π_6^t generates the broadcast via GDH_DOWN$(6, 4, \mathsf{Fl}_2, \mu_{23}) = \mathsf{Fl}_3, \mu_{31}, \mu_{32}, \mu_{33}$ where $\mathsf{Fl}_3 = (\mathcal{I}, \{g^{x_4x_6}, g^{x_2x_6}, g^{x_2x_4}\})$. The instance Π_2^t and Π_4^t compute the session key as GDH_KEY$(2, 6, g^{x_4x_6}, \mu_{31})$ and GDH_KEY$(4, 6, g^{x_2x_6}, \mu_{32})$, respectively[5]. Here the session key is computed from the common secret $g^{x_2x_4x_6}$.

4.6 Remove($\mathcal{I}, \mathcal{J}$)

This algorithm consists of a down-flow only (see Figure 5). Let n be the size of $\mathcal{I}$ and m be the size of $\mathcal{I} \backslash \mathcal{J}$. The group controller $\mathsf{GC}(\mathcal{I}')$ of the new set $\mathcal{I}' = \mathcal{I} \backslash \mathcal{J}$ invokes GDH_PICKS$^\star(\mathcal{I}_m')$ to get a *new* private exponent and then GDH_DOWN_AGAIN$(\mathcal{I}_m', \mathsf{Fl}')$ where Fl' is the saved previous broadcast; the function makes use of both exponents (the newly generated one and the old one) but erases the old one at the end. $\mathsf{GC}(\mathcal{I}')$ obtains a new set of intermediate values from which it simply deletes the elements related to the removed players (in

[5] To be correct, the function GDH_KEY$(\cdot)$ takes as a third input the entire set of values Fl, but here we wrote only the value that the player is going to use, to make the mechanism clearer.

the set $\mathcal{J}$) and updates the multicast group to be $\mathcal{I}'$. This produces the new broadcast flow Fl_m with some tags $\mu_1, \ldots, \mu_m$. Upon receiving the down-flow, $\Pi^t_{\mathcal{I}'_i}$ invokes GDH_KEY$(\mathcal{I}'_i, \mathcal{I}'_m, \mathsf{Fl}_m, \mu_i)$ and gets back the session key $\mathsf{sk}^t_{\mathcal{I}'_i}$ (and accepts the session). Here, is the reason why an oracle must store its private exponent and only erase its internal data when it leaves the group.

To illustrate this, assume U_6 wishes to leave the group built in the previous example. The new multicast group is now $\mathcal{I}' = \{\Pi^t_2, \Pi^t_4\}$. The group controller for $\mathcal{I}'$ is Π^t_4. It first chooses a new exponent x'_4 without erasing the previous x_4. From the saved broadcast $\mathsf{Fl}' = (\mathcal{I}, \{g^{x_4x_6}, g^{x_2x_6}, g^{x_2x_4}\})$, and using GDH_DOWN_AGAIN$(4, \mathsf{Fl}')$ it generates a "full" new broadcast $\{g^{x'_4x_6}, g^{x_2x_6}, g^{x_2x'_4}\}$ from which it deletes the term to be used by Π^t_6 (the leaving member). The new broadcasted values are thus $\mathsf{Fl}_2 = (\mathcal{I}', \{g^{x'_4x_6}, g^{x_2x_6}\})$, together with some authenticators μ_1, μ_2. The other player Π^t_2 can recover the common secret $g^{x_2x'_4x_6}$ with its old exponent x_2: that is, it does not have to pick a new exponent. Also note that the leaving player "left" its own exponent in the common secret, but cannot use it to get the session key.

4.7 Join$(\mathcal{I}, \mathcal{J})$

This algorithm also consists of the two stages: up-flow and down-flow (see Figure 6). Let n be the size of $\mathcal{I}$ and m be the size of $\mathcal{I} \cup \mathcal{J}$. On the up-flow, the group controller $\mathsf{GC}(\mathcal{I})$ of the old group invokes GDH_PICKS$^\star(\mathcal{I}_n)$, and then GDH_UP_AGAIN$(\mathcal{I}_n, j, \mathsf{Fl}')$ where Fl' and $j = \Pi^t_{\mathcal{J}_1}$ are respectively the saved previous broadcast and the index of the first joining player. It updates $\mathcal{I}$ into $\mathcal{I}'$, and forwards the result to the first joining player. From that point in the execution, the protocol works as the algorithm Setup, where the (temporary) group controller of the new group $\mathcal{I}' = \mathcal{I} \cup \mathcal{J}$ is the highest index player in $\mathcal{J}$, that is $\Pi^t_{\mathcal{J}_{|\mathcal{J}|}}$: the joining players will use GDH_UP until the group controller; the latter will use GDH_DOWN to perform the broadcast.

Again, to illustrate this, assume U_1 and U_3 wish to join the group built in the previous example. The new multicast group is now $\mathcal{I}' = \{\Pi^t_1, \Pi^t_2, \Pi^t_3, \Pi^t_4\}$. The group controller for $\mathcal{I}$ was Π^t_4. So it first chooses a new exponent x''_4 without erasing the previous x'_4. From the saved broadcast $\mathsf{Fl}'' = (\mathcal{I}, \{g^{x'_4x_6}, g^{x_2x_6}\})$, and using GDH_UP_AGAIN$(4, 1, \mathsf{Fl}'')$ it generates a "fresh" up-flow $\mathsf{Fl}_2 = (\mathcal{I}', \{g^{x''_4x_6}, g^{x_2x_6}, g^{x_2x''_4x_6}\})$ together with μ_{23}, and forwards these values to Π^t_1. Then Π^t_1 picks x_1 and, using GDH_UP$(1, 4, 3, \mathsf{Fl}_2, \mu_{23})$, produces $\mathsf{Fl}_3 = (\mathcal{I}', \{g^{x_1x''_4x_6}, g^{x_1x_2x_6}, g^{x_2x''_4x_6}, g^{x_1x_2x''_4x_6}\})$ and μ_{34}. The latter picks x_3 and, using GDH_DOWN$(3, 1, \mathsf{Fl}_3, \mu_{34})$, generates the broadcasted values $\{g^{x_1x_3x''_4x_6}, g^{x_1x_2x_3x_6}, g^{x_2x_3x''_4x_6}, g^{x_1x_2x''_4x_6}\}$.

5 Analysis of Security

In this section, we assert that the protocol AKE1 securely establishes 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 considering the standard corruption mode the protocol AKE1 is secure under standard assumptions. This proof can in turn be adapted to cope with the strong-corruption mode.

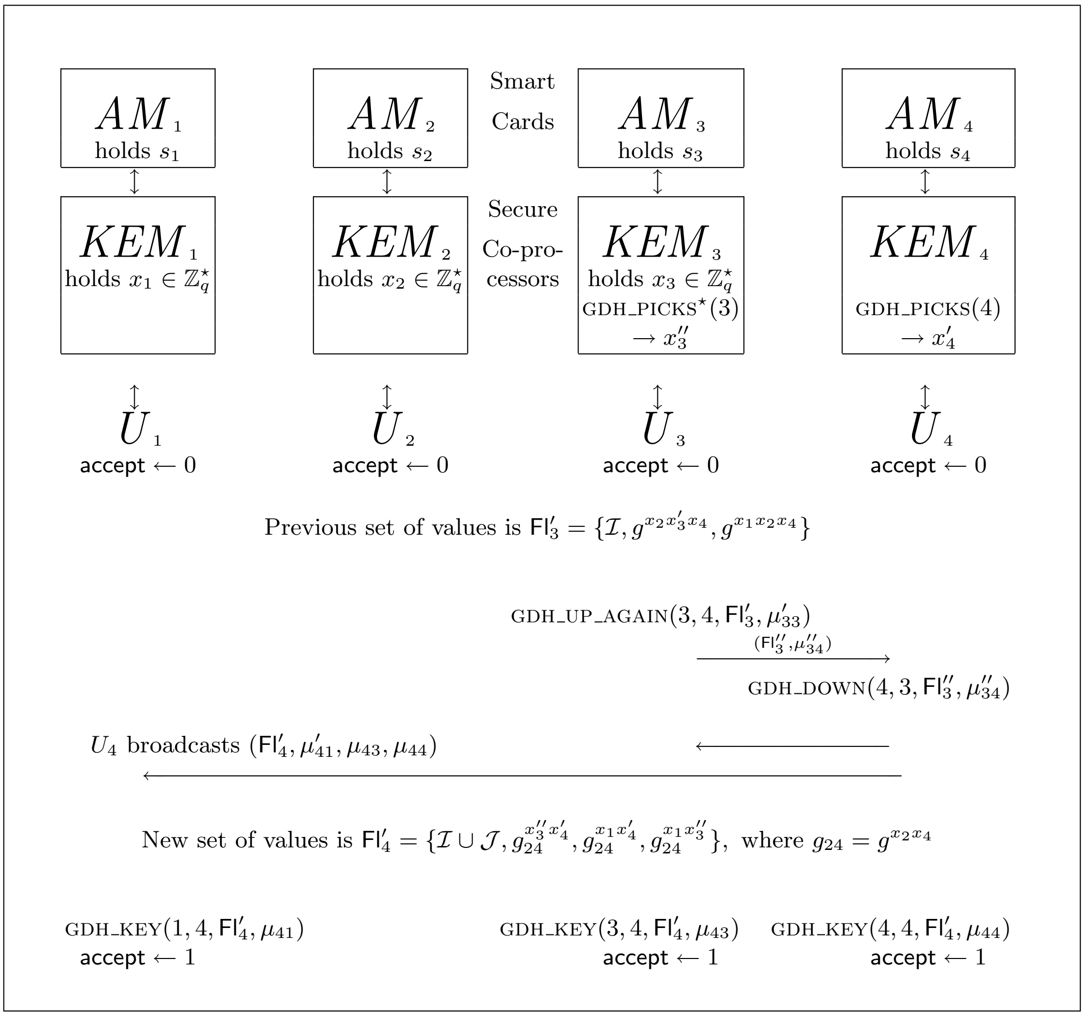

Fig. 6. Algorithm Join. A practical example with 4 players: $\mathcal{I} = \{U_1, U_3\}$, $\mathcal{J} = \{U_4\}$ and $\mathsf{GC} = U_3$. The new multicast group is $\mathcal{I} = \{U_1, U_3, U_4\}$.

