UCL
Université
catholique
de Louvain

Faculté des Sciences Appliquées
Laboratoire de Microélectronique

Authenticated Group Diffie-Hellman Key-Exchange : Theory and Practice

Olivier Chevassut

Thèse soutenue en vue de l'obtention du grade de
Docteur en Sciences Appliquées

Composition du jury:

Dr. D. A. Agarwal (Lawrence Berkeley National Laboratory, U.S.A)
Prof. J. M. Couveignes (Université de Toulouse, France)
Prof. J. D. Legat (Université Catholique de Louvain, Belgium) - Président
Prof. B. Macq (Université Catholique de Louvain, Belgium)
Prof. A. Magnus (Université Catholique de Louvain, Belgium)
Dr. D. Pointcheval (École Normale Supérieure, France)
Prof. J-J. Quisquater (Université Catholique de Louvain, Belgium) - Promoteur
Prof. J. Stern (École Normale Supérieure, France)

Louvain-la-Neuve, Belgique
Octobre 2002

Abstract

Authenticated two-party Diffie-Hellman key exchange allows two principals A and B, communicating over a public network, and each holding a pair of matching public/private keys to agree on a session key. Protocols designed to deal with this problem ensure A (B resp.) that no other principals aside from B (A resp.) can learn any information about this value. These protocols additionally often ensure A and B that their respective partner has actually computed the shared secret value.

A natural extension to the above cryptographic protocol problem is to consider a pool of principals agreeing on a session key. Over the years several papers have extended the two-party Diffie-Hellman key exchange to the multi-party setting but no formal treatments were carried out till recently. In light of recent developments in the formalization of the authenticated two-party Diffie-Hellman key exchange we have in this thesis laid out the authenticated group Diffie-Hellman key exchange on firmer foundations.

Keywords: Group Key-Exchange, Diffie-Hellman, Security Model

Acknowledgments

I could not have written this thesis without the help, input and support of many others. I would like to thank my advisor, Jean-Jacques Quisquater, for getting the ball rolling and my commitee members, Deborah Agarwal and David Pointcheval, for keeping it in play. Much appreciation goes to David Pointcheval, especially for his help in cryptography, and to Deborah Agarwal for believing in this project and helping make it a success. Thanks also to Jean-Marc Couveigne for pushing my thinking and leading my very first steps in research; many thanks as well to Jacques Stern for believing in this project while in its very early stages. Thanks also to Jean-Didier Legat, Benoit Macq, Alphonse Magnus who served me on committee thesis.

Of course, I am also very greateful to the members of the Complexity and Cryptography Research Group at the Ecole Normale Superieure, the Distributed Systems Department at Lawrence Berkeley National Laboratory, and the Cryptographic Group at the University Catholic of Louvain, who shared their knowledge and experiences with me. Without them, this thesis would never have came about. They are Emmanuel Bresson, Karlo Berket, Abdelilah Essiari, Mary Thompson, Sylvie Baudine for her invaluable logistic support, and the many others who for their own reasons did not wish their names mentionned. I also would like to thank my family, Alain, Marie-Claude and of course Lydie, for their understanding and assistance.

This work 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 Université Catholique de Louvain report number CG-2002/4 and Lawrence Berkeley National Laboratory report number LBNL-51150.

Biography and Publications

Olivier received is Diploma in computer science from the University of Bordeaux, France in 1998 and is working towards the Ph.D. degree in computer science from the University Catholic of Louvain, Belgium. He is also a research computer scientist in the Department of Distributed Systems at Ernest Orlando Lawrence Berkeley National Laboratory and in the UCL Cryptographic Group at the University Catholic of Louvain. His interests include theoretical and applied cryptography as well as secure and reliable multicast systems.

(1) D. A. Agarwal, O. Chevassut, M. R. Thompson and G. Tsudik, "An Integrated Solution for Secure Group Communication in Wide-Area Networks", Proceedings of the 6th IEEE Symposium on Computers and Communications, Hammamet, Tunisia, July 2001, pp 22-28

(2) G. Ateniese, O. Chevassut, D. Hasse, Y. Kim and G. Tsudik, "The Design of a Group Key Agreement API", Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX), IEEE Computer Society Press, 2000.

(3) K. Berket, D. A. Agarwal and O. Chevassut, "A Practical Approach to the InterGroup Protocols", Journal of Future Generation Computer Systems, volume 18, number 5, April 2002, pp 709-719.

(4) E. Bresson, O. Chevassut, D. Pointcheval and J. J. Quisquater, "Provably Authenticated Group Diffie-Hellman Key Exchange", Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, Nov 2001, pp 255-264

(5) E. Bresson, O. Chevassut and D. Pointcheval, "Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case", Proceedings of Asiacrypt, Gold Coast, Queensland, Australia, Dec 2001, pp 290-309

(6) E. Bresson, O. Chevassut and D. Pointcheval, "Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions",

Proceedings of Eurocrypt, Amsterdam, Netherlands, April 2002, pp 321-336.

(7) E. Bresson, O. Chevassut and D. Pointcheval, "The Group Diffie-Hellman Problems", Workshop on Selected Areas in Cryptography, St. John's, Newfoundland, Canada, August 2002.

(8) E. Bresson, O. Chevassut and D. Pointcheval, "Group Diffie-Hellman Key Exchange secure against Dictionary Attacks", Proceedings of Asiacrypt, Queenstown, New Zealand, Dec 2002.

Contents

Introduction

Grid technology offers the ability to bring together through the Internet a global network of computers, storage systems and other resources to access information and tackle complex analysis tasks. Grids enable large-scale parallel computations and exchange of data among processes belonging to the same computation [FK98, FKT01]. The Global Grid Forum, the umbrella organization for the grid projects under development around the world, has been working to define the architecture required to fulfill the security needs of a computational grid [FKTT98] and has adopted a client/server model for securing traffic over the grid. Although the client/server model is tempting, a server can be a performance bottleneck, a single point of failure, and it requires significant administration. Also, if the network partitions into different components due to for example a link failure only the component containing the server can continue.

Security solutions for collaborative environments have also traditionally been developed with the client/server model in mind. The Access Grid environment, for example, supports human-to-human group interactions [Gri]. It is a collection of virtual rooms, called venues, which enable large-scale distributed meetings, collaborative work sessions, seminars, lectures, tutorials and training. A participant joins a secure venue by connecting via the Hypertext Transfer Protocol (HTTP) to the venue server responsible for all membership and coordination activities of participants. This architecture works well for collaborations that run the server but leaves little room for the less well-healed collaboration. Small and fleeting collaborations are usually built in an ad-hoc manner and often there is no site in a position to run the server.

A more natural communication model for grid applications is a an architecture where the participants are treated as equals. Reconfiguring the grid aplications to use a peer group structure would allow the various clients participating in the application to independently organize and continue to operate in the presence of network failures or attacks. Peer-to-peer has the potential to provide a new model that will better

support collaborative self-organizing structures, but a critical issue that needs to be looked at before this vision can be realized is security.

The challenge with security services is how to best provide them to the application. The approach taken by *the Secure Socket Layer Protocol* (SSL/TLS), the current de facto standard protocol for securing the traffic over the Internet, is to interpose a security layer between the application and the transport layer protocol [FKK96, DR02] (see Figure 1). This protocol is easy to deploy since it only requires minor changes in the application to convey users' identity information/access privileges and leverages off the properties of the *Transmission Control Protocol* (TCP) [CK74] in transmitting its own messages.

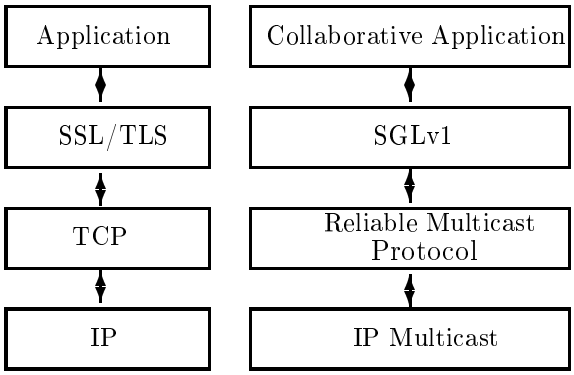

FIGURE 1. Networking protocol stack.

In the case of the Access Grid, the software operating in the venue (e.g, videoconferencing (vic, vat, and rat), session directory (sdr), and whiteboard (wb)) leverages off the multicast capabilities of the underlying network to send messages (see Figure 1). Securing these messages requires a layer similar to SSL but for multicast. In this thesis we describe such a layer protocol called *Secure Group Layer* (SGL) which provides the collaborative application with a security context within which messages multicast over the wire could be cryptographically protected.

The essential building block for setting up a secure multicast context is a key exchange protocol that allows the participants to exchange a session key as equals and, therefore, treats them as peers. The first step in solving this problem is to design an algorithm that allows a set of participants to agree on a session key. We refer to this kind of group genesis as the initial group Diffie-Hellman key exchange. The algorithm for initial group Diffie-Hellman key exchange depicted on Figure 2 proceeds in two stages and it is described in terms of the messages received in

each stages [STW96]. In the up-flow stage, the player raises the received intermediate values to the power of its private input and forwards the result to the next player in the ring. The down-flow takes place when the last player U_n in the multicast group receives the last up-flow and computes the session key. The last player U_n raises the intermediate values it has received to the power of its private key and broadcasts the result (i.e. Fl_n) to allow the other players to construct the session key.

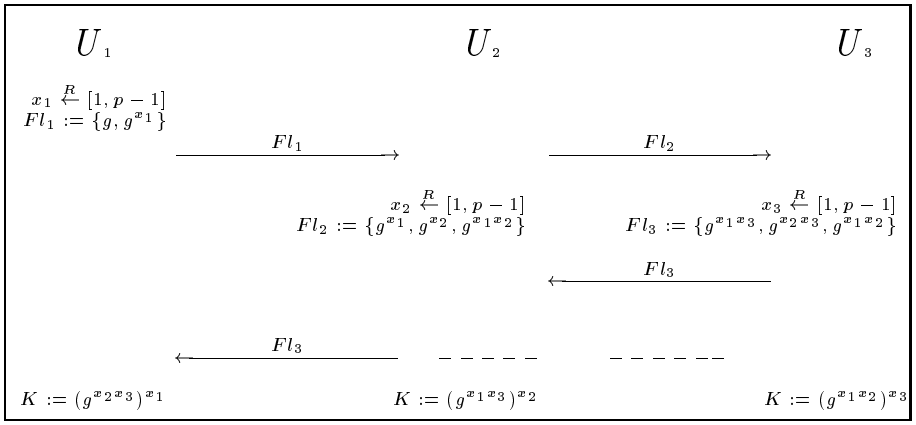

FIGURE 2. The group Diffie-Hellman protocol in a finite cyclic group $< g > \subsetneq \mathbb{Z}_p^*$. An example of an honest execution among 3 players U_1, U_2, U_3. The session key K is $g^{x_1 x_2 x_3}$.

This algorithm was originally proposed by Steiner et al. [STW96], however, the attacks found by Pereira and Quisquater [PQ01a, PQ01b] against this protocol have demonstrated that designing secure group Diffie-Hellman key exchange algorithms is difficult and fraught with many complications. Some security researchers have used formal specification tools to analyze their protocol but a logical proof does not imply a protocol is secure [BAN90, vO93, MY99, Mea00, PQ01a, PQ01b]. Numerous cryptographic researchers have over the years turned out to the provable-security approach to assess the security of their cryptographic constructions [BPR00, Sho99].

My first contribution in this thesis is to provide protocol designers with a provable-security framework to assess the security of group Diffie-Hellman key exchange protocols. I provide three models which capture the characteristics of the operational environments of group Diffie-Hellman key protocols. The first model captures the capabilities of the adversary and the security definitions for the initial Diffie-Hellman key

exchange. In initial group Diffie-Hellman key exchange, the members of the group come together and agree on a new session key without relying on any previous agreed session keys. This model is a stepping stone toward the second model.

In reality the membership for group of participants is not static, instead it is built incrementally. Participants join and leave the group at any time and the group may be split into disjoint components due to a network failures or even attacks. The second model is equipped with notions of dynamicity in the group membership to correctly capture the adversary's capabilities and the security definitions for the auxiliary group Diffie-Hellman key exchange. After the initial group Diffie-Hellman key exchange, the parties run the auxiliary group Diffie-Hellman key exchanges to update the session key after each join, leave, merge and partition events [AST00, ACH$^+$00, STW00]. Our third model captures even stronger capabilities: access to the internal memory of player and parallel executions of auxiliary group Diffie-Hellman key exchanges.

My second contribution in this thesis is to provide engineers with a serie of group Diffie-Hellman key exchange algorithms that are practical, provably-secure and easy to implement. Practiality is reached by modifying the algorithms from [AST00, ACH$^+$00, STW96, STW00]. Provable-security is reached by constructing reductions showing that in our formal model the algorithm achieves the security definitions under reasonable intractability assumptions. Since reasonable does not have the same meaning for everyone we provide various choices of intractability assumptions (decisional vs. computational) and session key derivation techniques (ideal-hash model vs. left-over-hash lemma). Our algorithms can easily be modified by pairing of a particular assumption/key derivation technique and a formal model. Easy of implementation is reached by giving hints on how to put them in practice.

This thesis is organized as follows. The basic principles and techniques developed by the field of modern cryptography for the task of exchanging a session key are introduced in Chapter 1. The next three chapters extend those concepts and techniques to tackle the group Diffie-Hellman key exchange. Chapter 2 addresses the initial group Diffie-Hellman key exchange problem. Chapter 3 adresses the broader scenario of auxiliary group Diffie-Hellman key exchange wherein the members join and leave the group at any time. Chapter 4 adds important security attributes to our two previous treatments. Alone the group Diffie-Hellman key exchange is of relatively little practical use. A mechanism to enforce restrictions on who can participate in the key exchange and, therefore, the multicast group is needed. Chapter 5 describes our integration of

the group Diffie-Hellman key exchange and the access control mechanism into the security layer SGLv1. Finally, we summarize the contributions of this thesis.

CHAPTER 1

Modern Cryptography

The theoretical concepts of public-key cryptography go back to Diffie and Hellman in 1976 [DH76] and the first public-key cryptosystem was proposed two years later by Rivest, Shamir and Adelman [RSA78]. In their seminal paper *New Directions in Cryptography*, Diffie and Hellman provided a method whereby two principals communicating over an insecure network can securely agree on a secret value.

Since the publication of this method, the cryptographic task of exchanging a secret value in presence of an adversary found a rigorous and systematic treatment in the framework of modern cryptography. *The field of modern cryptography provides a theoretical foundation based on which we may understand what exactly cryptographic problems are, how to evaluate protocols that purport to solve them, and how to build protocols in whose security we can have confidence* [GB01].

With the generalization of the Diffie-Hellman method to the multi-party setting [STW96], it is completely natural to provide a formal treatment for the group Diffie-Hellman key exchange problem in the framework of modern cryptography. In this chapter we introduce the cryptographic primitives and the modern cryptography notions used throughout the thesis.

1. Abstract Groups

A finite cyclic group in this thesis will be seen as an abstract group $\mathbb{G}$ with operation denoted multiplicatively, identity element denoted 1, and generator denoted g. In their original paper Diffie-Hellman presented a method that works in a finite cyclic group $\mathbb{Z}_p^*$ however since their publication other finite cyclic groups have been found suitable for application in the Diffie-Hellman protocol. Examples of such finite cyclic groups are the prime subgroups of $\mathbb{Z}_p^*$ [MW00], the (hyper)-elliptic curve based groups [Jou00, KMV00, Kob98, MW00], the XTR subgroups [SL01] or even the class group of an imaginary quadratic fields [BW88].

2. Diffie-Hellman Method

The Diffie-Hellman protocol works as follows. As illustrated in Figure 1, this protocol consists of an up-flow and a down-flow. In the up-flow, player U_1 chooses at random a value x_1 in $[1, |\mathbb{G}|]$, raises the value g to the power of x_1 and sends on the wire the resulting value to player U_2. The down-flow takes place when U_2 receives the up-flow. Player U_2 then chooses at random a value x_2 in $[1, |\mathbb{G}|]$, raises the value g to the power of x_2 and sends on the wire the resulting value to player U_2. It is then straightforward to check that both players can compute the shared secret value.

The motivation for running the Diffie-Hellman protocol is to implement a secure session over an insecure network connection. A secure session which provides an authenticated and private network channel between U_1 and U_2. More precisely, the value the players agreed on is used to achieve cryptographic goals like data integrity and/or message confidentiality. The players reach this aim by simply applying a function mapping elements of $\mathbb{G}$ to the space of either a Message Authentication Code [BCK96] and/or a conventional block cipher [DR00].

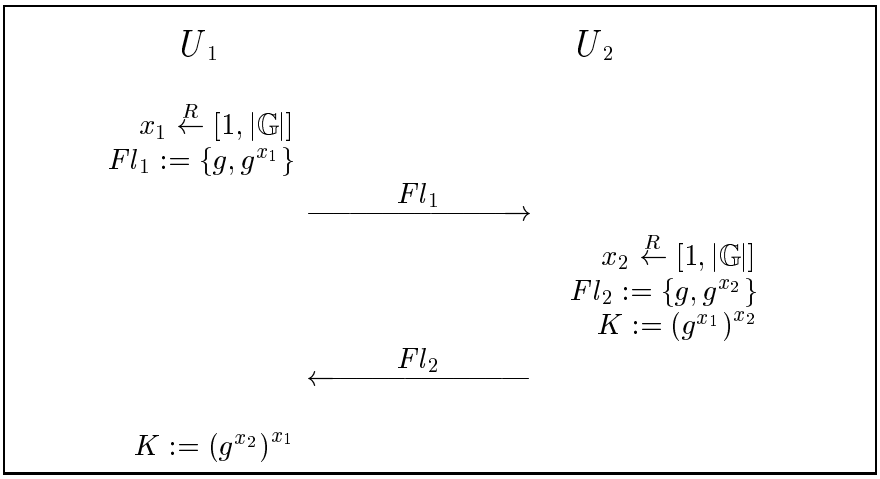

FIGURE 1. Diffie-Hellman Protocol. An example of an honest execution between U_1 and U_2. The shared secret key K is $g^{x_1 x_2}$.

In its original publication the Diffie-Hellman protocol was designed to protect against a *passive* adversary that only eavesdrops on messages, however when it comes to practical uses a much stronger adversary needs to be considered. In the real-world the adversary has complete control

over all the network communications: it may choose to relay, schedule, inject, alter messages between players; it may also choose to impersonate a player (i.e. man-in-the-middle); and so on. Hence the real-world network connection is folded into the adversary. One way to prevent these active attacks is to add authentication services to the Diffie-Hellman key exchange. We refer to it as authenticated Diffie-Hellman key exchange.