5.1 Security Results

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

Theorem 5 (– AKE Security in the Standard Corruption Model). *Let $\mathcal{A}$ be an adversary against protocol P, running in time T, allowed to make at most Q queries (Setup, Join, Remove, Send, $\mathsf{Corrupt}_{\mathsf{am}}$). The adversary is also restricted to not ask $\mathsf{Corrupt}_{\mathsf{kem}}$-queries. Let n be the number of players (among the N total number of players) involved in the operations which lead to the group on which $\mathcal{A}$ makes the Test-query. Then we have:*

$$\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) \leq 2nQ \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_\Gamma}(T') + 2N(N-1) \cdot \mathsf{Succ}_{\mathsf{auth}}^{\mathsf{cma}}(T, Q) + 2nQ \cdot \delta$$

where δ denotes the distance between the output of $\mathsf{KDF}(\cdot)$ 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 $\Gamma = \mathcal{E}_n$ corresponds to the elements adversary $\mathcal{A}$ can possibly learn (the extended trigon, see Figure 2):

$$\mathcal{E}_n = \bigcup_{1\leq j\leq n-2} \bigcup_{1\leq k\leq j} \{\, \{i \,|\, 1 \leq i \leq j, i \neq k\} \,\} \\ \cup \bigcup_{1\leq k<l\leq n} \{\, \{i \,|\, 1 \leq i \leq n, i \neq k, l\} \,\} \cup \bigcup_{1\leq k\leq n} \{\, \{i \,|\, 1 \leq i \leq n, i \neq k\} \,\}$$

Note 6. If the authentication means is a signature scheme (the verification is independent of the recipient), the security result becomes

$$\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) \leq 2nQ \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_\Gamma}(T') + 2N \cdot \mathsf{Succ}_{\mathsf{sign}}^{\mathsf{cma}}(T,Q) + 2nQ \cdot \delta.$$

Furthermore, we assume we know n, the number of players involved in the operations which lead to the group on which $\mathcal{A}$ makes the *Test*-query. It is indeed not a uniform reduction, but with an additional linear factor N, the reduction can be made uniform, first guessing n.

When we talk about the “players involved in the operations which lead to the group on which $\mathcal{A}$ makes the Test-query”, we mean a player an instance of whom has joined the group at least once since its setup. As an illustrative example, assume a multicast group is initialized by $\mathsf{Setup}(\Pi_2^t, \Pi_4^t, \Pi_6^t)$, then Π_6^t leaves, then Π_1^s, Π_3^s join, and the Test-query is asked to any of them $\Pi_1^s, \Pi_2^t, \Pi_3^s, \Pi_4^t$. Then the number of players that have been involved so far is $n = 5$, even if the size of the Test-ed group is 4. Note that we have $n \leq N$: a player which joins, leaves, then joins again is counted only once (though different instances of the player have to be considered).

5.2 Proof of the Main Theorem

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. Let b and b' be defined as in Section 3, namely the bit underlying the answer to the Test-query and the bit output by the adversary, respectively. We denote the event $b = b'$ by Guess.

Proof overview Intuitively, and in order to reduce the security to the **G-DDH**-problem, the flows sent by the players' instances in the crucial session (the one in which the session key will be Test-ed) will be somehow “replaced” by the lines of the extended trigon $\mathcal{D}$ (so that distinguishing the key from a random value corresponds to solving the **G-DDH**-problem). This replacement is viewed as successive modifications to the original game: 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[6]; bounding the probability that the "bad" event happens helps in relating successive games. In our proof the difficulty consists in replacing the flows with simulated values without changing the adversary's view "too much".

In our model, the adversary's capabilities are viewed as queries. These queries are answered by a simulator $\varDelta$. First of all, one may notice that Setup, Join and Remove-queries are essentially similar to Send-queries: in every case, an oracle instance is activated and must generate an outgoing message to either start a protocol execution or to continue it. Then the role of $\varDelta$, on receiving such kind of query, is to simulate the correct outgoing flow. The $\mathsf{Corrupt}$-query is also straightforward to simulate by having $\varDelta$ choosing all authentication keys by itself. The Reveal-query is the really problematic one, as soon as flows are simulated using values for which the discrete logarithm is not known (from the **G-DDH** instance).

Also, in order to answer the queries, the simulator $\varDelta$ will make use of several auxiliary inputs: in particular it will use two integers c_0 and i_0 (that will be introduced in game $\mathbf{G}_2$) as well as an instance $\mathcal{D}$ of size n of the **G-DDH** problem: $\mathcal{D}$ is drawn according to the distribution $\mathsf{GDH}^{\star}_{\varGamma}$ (this auxiliary data will be introduced in game $\mathbf{G}_3$), or $\mathsf{GDH}^{\$}_{\varGamma}$ (when we move to the game $\mathbf{G}_4$), where $\varGamma = \mathcal{E}_n$. The integers c_0 and i_0 will help $\varDelta$ to embed $\mathcal{D}$'s lines at the right place and at the right moment. For simplicity, we informally present the auxiliary inputs here, but they will be formally defined only in the games in which they are necessary. Before those games, $\varDelta$ simply ignores them.

Detailed proof We now describe each successive game. The core of the proof is in game $\mathbf{G}_3$, in which $\varDelta$ actually uses instance $\mathcal{D}$ to replace the real flows with the simulated ones.

Game $\mathbf{G_0}$ This game $\mathbf{G}_0$ is the real attack $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$, where $\varDelta$ simulates all the players and then all the queries knowing the authentication keys, and choosing the random coins. At the beginning of this game we set the bit b at random. By definition, we have:

$$\mathsf{Adv}^{\mathsf{ake}}_{P}(\mathcal{A}) = 2\Pr[\mathsf{Guess}_0] - 1 \qquad (1)$$

Game $\mathbf{G_1}$ The game $\mathbf{G}_1$ is identical to $\mathbf{G}_0$ except that we abort if a forgery for the authentication mechanism is detected before any $\mathsf{Corrupt}$-query: this happens when a valid tag appears in a flow (say, the adversary asks a $\mathsf{Send}(\varPi, (\mathsf{Fl}, \mu))$ query, with $\mathsf{Auth}.\textsc{Ver}(\mathsf{Fl}, \mu) = 1$), while the pair (Fl, μ) has not been produced by any instance (*i.e.*, was not generated by $\varDelta$ itself in answer to a previous query). We define the forgery event Forge. Using a well-know lemma we get:

$$|\Pr[\mathsf{Guess}_0] - \Pr[\mathsf{Guess}_1]| \leq \Pr[\mathsf{Forge}]. \qquad (2)$$

[6] This technique has been formalized by Shoup [65]. The point is in choosing the "bad" event.

Lemma 7 (– Probability of Event Forge).

$$\Pr[\mathsf{Forge}] \leq N(N-1) \times \mathsf{Succ}^{\mathsf{cma}}_{\mathsf{auth}}(T). \tag{3}$$

Proof. The proof uses a standard hybrid argument.

In this protocol, all the flows are authenticated by the sender. When the forgeries are excluded, active attacks are excluded too: only replay attacks are still possible. Of course, the adversary can also delay or reorder messages, then Δ only handle them if they are still meaningful: since they are signed, it knows which exponents they contain.

Game $\mathbf{G}_2$ Game $\mathbf{G}_2$ is the same as game $\mathbf{G}_1$ except that we make the simulation abort if certain conditions are not satisfied. Once the simulation does not abort (this implying a loss in the probabilities), we are thus ensured that the conditions are indeed satisfied in the next games. The reason why we abort might appear unclear in this game (indeed, there are purely formal here), but will become clear in the future games.

More precisely, we make use of the simulator's auxiliary input: a random index $i_0 \in [1, n]$ and a random integer $c_0 \in [1, Q]$. The value c_0 is a guess for the number of operations that will occur before the Test-ed session is built (remind that Q is an upper-bound for the total number of queries), while i_0 is a guess for the player instance who will send the broadcast flow of the "Test-ed" session. More precisely, it is its order in the list of the involved players in the series of operations that lead to the Test-ed group. Intuitively, c_0 and i_0 are thought to be as follows: if the c_0-th operation[7] is Join or Setup, then the simulator hopes that the i_0-th player involved will be the last joining player, otherwise the simulator hopes it will be the group controller's index. In the execution of the game, if the Test-ed session is not the one completed with the c_0-th operation, or if the corresponding broadcast flow is not operated by the i_0-th player, the simulator outputs "`Fail`" and sets b' randomly. Let E_2 be the event that these guesses are not correct. It can be noticed that the value c_0 and i_0 are chosen uniformly and at random in $[1, Q]$ and $[1, N]$ respectively. The probability of E_2 is thus $1 - 1/NQ$. Using the fact that E_2 and Guess_1 are independent, we have:

$$\begin{aligned}
\Pr[\mathsf{Guess}_2] &= \Pr[\mathsf{Guess}_2 \mid \mathsf{E}_2] \Pr[\mathsf{E}_2] + \Pr[\mathsf{Guess}_2 \mid \neg\mathsf{E}_2] \Pr[\neg\mathsf{E}_2] \\
&= \frac{1}{2} \Pr[\mathsf{E}_2] + \Pr[\mathsf{Guess}_1] \left(1 - \Pr[\mathsf{E}_2]\right) \\
\Pr[\mathsf{E}_2] &= 1 - \frac{1}{NQ}
\end{aligned}$$

Therefore,

$$\Pr[\mathsf{Guess}_2] - \frac{1}{2} = \frac{1}{NQ} \left(\Pr[\mathsf{Guess}_1] - \frac{1}{2} \right). \tag{4}$$

[7] By the "k-th operation", one means the k-th operation (Setup, Join or Remove) that has been initiated by $\mathcal{A}$ for building the Test-ed group. In a concurrent setting, the original groups continue to live whenever a membership change initiates a new group, and therefore a tree structure appears where nodes are the groups: a new operation creates a child. The value k is thus the depth of the Test-ed group.

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; for this reason, we will use the fact that the guesses in game $\mathbf{G}_2$ were correct: the Test-ed session is built by the c_0-th operation and the corresponding broadcast has been sent by the i_0-th player. Based on this information, Δ can correctly make use of the instance $\mathcal{D}$ to build the flows and answer the queries. Recall that the simulator Δ gets as an auxiliary input an instance $\mathcal{D}$ of size n from $\mathsf{GDH}^{\star}_{\Gamma}$, where Γ is the extended trigon $\mathcal{E}_n$.

Formally, the instance $\mathcal{D}$ plus its solution can be rewritten using the "lines", as follows.

$$\begin{aligned}\mathcal{D} &= \mathcal{V}iew^{*}_{\mathcal{E}_n}(x_1, x_2, \ldots, x_n)\\ &= \{S_1, S_2(x_1, x_2), \ldots, S_{n-2}(x_1, \ldots, x_{n-2}), S_{n-1}(x_1, \ldots, x_{n-2}, x_{n-1}),\\ &\qquad S_n(x_1, x_2, \ldots, x_{n-1}, x_n)\} \cup \{g^{x_1 \ldots x_n}\}\end{aligned}$$

wherein:

- $S_1 = \{g\}$;
- for $2 \leq j \leq n-2$ and $j = n$, $S_j(x_1, x_2, \ldots, x_{j-1}, x_j)$ is the set of all the $g^{\prod_k x_k}$, where k respectively enumerates the $j-1$-tuples one can build from $\{1, \ldots, j\}$;
- but $S_{n-1}(x_1, x_2, \ldots, x_{n-2}, x_{n-1})$ is the set of all the $g^{\prod_k x_k}$, where k respectively enumerates the $n-2$ tuples one can build from $\{1, \ldots, n\}$ (and not only from $\{1, \ldots, n-1\}$, as above, hence the extension).

Main ideas of this game. We now show how, based on the two values i_0 and c_0, the simulator is able to simulate the game many randomized instances, generated by (multiplicative) random self-reduction, from $\mathsf{GDH}^{\star}_{\Gamma}$ such that the Test-ed key is (a known power of) the GDH secret value $g^{x_1 \ldots x_n}$ relative to $\mathcal{D}$. That is all the elements of S_n (except the ones which correspond to removed players) will have been embedded into the protocol during the c_0-th operation, which leads to the Test-ed group.

The basic principle is that, whenever a Setup operation (for the Join and Remove operations, the technique follows similarly) is initiated on a group $\mathcal{I}$, Δ uses line S_1 for the first up-flow (which is always the same), $S_2(x_1, x_2)$ for the second one, etc. If the cardinality of $\mathcal{I}$ is greater than $n-1$, subsequent players instances will be simulated using exponents y_i chosen by Δ itself, so that it can still compute the further session keys, from $S_{n-1}(x_1, x_2, \ldots, x_{n-2}, x_{n-1})$ and the known y_i. While doing so, Δ maintains a list $\mathcal{L}$ (history) of involved players in $\mathcal{I}$ as well as the associations between the first n players in $\mathcal{I}$ and the indices of embedded exponent x_i, and between additional players and known exponents y_i. Indeed, for all the known and unknown exponents, the simulation must remain consistent, and always use the same exponent for an instance. More precisely, for each group constructed, or under construction (when an operation has been initiated), a sub-list of triples is maintained: the triples are of the form (i, j, x), where i is the index of the player (at most one triple exists for each player in a sub-list), j is the index of the embedded exponent x_j or $\perp$ if no exponent is embedded. In the former case, x is the randomization of x_j (see below), and in the latter case, x is the known exponent y_i.

Random self-reducibility. In order to handle concurrent executions of the protocol, Δ makes use of the (multiplicative) random self-reducibility of the GDH problem: any new instance with index i in the list $\mathcal{L}$ use a new randomized exponent $x'_i = r_i x_i$. To that goal, Δ stores in list $\mathcal{L}$, as many sublists as there are existing groups; and in each of these sublists, it stores up to n of these random "blinding" exponents r_i that keep trace of how the random self-reducibility was applied to the input instance $\mathcal{D} = \mathcal{V}iew_{\mathcal{E}_n}(x_1, x_2, \ldots, x_n)$ to get the new one $\mathcal{D}' = \mathcal{V}iew_{\mathcal{E}_n}(r_1x_1, r_2x_2, \ldots, r_nx_n)$, specific to the current group. More precisely, each time (an instance of) a player is assumed to pick at random a private exponent, Δ proceeds as follows. If the player's instance is associated (through list $\mathcal{L}$) to an unknown exponent x_i, a random r_i is chosen in $\mathbb{Z}_q^\star$ and stored, meaning that random self-reducibility is to be applied to $\mathcal{D}$ by (formally) replacing x_i with $r_i x_i$; if the player is not associated with any GDH exponent, a random, fresh exponent y_i is chosen in $\mathbb{Z}_q^\star$ and stored in $\mathcal{L}$. Of course, when a player instance is requested (by the protocol) to re-use a previous private exponent, Δ does not pick anything, but use $\mathcal{L}$ to perform the adequate computation, using the adequate elements from instance $\mathcal{D}$.

Since Δ knows the authentication keys, and with the specific form of the extended trigon, Δ can easily simulate answers to all the queries: a new exponent is either a new randomized exponent $x'_i = r_i x_i$ for an unknown x_i from $\mathcal{D}$ or a chosen y_i, the flows can be generated from the lines S_i and the random values $r_1, \ldots, r_n$ stored in $\mathcal{L}$. But some subtleties have to be detailed:

First difficulty. If, Δ embeds all the elements of S_n into the protocol execution the first time the size of the multicast group is n, Δ is not able to compute the session key value sk needed to answer to the Reveal-queries that can occur before c_0. More exactly, Δ would have then to use the value $g^{x_1 \cdots x_n}$, but we want to avoid this before c_0, in order to reduce the security to distinguishing this value from random.

Second difficulty. Δ needs to know in advance which player instance will send the last broadcast, in order not to embed the value $g^{x_1 \cdots x_n}$ prematurely in the *flows* themselves; this value must be embedded in the Test-ed session *key* only. Without caution, in particular if the i_0-th player is involved in the group at some time but then leave the group, we do not want that this temporary membership leads to embedding an exponent of instance $\mathcal{D}$; this player must be simulated using instance $\mathcal{D}$ at the c_0-th operation only. Otherwise there may be n unknown exponents x_1 through x_n embedded in the view and the secret value may be exposed in one of the flows.

Third difficulty. Assuming we manage to embed the GDH instance $\mathcal{D}$ exactly on time, when the Test-query is asked, and not before. One difficulty remains if we want to be able to perform the simulation. In effect, after having received a challenge (the answer to the Test-query), the adversary may continue to initiate some operations before terminating; if we do not want to expose the value $g^{x_1 \cdots x_n}$ during these future sessions, we need to be able to "go backward" and to simulate the flows with less than n exponents again.

How to overcome these points. In light of the previously identified difficulties, one can summarize the strategy of $\varDelta$ as follows:

> *Embed the successive elements of instance $\mathcal{D}$ (after some randomization) in the protocol flows in the order wherein the players join the group, until $n-1$ players have been involved and* except *for instances of the i_0-th player; during the c_0-th operation (creation of the* Test*-ed group), embed the last elements of instance $\mathcal{D}$ via the broadcast operated (hopefully) by the i_0-th player; and after that operation, simulate the flows using line S_{n-1} only, with session keys in line S_n.*

This last point, however, leads us to consider the *extended* trigon rather than the basic one, simply because we cannot know in advance which $(n-1)$-tuple of exponents will be involved in future session simulations.

We now show this strategy allows $\varDelta$ to deal with situations where n players are involved in the group *before* c_0, and are added and removed repeatedly. To prevent all the exponents x_i to be embedded prematurely, it is *sufficient* to prevent *one single* player of using such exponents, by simulating this player with a private exponent y_i that $\varDelta$ chooses by itself. But in order to have *all* the exponents involved in the session key of the Test-ed group, it is *necessary* to know who will be the last player to contribute (that is, which player will broadcast the last down-flow). Thus the "guess" made on a player index i_0.

Detailed steps of the simulation. We give some more details on what $\varDelta$ is doing at each step: before c_0, at c_0, after c_0. We will make intensive use of two counters k and η, specific to each group: any operation Setup, Join or Remove initiates a new group, and then each group structure owns counters k and η. In case of a Setup operation, the counters k and η are initialized to 0, for the two other operations, the new group keep the same values for k and η as the previous group.

First, at any time, and for any operation different from the c_0+1-th:

- for any new player U_i (never involved since the last Setup), if the index $k<n-1$ and $\eta \neq i_0-1$, $\varDelta$ increments both k and i_0, picks a randomizer r_k and thus uses $x'_k = r_k x_k$ as exponent for this player. The tuple (i, k, r_k) is stored in $\mathcal{L}$ for this group. This tuple will never be removed, even when the player leaves the group, but may be updated.
  This way, the up-flow or the broadcast flow involves a random self-reduction of the k-th line in the basic trigon: $S_k(r_1x_1, \dots r_kx_k)$ where all elements are put to power $\prod y_i$ for all the $(i, \perp, y_i)$ in $\mathcal{L}$ for this group. Similarly, the session key is derived from one element from the $k+1$-th line (where $k+1 \leq n$).
- for any new player U_i (never involved since the last Setup), when the index k is already equal to $n-1$, or $\eta = i_0-1$, $\varDelta$ increments i_0 and picks a random exponent y_i for this player. The tuple $(i, \perp, y_k)$ is stored in $\mathcal{L}$ for this group. This tuple will never be removed, even when the player leaves the group, but may be updated.
  This case is to ensure that we are not going to use (random self-reduced) line S_n of the trigon prematurely.

– for a player U_i already involved since the last Setup, one can get (i, k, y) from $\mathcal{L}$. In the case $k = \perp$, $x'_k = y_k = y$ can be used again, unless a new random has to be picked up. In such a case, Δ picks a new random exponent y'_k and thus uses $x''_k = y'_k$ as exponent for this player. The tuple $(i, \perp, y'_k)$ is used for updating $\mathcal{L}$ for this group.

  Otherwise, $x'_k = r_k x_k$ can be used again, unless a new random has to be picked up. In such a case, Δ picks a new randomizer r'_k and thus uses $x''_k = r'_k x_k$ as exponent for this player. The tuple (i, k, r'_k) is used for updating $\mathcal{L}$ for this group.

When the c_0-th operation occurs, the last broadcast flow is operated by the above i_0-th player, who now embeds some elements from the line S_n of the trigon; this means, in particular, that this player is always associated to the last exponent of instance $\mathcal{D}$. It follows that the corresponding session key (which is the Test-ed key) is the **G-CDH**$_\Gamma$ value $g^{x_1 \cdots x_n}$ relative to $\mathcal{D}$, blinded by some (known) random exponents: all the r_i and the y_i. Δ then answers the Test-query as in the real protocol, according to the value of bit b.

After c_0, however, Δ also needs to be able to answer to all queries and more specifically the Reveal-queries (adversary $\mathcal{A}$ may keep playing the game for more rounds). More precisely, we want Δ to do so without using the secret GDH value $g^{x_1 \cdots x_n}$. To this aim, Δ has to *un-embed* the elements of 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 $c_0 + 1$.

Technically speaking, this is feasible by having the initiator of the $c_0 + 1$-th operation choose a fresh private exponent y_i (and not simply blind his corresponding exponent in the instance $\mathcal{D}$ with a fresh randomizer). However depending on which player[8] performs that 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.

Therefore, for any player U_i initiating the $c_0 + 1$-th operation, Δ decrements k, picks a random exponent y_i for this player. The tuple $(i, \perp, y_k)$ is used to update $\mathcal{L}$ for this group.

This way, the up-flow or the broadcast flow involves a random self-reduction of the $n - 1$-th line in the extended trigon and the session key is derived from one element from the n-th line.

For all the subsequent operations (before a new Setup), $k = n - 1$ and thus Δ will use random private exponents for *all* the players, keeping all the x_i but one in the flows[9]. Therefore, the future session keys will still be derived from the n-th line, but the broadcasts may involve any element in the *extended* $n - 1$-th line.

[8] Note this is not obviously (an instance of) the i_0-th player, even if the latter did perform the previous broadcast.

[9] 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.

A comprehensive example of simulation Here we provide a complete example of how $\varDelta$ can correctly handle a set of executions of the protocol, according to the strategy described above. We represent the simulation by $\varDelta$ in the case $n = 4$ and according to the following "guesses": $c_0 = 4, i_0 = 2$. The instance $\mathcal{D}$ is $\{(), (g^{x_1}, g^{x_2}), (g^{x_1x_2}, g^{x_1x_2}, g^{x_2x_3}), (g^{x_1x_2x_3}, g^{x_1x_2x_4}, g^{x_1x_3x_4}, g^{x_2x_3x_4})\}$. Players' private exponents which are fully simulated by $\varDelta$ are denoted y_i, will the randomizers are denoted r_i. We note that U_2 (who performs the broadcast in the crucial session) will be associated with unknown exponent x_4 at that time only. Before that, U_2 is associated to a fully-controlled exponent y_2. As a consequence, indices are a bit tricky to follow, since U_1 is associated with x_1, U_3 with x_2 and U_4 with x_3 (but y_4 after the crucial query).