3. Digital Signatures

One way to add authentication services to the Diffie-Hellman key exchange is to have each principal sign the messages he sends out over the wire [GB01]. A signature scheme consists of a triplet of algorithms $(\mathcal{G}, \Sigma, V)$.

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

The fundamental security notion for a signature scheme is that it is computationally infeasible for an adversary to produce a valid forgery σ' with respect to a message m' under a (adaptively) chosen-message attack (CMA).

More formally, the signature scheme is (t, ϵ)-**CMA-secure** if there is no adversary $\mathcal{A}$ which can get a probability greater than ϵ in mounting an existential forgery under a CMA-attack within time t. We denote this probability ϵ as $\mathsf{Succ}^{cma}_{\Sigma}(\mathcal{A})$. An example of signature scheme that both achieves this security notion and allows for an efficient implementation is described in [PS00].

4. Message Authentication Code

Another way to add authentication services to the Diffie-Hellman key exchange is to compute an authentication tag using a symmetric cryptographic primitive called a Message Authentication Code [GB01]. Such MAC primitive consists of tuple of algorithms (MAC.Sgn,MAC.Vf).

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

The fundamental security notion for a MAC is that it is computationally infeasible for an adversary to produce a valid tag μ' with respect to a message m' under a (adaptively) chosen-message attack (CMA).

More formally, a (t, q, L, ϵ)-MAC-forger is a probabilistic Turing machine $\mathcal{F}$ running in time t that requests a MAC.Sgn-oracle up to q messages each of length at most L, and outputs an authenticated message (m', μ'), without having queried the MAC.Sgn-oracle on message m', with probability at least ϵ. We denote this success probability as $\mathsf{Succ}^{\mathsf{cma}}_{\mathsf{mac}}(t, q, L)$. The MAC scheme is (t, q, L, ϵ)-**CMA-secure** if there is no (t, q, L, ϵ)-MAC-forger. An example of MAC that both achieves this security notion and allows for an efficient implementation is the scheme of Bellare et al. [BCK96].

5. Provable Security

Despite the apparent simplicity of adding authentication services many proposed protocols for authenticated Diffie-Hellman key exchange have later found to be flawed [BGH+91, DvOW92, PQ01a] and in some cases the flaws even took years before being discovered. One way to avoid many of the flaws is to make use of the provable security methodology.

In the paradigm of "provable" security one finds a formal model and security definitions for a particular cryptographic task to solve [Poi01b]. For the task of exchanging a session key between players two models have received the most consideration in the literature.

The first model initiated by Bellare and Rogaway modeled the two-party and three-party key distribution [BR93a, BR95]. In this model, the instances of a player are modeled via oracles and the capabilities of the adversary are modeled through queries to these oracles. This model was extended to the authenticated two-party Diffie-Hellman key-exchange by Blake-Wilson et al. [BWJM97, BWM98a] and extended further by Bellare et al. to deal with the password-based authenticated two-party

key-exchange [BPR00]. Our formal treatment of the authenticated group Diffie-Hellman is derived from this later model.

The second formal model is based on the multi-party simulatability technique and was initiated by Bellare, Canetti and Krawczyk [BCK98]. In this model the two-party authenticated Diffie-Hellman key-exchange and the two-party encryption-based key-exchange are considered. Shoup refined this model and showed that under specific conditions the two models are equivalent [Sho99]. Building on the work of Shoup, Steiner recently proposed a formal treatment for the multi-party key agreement [SPW02].

With a formal model in hand one can then find security definitions capturing what it means to securely exchange on a session key among players. The fundamental security goal to achieve is certainly Authenticated Key Exchange (with "implicit" authentication) identified as **AKE**. In AKE, each player is assured that no other player aside from the arbitrary pool of players can learn any information about the session key. Another stronger but also highly desirable goal to achieve is Mutual Authentication identified as **MA**. In MA, each player is assured that its partners (or pool thereof) have actually computed the shared session key.

In the paradigm of "provable" security one then picks a protocol aiming to exchange a session key among players and analyzes it to see whether it satisfies the security definitions. One exhibits a proof of security to show that the scheme actually achieves the security goals. In the formalization of Bellare and Rogaway a proof is a *direct* reduction from the security of the scheme to an underlying intractability assumption (see Section 7). A reduction is a successful algorithm for the intractability assumption that uses the adversary of the scheme as a subroutine. On the other hand, a proof of security in the formalization of Bellare, Canetti and Krawczyk is a simulation argument.

6. Practical Security

In the 80's cryptographers were only looking for the minimal assumptions to solve cryptographic problems and to reach this aim used asymptotic notions from complexity theory such as *polynomial reduction* and *negligeable probability* [GGM86, GM84, GMR88]. During this period the reductions were algorithms for the underlying "hard" problem that takes several years to succeed however it soon turned out that such proofs

were not meaningful enough to validate cryptographic schemes in practice. Using security parameters suitable for practice the provably secure schemes could still be broken within a few hours.

Cryptographers have since begun to quantify their reductions to see how much security of the underlying "hard" problem was actually injected into the cryptographic schemes. Reductions carried out in this way provide an exact measurement of the security of a scheme as a function of both the probability of breaking the "hard" problem and the number of queries requested by the adversary [BR96, Poi01b]. Furthermore, cryptographers reach "practical" security when their reduction is an algorithm for a scheme that succeeds with almost the same probability as the best algorithm breaking the "hard" problem while running in the same amout of time [Poi01a]. "Practical" security is particularly relevant to practice since smaller security parameters can be used for the same level of security.

Our treatment of the authenticated group Diffie-Hellman is in the framework of exact security, however, we will in the thesis do our best to reach "practical" security. In order to quantify our reductions we define the advantage $\mathsf{Adv}^{ake}(\mathcal{A})$ that a (computationally bounded) adversary $\mathcal{A}$ will defeat the **AKE** goal of a protocol. The advantage is twice the probability that $\mathcal{A}$ defeats the **AKE** goal minus one. To defeat AKE security means for $\mathcal{A}$ distinguishing the session key from a random value. Hence, $\mathcal{A}$ can trivially defeat AKE with probability $1/2$, multiplying by two and substracting one rescales the probability. In order to quantify our reductions we will also need the probability $\mathsf{Succ}^{ma}(\mathcal{A})$ that adversary $\mathcal{A}$ will defeat the **MA** goal of a protocol. To defeat the MA security for $\mathcal{A}$ means impersonating a player.

7. Intractability Assumptions

The security of the Diffie-Hellman protocol was in its original publication based on the hardness of a certain computational problem, the so-called *Computational Diffie-Hellman assumption* (CDH) [DH76]. The intractability assumption derived from this problem states that the Diffie-Hellman secret $g^{x_1x_2}$ can not be computed by an adversary that only eavesdrops on the flows g^{x_1}, g^{x_2} exchanged by the two players during the Diffie-Hellman protocol.

It later turned out that even if the adversary could not compute the entire Diffie-Hellman secret he could still be able to compute some valuable information about it. And therefore that to securely design protocols for Diffie-Hellman key-exchange one has to rely on either the decisional

Diffie-Hellman problem [BWJM97, BWM98b, Bon98, Sho99] or ideal objects as we will see in the next section.

The generalized version of the Diffie-Hellman problems was first used by Steiner et al. to design protocols for group Diffie-Hellman key exchange [STW96]. The *Computational Group Diffie-Hellman problem* (G-CDH) and the *Decisional Group Diffie-Hellman problem* (G-DDH) allowed us to prove the security of protocols for authenticated group Diffie-Hellman key exchange [BCPQ01, BCP01, BCP02b, BCP02c].

7.1. Diffie-Hellman Assumptions

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

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

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

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

7.2. Group Diffie-Hellman Assumptions

Let $\mathbb{G} =< g >$ be a cyclic group of prime order q and n an integer. Let I_n be $\{1, \ldots, n\}$, $\mathcal{P}(I_n)$ be the set of all subsets of I_n and Γ be a subset of $\mathcal{P}(I_n)$ such that $I_n \notin \Gamma$. We define the *Group Diffie-Hellman distribution* relative to Γ as:

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

If $n = 2$, this $\mathsf{G\text{-}DH}_\Gamma$ distribution is the DDH distribution, and if $\Gamma = \mathcal{P}(I) \backslash \{I_n\}$ this $\mathsf{G\text{-}DH}_\Gamma$ distribution is the generalized Diffie-Hellman distribution [BCP02d, Bon98, NR97, STW96].

A (T, ϵ)-$\mathsf{G\text{-}DDH}_\Gamma$-distinguisher for $\mathbb{G}$ is a probabilistic Turing machine Δ running in time T that given an element X from either $\mathsf{G\text{-}DH}^{\$}_\Gamma$, where

the tuple of $\mathsf{G\text{-}DH}_\Gamma$ is appended a random element g^r, or $\mathsf{G\text{-}DH}^\star_\Gamma$, where the tuple is appended $g^{x_1 \cdots x_n}$, outputs 0 or 1 such that:

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

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

LEMMA 1. The DDH assumption implies the G-DDH assumption. The proof and the exact security reduction appear in [BCP02d].

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

LEMMA 2. The CDH and DDH assumptions imply the G-CDH assumption. The proof and the exact security reduction appear in [BCP02d].

7.3. Random Self-Reducibility

In a *prime-order* group $\mathbb{G}$, the CDH, G-CDH, DDH and G-DDH are all random self-reducible problems [NR97]. Informally, this property means that solving the problem on any original instance $\mathcal{D}$ can be reduced to solving the problem on a random instance $\mathcal{D}'$. This requires an efficient way to generate the random instances $\mathcal{D}'$ from the original instance $\mathcal{D}$ and an efficient way to compute the solution to the problem on $\mathcal{D}'$ from the solution to the problem on $\mathcal{D}$.

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

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

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

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

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

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

FIGURE 2. Hash-oracle simulation.

8. Ideal Objects

Provable security is unfortunately achieved at the cost of a loss of efficiency in terms of computation, communication, integration and engineers sometimes face environments where even a slight loss can not be tolerated. One way to achieve both provable security and efficiency is to analyze cryptographic algorithms in an ideal model of computation wherein concrete objects are identified to real ones. Such an analysis is useful to avoid attacks independent of the actual implementation of the ideal object. In the litterature this class of attacks is often referred to as *generic attacks*.

8.1. The Ideal-Hash Model

In the ideal hash model, also called the "random oracle model" [BR93b], cryptographic hash functions are viewed as random functions with the appropriate range. Security proofs in this model identify the hash functions as oracles which produce a truly random value for each new query and identical answers if the same query is asked twice (see Figure 2). Later, in practice, the random functions are instantiated using specific functions derived from standard cryptographic hash functions like SHA or MD5.

Analyses in this idealized model have been successful in ensuring security guarantees of numerous cryptographic algorithms provided that the hash function has no weaknesses [BR96, Ble98, CGH98, FOPS01, PS00, Sho01]. Security proofs in this model are superior to those provided by *heuristic* protocol designs although they do not provide the same security guarantees as those in the standard model [CS98, CS99].

8.2. The Ideal-Cipher Model

Security proofs in the ideal-cipher model see a (keyed) cipher as a family of random permutations which are queried via an oracle to encrypt and decrypt [BPR00]. The oracle produces a truly random value for each new query and identical answers if the same query is asked twice; furthermore, for each key, the injectivity is satisfied. In practice, the ideal-cipher is instantiated using deterministic symmetric encryption function such as AES [NIS00]. Although these encryption functions have been designed with different criteria from being an ideal-cipher, AES has been designed with unpredictability in mind.

Security proofs in the ideal-cipher model are superior to those provided by *ad-hoc* protocol designs although they do not provide the same security guarantees as those in the random oracle and the standard models. However, the ideal-cipher model allows for "elegant" and more efficient protocols. The ideal-cipher model has recently been used by Bellare et al. to ensure security guarantees of Diffie-Hellman key exchange when parties authenticated one another using a shared password [BPR00]. We also recently used it to lay out the problem of password-based authenticated group Diffie-Hellman key exchange on firmer foundation [BCP02c].

CHAPTER 2

Group Diffie-Hellman Key Exchange

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

1. Introduction

Group Diffie-Hellman schemes for Authenticated Key Exchange are designed to provide a pool of players communicating over an open network and each holding a pair of matching public/private keys with a shared secret key which may later be used to achieve some cryptographic goals like multicast message confidentiality or multicast data integrity. In this chapter we consider the scenario where the group membership is static and known in advance. At startup the participants would like to engage in a conversation at the end of which they have established a session key.

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

This chapter provides the first tier in the treatment of the group Diffie-Hellman key exchange problem using public/private key pairs. We first present a formal model to help manage the complexity of definitions and proofs for the authenticated group Diffie-Hellman key exchange. A model where a process controlled by a player running on some machine is modeled as an instance of the player, the various types of attacks are modeled by queries to these instances and the security of the session key is modeled through semantic security. Moreover, in order to be correctly formalized, the intuition behind mutual authentication requires definitions of session IDS and partner IDS.

Second, we define in this model the execution of a protocol modified from [STW96], we refer to it as AKE1, and show that AKE1 can be proven secure under reasonable and well-defined intractability assumptions. Third, we present a generic transformation for turning an AKE protocol into a protocol that provides MA and justify its security under reasonable and well-defined intractability assumptions.

The remainder of this chapter is organized as follows. In the following section we first review the related work. The chapter continues with a description of our model of a distributed environment in Section 3 and gives the precise security definitions that should be satisfied by a group Diffie-Hellman scheme in Section 4. Section 5 presents the protocol AKE1 and justifies its security in the random oracle model. Section 6 turns AKE1 into a protocol that provides MA and justifies its security in the random oracle model.

2. Related Work

Over the years, several papers [AST00, BW98, BD95, SSDW88, ITW82, JV96, Per99, STW96, Tze00] have attempted to extend the well-known Diffie-Hellman key exchange [DH76] to the multi-party setting. These protocols meet a variety of performance attributes but only exhibit an informal analysis showing that they achieve the desired security goals. Some of the papers exhibited an *ad-hoc* analysis for the security of their schemes and some of these schemes have later been found to be flawed [JV96, PQ01a]. Other papers only provided *heuristic evidence* of security without quantifying it. The remaining schemes assume authenticated links and thus do not consider the authentication as part of the protocol design.

The work of Ateniese et al. [AST00] is of particular interest since it identifies the fundamental and additional desirable security goals of authenticated group Diffie-Hellman key exchange. The authors offer provably

secure authenticated protocols and sketch informal proofs that their protocols achieve these goals. Unfortunately these protocols have also since been found to be flawed [PQ01a].

3. Model

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

3.1. Players

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

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

3.2. Long-Lived Keys

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

3.3. Session IDS

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

3.4. Accepting and Terminating

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

3.5. Security Model

3.5.1. *Oracle Queries*

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

- $\mathsf{Send}(\Pi_U^s, m)$: This query models adversary $\mathcal{A}$ sending messages to instances of players. The adversary $\mathcal{A}$ gets back from his query the response which oracle Π_U^s would have generated in processing message m. If oracle Π_U^s has not yet terminated and the execution of protocol P leads to accepting, variables SIDS are updated. A query of the form $\mathsf{Send}(\Pi_U^s,$ "`start`") initiates an execution of P.
- $\mathsf{Reveal}(\Pi_U^s)$: This query models the attacks resulting in the session key being revealed. The Reveal query is only available to adversary $\mathcal{A}$ if oracle Π_U^s has accepted. The Reveal-query unconditionally forces Π_U^s to release SK which otherwise is hidden to the adversary.
- $\mathsf{Corrupt}(U)$: This query models the attacks resulting in the player U's LL-key being revealed. Adversary $\mathcal{A}$ gets back LL_U but does not get the internal data of any instances of U executing P.
- $\mathsf{Test}(\Pi_U^s)$: This query models the semantic security of the session key SK, namely the following game, denoted by $\mathbf{Game}^{ake}(\mathcal{A}, P)$, between adversary $\mathcal{A}$ and the oracles Π_U^s involved in the executions of P. During the game, $\mathcal{A}$ can ask any of the above queries, and once, asks a Test-query. Then, one flips a coin b and returns SK if $b = 1$ or a random string if $b = 0$. At the end of the game, adversary $\mathcal{A}$ outputs a bit b' and *wins* the game if

$b = b'$. The Test-query is asked only once and is only available if Π_U^s is Fresh (see Section 4).

3.5.2. *Executing the Protocol*

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

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

3.5.3. *Discussion*

The group Diffie-Hellman-like protocols [AST00, BW98, BD95, SSDW88, ITW82, Per99, STW96] are generally specified using the broadcast communication primitive; the broadcast primitive allows a player to send messages to an arbitrary pool of players in a single round. However such a communication convention is irrelevant to our notions of security; for example, one can always turn a broadcast-based protocol P into a protocol P' which sends only one message in each round and which still meets our definitions of security as long as P does. The group Diffie-Hellman-like protocols also employ a different connectivity graph (e.g, ring or tree) to route messages among players. The connectivity graph allows the protocols to meet specific performance attributes. However the way the messages are routed among players does not impact our security definitions; one can always turn a protocol P into a protocol P' that differs only in its message routing.

4. Definitions

In this section we present the definitions that should be satisfied by a group Diffie-Hellman scheme and describe what *breaking* a group Diffie-Hellman scheme means. We uniquely define the partnering from the

session IDS and, thus, it is publicly available to the adversary[1]. We present each definition in a systematic way: we give an intuition and then formalize it.

Let's also recall that a function $\epsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that for all $k > k_c$, $\epsilon(k) < k^{-c}$.

4.1. Partnering using SIDS

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

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

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

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

with :

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

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

We complete in polynomial time (in $|V|$) the graph G_{SIDS} to obtain the graph of partnering : $G_{PIDS} = (V', E')$, where $V' = V$ and $E' = \{(\Pi_i^s, \Pi_j^t) \ : \ \Pi_i^s \rightsquigarrow \Pi_j^t\}$ (see [CLR90] for graph algorithms), and then define the partner IDS for oracle Π_i^s as:

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

[1]In the definition of partnering, we do not require that the session key SK computed by partnered oracles be the same since it can easily be proven that the probability that **partnered** oracles come up with different SK is negligible (see Section 6.4.1).

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

4.2. Security Notions

4.2.1. *Forward-Secrecy*

The notion of forward-secrecy entails that the loss of a LL-key does not compromise the semantic security of previously-distributed session keys. This definition captures *weak-corruption* attacks where the adversary gets the LL-key and not the random bits (i.e. internal data) used by a process. We refer to this definition of forward-secrecy as weak forward-secrecy. In Chapter 4, we will define a stronger notion of forward-secrecy.

4.2.2. *Freshness*

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

4.2.3. *AKE Security*

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

4.2.4. *Authentication Security*

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

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

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

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

4.3. Adversary's Resources

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

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

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

5. An Authenticated Group Diffie-Hellman Scheme

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

5.1. Preliminaries

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

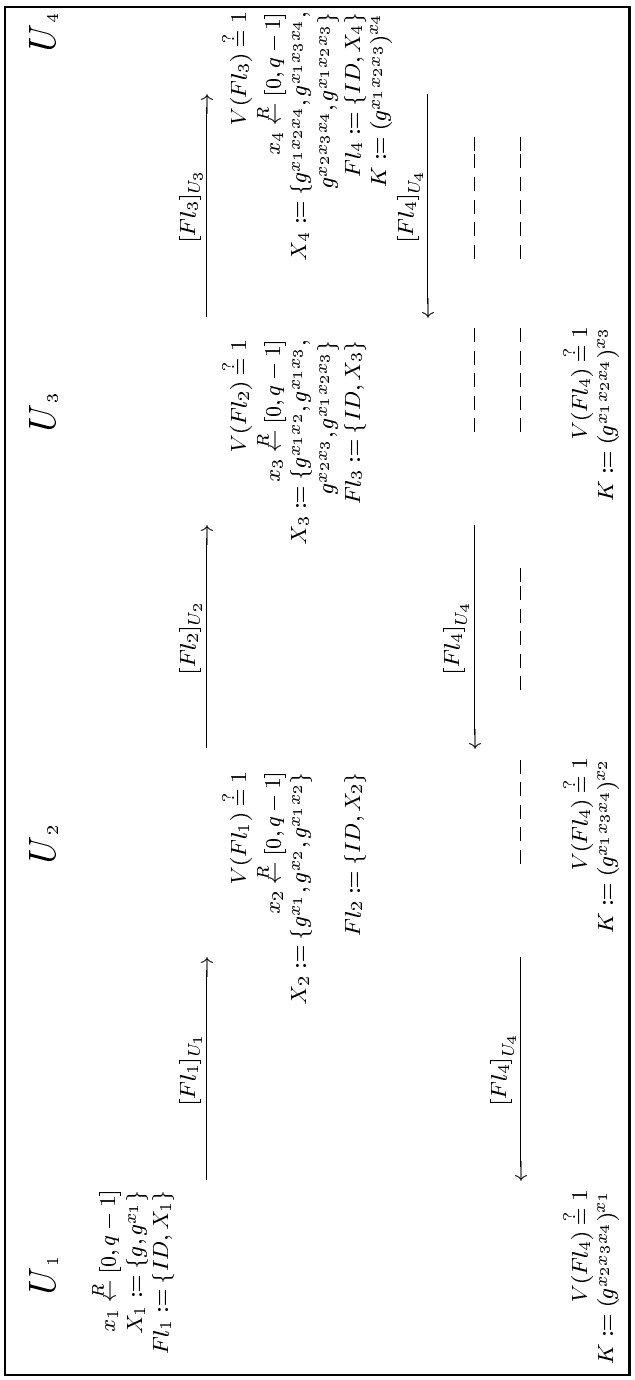

FIGURE 1. Protocol AKE1. An example of a honest execution with 4 players: $ID = \{U_1, U_2, U_3, U_4\}$. The shared session key SK is $sk = \mathcal{H}(U_1, U_2, U_3, U_4, Fl_4, g^{x_1 x_2 x_3 x_4})$.

is available to adversary $\mathcal{A}$; the adversary can submit an arbitrarily long bit string and obtain the value of $\mathcal{H}(m)$.

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

5.2. Scheme

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

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

5.3. Theorem of Security

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

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

$$\mathsf{Adv}_P^{ake}(t, q_{se}, q_h) \quad \leq$$

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

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

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

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

5.4. Proof of the Theorem

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

5.4.1. *Forger $\mathcal{F}$*

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

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

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

other reduction. When $\mathcal{A}$ makes a Hash-query, $\mathcal{F}$ answers the query as depicted on Figure 2 in Chapter 1.

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

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

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

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

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

5.4.2. *G-CDH$_\Gamma$-attacker* Δ

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

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

When $\mathcal{A}$ makes a Send-query to some instance of player U_i, then Δ increments c_i and proceeds as in protocol P using a random value. However if $c_i = u_i$ and m is the flow corresponding to the instance $\mathcal{D}$, Δ answers using the elements from the instance $\mathcal{D}$. When $\mathcal{A}$ makes a Corrupt-query, Δ answers in a straightforward way. When $\mathcal{A}$ makes a Hash-query, $\mathcal{F}$ answers the query as depicted on Figure 2 in Chapter 1. When $\mathcal{A}$ makes

a Reveal-query, Δ answers in straightforward way. However, if the session key has to be constructed from the instance $\mathcal{D}$, Δ halts and outputs "`Fail`". When $\mathcal{A}$ makes the Test-query, Δ answers with a random string.

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

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

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

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

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

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

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

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

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

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

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

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

□

5.5. Practical Security

The quality of the reduction measures how much security of the G-CDH$_\Gamma$ and the signature scheme is injected into AKE1. We view q_{se} as an upper bound on the number of queries we are willing to allow (e.g., $q_{se} = 2^{30}$ and $q_h = 2^{60}$) and n as the number of participants involved in the execution of AKE1 (e.g., current scientific collaborations involve up to 20 participants). Moreover, because of the network latency and computation cost, the practicability of AKE1 becomes an issue with groups larger than 40 members operating in a wide-area environment [ACTT01].

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

The reduction could be improved using a technique of Shoup [Sho97]. Shoup's technique runs two attackers, similar to the one above, in parallel on two different instances obtained by random self-reducibility, and a common value will appear in the $\mathcal{H}$-list of the attackers with overwhelming probability and thus leads to the right solution for G-CDH$_\Gamma$.

In the next chapters, we will not use Shoup's technique to carry out our proofs but a technique similar to the one used by Coron [Cor00]. We use the random self-reducibility of the group Diffie-Hellman problems to generate many instances $\mathcal{D}'$ from $\mathcal{D}$ such that all the $\mathcal{D}'$ lie in the same distribution as $\mathcal{D}$. Such instances are randomly used. But then, the resulting session key will be unknown. Therefore, the reduction will work if all the Reveal-queries are asked for known session keys, but the Test-query is asked to one involving an instance $\mathcal{D}'$. By correctly tuning the probability of using a $\mathcal{D}'$ instance or not, one can slightly improve the efficiency of the reduction[2]. If the session key is simply fixed to $g^{x_1 \cdots x_n}$

[2]However, such a proof gets complicated when one adds in the concern of forward-secrecy. Instead the ideas in the proof of Section 5.4 can easily be extended to show that AKE1 guarantees forward-secrecy.

the proof could additionaly be carried out in the standard model rather than the random one as done in Chapter 4.

6. Adding Authentication

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

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

6.1. Approach

The well-known approach uses the shared session key to construct a simple "authenticator" for the other parties. However, one has to be careful in the details and this is a common "error" in the design of authentication protocols. Actually the protocols offered by Ateniese et al. [AST00] are seen insecure under our definitions since the "authenticator" is computed as the hash of the session key sk and sk is the same as the final session key SK. The adversary learns some information about the session key sk – the hash of sk – and can use it to distinguish SK from a session key selected at random from session-key space **SK**. Therefore these protocols sacrifice the security goal that a protocol establishes a semantically secure session key.

6.2. Transformations

The transformation AddPPsA (adding player-to-players authentication) for player U consists of adding to protocol P one more round in such a way that the partners of U are convinced they share sk with U. As an example, on Figure 2 player U_n sends out $\mathcal{H}(sk, n)$.

More formally the transformation AddPPsA works as follows. Suppose that in protocol P player U_n has accepted holding $sk_{U_n}, sid_{U_n}, pid_{U_n}$ and has terminated. In protocol $P' = \text{AddPPs}(P)$, U_n sends out one additional flow $auth_{U_n} = \mathcal{H}(sk_{U_n}, n)$, accepts holding $sk'_{U_n} = \mathcal{H}(sk_{U_n}, 0)$,

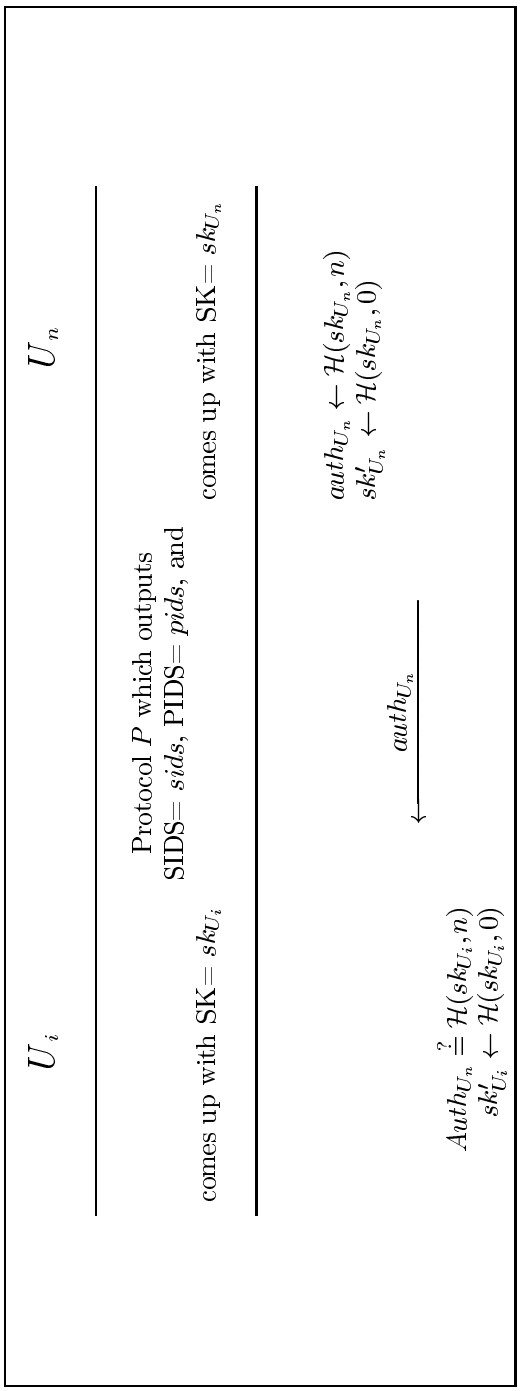

FIGURE 2. Transformation $P' = \text{AddPPsA}(P)$. The shared session key SK is $sk' = \mathcal{H}(sk, 0)$, SIDS and PIDS are unchanged.

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

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

6.3. Theorem of Security

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

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

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

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

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

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

6.4. Proof of the Theorem

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

advantage $\mathsf{Adv}_P^{ake}(t', q_{se}, q_h)$ in breaking the AKE security of protocol P within time t'.

6.4.1. *Disrupt Partnering*

We are not concerned with partnered oracles coming up with different session keys, since our definition of partnering implies the oracles have exchanged *exactly* the same flows. We also note that the probability that two instances of a given player come to be partnered is negligible; in fact, it would mean they have chosen the same random value in the protocols, which occurs with probability $\mathcal{O}(\frac{q_{se}^2}{2^k})$.

6.4.2. *AKE break*

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

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

The oracle queries made by $\mathcal{A}$ to $\mathcal{B}$ are relayed by $\mathcal{B}$ and the answers are subsequently returned to $\mathcal{A}$. However $\mathcal{B}$'s answers to Reveal and Test-queries go through the Hash-oracle to be padded with "0" before being returned to $\mathcal{A}$. The Hash-queries are answered as depicted in Figure 2 in Chapter 1.

In the ideal hash model, in which $\mathcal{H}$ is seen as a random function, $\mathcal{A}$ can not get any advantage in correctly guessing the bit involved in the Test-query without having made a query of the form $\mathcal{H}(sk, 0)$. So

$$\Pr\left[\mathcal{A}\text{asks}(sk, 0)\right] \geq \mathsf{Adv}_P^{ake}(\mathcal{A}) \geq \epsilon.$$

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

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

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

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

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

6.4.3. *MA break*

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

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

The oracle queries made by $\mathcal{A}$ to $\mathcal{B}$ are relayed by $\mathcal{B}$ and the answers are subsequently returned to $\mathcal{A}$. However $\mathcal{B}$'s answers to Reveal and Test-queries go through the Hash-oracle to be padded with "0" before being returned to $\mathcal{A}$. The Hash-queries are answered as usual Figure 2 in Chapter 1.

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

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

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

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

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

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

□

6.5. Practical Security

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

7. Conclusion

In this chapter we presented a model for the group Diffie-Hellman key exchange problem derived from the model of Bellare et al. [BPR00]. Some specific features of our approach that were introduced to deal with the Diffie-Hellman key exchange in the multi-party setting are: defining the notion of session IDS to be a set of session ID, defining the notion of partnering to be a graph of partner ID. Addressed in detail in this chapter were two security goals of the group Diffie-Hellman key exchange: the authenticated key exchange and the mutual authentication. For each we presented a definition, a protocol and a security proof in the ideal hash model that the protocol meets its goals. This chapter provided the first formal treatment of the authenticated group Diffie-Hellman key exchange problem.

CHAPTER 3

Dynamic Group Diffie-Hellman Key-Exchange

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

1. Introduction

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

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

The fundamental security goal for a group Diffie-Hellman scheme to achieve is Authenticated Key Exchange (with "implicit" authentication) identified as AKE. In AKE, each player is assured that no other player

aside from the arbitrary pool of players can learn the session key. Another stronger highly desirable goal for a group Diffie-Hellman scheme to provide is Mutual Authentication (MA). In MA, each player is assured that only its partners actually have possession of the distributed session key. With these security goals in hand, the security of a group Diffie-Hellman scheme can then be analyzed to see how it meets the definitions.

This chapter provides the second tier in the formal treatment of the group Diffie-Hellman key exchange problem using public/private keys. We present the first formal model to help manage the complexity of definitions and proofs for the authenticated group Diffie-Hellman key exchange when the group membership is *dynamic*. This model is equipped with some notions of dynamicity in the group membership where the various types of attacks are modeled by queries to the players. This model does not yet encompass attacks involving multiple player's instances activated concurrently and simultaneously by the adversary. Also, in order to be correctly formalized, the intuition behind mutual authentication requires cumbersome definitions of session IDS and partner IDS which may be skipped at the first reading.

We start with the model and definitions introduced in the previous chapter and extend them to deal with the authenticated *dynamic* group Diffie-Hellman key exchange. We define the partnering, freshness of session key and measures of security for AKE. In this model we define the execution of a protocol, we refer to it as AKE1, modified from [AST00] and show that it can be proven secure under reasonable and well-defined intractability assumptions.

The chapter is organized as follows. In the remainder of this section we summarize the related work. In Section 3 we define our security model. We use it in Section 4 to define the security definitions that should be satisfied by a group Diffie-Hellman scheme. We present the AKE1 protocol in Section 5 and justify its security in the random oracle model in Section 6. Finally in Section 7 we briefly deal with MA in the random oracle model.

2. Related Work

Many group Diffie-Hellman protocols [AST00, BW98, BD95, ITW82, JV96, SSDW88, STW96, Tze00] aim to distribute a session key among the multicast group members for a scenario in which the membership is *static* and known in advance. However these protocols are not well-suited for a scenario in which members join and leave the multicast

group at a relatively high rate. Fortunately, several papers have shown how to extend the protocols [AST00, KPT00, KPT01, STW00] and one extension is the system offered in [ACTT01]. However these protocols, and this existing system, are based on or use an informal approach and do not rely on proofs of security. The protocol presented in [AST00] has been found to be flawed in [PQ01a] and the other papers assume authenticated links, or more specifically do not consider the AKE and MA goals as part of the protocols. These goals need to be addressed separately.

3. Model

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

3.1. Players

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

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

3.2. Long-Lived Keys

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

3.3. Generic Group Diffie-Hellman Schemes

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

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

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

3.4. Session IDS

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

3.5. Accepting and Terminating

A player U accepts when it has enough information to compute a session key SK_U. At any time a player U who is in "expecting state" can accept and it accepts at most once in executing an operation. As soon as U accepts in executing an operation, SK and SIDS are defined. Having accepted, U does not yet terminate this execution. Player U may want

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

3.6. Security Model

3.6.1. *Queries*

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

- Setup($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *setup* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated and are thus in a "stand by" state.. $\mathcal{A}$ gets back from the first player U in $\mathcal{J}$ the flow initiating the *setup* execution. Other players are aware of the *setup* and move to an "expecting state" but do not reply to any messages.
- Remove($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *remove* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated. $\mathcal{A}$ gets back from the group controller U_{GC} the flow initiating the *remove* execution. Other players are aware of the *remove* operation but do not reply. They move from a "stand by" state to an "expecting state".
- Join($\mathcal{J}$): This query models adversary $\mathcal{A}$ initiating the *join* operation. The query is only available to adversary $\mathcal{A}$ if all the players in $\mathcal{U}$ have terminated. $\mathcal{A}$ gets back from the group controller U_{GC} the flow initiating the *join* execution. Other players are aware of the *join* operation but do not reply. They move from a "stand by" state to an "expecting state".
- Send(U, m): This query models adversary $\mathcal{A}$ sending a message to a player. The adversary $\mathcal{A}$ gets back from his query the response which player U would have generated in processing message m (this could be the empty string if the message is incorrect or unexpected). If player U has not yet terminated

and the execution of protocol P leads to accepting, variable SIDS(U) is updated as explained above.

- Reveal(U): This query models the attacks resulting in the misuse of the session key, which may then be revealed. The query is only available to adversary $\mathcal{A}$ if player U has accepted. The Reveal-query unconditionally forces player U to release SK_U which is otherwise hidden to the adversary.
- Corrupt(U): This query models the attacks resulting in the player U's LL-key been revealed. $\mathcal{A}$ gets back LL_U but does not get any internal data of U executing P.
- Test(U): This query models the semantic security of the session key SK, namely the following game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ between adversary $\mathcal{A}$ and the players U involved in an execution of the protocol P. The Test-query is only available if U is **Fresh** (see Section 4). In the game $\mathcal{A}$ asks any of the above queries, however, it can only ask a Test-query once. Then, one flips a coin b and returns sk_U if $b = 1$ or a random string if $b = 0$. At the end of the game, adversary $\mathcal{A}$ outputs a bit b' and *wins* the game if $b = b'$.

3.6.2. *Executing the Game*

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

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

4. Definitions

In this section we present the definitions that should be satisfied by a group Diffie-Hellman scheme. We define the partnering from the session

IDS and use it to define security measurements that an adversary will defeat the security goals. We also recall that a function $\varepsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that for all $k > k_c$, $\varepsilon(k) < k^{-c}$.