Setup$\{U_1, U_2, U_3\}$	$c = 1, \mathcal{I} = \{1, 2, 3\}$

$U_1 \qquad U_2 \qquad U_3 \qquad U_4$

$U_1 \xrightarrow{g,\ g^{r_1x_1}} U_2 \xrightarrow{g^{y_2},\ g^{r_1x_1},\ g^{r_1x_1y_2}} U_3$

$U_1 \xleftarrow{g^{y_2r_2x_2},\ g^{r_1x_1r_2x_2},\ g^{r_1x_1y_2}} U_3$

SK $= (g^{x_1x_2})^{r_1r_2y_2}$ is known to $\varDelta$

The adversary first builds a group with the following successive queries: Setup$\{U_1, U_2, U_3\}$, Send(U_2, m_1), Send(U_3, m_2), Send(U_1, m_3), Send(U_2, m_3). To answer the first query, $\varDelta$ simulates player U_1, associating his values with the first term g^{x_1} in instance $\mathcal{D}$; that is $\varDelta$ add $(1, 1, r_1)$ to the first sublist; it can thus construct the message m_1, which is made of the first flow and the appropriate authentication data. Then the adversary asks Send(U_2, m_1). The simulator processes this query as follows: since the player being simulated is U_{i_0} but this Setup is not the c_0-th operation, U_2 is simulated with a fully controlled exponent: $\varDelta$ chooses y_2 by itself and add $(2, \perp, y_2)$ to the current sublist. Finally, when the generated flow m_2 is sent to U_3 via the appropriate query, $\varDelta$ processes it by associating U_3 with the second term of instance $\mathcal{D}$ (modulo some known randomizer r_3): the tuple $(3, 2, r_2)$ is added to the sublist. The computed broadcast can thus be addressed to U_1 and U_2 (simulation is straightforward there). The sublist for this execution is then: $\{(1, 1, r_1), (2, \perp, y_2), (3, 2, r_2)\}$.

Remove$\{U_2\}$	$c = 2, \mathcal{I} = \{1, 3\}$

$U_1 \qquad U_2 \qquad U_3 \qquad U_4$

$U_1 \xleftarrow{g^{y_2r'_2x_2},\ g^{r_1x_1y_2}} U_3$

SK $= (g^{x_1x_2})^{r_1r'_2y_2}$ is known to $\varDelta$

To remove a player from the existing group, the adversary first asks a Remove$\{U_2\}$ query. The simulator can easily simulates the group controller to build a well-formed broadcast: indeed, $\varDelta$ just refreshes the randomizer for U_3. The sublist for this group then becomes $\{(1,1,r_1),(2,\perp,y_2),(3,2,r_2')\}$. The broadcast is sent to U_1 via a Send-query, and that latter is easily processed by Δ to compute the session key from $g^{x_1x_2}$ and the data in $\mathcal{L}=\{(1,1,r_1),(2,\perp,y_2),(3,2,r_2')\}$.

Join$\{U_4\}$ $\qquad$ $c=3, \mathcal{I}=\{1,3,4\}$

$U_1 \qquad U_2 \qquad U_3 \qquad U_4$

$U_3 \xrightarrow{g^{y_2 r_2'' x_2},\ g^{r_1 x_1 y_2},\ g^{r_1 x_1 y_2 r_2'' x_2}} U_4$

$U_1 \xleftarrow{g^{y_2 r_2'' x_2 r_3 x_3},\quad g^{r_1 x_1 y_2 r_3 x_3},\quad g^{r_1 x_1 y_2 r_2'' x_2}} U_4$

SK $=(g^{x_1x_2x_3})^{r_1 r_2'' r_3 y_2}$ is known to Δ

In this step the adversary add a new player to the group, with Join$\{U_4\}$. The simulator Δ will thus generate flows that will be sent successively from U_3 (the group controller) to U_4 (the joining player) and thereafter broadcasted by U_4 (newly group controller) to all other members. The up-flow is computed by Δ using a refreshed randomizer r_2'' and the broadcast is constructed by associating U_4 to the next term of instance $\mathcal{D}$: thus a tuple $(4,3,r_3)$ will be added to the sublist. The sublist representing this execution is $\mathcal{L}=\{(1,1,r_1),(2,\perp,y_2),(3,2,r_2''),(4,3,r_3)\}$. Here we can see the aforementioned *first difficulty*: if we had (in the Setup operation) associated U_2 with a term of the GDH instance, the session key here would have involved the secret value $g^{x_1x_2x_3x_4}$, and Δ would have not be able to answer a possible Reveal-query.

Join$\{U_2\}$ $\qquad$ $c=4, \mathcal{I}=\{1,2,3,4\}$

$U_1 \qquad U_2 \qquad U_3 \qquad U_4$

$U_2 \xleftarrow[g^{r_1 x_1 y_2 r_2'' x_2},\ g^{r_1 x_1 y_2 r_2'' x_2 r_3' x_3}]{g^{y_2 r_2'' x_2 r_3' x_3},\ g^{r_1 x_1 y_2 r_3' x_3}} U_4$

Test-query guessed now

$\xrightarrow{g^{r_4 x_4 r_2'' x_2 r_3' x_3},\quad g^{r_1 x_1 r_2'' x_2 r_3' x_3},\quad g^{r_1 x_1 r_4 x_4 r_3' x_3},\quad g^{r_1 x_1 r_4 x_4 r_2'' x_2}}$

broadcast sent by U_2

SK $=(g^{x_1x_2x_3x_4})^{r_1 r_2'' r_3' r_4}$ is derived from the DH secret

Note that $c=c_0=4$. Here U_2 joins the group again. Before this step, it was not associated with one exponent x_i from $\mathcal{D}$, to prevent premature exposure of the GDH secret. However this time, the simulator has to inject the last exponent: player U_2 will thus be associated with the last exponent x_4, and the current sublist will contain a tuple $(2,4,r_4)$. Note that, knowing the value of y_2, the simulator was able to remove it when injecting x_4 instead. The scenario illustrates the afore mentioned *second difficulty*: why Δ needs to guess which player will perform the crucial broadcast. In effect the up-flow sent by U_4 (current group controller) to U_2 (joining member) must not contain the last exponent x_4, otherwise the secret GDH value is exposed. Thus U_{i_0} is the only player that can be associated with x_4 in the Test-ed session. Consequently his identity must have been guessed before in order to perform a special treatment in earlier sessions. The current sublist here is $\{(1,1,r_1),(2,4,r_4),(3,2,r_2''),(4,3,r_3')\}$.

Remove$\{U_1, U_2\}$	$c = 5, \mathcal{I} = \{3,4\}$

U_1 $\quad$ U_2 $\quad$ U_3 $\quad$ U_4

$$U_3 \xleftarrow{\; g^{r_1 x_1 r_4 x_4 y_4} \;,\; g^{r_1 x_1 r_4 x_4 r_2'' x_2} \;} U_4$$

$\boxed{\mathrm{SK} = (g^{x_1 x_2 x_4})^{r_1 r_2'' r_4 y_4} \text{ is known to } \Delta \text{ again}}$

This scenario explains the *third difficulty*: why the instance $\mathcal{D}$ must follow the "extended trigon" distribution. Because the $c_0 + 1$-th operation removes U_2 from the group (thus making him inactive), Δ cannot update in the sublist the tuple $(2,4,r_4)$ which is relative to U_2; the only exponent that can be refreshed is that of the group controller U_4 when sending the broadcast. This means that Δ will *dissociate* U_4 from the unknown (randomized) exponent x_3 and use a fully controlled exponent y_4 instead. As a consequence, a term derived from $g^{x_1 x_4}$ appears in the broadcast, and that is why Δ needs the extended trigon as his auxiliary input (remind that no such term appears in the basic trigon). The sublist used by Δ is $\{(1,1,r_1),(2,4,r_4),(3,2,r_2''),(4,\perp,y_4)\}$.

Join$\{U_2\}$	$c = 6, \mathcal{I} = \{2,3,4\}$

U_1 $\quad$ U_2 $\quad$ U_3 $\quad$ U_4

$$U_2 \xleftarrow{\; g^{r_1 x_1 r_4 x_4 y_4'} \;,\; g^{r_1 x_1 r_4 x_4 r_2'' x_2} \;,\; g^{r_1 x_1 r_4 x_4 r_2'' x_2 y_4'} \;} U_4$$

$$\xrightarrow{\; g^{r_1 x_1 r_2'' x_2 y_4'} \;,\; g^{r_1 x_1 r_4' x_4 y_4'} \;,\; g^{r_1 x_1 r_4' x_4 r_2'' x_2} \;}$$

$\boxed{\mathrm{SK} = (g^{x_1 x_2 x_4})^{r_1 r_2'' r_4' y_4'} \text{ is known to } \Delta}$

This last operation is mainly for completeness. From now on, the simulator will never use exponent x_3 again: only terms derived from x_1, x_2 and x_4 will be used (with session keys derived from the (known) value $g^{x_1 x_2 x_4}$). Therefore, when simulating the answers to the queries for U_2, Δ simply refreshes the randomizer r_4; the corresponding sublist is $\{(1,1,r_1),(2,4,r_4'),(3,2,r_2''),(4,\perp,y_4')\}$.

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

$$\Pr[\mathsf{Guess}_2] = \Pr[\mathsf{Guess}_3]. \tag{5}$$

Game $\mathbf{G_4}$ Game $\mathbf{G}_4$ is the same as game $\mathbf{G}_3$ except that the simulator is now given as an auxiliary input an instance $\mathcal{D}$ of size n from $\mathsf{GDH}_{\Gamma}^{\$}$, where Γ is the extended trigon $\mathcal{E}_n$:

$$\begin{aligned}\mathcal{D} &= \mathcal{V}iew^{\$}_{\mathcal{E}_n}(x_1, x_2, \ldots, x_n, r)\\ &= \{S_1, S_2(x_1, x_2), \ldots, S_{n-2}(x_1, \ldots, x_{n-2}), S_{n-1}(x_1, \ldots, x_{n-2}, x_{n-1}),\\ &\qquad\qquad S_n(x_1, x_2, \ldots, x_{n-1}, x_n)\} \cup \{g^r\}\end{aligned}$$

Therefore, in case $b = 1$, it uses the value g^r to answer the Test-query. Note this value is used only to answer the Test-query and is never used elsewhere in the simulation described above. In such game, the Reveal-queries can be answered exactly the same way as in the previous game. Straightforwardly, distinguishing between games $\mathbf{G}_3$ and $\mathbf{G}_4$ is at most as hard as solving the $\mathbf{G\text{-}DDH}_{\Gamma}$-problem:

$$|\Pr[\mathsf{Guess}_3] - \Pr[\mathsf{Guess}_4]| \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma}}(T'). \tag{6}$$

The running time of simulator in games $\mathbf{G}_3$ and $\mathbf{G}_4$ is essentially the same as in the first game, except that each query may imply computation of up to n exponentiation needed for the multiplicative random 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 $\Pr[\mathsf{Guess}_5] = 1/2$. Let δ be the distance between the output of $\mathsf{KDF}(\cdot)$ and the uniform distribution, we have:

$$|\Pr[\mathsf{Guess}_5] - \Pr[\mathsf{Guess}_4]| \leq \delta. \tag{7}$$

Conclusion Putting all together equations (2), (3), (4), (5), (6), (7), we get

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

The theorem then follows from lemma 7. □

Remark Recall that this proof is considering an adversary that is restricted not to ask $\mathsf{Corrupt}_{\mathsf{kem}}$-queries. When dealing with strong-corruption we have to answer to all the $\mathsf{Corrupt}_{\mathsf{kem}}$-queries made by the adversary along the games but we can only do so if we know the private exponents involved in the games — these exponents must be given to $\mathcal{A}$ on Corruption queries—. To reach this aim, we can no longer benefit from the random self-random reducibility property of **G-DDH** and have to "guess" the moments at which the adversary will initiate the operations leading to the Test-ed group. Unfortunately, reductions carried out in such a way add an exponential factor in the size of the multicast group: indeed for each of the n players, we will have to guess (among up to Q messages sent) the flow that will be involved to build the Test-ed key; the loss in the probability is thus $O(Q^n)$.

6 Conclusion

In the present paper we have provided a formal model and security definitions, as well as methods, for authenticated group Diffie-Hellman key exchange. Our work should allow cryptographic experts to properly analyze the security of a group key exchange protocol, to address in a rigorous way the security requirements a given method aims to achieve, and to come up with provably secure protocols. The proposed model is sufficiently generic to be adapted to many cryptographic scenarios well-suited for key exchange in a group.

In addition, we have performed a security analysis a protocol suite already proposed for dynamic group Diffie-Hellman key exchange; we have enhanced it with authentication services, proposed a modular implementation that can be used to abstract out the use of cryptographic devices, and exhibit a formal security proof under standard computational assumptions.

This paper, we hope, will enable security architects to pick a method based not only on its efficiency but also on its (provable) security.

Acknowledgments

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-54708. Disclaimer available at http://www-library.lbl.gov/disclaimer.

References

1. Agarwal, D., Chevassut, O., Thompson, M. R., and Tsudik, G. 2001. An integrated solution for secure group communication in wide-area networks. In *Proc. of 6th IEEE Symposium on Computers and Communications*. IEEE Computer Society Press, 22–28. Also Technical Report LBNL-47158, Lawrence Berkeley National Laboratory.
2. Amir, Y., Kim, Y., Nita-Rotaru, C., Schultz, J., Stanton, J., and Tsudik, G. 2004. Secure group communication using robust contributory key agreement. *IEEE Transactions on Parallel and Distributed Systems 15,* 5 (May), 468–480.
3. Amir, Y. and Stanton, J. 1998. The spread wide area group communication system. Tech. rep., CNDS-98-4.
4. Ateniese, G., Steiner, M., and Tsudik, G. 1998. Authenticated group key agreement and friends. In *Proc. of ACM CCS '98*. ACM Press, 17–26.
5. Ateniese, G., Steiner, M., and Tsudik, G. 2000. New multi-party authentication services and key agreement protocols. *IEEE Journal of Selected Areas in Communications 18,* 4 (April), 628–639.

6. Backes, M. and Cachin, C. 2003. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *Proc. of Intl. Conference on Dependable Systems and Networks (DSN-2003).* 37–46.
7. Becker, K. and Wille, U. 1998. Communication complexity of group key distribution. In *Proc. of ACM CCS '98.* ACM Press, 1–6.
8. Bellare, M., Canetti, R., and Krawczyk, H. 1996. Pseudo-random functions revisited: The cascade construction and its concrete security. In *Proc. of FOCS '96.* IEEE Computer Society Press, 514–523.
9. Bellare, M., Canetti, R., and Krawczyk, H. 1998. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. of STOC '98.* ACM Press, 419–428.
10. Bellare, M., Pointcheval, D., and Rogaway, P. 2000. Authenticated key exchange secure against dictionary attacks. In *Proc. of Eurocrypt '00*, B. Preneel, Ed. LNCS, vol. 1807. Springer-Verlag, Berlin, 139–155.
11. Bellare, M. and Rogaway, P. 1993a. Entity authentication and key distribution. In *Proc. of Crypto '93*, D. R. Stinson, Ed. LNCS, vol. 773. Springer-Verlag, Berlin, 232–249.
12. Bellare, M. and Rogaway, P. 1993b. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of ACM CCS '93.* ACM Press, 62–73.
13. Bellare, M. and Rogaway, P. 1995. Provably secure session key distribution: The three party case. In *Proc. of STOC '95.* ACM Press, 57–66.
14. Berket, K., Agarwal, D., and Chevassut, O. 2002. A practical approach to the intergroup protocols. *Future Generation Computer Systems 18,* 5 (April), 709–719.
15. Berman, F., Fox, G., and Hey, T. 2003. *Grid Computing: Making The Global Infrastructure a Reality.* Wiley.
16. Bird, R., Gopal, I. S., Herzberg, A., Janson, P. A., Kutten, S., Molva, R., and Yung, M. 1991. Systematic design of two-party authentication protocols. In *Proc. of Crypto '91*, J. Feigenbaum, Ed. LNCS, vol. 576. Springer-Verlag, Berlin, 44–61.
17. Birman, K. P. 1999. A review experience with reliable multicast. *Software – Practice and Experience 29,* 9 (July), 741–774.
18. Blake-Wilson, S., Johnson, D., and Menezes, A. J. 1997. Key agreement protocols and their security analysis. In *Proc. of 6^{th} IMA International Conference on Cryptography and Coding*, M. Darnell, Ed. LNCS, vol. 1355. Springer-Verlag, Berlin, 30–45.
19. Blake-Wilson, S. and Menezes, A. J. 1997. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proc. of SPW '97*, B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, Eds. LNCS, vol. 1361. Springer-Verlag, Berlin, 137–158.
20. Boneh, D. 1998. The decision Diffie-Hellman problem. In *Proc. of ANTS III*, J. P. Buhler, Ed. LNCS, vol. 1423. Springer-Verlag, Berlin, Portland, Oregon, 48–63.
21. Boyd, C. 1995. Towards a classification of key agreement protocols. In *Proc. of CSFW '95.* IEEE Computer Society Press, 38–43.
22. Boyd, C. 1997. On key agreement and conference key agreement. In *Workshop on Information Security and Privacy.* LNCS, vol. 1270. Springer-Verlag, Berlin, 294–302.
23. Boyd, C. and Mathuria, A. 2003. *Protocols for Authentication and Key Establishment.* Springer-Verlag, Berlin.
24. Boyd, C. and Nieto, J. M. 2003. Round-optimal contributory conference key agreement. In *Proc. of PKC '03*, Y. G. Desmedt, Ed. LNCS, vol. 2567. Springer-Verlag, Berlin, 161–174.
25. Bresson, E. and Catalano, D. 2004. Constant round authenticated group key agreement via distributed computation. In *Proc. of PKC '04*, F. Bao, R. H. Deng, and J. Zhou, Eds. LNCS, vol. 2947. Springer-Verlag, Berlin, 115–129.
26. Bresson, E., Chevassut, O., and Pointcheval, D. 2001. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In *Proc. of Asiacrypt '01*, C. Boyd, Ed. LNCS, vol. 2248. Springer-Verlag, Berlin, 290–309. Full version available from authors' web pages.
27. Bresson, E., Chevassut, O., and Pointcheval, D. 2002a. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Proc. of Eurocrypt '02*, L. R. Knudsen, Ed. LNCS, vol. 2332. Springer-Verlag, Berlin, 321–336. Full version available from authors' web pages.
28. Bresson, E., Chevassut, O., and Pointcheval, D. 2002b. The group Diffie-Hellman problems. In *Proc. of SAC '02*, K. Nyberg and H. Heys, Eds. LNCS, vol. 2595. Springer-Verlag, Berlin, 325–338.
29. Bresson, E., Chevassut, O., Pointcheval, D., and Quisquater, J.-J. 2001. Provably authenticated group Diffie-Hellman key exchange. In *Proc. of ACM CCS '01*, P. Samarati, Ed. ACM Press, 255–264.
30. Burmester, M. and Desmedt, Y. G. 1994. A secure and efficient conference key distribution system. In *Proc. of Eurocrypt '94*, A. D. Santis, Ed. LNCS, vol. 950. Springer-Verlag, Berlin, 275–286.