4.1. Partnering using SIDS

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

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

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

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

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

We complete in polynomial time (in $|V|$) the graph G_{SIDS} to obtain the graph of partnering: $G_{PIDS} = (V', E')$, where $V' = V$ and $E' = \{(U_i, U_j) \ : \ U_i \leftrightsquigarrow U_j\}$, and then define the partner IDS for oracle U_i as:

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

4.2. Security Notions

4.2.1. *Freshness*

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

4.2.2. *Forward-Secrecy*

The notion of forward-secrecy entails that the loss of a LL-key does not compromise the semantic security of previously-distributed session keys. This definition captures *weak-corruption* attacks where the adversary gets the LL-key and not the random bits (i.e. internal data) used by a process. We refer to this definition of forward-secrecy as weak forward-secrecy. In Chapter 4, we will define a stronger notion of forward-secrecy.

4.2.3. *AKE Security*

In an execution of P, we say an adversary $\mathcal{A}$ *wins* if she asks a single Test-query to a **Fresh** player U and correctly guesses the bit b used in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$. We denote the **ake advantage** as $\mathsf{Adv}_P^{ake}(\mathcal{A})$; the advantage is taken over all bit tosses. (The advantage is twice the probability that $\mathcal{A}$ will defeat the AKE security goal of the protocol minus one[1].) Protocol P is an **$\mathcal{A}$-secure AKE** if $\mathsf{Adv}_P^{ake}(\mathcal{A})$ is negligible.

4.2.4. *MA Security*

In an execution of P, we say adversary $\mathcal{A}$ violates mutual authentication (MA) if there exists an operation execution wherein a player U terminates holding $\mathrm{SIDS}(U)$, $\mathrm{PIDS}(U)$ and $|\mathrm{PIDS}(U)| \neq |\mathcal{I}| - 1$. We denote the **ma success** as $\mathsf{Succ}_P^{ma}(\mathcal{A})$ and say protocol P is an **$\mathcal{A}$-secure MA** if $\mathsf{Succ}_P^{ma}(\mathcal{A})$ is negligible.

Therefore to deal with mutual authentication, we consider a new game, we denote $\mathbf{Game}^{ma}(\mathcal{A}, P)$, wherein the adversary exactly plays the same way as in the game $\mathbf{Game}^{ake}(\mathcal{A}, P)$ with the same player accesses but with a different goal: to violate the mutual authentication.

4.3. Adversary's Resources

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

- T-time of computing;
- $q_s, q_r, q_c, Q_S, Q_R, Q_J$ numbers of Send, Reveal, Corrupt, Setup, Remove and Join queries the adversary $\mathcal{A}$ respectively makes.

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

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

5. An Authenticated Dynamic Group Diffie-Hellman Scheme

In the following theorem and proof we assume the random oracle model [BR93b] and denote $\mathcal{H}$ a hash function from $\{0,1\}^*$ to $\{0,1\}^\ell$, where ℓ is a security parameter. The session-key space **SK** associated to this protocol is $\{0,1\}^\ell$ equipped with a uniform distribution. The arithmetic is in a finite cyclic group $\mathbb{G} =< g >$ of order a k-bit prime number q and the operation is denoted multiplicatively. This group could be a prime subgroup of $\mathbb{Z}_p^*$, or it could be an (hyper)-elliptic curve based group.

5.1. Scheme

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

Unlike [AST00], this is a protocol wherein the player with the highest-index in $\mathcal{I}$ is the group controller, the flows are signed using the long-lived key LL_U, the names of the players are in the protocol flows, and the session key SK is $sk = \mathcal{H}(\mathcal{I} \| Fl_{max(\mathcal{I})} \| g^{x_1 \cdots x_{\max(\mathcal{I})}})$; $Fl_{max(\mathcal{I})}$ is the down-flow, SIDS and PIDS are appropriately defined.

The notion of index models "pre-existing" relationships among players: for example, it may capture different levels of reliability (i.e. the higher the index is, the more reliable the player). This is also a protocol, unlike [AST00], where the set of values from the down-flow is included in the flows of REMOVE1 and JOIN1, which avoids replay attacks.

5.1.1. *Algorithm SETUP1*

The algorithm consists of two stages: up-flow and down-flow. The multicast group $\mathcal{I}$ is set to $\mathcal{J}$. As illustrated by the example in Figure 1, in the up-flow the player U_i receives a set (Y, Z) of intermediate values, with

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

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

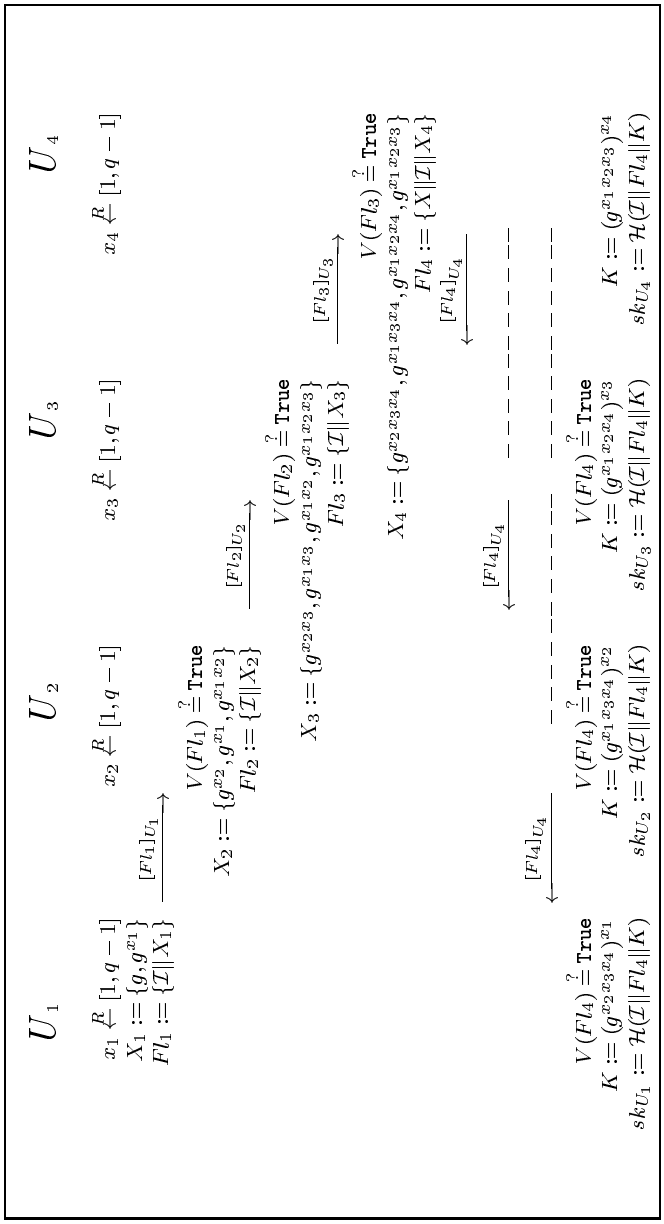

FIGURE 1. Algorithm SETUP1. An example of an honest execution with 4 players: $\mathcal{J} = \{U_1, U_2, U_3, U_4\}$. The multicast group is $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_4 \| g^{x_1 x_2 x_3 x_4})$. The partner IDS for U_1 is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

values

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

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

5.1.2. *Algorithm REMOVE1*

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

5.1.3. *Algorithm JOIN1*

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

5.2. Theorem of Security

THEOREM 3. Let P be the AKE1 protocol, **SK** be the session-key space and $\mathcal{G}$ be the associated LL-key generator. Let $\mathcal{A}$ be an adversary against the AKE security of P within a time bound T, on a multicast group of size s among the n players in $\mathcal{U}$, after $Q = Q_S + Q_J + Q_R$ interactions with the parties, q_s send-queries and q_h hash-queries. Then we have:

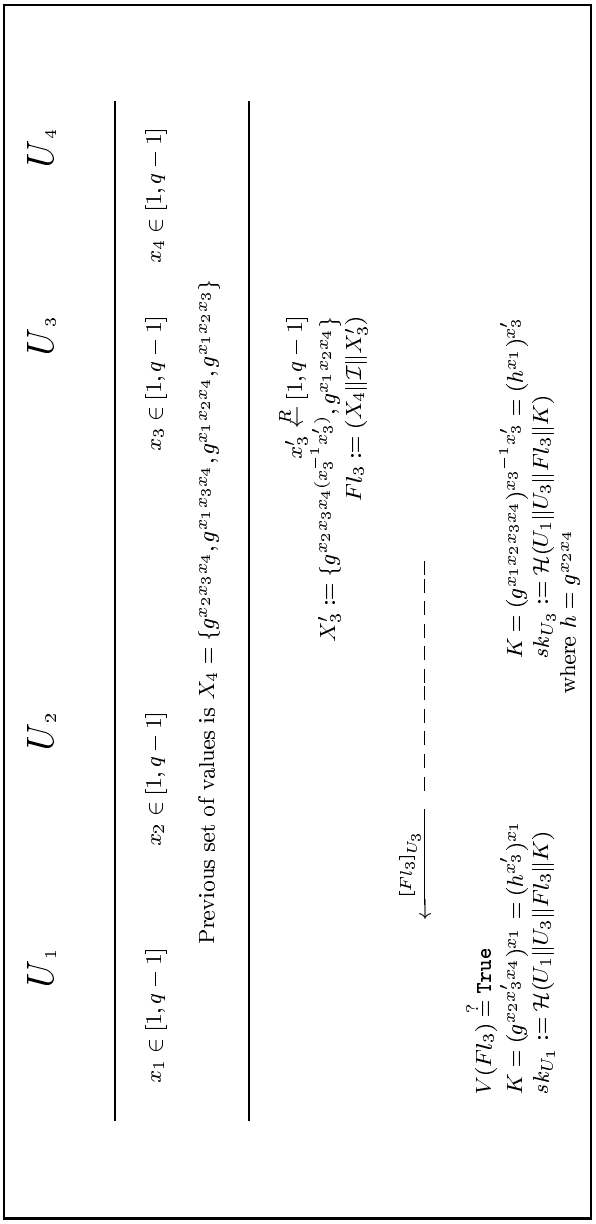

FIGURE 2. Algorithm REMOVE1. An example of an honest execution with 4 players: $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$, $\mathcal{J} = \{U_2, U_4\}$. The new multicast group is $\mathcal{I} = \{U_1, U_3\}$, $U_{GC} = U_3$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_3 \| g^{x_1x_2x'_3x_4})$, the partner IDS for U_1 is $pids_{U_1} = \{U_3\}$, for U_3 is $pids_{U_3} = \{U_1\}$.

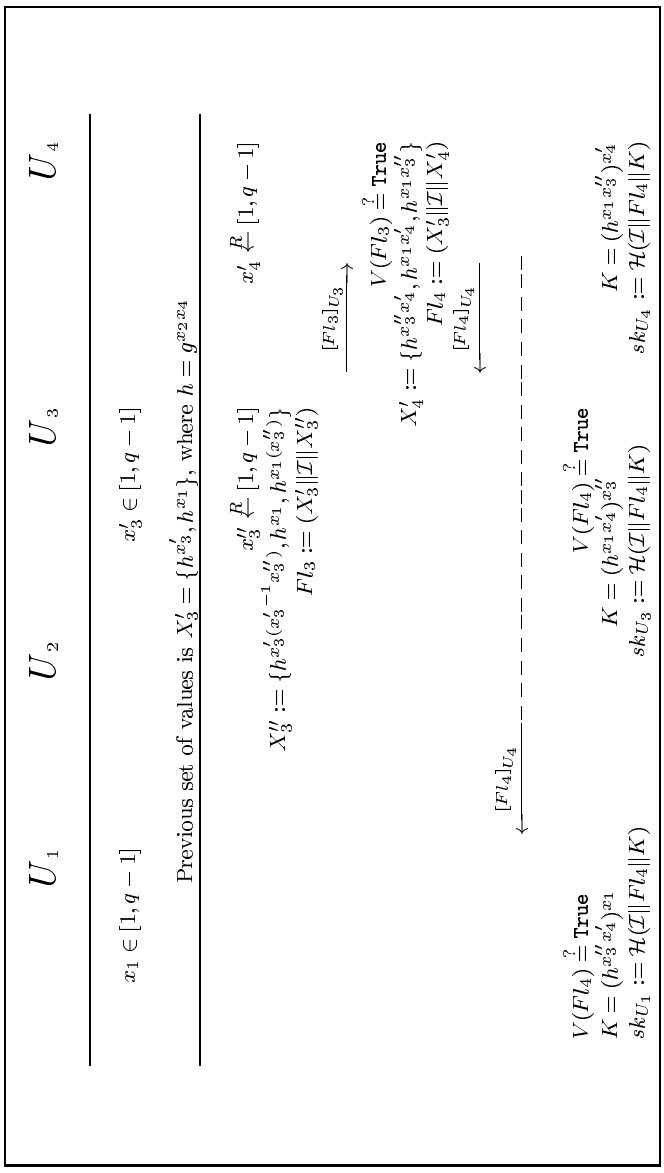

FIGURE 3. Algorithm JOIN1. An example of an honest execution with 4 players: $\mathcal{I} = \{U_1, U_3\}$, $\mathcal{J} = \{U_4\}$ and $U_{GC} = U_3$. The new multicast group is $\mathcal{I} = \{U_1, U_3, U_4\}$ and the shared session key is $sk = \mathcal{H}(\mathcal{I} \| Fl_4 \| g^{x_1 x_2 x_3'' (x_4 x_4')})$ The partner IDS for U_1 is $pids_{U_1} = \{U_3, U_4\}$, for U_3 is $pids_{U_3} = \{U_1, U_4\}$ and for U_4 is $pids_{U_4} = \{U_1, U_3\}$.

$$\mathsf{Adv}_P^{ake}(T, Q, q_s, q_h) \quad \leq$$

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

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

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

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

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

The specific structure of Γ_s (see Figure 4 for Γ_4) makes the simulation perfectly indistinguishable from the adversary point of view if our guesses are all correct. But then, because of the random oracle $\mathcal{H}$, to have any information about the session key the adversary wants to test, she has

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

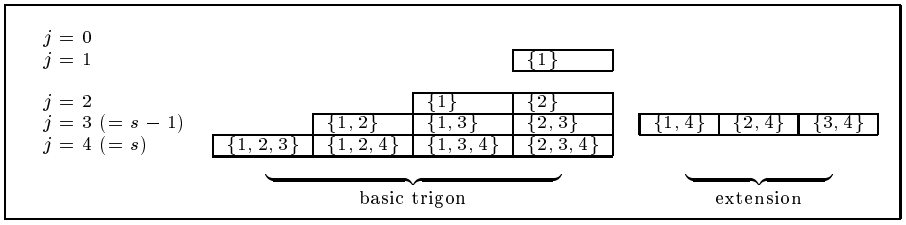

FIGURE 4. Extended Trigon for Γ_4

to have asked for $\mathcal{H}(\mathcal{I}\|Fl_{last}\|K)$, where K is the value we are looking for. Therefore, if the adversary has some advantage in breaking the AKE security, this value K can be found in the list of the queries asked to $\mathcal{H}$. The details of the simulation can be found in Section 6.

6. Proof of the Theorem

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

6.1. Forger $\mathcal{F}$

Let's assume that $\mathcal{A}$ breaks the protocol P by forging, with probability greater than ν, a signature with respect to some player's (public) LL-key (Of course before $\mathcal{A}$ corrupts U). We construct from it a (T'', ε'')-forger $\mathcal{F}$ which outputs a forgery (σ, m) with respect to a given (public) LL-key K_p, produced by $\mathcal{G}_{LL}(1^k)$.

$\mathcal{F}$ receives as input K_p and access to a signing oracle. $\mathcal{F}$ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$ and all U_i. $\mathcal{F}$ picks at random $i_0 \in [1, n]$ and runs $\mathcal{G}_{LL}(1^k)$ to set the players' LL-keys. However for player i_0, $\mathcal{F}$ sets LL_{i_0} to K_p. $\mathcal{F}$ then starts running $\mathcal{A}$ as a subroutine and answers the oracle queries made by $\mathcal{A}$ as explained below. $\mathcal{F}$ also uses a variable $\mathcal{K}$, initially set to $\emptyset$.

The Send-queries, Setup-queries, Join-queries and Remove-queries are answered in a straightforward way, except if the query is made to player U_{i_0}. In this latter case the answers go through the signing oracle, and $\mathcal{F}$ stores in $\mathcal{K}$ the oracle query and the oracle reply. The Reveal-queries and Test-query are answered in a straightforward way as well. Eventually, the Corrupt-query is also answered in a straightforward way, except if the query is made to player U_{i_0}. In this latter case since $\mathcal{F}$ does not know the LL-key K_s for player i_0, $\mathcal{F}$ stops and outputs "`Fail`". But anyway,

no signature forgery occurred before, and so, such an execution can be used with the other reduction. The Hash-query is simulated as depicted on Figure 2 in Chapter 1.

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

The probability that $\mathcal{F}$ outputs a forgery is the probability that $\mathcal{A}$ produces a valid flow by itself multiplied by the probability of "correctly guessing" the value of i_0: $\mathsf{Succ}^{cma}_{\Sigma}(\mathcal{F}) \geq \nu/n$.

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

6.2. G-CDH$_{\Gamma_s}$-attacker Δ

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

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

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

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

The aim of the simulation is to have all the elements of S_s, embedded into the protocol when the adversary $\mathcal{A}$ asks the Test-query. In this

case, $\mathcal{A}$ will not be able to get any information about the value sk of the session key without having previously queried the random hash oracle $\mathcal{H}$ on the Diffie-Hellman secret value $g^{x_1 \cdots x_s}$. Thus, to break the security of P the adversary $\mathcal{A}$ would have to have asked a query of the form $\mathcal{H}(\mathcal{I}, Fl_{last}, g^{x_1 \cdots x_s})$ which as a consequence will be in the list of queries asked to $\mathcal{H}$.

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

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

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

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

We now show that the above simulation and the random self-reducibility of G-CDH allows Δ to answer all the queries until $\mathcal{A}$ asks the Test-query at c_0. Since Δ embeds elements of S_i when a player u_i from $\mathcal{I}_0$ (except u_{i_0}) is added to the group and Δ does not remove it when u_i leaves, each protocol flow consists of a random self-reduction on one line (line

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

0, i.e. S_0 down to line $s-1$, i.e. $_{s-1}$) of the basic trigon. The trigon is illustrated on Figure 4. Thus, Δ can derivate the value sk of the session key from one of the values in the line below (line 1, i.e. S_1 up to line s, i.e. S_s).