31. Cachin, C. and Strobl, R. 2004. Asynchronous group key exchange with failures. In *Proc. of PODC '04*. ACM Press, 357–366.
32. Canetti, R. 2000. Security and composition of multi-party cryptographic protocols. *J. of Cryptology 13,* 1 (Winter), 143–202.
33. Canetti, R., Goldreich, O., and Halevi, S. 1998. The random oracle methodology, revisited. In *Proc. of STOC '98*. ACM Press, 209–218.
34. Canetti, R. and Krawczyk, H. 2001. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. of Eurocrypt '01*, B. Pfitzmann, Ed. LNCS, vol. 2045. Springer-Verlag, Berlin, 453–474.
35. Canetti, R. and Krawczyk, H. 2002. Universally composable notions of key exchange and secure channels. In *Proc. of Eurocrypt '02*, L. R. Knudsen, Ed. LNCS, vol. 2332. Springer-Verlag, Berlin, 337–351.
36. Chockler, G. V., Keidar, I., and Vitenberg, R. 2001. Group communication specifications: A comprehensive study. *ACM Computing Surveys 33,* 4 (December), 427–469.
37. Crescenzo, G. D., Ferguson, N., Impagliazzo, R., and Jakobsson, M. 1999. How to forget a secret. In *Proc. of STACS '99*, C. Meinel and S. Tison, Eds. LNCS, vol. 1563. Springer-Verlag, Berlin, 500–509.
38. Diffie, W. and Hellman, M. E. 1976. New directions in cryptography. *IEEE Trans. on Information Theory IT-22,* 6 (November), 644–654.
39. Diffie, W., van Oorschot, P. C., and Wiener, M. J. 1992. Authentication and authenticated key exchange. *Designs, Codes and Cryptography 2,* 2 (June), 107–125.
40. Dupont, R. and Enge, A. 2002. Practical non-interactive key distribution based on pairings. Cryptology ePrint Archive.
41. Foster, I. and Kesselman, C. 2004. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann.
42. Goldwasser, S. and Micali, S. 1984. Probabilistic encryption. *J. of Computer and System Sciences 28,* 2 (April), 270–299.
43. Håstad, J. 1990. Pseudo-random generators under uniform assumptions. In *Proc. of STOC '90.* ACM Press, 395–404.
44. Håstad, J., Impagliazzo, R., Levin, L. A., and Luby, M. 1999. A pseudo-random generator from any one-way function. *SIAM J. of Computing 28,* 4 (August), 1364–1396. Combination of [45] and [43].
45. Impagliazzo, R., Levin, L. A., and Luby, M. 1989. Pseudo-random generation from one-way functions. In *Proc. of STOC '89.* ACM Press, 12–24.
46. Ingemarsson, I., Tang, D. T., and Wong, C. K. 1982. A conference key distribution system. *IEEE Trans. on Information Theory IT-28,* 5 (September), 714–720.
47. Joux, A. 2000. A one-round protocol for tripartite Diffie-Hellman. In *Proc. of ANTS IV*, W. Bosma, Ed. LNCS, vol. 1838. Springer-Verlag, Berlin, 385–394.
48. Joye, M. and Quisquater, J.-J. 1997. On the importance of securing your bins: The garbage-man-in-the-middle attack. In *Proc. of ACM CCS '97.* ACM Press, 135–141.
49. Just, M. and Vaudenay, S. 1996. Authenticated multi-party key agreement. In *Proc. of Asiacrypt '96*, K. Kim and T. Matsumoto, Eds. LNCS, vol. 1163. Springer-Verlag, Berlin, 36–49.
50. Katz, J. and Yung, M. 2003. Scalable protocols for authenticated group key exchange. In *Proc. of Crypto '03*, D. Boneh, Ed. LNCS, vol. 2729. Springer-Verlag, Berlin, 110–125.
51. Kim, Y., Perrig, A., and Tsudik, G. 2000. Simple and fault-tolerant key agreement for dynamic collaborative group. In *Proc. of ACM CCS '00*, S. Jajodia, Ed. ACM Press, 235–244.
52. Kim, Y., Perrig, A., and Tsudik, G. 2001. Communication-efficient group key agreement. In *Proc. of International Federation for Information Processing (IFIP SEC 2001)*, M. Dupuy and P. Paradinas, Eds. IFIP Conference Proceedings, vol. 193. International Federation for Information Processing, Kluwer, 229–244.
53. Li, C.-H. and Pieprzyk, J. 1999. Conference key agreement from secret sharing. In *Proc. of ACISP '99*, J. Pieprzyk, R. Safavi-Naini, and J. Seberry, Eds. LNCS, vol. 1587. Springer-Verlag, Berlin, 64–76.
54. McGrew, D. A. and Sherman, A. T. 1998. Key establishment in large dynamic groups using one-way function trees. Manuscript.
55. Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. 1997. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, Florida. `http://cacr.math.uwaterloo.ca/hac/`.
56. Naor, M. and Reingold, O. 1997. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. of FOCS '97.* IEEE Computer Society Press, 458–467.
57. NIST. 1994. *Federal Information Processing StandardsPublication 140-1: Security Requirements for Cryptographic Modules.* U. S. National Institute of Standards and Technology.

58. Palmer, E. R., Smith, S. W., and Weingart, S. H. 1998. Using a high-performance, programmable secure coprocessor. In *Proc. of Financial Crypto '98*, R. Hirschfeld, Ed. LNCS, vol. 1465. Springer-Verlag, Berlin, 73–89.
59. Pereira, O. and Quisquater, J.-J. 2001. A security analysis of the cliques protocol suites. In *Proc. of CSFW '01*. IEEE Computer Society Press, 73–81.
60. Perrig, A. 1999. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *International Workshop on Cryptographic Techniques and E-Commerce CrypTEC '99*. Hong-Kong City University Press, Hong-Kong, HK.
61. Pfitzmann, B. and Waidner, M. 2001. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. of the 22nd IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 184–200.
62. Rodeh, O., Birman, K. P., and Dolev, D. 2001. The architecture and performance of the security protocols in the ensemble group communication system. *ACM Trans. on Information and System Security 4,* 3 (August), 289–319.
63. Rubin, A. D. and Shoup, V. 1996. Session-key distribution using smart cards. In *Proc. of Eurocrypt '96*, U. M. Maurer, Ed. LNCS, vol. 1070. Springer-Verlag, Berlin, 321–331.
64. Shoup, V. 1999. On formal models for secure key exchange. Technical Report RZ 3120, IBM Zürich Research Lab, Zürich, CH. November.
65. Shoup, V. 2001. OAEP reconsidered. In *Proc. of Crypto '01*, J. Kilian, Ed. LNCS, vol. 2139. Springer-Verlag, Berlin, 239–259.
66. Steer, D. G., Strawczynski, L., Diffie, W., and Wiener, M. J. 1988. A secure audio teleconference system. In *Proc. of Crypto '88*, S. Goldwasser, Ed. LNCS, vol. 403. Springer-Verlag, Berlin, 520–528.
67. Steiner, M., Tsudik, G., and Waidner, M. 1996. Diffie-Hellman key distribution extended to group communication. In *Proc. of ACM CCS '96*. ACM Press, 31–37.
68. Steiner, M., Tsudik, G., and Waidner, M. 2000. Key agreement in dynamic peer group. *IEEE Transactions on Parallel and Distributed Systems 11,* 8 (August), 769–780.
69. Tzeng, W.-G. 2000. A practical and secure fault-tolerant conference-key agreement protocol. In *Proc. of PKC '00*, H. Imai and Y. Zheng, Eds. LNCS, vol. 1751. Springer-Verlag, Berlin, 1–13.
70. von Renesse, R., Birman, K. P., Hayden, M., Vaysburd, A., and Karr, D. 1998. Building adaptive systems using ensemble. *Software–Practice and Experience 28,* 9 (August), 963–979.
71. Vedder, K. and Weikmann, F. 1997. Smart cards requirements, properties, and applications. In *State of the Art in Applied Cryptography*, B. Preneel and V. Rijmen, Eds. LNCS, vol. 1528. Springer-Verlag, Berlin, 307–331.
72. Weingart, S. H. 2000. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Proc. of CHES '00*, ¸C. K. Ko¸c and C. Paar, Eds. LNCS, vol. 1965. Springer-Verlag, Berlin, 302–317.

A Proofs of theorems 1 and 2

A.1 Preliminaries

Remind the GDH-distribution associated to a structure Γ made of proper subsets of $I_n = \{1, \ldots, n\}$:

$$\mathsf{GDH}_\Gamma = \{\mathcal{D}_\Gamma(x_1, \ldots, x_n) \,|\, x_1, \ldots, x_n \in_R \mathbb{Z}_q\}, \quad \text{where} \quad \mathcal{D}_\Gamma(x_1, \ldots, x_n) = \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \, J \in \Gamma \right\}.$$

The γ function denotes the cardinality of any structure Γ:

- for $\mathcal{T}_n$, we have $\tau_n = \gamma(\mathcal{T}_n) = \sum_{i=1}^{n} i = n(n+1)/2$ since the i-th "line" of this structure has exactly i elements.
- the cardinality of $\mathcal{E}_n$ is $\epsilon_n = \gamma(\mathcal{E}_n) = \gamma(\mathcal{T}_n) + \binom{n-2}{n} - n + 1 = n^2 - n + 1$ since the extension of the $n-1$-th line of this structure has exactly $\binom{n-2}{n} - (n-1)$ elements.
- it is also worthwhile to mention that the cardinality of the *Generalized* one is $2^n - 2$.

The later is exponential in n, while the two others are quadratic.

Good Structure Families Our goal is to prove that the hardness of the $\mathbf{G\text{-}DDH}_{\Gamma}$-problem can be reduced to that of the **DDH** one. Given an indexed family $\mathbf{\Gamma} = \{\Gamma_n\}$, we proceed by induction over n: we prove that solving the $\mathbf{G\text{-}DDH}_{\Gamma_n}$-problem reduces to solving the $\mathbf{G\text{-}DDH}_{\Gamma_{n-1}}$-problem. The intuitive (and simple) idea is to replace, in an instance of Γ_n, all occurrences of $x_1 x_2$ by an independent variable x_{12}, so that the number of variables decreases by one, while the computational distance increases by at most $\mathsf{Adv}^{\mathsf{ddh}}$. However, re-mapping the new variable x_{12} to a variable in Γ_{n-1} assumes that the subsets defining the $\mathbf{\Gamma}$ family are well suited for that. To do so, we examine the re-mapping of modified subsets in Γ_n into subsets of Γ_{n-1}. For any indexed structure $\mathbf{\Gamma} = \{\Gamma_n\}_n$, we consider an auxiliary structure $\hat{\mathbf{\Gamma}} = \{\hat{\Gamma}_n\}_n$, where $\hat{\Gamma}_n$ is built from the set $\{0, 3, \ldots, n+1\}$ in the same way Γ_n is built from the set I_n through the map $1 \to 0$, $2 \to 3$, $\ldots$, $n \to n+1$.

Definition 8 (– Good Structure Family). *A family $\mathbf{\Gamma} = \{\Gamma_n\}_n$ is **good** if for any integer n greater than 3 the following four conditions are satisfied:*

1. $\forall J \in \Gamma_n, \{1,2\} \subseteq J \Rightarrow J_{12} \cup \{0\} \in \hat{\Gamma}_{n-1}$
2. $\forall J \in \Gamma_n, 1 \notin J, 2 \in J \Rightarrow J_2 \in \hat{\Gamma}_{n-1}$
3. $\forall J \in \Gamma_n, 1 \in J, 2 \notin J \Rightarrow J_1 \in \hat{\Gamma}_{n-1}$
4. $\forall J \in \Gamma_n, 1 \notin J, 2 \notin J \Rightarrow J \in \hat{\Gamma}_{n-1}$

where for any J, we denote by J_1, J_2 and J_{12} the sets $J\backslash\{1\}$, $J\backslash\{2\}$ and $J\backslash\{1,2\}$ respectively.

In other words, this means that

$$\Gamma_n \subseteq \left\{ J_0 \cup \{1,2\} \middle| J \in \hat{\Gamma}_{n-1}, 0 \in J \right\} \bigcup \left\{ J \cup \{2\}, J \cup \{1\}, J \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J \right\},$$

where for any J, we denote by J_0 the set $J\backslash\{0\}$.

Note 9. The basic trigon $\mathcal{T} = \{\mathcal{T}_n\}$ and extended trigon $\mathcal{E} = \{\mathcal{E}_n\}$ are *good structure families.*

Note 10. In [56] it is proved that the generalized (Decisional) Diffie-Hellman problem is polynomially equivalent to DDH. While it is straightforward that the generalized structure is a good one, we mention that our generic technique described in this section could not be used to establish such reduction for the generalized structure, due to the exponential size of that latter.

Group Random Distributions For proving our result, we need to alter Group Diffie-Hellman tuples, introducing some randomness. This leads to the *group random (GR) distributions* in which some elements are independently random in the *group Diffie-Hellman distributions.*

First we split the tuples in two parts:

$$\begin{aligned}
\mathcal{D}_{\Gamma_n}(x_1, \ldots, x_n) &= \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| J \in \Gamma_n, \{1,2\} \nsubseteq J \right\} \\
&\quad \cup \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| J \in \Gamma_n, \{1,2\} \subseteq J \right\} \\
&= \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \{1,2\} \nsubseteq J \right\} \\
&\quad \cup \left\{ \left(J, g^{x_1 x_2 \prod_{j \in J_{12}} x_j}\right) \middle| \{1,2\} \subseteq J \right\}.
\end{aligned}$$

We can now define an additional distribution:

$$\mathsf{GR}_{\Gamma_n} = \{\mathcal{V}_{\Gamma_n}(x_1, \ldots, x_n, \alpha) \,|\, x_1, \ldots, x_n, \alpha \in_R \mathbb{Z}_q\},$$

where (recall that J_{12} is the set $J \backslash \{1, 2\}$)

$$\begin{aligned} \mathcal{V}_{\Gamma_n}(x_1, \ldots, x_n, \alpha) = &\left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \, J \in \Gamma_n, \{1, 2\} \nsubseteq J \right\} \\ &\bigcup \left\{ \left(J, g^{\alpha \prod_{j \in J_{12}} x_j}\right) \middle| \, J \in \Gamma_n, \{1, 2\} \subseteq J \right\}. \end{aligned}$$

Similarly to what is done for the Group Diffie-Hellman distributions, we define the two tuples $\mathcal{V}^{\star}_{\Gamma_n}(x_1, \ldots, x_n, \alpha)$ and $\mathcal{V}^{\$}_{\Gamma_n}(x_1, \ldots, x_n, \alpha, r)$, the extensions of $\mathcal{V}_{\Gamma_n}(x_1, \ldots, x_n, \alpha)$ where one appends $\{(I_n, g^{\alpha x_3 \cdots x_n})\}$ and $\{(I_n, g^r)\}$ respectively. Then,

$$\begin{aligned} \mathsf{GR}^{\star}_{\Gamma_n} &= \left\{ \mathcal{V}^{\star}_{\Gamma_n}(x_1, \ldots, x_n, \alpha) \,|\, x_1, \ldots, x_n, \alpha \in_R \mathbb{Z}_q \right\}, \\ \mathsf{GR}^{\$}_{\Gamma_n} &= \left\{ \mathcal{V}^{\$}_{\Gamma_n}(x_1, \ldots, x_n, \alpha, r) \,|\, x_1, \ldots, x_n, \alpha, r \in_R \mathbb{Z}_q \right\}. \end{aligned}$$

Note 11. We notice that under the constraint $\alpha = x_1 x_2$, for any $x_1, \ldots, x_n, r \in_R \mathbb{Z}_q$, one would have,

$$\begin{aligned} \mathcal{V}_{\Gamma_n}(x_1, \ldots, x_n, \alpha) &= \mathcal{D}_{\Gamma_n}(x_1, \ldots, x_n) \\ \mathcal{V}^{\star}_{\Gamma_n}(x_1, \ldots, x_n, \alpha) &= \mathcal{D}^{\star}_{\Gamma_n}(x_1, \ldots, x_n) \\ \mathcal{V}^{\$}_{\Gamma_n}(x_1, \ldots, x_n, \alpha, r) &= \mathcal{D}^{\$}_{\Gamma_n}(x_1, \ldots, x_n, r) \end{aligned}$$

and thus,

$$\mathsf{GR}_{\Gamma_n} \equiv \mathsf{GDH}_{\Gamma_n} \quad \mathsf{GR}^{\star}_{\Gamma_n} \equiv \mathsf{GDH}^{\star}_{\Gamma_n} \quad \mathsf{GR}^{\$}_{\Gamma_n} \equiv \mathsf{GDH}^{\$}_{\Gamma_n}.$$

Definition 12 (– Group Random Adversaries). *A Group Computational Random- or (t, ϵ)-GCR_{Γ_n}-attacker in $\mathbb{G}$ is a probabilistic Turing machine Δ running in time t such that*

$$\mathsf{Succ}^{\mathsf{gcr}_{\Gamma_n}}_{\mathbb{G}}(\Delta) = \Pr_{x_i, \alpha}\left[\Delta(\mathcal{V}_{\Gamma_n}(x_1, \ldots, x_n, \alpha)) = g^{\alpha x_3 \cdots x_n}\right] \geq \epsilon.$$

A Group-Decisional-Random- or (t, ϵ)-GDR_{Γ_n}-distinguisher in $\mathbb{G}$ is a probabilistic Turing machine Δ running in time t such that its advantage $\mathsf{Adv}^{\mathsf{gdr}_{\Gamma_n}}_{\mathbb{G}}(\Delta)$ defined by

$$\left| \Pr_{x_i, \alpha}\left[\Delta\big(\mathcal{V}^{\star}_{\Gamma_n}(x_1, \ldots, x_n, \alpha)\big) = 1\right] - \Pr_{x_i, \alpha, r}\left[\Delta\big(\mathcal{V}^{\$}_{\Gamma_n}(x_1, \ldots, x_n, \alpha, r)\big) = 1\right] \right|$$

is greater than ϵ.

A.2 Proof of theorem 1

Now we provide a reduction of the Decisional Diffie-Hellman (DDH) problem to the group Decisional Diffie-Hellman (GDDH) problem, but for the good structure families only. We first (re)state the theorem more formally.

THEOREM 1. *Let $\mathbb{G}$ be a cyclic multiplicative group of prime order q and $t_{\mathbb{G}}$ the time needed for an exponentiation in $\mathbb{G}$. For any good structure family $\mathbf{\Gamma} = \{\Gamma_n\}_n$ of cardinality $\gamma = \{\gamma_n\}_n$ and any integer n, we have:*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(t) \leq (2n-3)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ where } t' \leq t + t_{\mathbb{G}} \sum_{i=3}^{n} \gamma_i.$$

The proof results, by induction, from the following two lemmas 13 and 14 which lead to

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(t) \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) + 2\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}).$$

However before to prove it let's plug in some numerical values for the time of computation:

- for the structure of basic trigon $\mathcal{T}_n$, the time t' is less than $t + n^3 t_{\mathbb{G}}/3$;
- for the structure of extended trigon $\mathcal{E}_n$, the time t' is less than $t + 2n^3 t_{\mathbb{G}}/3$.

Lemma 13 (– Relating GDDH and GDR). *For any integer n and any structure Γ_n, we have*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(t) \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gdr}_{\Gamma_n}}(t) + 2\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}).$$

Proof. We consider an adversary $\mathcal{A}$ against the $\mathbf{G\text{-}DDH}_{\Gamma_n}$ problem. Such an adversary, on input a distribution depending on a bit b, replies with a bit b' which is a guess for b. We assume that $\mathcal{A}$ runs in maximal time t, in particular it always terminates, even if the input comes from neither $\mathsf{GDH}_{\Gamma_n}^{\star}$ nor from $\mathsf{GDH}_{\Gamma_n}^{\$}$. Then we define the following two games: $\mathbf{G}_0$, $\mathbf{G}_1$ and consider the event S_i in game $\mathbf{G}_i$ as $b = b'$.

Game $\mathbf{G_0}$ In this game, we are given a Diffie-Hellman triple $(A, B, C) = (g^{x_1}, g^{x_2}, g^{x_1 x_2})$. Then we choose at random $(x_3, \ldots, x_n)$ in $\mathbb{Z}_q^*$ and compute (within time $O(\gamma_n t_{\mathbb{G}})$) a tuple U_n which follows the distribution GDH_{Γ_n}, as follows

$$\begin{aligned} \mathsf{U}_n = & \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \, J \in \Gamma_n, 1 \notin J, 2 \notin J \right\} \\ & \bigcup \left\{ \left(J, A^{\prod_{j \in J_1} x_j}\right) \middle| \, J \in \Gamma_n, 1 \in J, 2 \notin J \right\} \\ & \bigcup \left\{ \left(J, B^{\prod_{j \in J_2} x_j}\right) \middle| \, J \in \Gamma_n, 1 \notin J, 2 \in J \right\} \\ & \bigcup \left\{ \left(J, C^{\prod_{j \in J_{12}} x_j}\right) \middle| \, J \in \Gamma_n, \{1, 2\} \subseteq J \right\}. \end{aligned}$$

Then if $b = 1$, one appends to U_n the value $C^{x_3 \cdots x_n}$; and if $b = 0$, one appends to U_n a value g^r, where r is a random exponent: the computed tuple follows exactly the distribution $\mathsf{GDH}_{\Gamma_n}^{\star}$ (resp. $\mathsf{GDH}_{\Gamma_n}^{\$}$) if $b = 1$ (resp. $b = 0$). Thus by definition, if we feed the attacker $\mathcal{A}$ with this tuple, we have

$$\Pr[S_0] = \frac{\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(\mathcal{A}) + 1}{2}.$$

Game $\mathbf{G_1}$ It is the same as game $\mathbf{G}_0$ except that we are given a tuple $(A, B, C) = (g^{x_1}, g^{x_2}, g^{\alpha})$, where α is a random exponent. It is easy to see that the tuple given to the attacker $\mathcal{A}$ follows the distribution $\mathsf{GR}^{\star}_{\Gamma_n}$ (resp. $\mathsf{GR}^{\$}_{\Gamma_n}$) if $b = 1$ (resp. $b = 0$). Then,

$$\Pr[S_1] = \frac{\mathsf{Adv}^{\mathsf{gdr}_{\Gamma_n}}_{\mathbb{G}}(\mathcal{A}) + 1}{2} \leq \frac{\mathsf{Adv}^{\mathsf{gdr}_{\Gamma_n}}_{\mathbb{G}}(t) + 1}{2}.$$

Also, the difference in the probability distributions in the two games is upper-bounded by:

$$\Pr[S_0] \leq \Pr[S_1] + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + \gamma_n t_{\mathbb{G}}).$$

The lemma follows. $\square$

Lemma 14 (– Induction Step). *For any good structure family $\mathbf{\Gamma} = \{\Gamma_n\}$ and any integer n, we have*

$$\mathsf{Adv}^{\mathsf{gdr}_{\Gamma_n}}_{\mathbb{G}}(t) \leq \mathsf{Adv}^{\mathsf{gddh}_{\Gamma_{n-1}}}_{\mathbb{G}}(t + \gamma_n t_{\mathbb{G}}).$$

Proof. We consider a GDR_{Γ_n}-distinguisher $\mathcal{A}$ running in time t and we use it to built a $\mathbf{G\text{-}DDH}_{\Gamma_{n-1}}$-distinguisher. To reach that goal, we receive as input a tuple drawn from either $\mathsf{GDH}^{\star}_{\Gamma_{n-1}}$ or $\mathsf{GDH}^{\$}_{\Gamma_{n-1}}$. We use $\mathcal{A}$ to guess the underlying bit b. In the given tuple, we denote by (I_{n-1}, u_{n-1}) the last value and by U_{n-1} the first values of this input tuple:

$$\begin{aligned}
&\mathsf{U}_{n-1} = \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| \, J \in \Gamma_{n-1} \right\} = \mathcal{D}_{\Gamma_{n-1}}(x_1, \ldots, x_{n-1}) \in \mathsf{GDH}_{\Gamma_{n-1}} \\
&u_{n-1} = g^{x_1 \ldots x_{n-1}} \text{ if } b = 1, \text{ or } g^r \text{ if } b = 0.
\end{aligned}$$