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

Formally, the simulator works as follows. Δ provides coin tosses to $\mathcal{G}_{LL}$, $\mathcal{A}$, all U_i and runs $\mathcal{G}_{LL}(1^k)$ to set the player's LL-keys. Δ sets an operation counter c to 0, and two variable $\mathcal{K}$ and $\mathcal{T}$ to $\emptyset$. Δ will use variable $\mathcal{K}$ to store (all) the random exponents involved in the game $\mathbf{Game}^{ake}(\mathcal{A},P)$ and variable $\mathcal{T}$ to store which exponents of instance $\mathcal{D}$ have been injected in the game so far. Then, Δ starts running $\mathcal{A}$ as a subroutine and answers the queries as depicted in Figure 7.

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

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

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

Another technical difficulty may show up if the adversary $\mathcal{A}$ does not output the bit b' right away after asking the Test-query and keeps playing the game for more rounds. Indeed, the session key is derived from the

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

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

G-CDH one is looking for. And forthcoming session keys would as well. Therefore, Δ would be unable to answer Reveal-queries. Δ has to reduce the number of exponents taken from the instance $\mathcal{D}$: basically, we go down in the basic trigon while players join the group. Until having involved $s-1$ exponents from instance $\mathcal{D}$. At the very last broadcast before the Test-query, we inject the last exponent (x_s, s being the size

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

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

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

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

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

Indeed, if one assumes that the adversary $\mathcal{A}$ has made a Test-query and has terminated outputing a bit b' at some point. Δ then looks in the $\mathcal{H}$-list to see if queries of the form $\mathsf{Hash}(\mathcal{I}_0 \| Fl_0 \| *)$ have been asked (Fl_0

is the flow broadcast in the execution of the c_0-th operation). If so, Δ chooses at random one of them and then looks in variable $\mathcal{K}$ (thanks to the flow Fl_0) for the corresponding random exponents Δ had used with the random self-reducibility to blind. Δ then unblinds' the remaining part "$*$" of the Hash-query and outputs it.

The probability that Δ correctly "guesses" the moment of the Test-query is the probability that $\mathcal{A}$ makes its Test-query after c_0 operations (proba $\geqslant 1/Q$) multiplied by the probability that at c_0 the multicast group is $\mathcal{I}_0$ (proba $\geqslant 1/\binom{n}{s}$). The probability Δ correctly "guesses" the index i_0 player is at least $1/s$. Also the solution is correctly extracted from the $\mathcal{H}$-list with probability $1/q_h$, since one just picks one candidate at random. Otherwise, one could output all the possible unblinded candidates and use the by now classical reduction from [Sho97].

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

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

Finally, we have:

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

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

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

□

6.3. AKE1 in Practice

We want our results to be practical. This means that when system designers choose a scheme they will take into account its security but also its efficiency in terms of computation, communication, ease of integration

and so on. However, if provable security is achieved at the cost of a loss of efficiency, system designers will often prefer the heuristic schemes.

AKE1 was to date the first group Diffie-Hellman scheme to exhibit a proof that it achieves a strong notion of security. It is secure in the random oracle model under the G-CDH assumption. It thus provides stronger security guarantees than other schemes [AST00, BD95, JV96] while being more efficient than [AST00]. However security proofs for existing schemes or slight variants may show up.

On the integration front, the question that may be raised is what happens when several groups merge to form a larger group. A scenario that occurs in practice when a network failure partitions the multicast group into several disjoint sub-groups which will later need to merge when the network is be repaired [ACTT01]. The most efficient way in terms of computation and communication is to add players from the smaller sub-groups into the largest of the merging sub-groups. That is, U_{GC} is chosen as the player with the highest-index in the largest merging sub-group and the players from the smaller sub-groups are added via the JOIN1 algorithm.

7. Mutual Authentication

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

8. Conclusion

This chapter provides the first formal treatment of the authenticated group Diffie-Hellman key exchange problem in a scenario in which the membership is *dynamic* rather than static. Addressed in this chapter were two security goals of the group Diffie-Hellman key exchange: the authenticated key exchange and the mutual authentication. For each we presented a definition, a protocol and a security proof in the random oracle model that the protocol meets its goals.

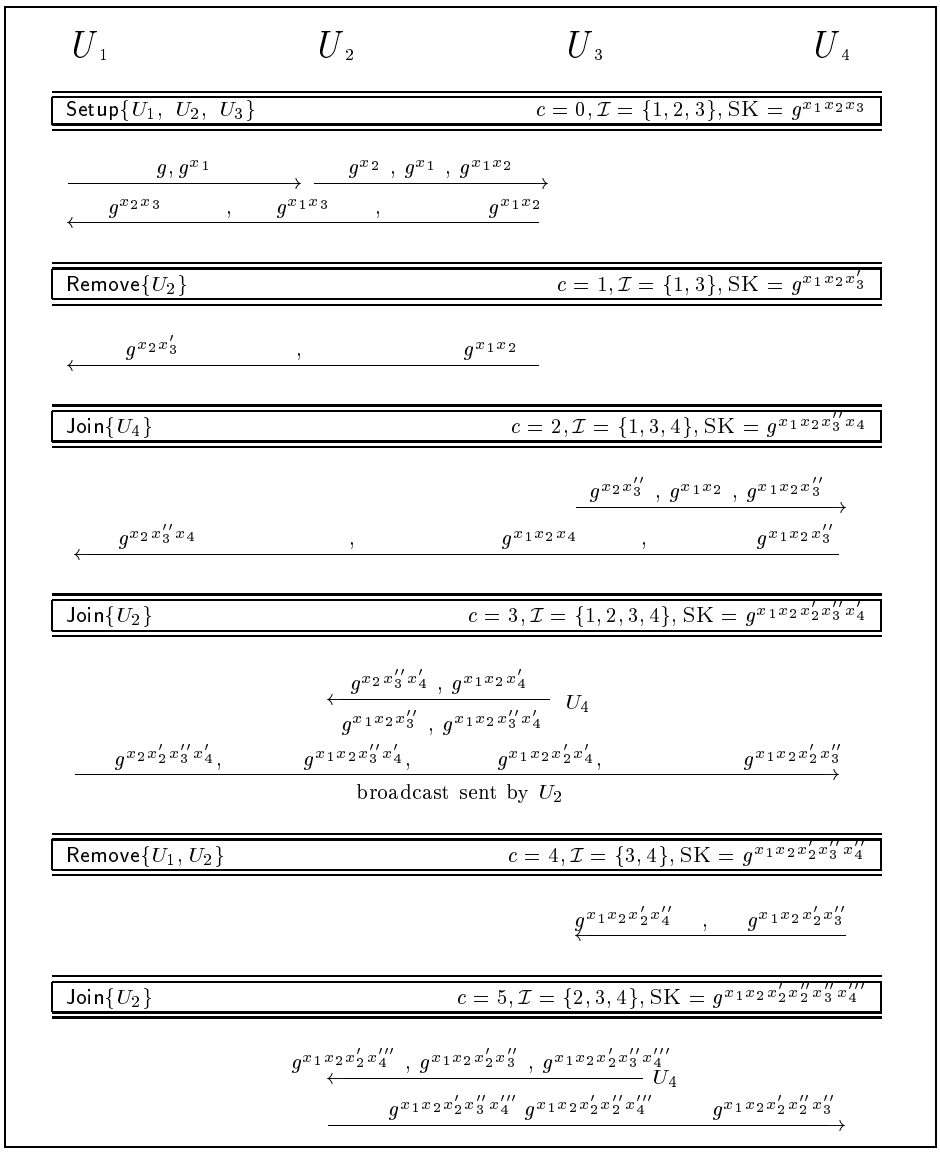

FIGURE 6. An example of an execution of the real protocol P=AKE1

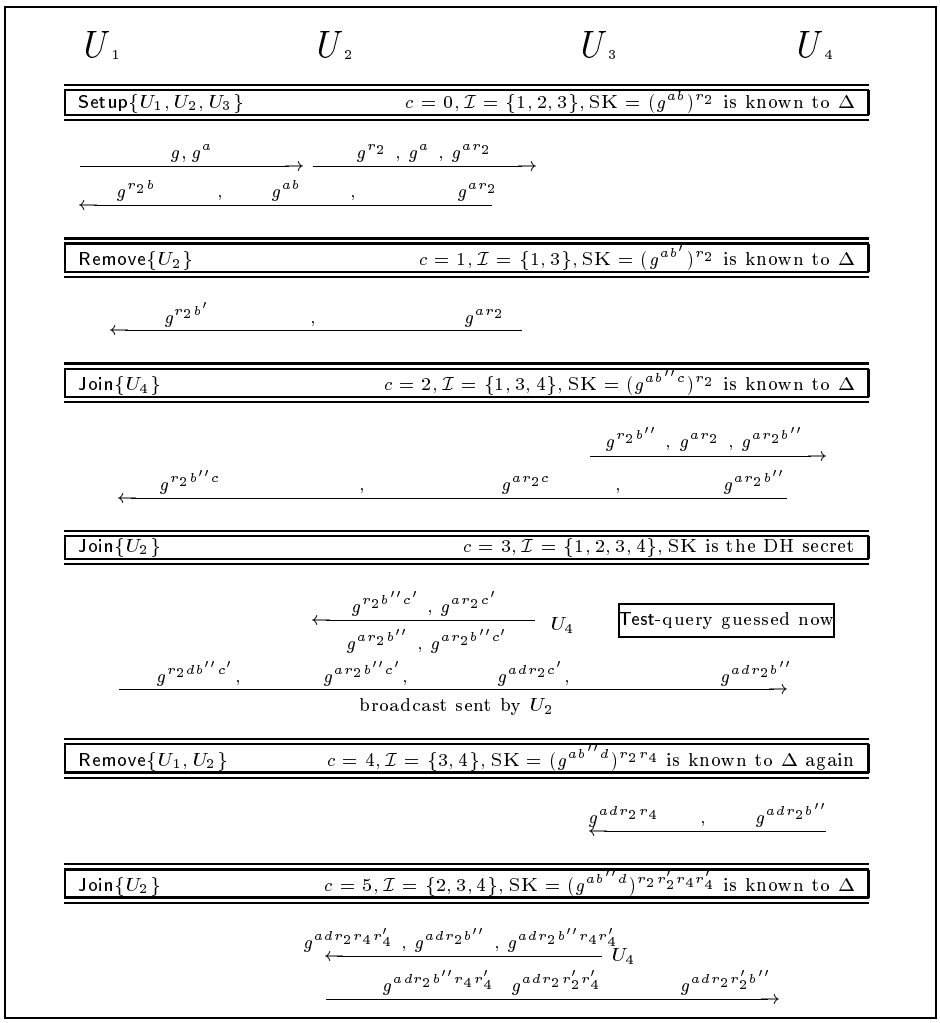

FIGURE 7. An example of an execution of the protocol P=AKE1 with the adversary. We represent the simulation by Δ according to the following "guesses": $c_0 = 3, s = 4, \mathcal{I}_0 = \{1,2,3,4\}, i_0 = 2$. We denote by b, b', b'' etc. some blinding exponents used in the self-reduction of G-CDH (think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

CHAPTER 4

Group Diffie-Hellman Key-Exchange under Standard Assumptions

Authenticated dynamic group Diffie-Hellman key exchange allows a pool of principals communicating over a public network, and each holding public/private keys, to agree on a shared secret value. In this chapter we refine our previous study of this problem to incorporate major missing details (e.g., strong-corruption and concurrent sessions). Within our new model we define the execution of a protocol for authenticated dynamic group Diffie-Hellman and show that it is provably secure under the decisional Diffie-Hellman assumption. Our security result here holds in the standard model and thus provides better security guarantees than our result in the chapter.

1. Introduction

This chapter is the third tier in the formal treatment of the group Diffie-Hellman key exchange using public/private key pairs. We start with the model from the previous chapter and refine it to add important attributes. In the present chapter, we model instances of players via oracles available to the adversary through queries. The queries are available to use at any time to allow model attacks involving multiple instances of players activated concurrently and simultaneously by the adversary. In order to model two modes of corruption, we consider the presence of two cryptographic devices which are made available to the adversary through queries. Hardware devices are useful to overcome software limitations however there has thus far been little formal security analysis [CFIJ99, SR96].

The types of crypto-devices and our notion of forward-secrecy leads us to modifications to the protocol AKE1 of the previous chapter to obtain a protocol, we refer to it as $\mathsf{AKE1}^{+}$, secure against strong corruptions. Due to the very limited computational power of a smart card chip, the smart card is used as an authentication token while a secure coprocessor is used to carry out the key exchange operations. We show that within

our model the protocol $\mathsf{AKE1}^+$ is secure assuming the decisional Diffie-Hellman problem and the existence of a pseudo-random function family. Our security theorem does not need a random oracle assumption anymore since it holds in the standard model. A proof in the standard model provides better security guarantees than one in an idealized model of computation. Furthermore we exhibit a security reduction with a much tighter bound than in the previous chapter, namely we suppress the exponential factor in the size of the group. Therefore the security result is meaningful even for large groups. However the protocols are not practical for groups larger than 100 members.

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

2. Related Work

Several papers [AST98, BD95, JV96, SSDW88, Tze00] have extended the 2-party Diffie-Hellman key exchange [DH76] to the multi-party setting, however, a formal analysis has only been proposed recently. In the previous chapters we defined a formal model for the authenticated (dynamic) group Diffie-Hellman key exchange and proved secure protocols within this model. In both chapters we use an ideal hash function [BR93b], without dealing with dynamic group changes or concurrent executions of the protocol.

However security can sometimes be compromised even when using a proven secure protocol: the protocol is incorrectly implemented or the model is insufficient. Cryptographic protocols assume, and do not usually explicitly state, that secrets are *definitively and reliably erased* (only the most recent secrets are kept) [CFIJ99, JQ97]. Recently, formal models have been refined to incorporate the cryptographic action of erasing a secret, and thus protocols achieving forward-secrecy in the strong-corruption sense have been proposed [BPR00, Sho99].

Protocols for group Diffie-Hellman key exchange achieve the property of forward-secrecy [DvOW92, Gun89] in the strong-corruption sense assuming that "ephemeral" private keys are erased upon completion of a protocol run. However protocols for dynamic group Diffie-Hellman key

exchange do not, since they reuse the "ephemeral" keys to update the session key. Fortunately, these "ephemeral" keys can be embedded in some hardware cryptographic devices which are at least as good as erasing a secret [PSW98, U. 94, VW97].

3. Model

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

3.1. Players

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

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

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

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

3.2. Abstract Interface

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

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

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

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

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

3.3. Security Model

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

3.3.1. *Instance Oracle Queries*

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

- $\mathsf{Send}(\Pi_U^t, m)$: This query models $\mathcal{A}$ sending messages to instance oracles. $\mathcal{A}$ gets back from his query the response which Π_U^t would have generated in processing message m according to P.

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

3.3.2. *Secure Coprocessor Queries*

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

- $\mathsf{Send}_c(\Pi_U^t, m)$: This query models $\mathcal{A}$ *directly* sending messages to the secure coprocessor. $\mathcal{A}$ gets back from his query the response which the secure coprocessor would have generated in processing message m. The adversary could directly interact with the secure coprocessor in a variety of ways: for instance, the adversary may have broken into a computer without being detected (e.g., bogus softwares, trojan horses and viruses).
- $\mathsf{Corrupt}_c(\Pi_U^t)$: This query models $\mathcal{A}$ having access to the private memory of the device. $\mathcal{A}$ gets back the internal data stored on the secure coprocessor. This query can be seen as an attack wherein $\mathcal{A}$ gets physical access to a secure coprocessor and bypasses the tamper detection mechanism [Wei00]. This query is only available to the adversary when considering the *strong-corruption model* (see Section 5). The $\mathsf{Corrupt}_c$-query also reveals the flows the secure coprocessor and the smart card have exchanged.

3.3.3. *Smart Card Queries*

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

- $\mathsf{Send}_s(U, m)$: This query models $\mathcal{A}$ sending messages to the smart card and receiving messages from the smart card.
- $\mathsf{Corrupt}_s(U)$: This query models the attacks in which the adversary gets access to the smart card and gets back the player's LL-key. This query models attacks like differential power analysis or other attacks by which the adversary bypasses the tamper detection mechanisms of the smart card [Wei00].

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

4. An Authenticated Dynamic Group Diffie-Hellman Scheme

In this section, we describe the protocol $\mathsf{AKE1}^+$ by splitting it into functions that implement the GDH abstract interface. These functions specify how certain cryptographic transformations have to be performed and abstract out the details of the devices (software or hardware) that will carry out the transformations.

In the following we identify the multicast group to the set of indices of players (instances of players) in it. We use a security parameter ℓ and, to make the description easier see a player U_i not involved in the multicast group as if his private exponent x_i were equal to 1.

4.1. Overview

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

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

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

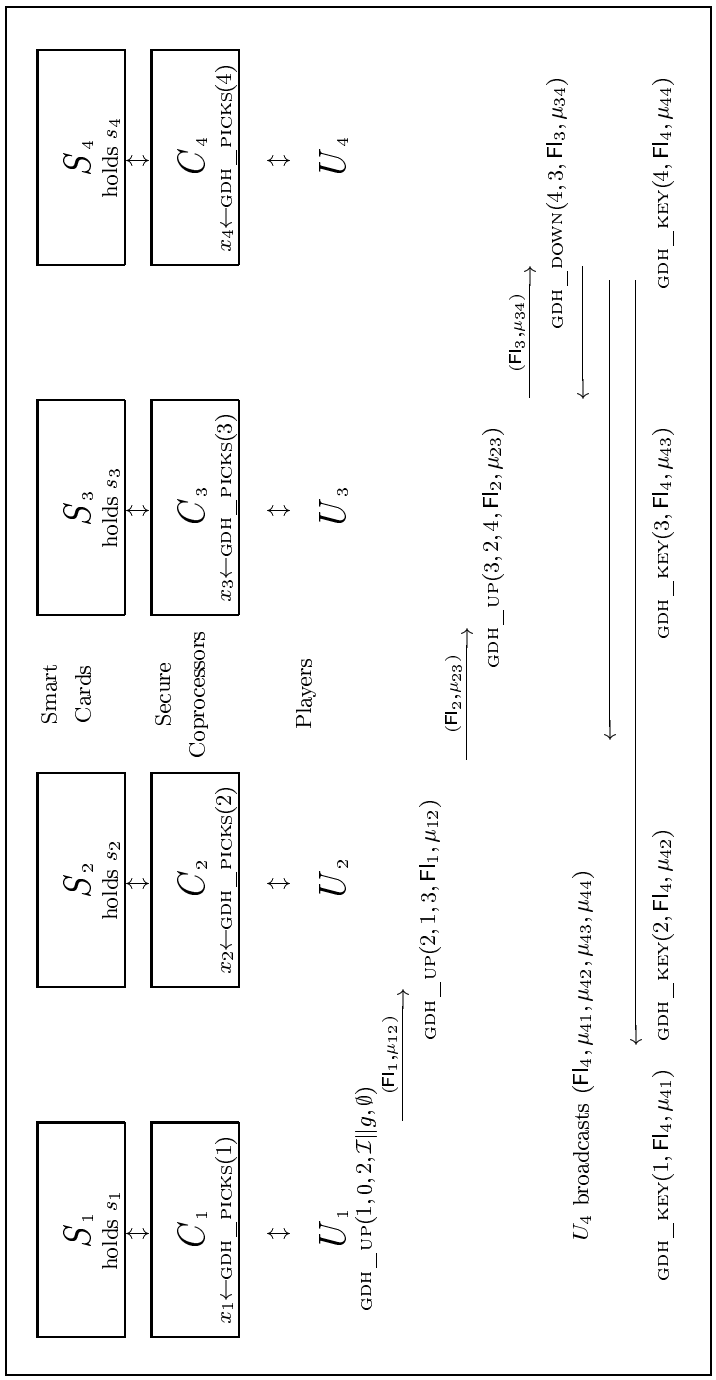

FIGURE 1. Algorithm $\mathsf{Setup1}^+$. A practical example with 4 players $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$.