First, we split the tuple U_{n-1} in two blocks, depending whether $1 \in J$:

$$\mathsf{U}_{n-1} = \left\{ \left(J, g^{x_1 \prod_{j \in J_1} x_j}\right) \middle| J \in \Gamma_{n-1}, 1 \in J \right\} \cup \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \middle| J \in \Gamma_{n-1}, 1 \notin J \right\}.$$

Then we write this tuple by renaming the variables $x_1, \ldots, x_{n-1}$ to be respectively $X_0, X_3, \ldots, X_n$. It then follows that the elements of U_{n-1} are indexed by the elements of $\hat{\Gamma}_{n-1}$ rather than Γ_{n-1}:

$$\left\{ \left(J, g^{X_0 \prod_{j \in J_0} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \in J \right\} \cup \left\{ \left(J, g^{\prod_{j \in J} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J \right\}.$$

Now we pick at random two values X_1, X_2 in $\mathbb{Z}_q^*$ and use them to construct the following tuple, in which the last block in the above equation is used to derive the last three blocks of W_{n-1}:

$$\begin{aligned}
\mathsf{W}_{n-1} = & \left\{ \left(J, g^{X_0 \prod_{j \in J_0} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \in J \right\} \\
& \bigcup \left\{ \left(J, g^{X_2 \prod_{j \in J} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J \right\} \\
& \bigcup \left\{ \left(J, g^{X_1 \prod_{j \in J} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J \right\} \\
& \bigcup \left\{ \left(J, g^{\prod_{j \in J} X_j}\right) \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J \right\}.
\end{aligned}$$

Remember that $\mathbf{\Gamma}$ is a "good" structure family:

$$\Gamma_n \subseteq \left\{J_0 \cup \{1,2\} \middle| J \in \hat{\Gamma}_{n-1}, 0 \in J\right\} \bigcup \left\{J \cup \{2\}, J \cup \{1\}, J \middle| J \in \hat{\Gamma}_{n-1}, 0 \notin J\right\}.$$

It follows that one can build the following tuple V_n which is also included in W_{n-1}:

$$\begin{aligned}\mathsf{V}_n = &\left\{\left(J, g^{X_0 \prod_{j \in J_{12}} X_j}\right) \middle| J \in \Gamma_n, \{1,2\} \subseteq J\right\} \\ &\bigcup \left\{\left(J, g^{\prod_{j \in J} X_j}\right) \middle| J \in \Gamma_n, \{1,2\} \nsubseteq J\right\}.\end{aligned}$$

We note that

$$\mathsf{V}_n = \mathcal{V}_{\Gamma_n}(X_1, \ldots, X_n, X_0) \in \mathsf{GR}_{\Gamma_n}.$$

Then V_n is appended (I_n, u_{n-1}) and given to $\mathcal{A}$. The latter returns a bit b' that we relay back as an answer to the original $\mathbf{G\text{-}DDH}_{\Gamma_{n-1}}$ problem. The computation time needed to properly generate V_n from the input U_{n-1} is at most $\gamma_n t_{\mathbb{G}}$.

Thus, we have

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) \geq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gdr}_{\Gamma_n}}(t).$$

The lemma follows. □

Putting all together, we obtain:

$$\begin{aligned}\mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_n}}(t) &\leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gdr}_{\Gamma_n}}(t) + 2\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ &\leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{gddh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) + 2\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ &\leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}\left(t + \sum_{i=3}^{n} \gamma_i t_{\mathbb{G}}\right) + 2\sum_{i=3}^{n} \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}\left(t + \sum_{j=i}^{n} \gamma_j t_{\mathbb{G}}\right) \\ &\leq (2n-3)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ where } t' \leq t + t_{\mathbb{G}} \sum_{i=3}^{n} \gamma_i.\end{aligned}$$

A.3 Proof of theorem 2

Now we show the GCDH is a standard assumption by relating it to both the CDH and the DDH.

Theorem 2. *Let $\mathbb{G}$ be a cyclic multiplicative group of prime order q and $t_{\mathbb{G}}$ the time needed for an exponentiation in $\mathbb{G}$. For any good structure family $\mathbf{\Gamma} = \{\Gamma_n\}_n$ of cardinality $\gamma = \{\gamma_n\}_n$ and any integer n, we have:*

$$\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(t) \leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}(t') + (n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ where } t' \leq t + \sum_{i=3}^{n} \gamma_i t_{\mathbb{G}}.$$

As for the previous theorem, the result comes, by induction, from both

$$\begin{aligned}\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(t) &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcr}_{\Gamma_n}}(t) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcr}_{\Gamma_n}}(t) &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}).\end{aligned}$$

We consider an adversary $\mathcal{A}$ against the $\mathbf{G\text{-}CDH}_{\Gamma_n}$ problem. Such an adversary, on input a tuple drawn from the GDH_{Γ_n} distribution, replies with a single value which is a guess for the corresponding secret. We assume that $\mathcal{A}$ runs in maximal time t, in particular it always terminates, even if the input does not come from GDH_{Γ_n}.

We then define a sequence of games $\mathbf{G}_0$, $\mathbf{G}_1$, ... In each game, given a triple (A, B, C) and $n-2$ random elements $(x_3, \dots, x_n)$ in $\mathbb{Z}_q^*$ (which are not necessarily known), we consider S_i as the event that the adversary $\mathcal{A}$ outputs $C^{x_3 \cdots x_n}$.

Game $\mathbf{G_0}$ In this game, we are given a Diffie-Hellman triple $(A, B, C) = (g^{x_1}, g^{x_2}, g^{x_1 x_2})$. Then by randomly choosing $(x_3, \dots, x_n)$ we can compute:

$$\begin{aligned}\mathsf{U}_n = & \left\{ \left(J, g^{\prod_{j \in J} x_j}\right) \Big| \, J \in \Gamma_n, 1 \notin J, 2 \notin J \right\} \\ & \bigcup \left\{ \left(J, A^{\prod_{j \in J_1} x_j}\right) \Big| \, J \in \Gamma_n, 1 \in J, 2 \notin J \right\} \\ & \bigcup \left\{ \left(J, B^{\prod_{j \in J_2} x_j}\right) \Big| \, J \in \Gamma_n, 1 \notin J, 2 \in J \right\} \\ & \bigcup \left\{ \left(J, C^{\prod_{j \in J_{12}} x_j}\right) \Big| \, J \in \Gamma_n, \{1, 2\} \subseteq J \right\}.\end{aligned}$$

It is easy to see that $\mathsf{U}_n = \mathcal{D}_{\Gamma_n}(x_1, \dots, x_n)$, and thus follows exactly the distribution GDH_{Γ_n}. Then the tuple U_n is provided to the adversary. By definition, since $C^{x_3 \cdots x_n} = g^{x_1 \cdots x_n}$, we have

$$\Pr[S_0] = \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(\mathcal{A}).$$

Game $\mathbf{G_1}$ It is the same as game $\mathbf{G}_0$ except that we are given a tuple $(A, B, C) = (g^{x_1}, g^{x_2}, g^{\alpha})$, where α is a random element in $\mathbb{Z}_q^*$. We then perform the same operations as in game $\mathbf{G}_0$ to obtain a tuple which follows the distribution GR_{Γ_n}: $\mathsf{U}_n = \mathcal{V}_{\Gamma_n}(x_1, \dots, x_n, \alpha)$. This tuple is provided to the adversary, which computes $g^{\alpha x_3 \dots x_n}$. By definition, we have:

$$\Pr[S_1] = \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcr}_{\Gamma_n}}(\mathcal{A}) \le \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcr}_{\Gamma_n}}(t).$$

In both games the computation time needed for generating the tuple from the input a triple (A, B, C) is at most $(\gamma_n - 1) t_{\mathbb{G}}$ where $t_{\mathbb{G}}$ is the time required for an exponentiation in $\mathbb{G}$. Another exponentiation is needed to compute $C^{x_3 \cdots x_n}$. Clearly the computational distance between the games is upper-bounded by $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}})$, then:

$$\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(\mathcal{A}) \le \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcr}_{\Gamma_n}}(t) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}).$$

Game $\mathbf{G_2}$ It is the same as game $\mathbf{G}_1$ except that we choose x_1 and x_2 by ourselves. Therefore $(A, B, C) = (g^{x_1}, g^{x_2}, g^{\alpha})$ where x_1 and x_2 are known, but α is not. The remaining of this game is distributed exactly as in the previous one, so $\Pr[S_2] = \Pr[S_1]$.

Game $\mathbf{G_3}$ It is the same as game $\mathbf{G}_2$ except that we do not know the elements $(x_3, \ldots, x_n)$. Instead, we are given an instance U_{n-1} of the $\mathbf{G\text{-}CDH}_{\Gamma_{n-1}}$ problem, built from the (unknown) exponents $(\alpha, x_3, \ldots, x_n)$, where α is the same than the underlying (hidden) exponent in C. By operating as in the previous section, granted the property of good structure family, we can complete the given tuple by using x_1 and x_2 (which are known) to obtain a tuple V_n following the distribution GR_{Γ_n}.

The variables are distributed exactly as in the previous game, so we have $\Pr[S_3] = \Pr[S_2]$. Note that since we do not know $x_3, \ldots, x_n$, we are no longer able to decide whether the value the adversary outputs is $C^{x_3 \cdots x_n}$. But it is not a problem since the two games are *perfectly* identical.

Anyway, since $C^{x_3 \cdots x_n} = g^{\alpha x_3 \cdots x_n}$ is the Diffie-Hellman secret associated to the given $\mathbf{G\text{-}CDH}_{\Gamma_{n-1}}$ instance, the adversary outputs $C^{x_3 \cdots x_n}$ with probability at most $\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}})$:

$$\Pr[S_3] \leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}).$$

Putting all these together gives us

$$\begin{aligned}\Pr[S_0] &= \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(\mathcal{A}) \leq \Pr[S_1] + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ &\leq \Pr[S_3] + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}})\end{aligned}$$

Since it is true for any adversary running within time t,

$$\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(t) \leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}).$$

By induction, it follows:

$$\begin{aligned}\mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_n}}(t) &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-1}}}(t + \gamma_n t_{\mathbb{G}}) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{gcdh}_{\Gamma_{n-2}}}(t + (\gamma_n + \gamma_{n-1}) t_{\mathbb{G}}) \\ &\qquad + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + (\gamma_n + \gamma_{n-1}) t_{\mathbb{G}}) + \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + \gamma_n t_{\mathbb{G}}) \\ &\leq \ldots \\ &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}\left(t + \sum_{i=3}^{n} \gamma_i t_{\mathbb{G}}\right) + \sum_{i=3}^{n} \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}\left(t + \sum_{j=i}^{n} \gamma_j t_{\mathbb{G}}\right) \\ &\leq \mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}(t') + (n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') \text{ where } t' \leq t + \sum_{i=3}^{n} \gamma_i t_{\mathbb{G}}.\end{aligned}$$

□