4.2. Authentication Functions

The authentication mechanism supports the following functions:

- AUTH_KEY_DERIVE(i, j). This function derives a secret value K_{ij} between U_i and U_j. In our protocol, $K_{ij} = F_1(g^{s_i s_j})$, where the map F_1 is specified in Section 4.4. (K_{ij} is never exposed.)
- AUTH_SIG(i, j, m). This function invokes MAC.SGN(K_{ij}, m) to obtain tag μ, which is returned.
- AUTH_VER(i, j, m, μ). This function invokes MAC.VF(K_{ij}, m, μ) to check if (m, μ) is correct w.r.t. key K_{ij}. The boolean answer is returned.

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

4.3. Key-Exchange Functions

The key-exchange mechanism supports the following functions:

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

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

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

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

  Third, $\mathsf{Fl}' = (\mathcal{I}, Y', Z')$ is authenticated, by invoking AUTH_SIG(i, k, Fl') to obtain tag μ'. The flow (Fl', μ') is returned.
- GDH_DOWN(i, j, Fl, μ). First, the authenticity of (Fl, μ) is checked, by invoking AUTH_VER(j, i, Fl, μ). Then the flow Fl' is computed as in GDH_UP, from $\mathsf{Fl} = (\mathcal{I}, Y, Z)$ but without the last element Z' (i.e. $\mathsf{Fl}' = (\mathcal{I}, Y')$). Finally, the flow Fl' is appended tags $\mu_1, \ldots, \mu_n$ by invoking AUTH_SIG(i, k, Fl'), where k ranges in $\mathcal{I}$. The tuple $(\mathsf{Fl}', \mu_1, \ldots, \mu_n)$ is returned.

- GDH_UP_AGAIN$(i, k, \mathsf{Fl} = (\mathcal{I}, Y'))$. From Y' and the previous random x_i', one can recover the associated Z'. In this tuple (Y', Z'), one replaces the occurrences of the old random x_i' by the new one x_i (by raising some elements to the power x_i/x_i') to obtain Fl'. The latter is authenticated by computing via AUTH_SIG(i, k, Fl') the tag μ. The flow (Fl', μ') is returned. From now the old random x_i' is no longer needed and, thus, can be erased.
- GDH_DOWN_AGAIN$(i, \mathsf{Fl} = (\mathcal{I}, Y'))$. In Y', one replaces the occurrences of the old random x_i' by the new one x_i, to obtain Fl'. This flow is appended tags $\mu_1, \ldots, \mu_n$ by invoking AUTH_SIG(i, k, 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 (Fl, μ) is checked with AUTH_VER(j, i, Fl, μ). Second, the value $\alpha = g^{\prod_{j \in \mathcal{I}} x_j}$ is computed from the private exponent x_i, and the corresponding value in Fl. Third, sk is defined to be $F_2(\mathcal{I} \| \mathsf{Fl} \| \alpha)$, where the map $F_2(\cdot)$ is defined below.

4.4. Key Derivation Functions

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

LEMMA 3 (Left-Over-Hash Lemma [HILL99]). Let $\mathcal{D}_s : \{0,1\}^s$ be a probabilistic space with entropy at least σ. Let e be an integer and $\ell = \sigma - 2e$. Let $h : \{0,1\}^k \times \{0,1\}^s \to \{0,1\}^\ell$ be a universal hash function. Let $r \in_{\mathcal{U}} \{0,1\}^k$, $x \in_{\mathcal{D}_s} \{0,1\}^s$ and $y \in_{\mathcal{U}} \{0,1\}^\ell$. Then the statistical distance δ is:

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

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

The map $F_1(\cdot)$ is implemented as follows through *public certified random strings*. In a Public-Key Infrastructure (PKI), each player U_i is given $N-1$ random strings $\{r_{ij}\}_{j \neq i}$ each of length k when registering his

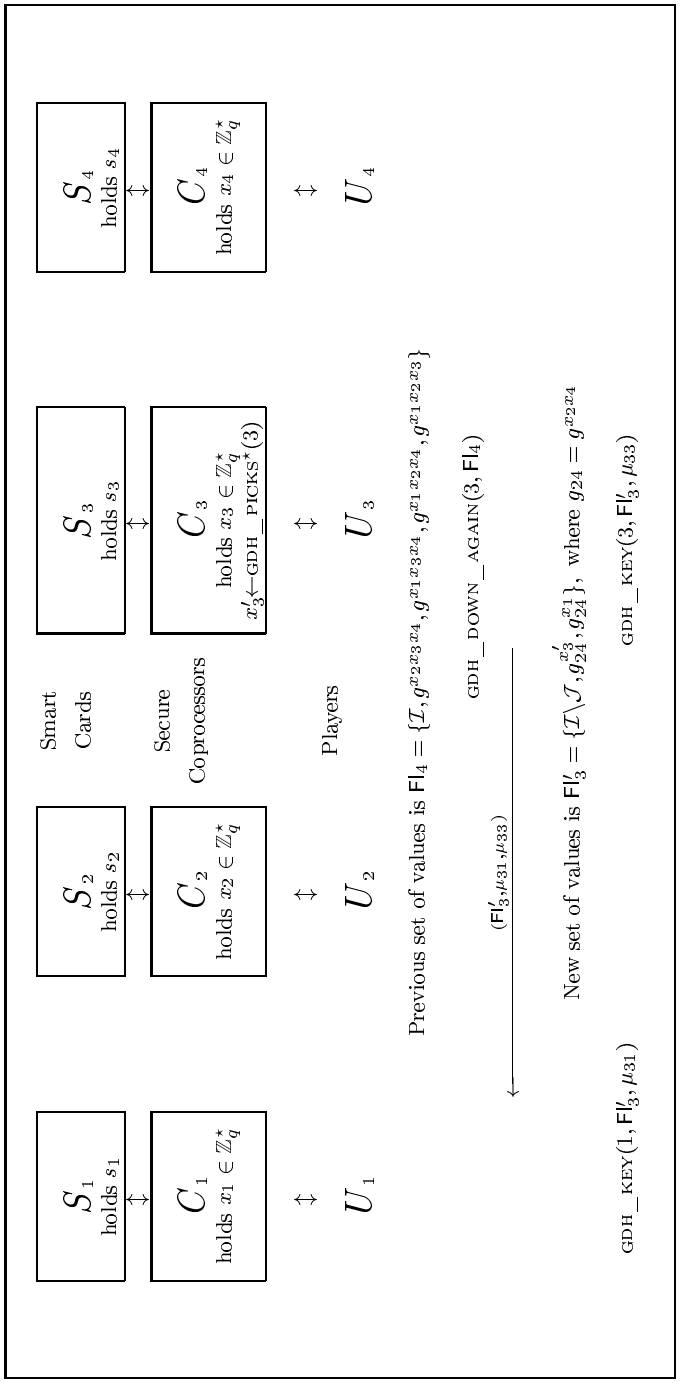

FIGURE 2. Algorithm $\mathsf{Remove1}^+$. A practical example with 4 players: $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$ and $\mathcal{J} = \{U_2, U_4\}$. The new multicast group is $\mathcal{I} = \{U_1, U_3\}$ and $\mathsf{GC} = U_3$.

identity with a Certification Authority (CA). Recall that $N = |\mathcal{U}|$. The random string $r_{ij} = r_{ji}$ is used by U_i and U_j to derivate from input value x a symmetric-key $K_{ij} = F_1(x) = h_{r_{ij}}(x)$.

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

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

4.5. Scheme

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

4.5.1. *Setup1*$^+(\mathcal{I})$

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

4.5.2. *Remove1*$^+(\mathcal{I}, \mathcal{J})$

This algorithm consists of a down-flow only (see Figure 2). The group controller $\mathsf{GC}(\mathcal{I})$ of the new set $\mathcal{I} = \mathcal{I} \backslash \mathcal{J}$ invokes GDH_PICKS$^\star(\mathcal{I}_n)$ to get a *new* private exponent and then GDH_DOWN_AGAIN$(\mathcal{I}_n, \mathsf{Fl}')$ where Fl' is the saved previous broadcast. $\mathsf{GC}(\mathcal{I})$ obtains a new set of intermediate values from which it deletes the elements related to the removed players (in the set $\mathcal{J}$) and updates the multicast group. This produces the new broadcast flow Fl_n. Upon receiving the down-flow, $\Pi(\mathcal{I}, i)$ invokes GDH_KEY$(\mathcal{I}_i, \mathcal{I}_n, \mathsf{Fl}_n, \mu_i)$ and gets back the session key

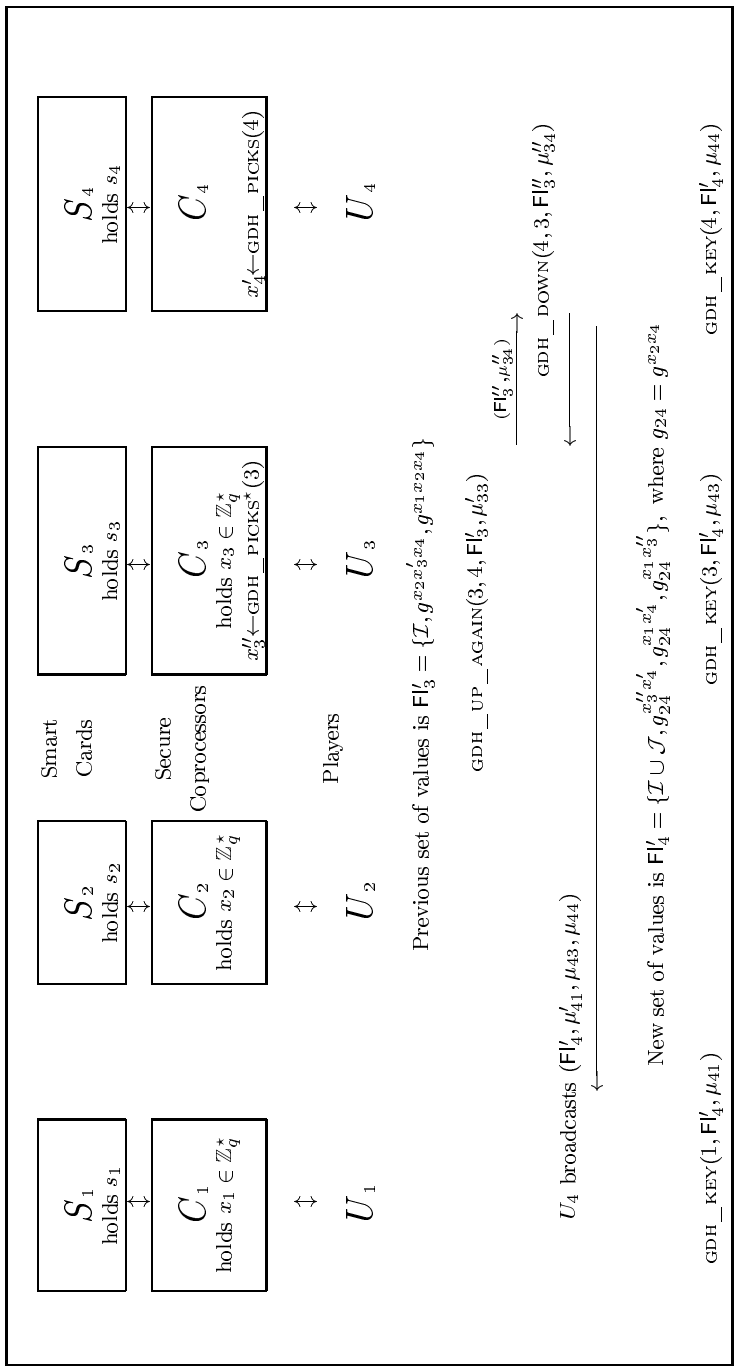

FIGURE 3. Algorithm $\mathsf{Join1}^+$. A practical example with 4 players: $\mathcal{I} = \{U_1, U_3\}$, $\mathcal{J} = \{U_4\}$ and $\mathsf{GC} = U_3$. The new multicast group is $\mathcal{I} = \{U_1, U_3, U_4\}$.

$\mathrm{SK}_{\Pi(\mathcal{I},i)}$. Here, is the reason why an oracle must store its private exponent and only erase its internal data when it leaves the group.

4.5.3. *Join1*$^+(\mathcal{I}, \mathcal{J})$

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

4.6. Practical Considerations

When implementors choose a protocol, they take into account its security but also its ease of integration. For a minimal disruption to a current security infrastructure, it is possible to modify $\mathsf{AKE1}^+$ so that it does not use *public certified random strings*. In this variant, the key derivation functions are both seen as ideal functions (i.e. the output of $F_1(\cdot)$ and $F_2(\cdot)$ are uniformly distributed over $\{0,1\}^\ell$) and are instantiated using specific functions derivated from cryptographic hash functions like SHA-1 or MD5. The analogue of Theorem 6 in the random oracle model can then easily be proven from the security proof of $\mathsf{AKE1}^+$.

5. Analysis of Security

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

5.1. Security Notions

5.1.1. *Forward-Secrecy*

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

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

5.1.2. *Freshness*

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

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

5.1.3. *AKE Security*

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

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

5.1.4. *Multi Decisional Diffie-Hellman Assumption (M-DDH)*

We introduce a new decisional intractability assumption which is based on the Diffie-Hellman assumption. Let us define the *Multi Diffie-Hellman* M-DH and the *Random Multi Diffie-Hellman* $\mathsf{M\text{-}DH}^{\$}$ distributions of size n as:

$$\begin{aligned}
\mathsf{M\text{-}DH}_n &= \{\left(\{g^{x_i}\}_{1\le i\le n}, \{g^{x_i x_j}\}_{1\le i<j\le n}\right) \mid x_1, \dots, x_n \in_R \mathbb{Z}_q\} \\
\mathsf{M\text{-}DH}_n^{\$} &= \{\left(\{g^{x_i}\}_{1\le i\le n}, \{g^{r_{j,k}}\}_{1\le j<k\le n}\right) \mid x_i, r_{j,k} \in_R \mathbb{Z}_q, \\
&\qquad\qquad \forall i, 1 \le j < k \le n\}.
\end{aligned}$$

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

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

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

LEMMA 4. For any group $\mathbb{G}$ and any integer n, the $\mathsf{M\text{-}DDH}_n$ problem can be reduced to the DDH problem and we have: $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{mddh}_n}(T) \leq n^2 \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(T)$.

The proof of this lemma follows from an easy hybrid argument [NR97]. □

5.2. Theorem of Security

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

THEOREM 4. Let $\mathcal{A}$ be an adversary against protocol P, running in time T, allowed to make at most Q queries, to any instance oracle. Let n be the number of players involved in the operations which lead to the group on which $\mathcal{A}$ makes the Test-query. Then we have:

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

where δ_i denotes the distance between the output of $F_i(\cdot)$ and the uniform distribution over $\{0,1\}^{\ell}$, $T' \leq T + QnT_{exp}(k)$, where $T_{exp}(k)$ is the time

of computation required for an exponentiation modulo a k-bit number, and Γ_n corresponds to the elements adversary $\mathcal{A}$ can possibly view:

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

Before to get into the details of the proof in we provide the main ideas. Let the notation $\mathbf{G}_0$ refer to $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, P)$. Let b and b' be defined as in Section 3 and S_0 be the event that $b = b'$. We incrementally define a sequence of games starting at $\mathbf{G}_0$ and ending up at $\mathbf{G}_5$. We define in the execution of $\mathbf{G}_{i-1}$ and $\mathbf{G}_i$ a certain "bad" event E_i and show that as long as E_i does not occur the two games are identical [Sho01]. The difficulty is in choosing the "bad" event. We then show that the advantage of $\mathcal{A}$ in breaking the **AKE** security of P can be bounded by the probability that the "bad" events happen. We now define the games $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3, \mathbf{G}_4, \mathbf{G}_5$. Let S_i be the event $b = b'$ in game $\mathbf{G}_i$.

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

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

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

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

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

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

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

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

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

□

Remark. When considering strong-corruptions we have to answer to all the $\mathsf{Corrupt}_c$-queries made by the adversary along the games but we can only do so if we know the private exponents involved in the games. To reach this aim, we can no longer benefit from the self-random reducibility property of G-DDH and have to "guess" the moment at which the adversary will ask the Test-query. Unfortunately, reductions carried out in such a way add an exponential factor in the size of the multicast group [BCPQ01, BCP01].

6. Proof of the 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. By *player involved in a group*, we mean a player who has joined the group at least once since its setup.

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

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

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

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

$$\left|\Pr\left[\mathsf{S}_0\right] - \Pr\left[\mathsf{S}_1\right]\right| \leq \Pr\left[\mathsf{Forge}\right]. \tag{6.1}$$

LEMMA 6. Let δ_1 be the distance between the output of the map F_1 and the uniform distribution. Then, we have:

$$\Pr\left[\mathsf{Forge}\right] \leq \mathsf{Adv}^{\mathsf{mddh}_n}_{\mathbb{G}}(T) + \frac{n(n-1)}{2}\mathsf{Succ}^{\mathsf{cma}}_{\mathsf{mac}}(T) + \frac{n(n-1)}{2}\delta_1. \tag{6.2}$$

The proof of this lemma appears later on in section 7.

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

$$\begin{aligned}
\Pr\left[\mathsf{S}_2\right] &= \Pr\left[\mathsf{S}_2 \wedge \mathsf{E}_2\right] + \Pr\left[\mathsf{S}_2 \wedge \neg\mathsf{E}_2\right] && (6.3)\\
&= \Pr\left[\mathsf{S}_2 \mid \mathsf{E}_2\right]\Pr\left[\mathsf{E}_2\right] + \Pr\left[\mathsf{S}_2 \mid \neg\mathsf{E}_2\right]\Pr\left[\neg\mathsf{E}_2\right] \\
&= \frac{1}{2}\Pr\left[\mathsf{E}_2\right] + \Pr\left[\mathsf{S}_1\right]\left(1 - \Pr\left[\mathsf{E}_2\right]\right), && (6.4)
\end{aligned}$$

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

Game $\mathbf{G}_3$. Game $\mathbf{G}_3$ is the same as game $\mathbf{G}_2$ except that we modify slightly the way queries made by $\mathcal{A}$ are answered. Δ receives as input

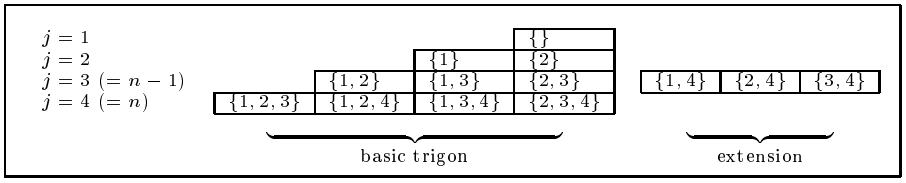

FIGURE 4. Extended Trigon for Γ_4

an instance $\mathcal{D}$ of size n from $\mathsf{G\text{-}DH}^{\star}_{\Gamma_n}$, with its solution $g^{x_1 \cdots x_n}$:

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

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

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

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

This strategy allows Δ to deal with situations where n players are involved in the group *before* c_0, and are added and removed repeatedly. If, in effect, Δ embeds all the elements of S_n into the protocol execution the first time the size of $\mathcal{L}_0$ is n, Δ is not able to compute the session key value sk needed to answer to the Reveal-query. Before c_0, Δ uses truly random values instead of instance $\mathcal{D}$ for player U_{i_0} or if there are already $n-1$ players involved in the group. Note that Δ embeds elements of S_i when a new player U_i (except U_{i_0}) is added to the group $\mathcal{I}$ for the first time and Δ does not remove it when U_i leaves. This way, after the joining operation of the j-th player from $\mathcal{L}_0$, U_{i_0} excepted, the broadcast flow involves a random self-reduction of the j-th line in the basic trigon (see figure 4), the up-flows involve elements in the $j-1$-th line, and the

session key one element from the $j+1$-th line. Thus, before operation c_0, Δ is able to answer the Join and Remove-queries and knows all the session keys needed to answer the Reveal-queries. To correctly deal with self-reducibility, Δ make use of variable $\mathcal{T}$ to reconstruct well-formatted (blinded) flows from $\mathcal{D}$.

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

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

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

$$\Pr[\mathsf{S}_2] = \Pr[\mathsf{S}_3]. \tag{6.5}$$

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

$$\left|\Pr[\mathsf{S}_3] - \Pr[\mathsf{S}_4]\right| \leq \mathsf{Adv}^{\mathsf{gddh}_{\Gamma_n}}_{\mathbb{G}}(T'). \tag{6.6}$$

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

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

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

$$\left|\Pr[\mathsf{S}_5] - \Pr[\mathsf{S}_4]\right| \leq \delta_2. \tag{6.7}$$

Putting together Equations (6.1), (6.2), (6.4), (6.5), (6.6), (6.7), we get

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

The theorem then follows from lemma 6. □

7. Proof of Lemma 6

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

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

Game $\mathbf{G}'_1$. The game $\mathbf{G}'_1$ is identical to $\mathbf{G}'_0$, except that each MAC key K_{ij} is computed as $F_1(g^{r_{ij}})$, where r_{ij} is a random value, instead

of $F_1(g^{x_i x_j})$. It follows that the difference between the two games is upper-bounded by the computational distance $\mathsf{Adv}^{\mathsf{mddh}_n}_{\mathbb{G}}(T)$:

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

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

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

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

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

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

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

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

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

Finally, we easily get:

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

□

8. Conclusion

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

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

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

CHAPTER 5

Practical Aspects of Group Diffie-Hellman Key Exchange

A group communication system provides a platform upon which distributed applications can rely on to achieve reliable coordination among their components, however when operating over public wires security becomes an issue to address. The problem that immediatly emerges is how to integrate cryptographic mechanisms, such as an authenticated group Diffie-Hellman key exchange algorithm, with a group communication system.

In this chapter we provide a security framework to put cryptographic algorithms to practical use. Our framework is a security layer that bundles an authenticated dynamic group Diffie-Hellman algorithm, a distributed authorization/access control mechanism, and a reliable group communication system to provide a comprehensive and practical secure group communication platform. This layer also encapsulates the standard multicast security services (i.e multicast message confidentiality, and multicast data integrity). A number of challenging issues encountered in the design of the layer are brought to light and experimental results obtained with a prototype implementation are discussed in this chapter.

1. Introduction

Many current applications are implemented as distributed systems. Some are distributed by nature (e.g., collaboratory and conferencing software) while others are distributed to meet load-balancing and fault-tolerance requirements (e.g., content servers and fault-tolerant CORBA). Such applications often rely on reliable group communication to provide coordination between processes.

One example of an application class that can benefit from, and make extensive use of, a reliable group communication platform is scientific collaboration software. Applications such as distributed white boards, remote instrument control, messaging systems, electronic notebooks, and data sharing are natural users of group communication. Applications of

this type normally involve users spread across a wide-area network and may utilize multiple groups. Unfortunately, few group communication systems can operate over a wide-area network and even fewer incorporate the access control and other security services that these applications require.

Although acceptable solutions (e.g., SSL [DR02] and Kerberos [SNS98]) are available for securing unicast connections, they do not extend to securing group communication. One of the main reasons is key management. Unicast communication involves only two parties and consensus on a shared key is relatively easy to reach. Each time a new unicast connection is created the consensus process starts from scratch and, if either party in a unicast communication session quits, fails, or drops the connection, the other party also quits. In group communication, consensus on a shared key is more complex since group membership is dynamic. Once a group is formed, members may join or leave the group due to failures, network partitions and voluntary membership changes.

Controlling access to a group requires authentication of users and definition of group access policy. Authentication and authorization for groups present a more complicated set of problems than the typical client-server access control. Authentication is more difficult because each group member must be able to authenticate all the other members. In a server-based access control model the policy normally only controls access and is only enforced by the server[1].

Another challenge in introducing group security services is how best to provide them to the application. One approach is to integrate them into the underlying group communication system. This approach makes the security services invisible to the applications but makes providing authenticity, authorization/access control based on user credentials very difficult. This solution also would not be portable across different group communication systems.

An alternate approach is to interpose a security layer between the application and the group communication system. This approach introduces minor changes in the application to convey a user's credentials (access privileges and user identity information) to the security layer and has the advantage of being largely independent of the group communication system implementation. This approach also allows the security layer to leverage off the properties of the group communication system in transmitting its own messages.

[1]The scope of group security policy is still a research topic as are the methods of group policy enforcement.

The contribution of this chapter is in the design of a Secure Group Layer aimed at wide-area environments. This layer, we refer to it as SGLv1, protects against attacks like eavesdropping and spoofing[2] by integrating a reliable group communication system, a group authorization and access control mechanism to determine who knows the key, and an authenticated dynamic group Diffie-Hellman key exchange algorithm which facilitates the standard message security services (i.e. confidentiality, authenticity and integrity). However, denial of service attacks through corruption of the underlying group communication system can still be a problem.

The remainder of this paper is organized as follow. Section 2 defines the group communication terminology. Section 3 summarizes the related work. Section 4 describes the SGLv1 architecture. Finally Section 5 presents some experimental results obtained with the prototype implementation.

2. Group Communication

Group communication systems are designed to support communication between processes cooperating in groups. The group communication system provides an underlying layer that does the work of maintaining membership of the process group and reliably delivering messages sent to the group in an asynchronous distributed system. Example group communication systems which provide these properties include Totem [MMSA+96], Spread [ADS00], and InterGroup [BAC02, BAMSM01].

There are several message ordering properties that group communication systems provide to the application. A very basic property is *causal ordering* of messages. Messages sent by the same application process are received by the application in the order they were sent and messages received by an application process before sending a new message (m1) are ordered before m1 at all the members of the group. Many systems also provide a *total order* on messages within the group so that messages are received by the application in a single linear total order that is the same at all the members of the group.

Group membership maintenance is a critical component of the group communication system since the membership of the group is the basis for the determination of reliable delivery of messages and message order. A particular instance of the group membership is referred to as a *view*. Each application receives messages within the context of a view. It

[2]Spoofing is an integral part of many network attacks. In a group communication setting, spoofing attacks refer to the impersonation of a group member.

is important that the delivery order of messages and view changes are consistent across the members of a particular view.

There are several consistency definitions that are in use by group communication systems. Some common consistency definitions are *sending view delivery*, *view synchrony*, and *extended virtual synchrony* (see [VKCD99]). *Sending view delivery* means that messages are received in the view in which they were sent. *Virtual synchrony* [BJ87] and *extended virtual synchrony* [MAMSA94] (EVS) define message order, message delivery and view change consistency constraints. In the case of EVS these consistency constraints are system wide.

3. Related Work

Research on protocols for secure reliable multicast communication could be classified according to their threat model. One approach shares the same security model as the protocol SSL/TLS which assumes that the end-points will not get corrupted. Ensemble security [RBH+98] relies on a static group leader to generate session keys and distribute them via extensions of two-party cryptographic tools such as SSL/TLS, PGP [Zim95] or Kerberos. The group leader is static and changes only when the current group leader leaves or becomes unreachable. The drawback of such an approach is two fold. The trusted server is a single point of failure in the overall system and using secure unicast communication leads to inefficient secure group communication.

Secure Spread [AAH+00] differs from Ensemble since it uses an authenticated group Diffie-Hellman algorithm [AST00, ACH+00]. This algorithm does not achieve provable security and flaws have been discovered in it [BCP+02a, PQ01a]. Secure Spread is placed above the Spread group communication system and relies on the property commonly known as *view synchrony*. View synchrony is a stronger property than the *sending view delivery* we require for SGLv1. Secure Spread also does not provide any authentication/access control mechanisms, and focuses primarily on LAN and interconnected LAN environments.

Another approach assumes that the group members can be corrupted, i.e. the so-called Byzantine failure model. Rampart [Rei94] was the first to demonstrate the feasibility of reliable and atomic group multicast for asynchronous distributed systems in the presence of Byzantine failures. It uses public key cryptography to establish authenticated communication between a pair of processors and implements the reliable and atomic group multicast protocols over a secure group membership protocol [Rei96]. Immune [KMMS98] uses public key cryptography to secure

the Totem daemon. Immune secures the low-level ring protocol against Byzantine failure and hence maintains the reliable ordered message delivery and group membership services despite the corruption of some group communication servers by an adversary. The extension of Rampart and Immune to wide-area environments has been done in [MMR97].

Another important aspect of securing group communication is the group policy issues such as requirements for group rekeying and levels of message security. The Antigone framework [MPH99] provides interfaces for the definition and implementation of policies for secure groups. Policies are implemented by composition and configuration of mechanisms which provide basic services for secure groups. For the purpose of this paper we will assume that a simple policy exists but we will still need to identify the repository for group policies.

4. The Secure Group Layer

The design of our secure group layer SGLv1 uses the properties provided by the underlying group communication system. These properties, which are usually refered as *extended virtual synchrony*, ensure that messages are consistently ordered and delivered within the group. The *view change events* emanating from the group communication system notify SGLv1 of membership changes due to a join, leave, fail, partition, or merge events.

In addition to the existing properties of the group communication system, SGLv1 provides applications with the property of *sending view delivery* [CKV01]. This property is useful for implementing group security services since it guarantees the application that messages are delivered in the same view as they were sent. By using sending view delivery, SGLv1 can use one session key at a time and thus bind a session key to each new view.

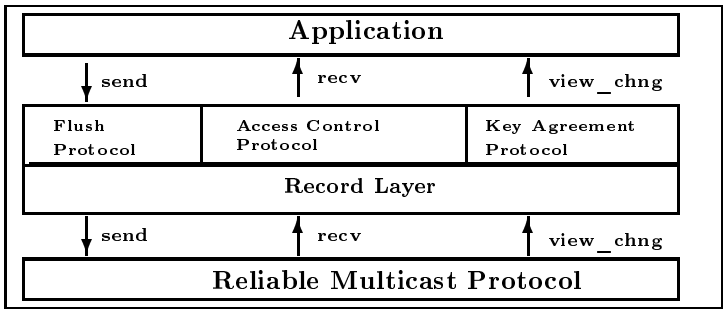

FIGURE 1. The architecture of SGLv1.

The SGLv1 architecture consists of four main components each implementing a separate protocol (see Fig. 1). The *record layer* provides standard message security services (i.e., confidentiality, integrity and authenticity). The *access control protocol* enforces restrictions on group membership. The *flush protocol* provides a mechanism for delineating membership views where each view corresponds to the lifetime of a specific session key and any keys derived from it. The *key agreement protocol* creates a session key which is then used to derive a key for a symmetric-encryption algorithm and a key for a message authentication code (MAC). These two keys are subsequently made available to the record layer.[3]

4.1. Record Layer

The record layer supports message transmission with confidentiality, integrity and authenticity. It takes an application message, applies an integrity algorithm using the MAC group key, encrypts it using the symmetric group encryption key and sends it out using the group communication system.

On receipt, a message is decrypted with the symmetric group decryption key, verified using the MAC group key and delivered to the application. The current SGLv1 implementation uses the Rijndael cipher [DR00] for encryption and the HMAC method [KBC97] for MAC computation.

4.2. Flush Protocol

As mentioned above, the flush protocol implements sending view delivery for SGLv1 and applications. It defines the end of a membership view and thus guarantees that no further messages encrypted with a particular session key will be received. Coordination is attained with special flush messages marking the end of a view.

The flush protocol is invoked by a view change event. Recall that *sending view delivery* means that all messages that the application has sent in a given view are received in that same view. To accomplish this, the flush protocol sends any pending messages (which have been accepted from the application) and blocks any new messages from the application before it sends a *flush* message.

The protocol waits until a flush message from every process in the new view has been received and verified. Since the flush marks the end of

[3]The flush, access control, and key agreement protocols are invoked by each view change event.

messages in a view receipt of a flush message represents for the recipient a "promise" by the sender not to send any more messages encrypted with the group key corresponding to the old view. When all flush messages are received, the process can conclude that no further messages protected by the old group key will be received. View change events reset and restart the flush protocol.

It is important to note that, if SGLv1 were be built on an underlying group communication system that provided sending view delivery, the flush layer would still be needed. Otherwise, the group communication system would need to accept, for sending in the old view, application messages buffered (e.g., during encryption) in SGLv1 and not yet passed to the group communication system.

4.3. Access Control Protocol

A group access control mechanism enforces restrictions on group membership. Without it, other security services (including key agreement and data integrity/privacy) are basically ineffective. Our access control approach uses *membership certificates* that authorize entry into the key agreement protocol and, hence, the group itself.

The Authorization Authority is responsible for collecting group policies and using them to determine who is allowed to join a group. The Akenti server is such an authority [TJM+99]. Akenti determines if a user is allowed to join a group and issues *membership certificates* containing that information.

A *membership certificate* (see Figure 2) associates a public key with the X.509 identity of a user and contains the access privilege granted to the user with respect to the group, the validity period for the certificate as determined from the policy, the identification of the Authorization Authority that issued the certificate and additional information for the Authorization Authority's use, such as a serial number.

Subject:	Distinguished Name, Public Key
Access Privilege:	Group, Authorized or Denied Access
Issuer:	Distinguished Name, Signature
Administrative Information:	Version, Serial Number.
Period of Validity:	Start and Expire Dates/Times

FIGURE 2. Membership certificate format.

The Akenti server not only issues membership certificates, it also manages them. It keeps a list of all non-expired certificates that have been

issued for a group. Akenti also keeps a list of all the revoked certificates called the Certificate Revocation List (CRL). When examining membership certificates for validity, therefore, it is necessary to contact the issuing Akenti server to check the Certificate Revocation List. At this time this is not an automated part of the group access control protocol.

In order to begin, a user needs to first obtain a membership certificate from the Akenti server or needs to request a new certificate if its certificate has expired or has been revoked. When a user wants to join the secure group, he will start by joining the reliable group. This join causes a view change and intitates the flush protocol.

During the flush protocol each member, including the new joinee, broadcasts its membership certicate. Each member will verify every other members' certificate by checking that the message is signed by the subject of the certificate, that the certificate is signed by Akenti, the certificate is within its validity period and the certificate grants joining access.

Once the group controller has verified all the members, it will start a new key agreement protocol. The group controller is a group member who enforces group access control policy by creating and disseminating the group keying material to authorized members. The group controller role is determined by the group key agreement as we will see in the next section. If any member sees key agreement messages from a user that it does not trust, it can refuse to participate.

4.4. Group Key Agreement Protocol

A group key agreement mechanism establishes a session key between members of a group. It allows the members to agree upon and begin computing a session key without relying on any centralized trusted third party (TTP) which could be a single point of vulnerability for the overall system. Authenticated group Diffie Hellman key exchange are such a mechanism.

In a preliminary work we implemented two group Diffie-Hellman key algorithms [ACH+00]. Using the terminology from [ACH+00] these algorithm are respectively referred as IKA.1 and IKA.2. The IKA.1 algorithm, depicted in Figure 2 in the introduction, trades off minimal round (and number of messages) complexity in return for higher computational cost. In contrast, IKA.2 minimizes computational cost at the expense of more protocol rounds (and more messages).

These two Initial Key Agreement protocols can be extended into algorithms for authenticated dynamic group Diffie-Hellman key exchange but IKA.1 is the only algorithm that was formally analyzed and modified to

achieve provable security. Such a formal treatment was carried out in the previous chapters and IKA.1 was renamed AKE1. We emphasize that IKA.1 and AKE1 have the same computational cost and the same number of rounds.

These protocols dynamically determine one of the members to serve as the group controller whose main task is to coordinate the generation of partial keys and to disseminate them to other group members. The group controller is always the newest (or most recent) group member. This selection criterion has an important benefit as it can be performed without any message exchange. Note that the concept of newest is not meaningful in an execution model where different processes observe group views in differents orders or with gaps. We postpone further discussion of this issue until Section 4.4.2.

4.4.1. *Performance Analysis*

The overall time-to-completion for IKA.1 and IKA.2 is dominated by two factors: network communications and cryptographic processing times; primarily, exponentiation with large numbers which is quite costly. In order to compare the costs of the two protocols we need to consider the steps of each protocol.

IKA.1 (Fig 2) operates in k rounds and requires $k-1$ unicast messages followed by a single broadcast. Each round i $(1 \leq i < k)$ involves each member M_i performing i exponentiations. This is followed by M_i unicasting a set of $(i+1)$ partial keys on to M_{i+1}, except for the last (k-th) round when M_{n+k} broadcasts the partial keys. Finally, each M_i performs a single exponentiation upon receipt of the broadcast. Assuming that all members exponentiate in approximately the same time, the total protocol delay is thus $COST(IKA.1)$. Similarly for IKA.2, the total protocol delay is $COST(IKA.2)$.

$$COST(IKA.1) = \frac{E}{2}\ k^2\ +\ (En + \frac{E}{2} + D)\ k\ +\ D$$

$$COST(IKA.2) = (2E + D)\ k\ + (En - E + 3D)$$

where D is the network delay, E the cost of a single exponentiation, n the number of members in the group and k the number of joining members.

We are now ready to compare the relationship between the cost of IKA.1 and the cost of IKA.2. We assume a 2ms exponentiation delay[4] and a 100ms wide-area network delay[5] and thus obtain the relation represented in Figure 3. The curve represents the values for which IKA.1 and IKA.2 have the same cost.

In the typical underlying group communication system most view changes require consensus among the new views. For this reason, the time to complete a membership change becomes prohibitive as the group size grows. This is particularly true when group members are spread across a wide-area network (WAN) since a WAN involves an increased round-trip time between group members and a greater likelihood of lost and thus resent messages due to the number of hops and sheer distances traversed.

We postulate that a practical limit for process group membership size in a wide-area network is likely to be around 40. In our experience, current scientific collaborations typically involve even smaller groups for example less than 20 members. Thus, most membership increases in a 20 member group are likely to involve a relatively small number of members merging into an existing group (either new members or heals of prior network partitions). Figure 3 clearly demonstrates that, under these assumptions, IKA.1 offers better performance than IKA.2.

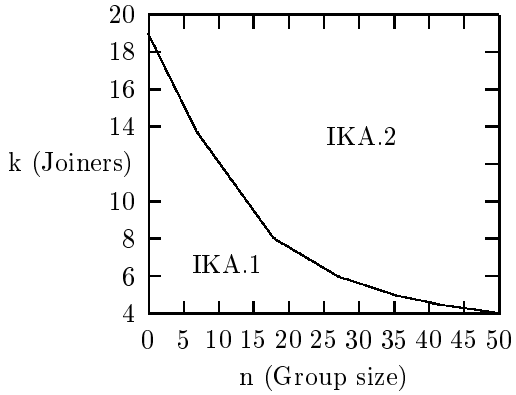

FIGURE 3. Tradeoff between group size and number of joining members. In the area below the curve, IKA.1 is faster than IKA.2.

[4]The performance for the 512-bit moduli exponentiation was obtained using the big number library in OpenSSL on a 450MHZ Pentium II PC.

[5]The average point-to-point delay for a US coast-to-coast round-trip at the application level.

As shown in Figure 3, on one extreme all 20 members join the collaborative session as soon as it starts. At $n = 0$ and $k = 20$, IKA.1 is as fast as IKA.2. Later, members may leave and join the group or the group may become partitioned and such a partioned group may merge. Suppose a 5 member group and a 15 member group were to merge. As shown in Figure 3 at $n = 5$, $k = 15$ and n=15, $k = 5$, IKA.1 is faster than IKA.2.

Figure 3 above is based on our current measurements for exponentiation and wide-area network delay. The curve in figure 3 will move up as the time for exponentiation goes down. In the future, due to faster computers, the exponentiation delay is more likely to decrease than the wide-area network delay. Moreover faster exponentiation algorithms exist; Hankerson et al. [HHM00] obtained an exponentiation delay of 1.5 ms[6] with a level of security equivalent to twice (i.e. 1024-bit security) the 512-bit security that we require for scientific collaboration software.

4.4.2. *Group Controller*

In the event of a network failure, a group may become partitioned into several disjoint components. These components may subsequently need to merge when the failure is repaired (i.e., a partition heals). However, this brings up the question of how to select the group controller following a merge event.

In our framework, the new group controller is selected as the prior controller of the largest merging sub-group (largest in terms of number of *authorized* members). Adding members from the smaller group into the larger one has some obvious advantages. As an example, consider the merge of a 5-member group and a 15-member group. Assuming all 20 members are previously authorized, a 2ms single exponentiation delay and a 100ms WAN round-trip delay, merging 5 members into a 15-member group costs 0.780 sec, while merging 15 members into a 5-member group costs 1.900 sec. More generally, $COST(IKA.1)$ grows linearly with n and is quadratic in k, thus, merging a smaller group into a larger one is always faster.

The problem now is how to agree on which group has the larger number of authorized members. Since the underlying group communication system is acting independently of SGLv1 its membership may be a superset of the secure group membership. Consequently, relative sizes of the merging

[6]Hankerson et al. [HHM00] obtain a 1.5ms exponentiation delay on NIST-recommended elliptic curves K-163 using the big number library in OpenSSL on a 400MHZ Pentium II PC.

secure groups cannot be determined from the information provided by the underlying group communication system. Additionally, there may even be view changes where one of the merging groups has no authorized members.

With a small modification, our flush protocol can provide the information about sizes of merging groups. Each member adds to the flush (*flush.view*) a list of the processes in its secure group communication session that are also in the new unsecure group view. Thus, on receipt of a flush message from each member in the new view, all members can determine the largest secure group and hence dynamically determine a member to serve as the merged group controller (using the process identifier to break ties).

5. Experimental Results

5.1. Prototype

A prototype implementation of the SGLv1 in the "C" programming language has been completed. It currently runs on Sun UltraSparc workstations with the Solaris 5.7 operating system. The Totem system [MMSA+96] is utilized as the underlying group communication system, the Akenti server [TJM+99] serves as the authorization server and the IKA.1 protocol has been implemented using the functions provided by the toolkit [ACH+00]. We also use the implementation of DSA provided by OpenSSL [Res01].

The Totem system [MMSA+96] provides all the properties required by SGLv1 and some additional properties such as totally ordered messages. The Totem system runs as a daemon and a light-weight user interface layer. The remote users connect via the light-weight layer to the Totem daemon using a TCP/IP connection - or through an SSL connection - across the high latency link since the Totem daemons were not designed to operate over a high latency link. One advantage of the Totem system is that it can be replaced, if needed, by its secure version called Immune [KMMS98] which is designed to protect against Byzantine failures.

The Akenti server issues the membership certificates used for group admission. For the sake of fault tolerance, it can be run as a set of mirrored servers. Note that Akenti can be administered independently. Akenti provides an interface to allow stakeholders to create digitally signed policy certificates.

5.2. Experiment

Initial performance tests with our prototype implementation were performed between sites in Berkeley, California (LBNL) and Argonne, Illinois (ANL). In these tests we measured the performance of the SGLv1 when one member joins the group (Fig 4), leaves the group, and group merge operation with various component sizes (Fig 6). In each case the graphs show the results from the worst case scenario (e.g. the joining member is separated from the group by a high-latency link).

We now describe our experiments. At Lawrence Berkeley Laboratory (Berkeley), one Sun UltraSparc 5s is running a Totem daemon and one group member. The second Sun UltraSparc 5 is running a Totem daemon and the rest of the Berkeley group members. At Argonne, a Sun UltraSparc 2 is running one user who connects to a Totem daemon at Berkeley. On a Sun UltraSparc 5, a 512-bit moduli exponentiation, DSA signature and DSA verification [7] provided by OpenSSL [Res01] costs respectively 0.010 seconds, 0.010 seconds and 0.030 seconds.

5.3. Measurements

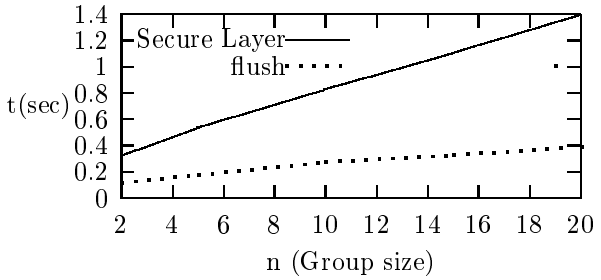

FIGURE 4. Performance of SGLv1 when one member located at Argonne joins the group.

Figure 4 shows the performance of SGLv1 when the member in Argonne joins an existing group in Berkeley. As an example, adding one member to a group of 19 members takes 1.4 seconds. Examining the steps after the flush protocol is finished, the group controller in Berkeley computes 19 exponentiations, signs the message and sends the values to the joining

[7] It is worth noting that DSA operations (i.e. signing and verification) are more symmetric than the RSA operations. RSA verification is roughly an order of magnitude faster than RSA signing and RSA signing is roughly as fast as DSA signing. For SGLv1, DSA is clearly a bottleneck.

member in Argonne; 0.200 seconds. Upon reception of the message, the joining member verifies the signature, computes 19 exponentiations, signs the message, and sends the values to the daemon in Berkeley, 0.303 seconds. At this point, the total is 0.503 seconds.

As soon as the daemon receives the message, it forwards it to the users. The user in Argonne gets the message after 0.070 seconds, verifies the signature, and computes one exponentiation; 0.110 seconds. The user in Argonne computes the group secret in a total time of 0.613 seconds. Adding the flush protocol we get 0.993 seconds. Each of the 18 users on the Sun UltraSparc 5 have to get the message, verify the signature and compute one exponentiation; $0.035 + 18 * 0.04 = 0.755$ seconds. Adding the cost of the flush protocol, the 1st member computes the group key in 0.958 seconds while the 18th member computes the group key in 1.638 seconds.

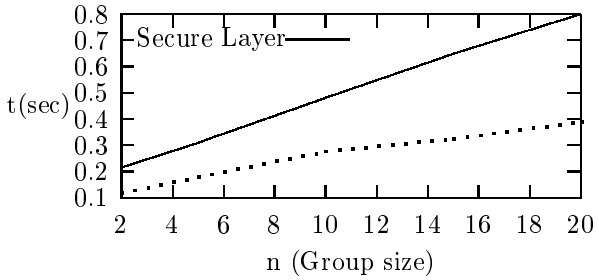

FIGURE 5. Performance of SGLv1 when one member leaves the group and the group controller is located at Argonne.

Figure 5 shows the performance of SGLv1 when one member leaves the group communication and the group controller is in Argonne. As an example, a member leaving an existing group with 20 members costs 0.8 seconds. Once the flush protocol completes, the group controller located in Argonne computes 19 exponentiations, signs the message and sends it to the daemon at Berkeley; 0.238 seconds. The user in Argonne gets the message after 0.070 seconds, verifies the signature, and computes one exponentiation; 0.110 seconds. By adding the flush protocol we get 0.728 seconds. For each of the 18 users that the Sun UltraSparc 5 contains, it has to get the message, verify the signature and compute one exponentiation; 0.035+18*0.04 = 0.755 seconds. By adding the cost of the flush protocol, the 1st member computes the group key in 0.693 seconds while the 18th member computes the group key in 1.373 seconds.

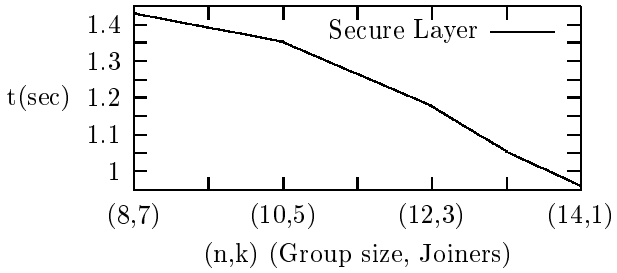

FIGURE 6. Performance of SGLv1 on a group merge with variable-size merging components. The main group size is constant at 15 members. The cost of the flush is not included.

Figure 6 shows the performance of the group merge operation with various partition sizes. As an example a group with k=3 members is added to an existing group with n=12 members. The 12 members in the existing group are on one computer and the other group has one member at Argonne and the other two on a computer in Berkeley. Once the flush protocol completes, the group controller of the larger group computes 12 exponentiations, signs the message and sends the value to the 1st member in the group with 3 members; 0.13 seconds. The 1st member receives the message, verifies the signature, computes 13 exponentiations, signs the message and sends it to the member located at Argonne; 0.17 seconds. The member located at Argonne receives the message, verifies the signature, computes 14 exponentiations, signs the message and sends it to the 3rd member; 0.215 seconds. The 3rd member receives the message, verifies the signature, computes 14 exponentiations, signs the message and sends it to the group; 0.215 seconds. At this point the total is 0.73 seconds. The member located in Argonne, computes the group secret in a total time of 0.805 seconds. Each of the first group's 12 members need to get the message, verify the signature and compute one exponentiation; 0.48 seconds. So, the 1st member computes the group key in 0.77 seconds while the 12th member computes the group key in 1.21 seconds. The other two members of the second group get the message, verify the signature and compute an exponentiation; 0.81 ms. The graph shows the average experimental value obtained for this group merge operation.

6. Conclusion

This paper presented the design of a secure group layer aimed at wide-area environments. The security layer offers protection against attacks like eavesdropping by integrating an access control mechanism, authenticated dynamic group Diffie-Hellman key exchange algorithm, and a reliable multicast transport protocol. The application information is passed securely inside the group since the group key agreement protocol is proved to securely distribute a session key among the legitimate group members only and the application messages are all encrypted. Even the lack of reliable and ordered delivery of messages will not disclose the application information.

If the reliable group communication system is corrupted, however, our secure group layer can no longer count on the reliable delivery of messages, which may cause the communication to be useless in some settings. These are denial of services attacks. To reduce the chances of the communication failing, our security layer could be used in conjunction with a group communication system resistant to Byzantine failures such as Immune [KMMS98].

Although the emphasis in building the first prototype SGLv1 was not on performance, this prototype has served as a good platform for understanding the issues involved in providing a secure group layer. These are robustness features, security issues, efficiency improvements and interface definitions. Significant performances improvement could for example be realized by exploiting faster platforms to speed-up the cryptographic operations and by using elliptic curve cryptography to implement the authenticated group Diffie-Hellman algorithm [HHM00]. More efficient support of groups spread across wide-area networks would also result from using recent group communication system intended for wide-area networks [BAMSM01].

Conclusion and Further Research

In this thesis we carried out a complete treatment of the authenticated group Diffie-Hellman key exchange using public/private keys. Through a number of incremental steps we extended the work of Bellare et al. [BPR00] on the two-party Diffie-Hellman key exchange to the multi-party setting. We first provided a formal treatment for the scenario wherein the group membership is static and then addressed the scenario wherein the group membership is dynamic. Our formal treatment consists of defining models to manage the complexity of security definitions in this area, and showing that the protocols for authenticated group Diffie-Hellman are secure within these models. We also enhanced our formal treatment with important missing security attributes. Finally, we put our protocols for authenticated group Diffie-Hellman to practical use within a security framework designed to be similar to the well-know SSL protocol.

As parties communicate according to a common set of protocols, many standards have been developed to provide a basis for interoperability of the different systems. For security/cryptographic protocols the standards serve as assurance of some level of safety and have an impact on their deployment. The IEEE P1363, IETF multicast security working group, IRTF secure multicast research group, ISO/IEC JTC1 SC27 and NIST organizations have developed comprehensible classifications of types of cryptosystems but have not yet developed a classification for multi-party cryptographic algorithms. A logical follow-on to the work carried out in this thesis is to extend our formal treatment to other group Diffie-Hellman-like protocols and to develop standards for multi-party cryptography to get them standardized.

Bibliography

[AAH+00] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik. Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments. In *Proc of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 330–343, April 2000.

[ACH+00] G. Ateniese, O. Chevassut, D. Hasse, Y. Kim, and G. Tsudik. The Design of a Group Key Agreement API. In *DARPA Information Survivability Conference and Exposition (DISCEX)*. IEEE Computer Society, Jan 2000.

[ACTT01] D. A. Agarwal, O. Chevassut, M.R. Thompson, and G. Tsudik. An Integrated Solution for Secure Group Communication in Wide-Area Networks. In *Proc. of the 6th IEEE Symposium on Computers and Communications (ISCC)*, pages 22–28, July 2001.

[ADS00] Y. Amir, C. Danilov, and J. Stanton. A Low Latency, Loss Tolerant Architecture and Protocol For Wide Area group Communication. In *Proc of the International Conference on Dependable Systems and Networks (FTCS)*, June 2000.

[AST98] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proc of the 5th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 17–26. ACM Press, November 1998.

[AST00] G. Ateniese, M. Steiner, and G. Tsudik. New Multiparty Authentication Services and Key Agreement Protocols. *IEEE Journal of Selected Areas in Communications (JSAC)*, 18(4), April 2000.

[BAC02] K. Berket, D. A. Agarwal, and O. Chevassut. A Practical Approach to the Intergroup Protocols. In *J.of Future Generation Computer Systems*, volume 18, pages 709–719, April 2002.

[BAMSM01] K. Berket, D. A. Agarwal, P. M. Melliar-Smith, and L. E. Moser. Overview of the InterGroup Protocols. In *Prof of the International Conference on Computational Science (ICCS)*, volume 2073 of *Lecture Notes in Computer Science*, pages 316–325. Springer-Verlag, May 2001.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In *Proc. of Crypto*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1996.

[BCK98] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols.

In *Proc. of the 30th Annual Symposium on the Theory of Computing (STOC)*. ACM Press, 1998.

[BCP01] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Group Diffie-Hellman Key Exchange – The Dynamic Case. In *Proc. of Asiacrypt*, volume 2248 of *Lecture Notes in Computer Science*, pages 290–309. Springer-Verlag, Dec 2001.

[BCP+02a] E. Bresson, O. Chevassut, O. Pereira, D. Pointcheval, and J.J. Quisquater. Two Views of Authenticated Group Diffie-Hellman Key Exchange. In *Proc of the Workshop on Cryptographic Protocols in Complex Environments*. DIMACS Center, Rutgers University, May 15 - 17 2002.

[BCP02b] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. In *Proc. of Eurocrypt*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer-Verlag, May 2002.

[BCP02c] E. Bresson, O. Chevassut, and D. Pointcheval. Goup Diffie-Hellman Key Exchange Secure Against Dictionary Attacks. In *Asiacrypt*, Dec 2002.

[BCP02d] E. Bresson, O. Chevassut, and D. Pointcheval. The Group Diffie-Hellman Problems. In *Selected Areas in Cryptography (SAC)*, August 2002.

[BCPQ01] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably Group Diffie-Hellman Key Exchange. In *Proc. of the 8th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 255–264. ACM Press, Nov 2001.

[BD95] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Proc of Eurocrypt*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1995.

[BGH+91] R. Bird, I. Gopal, A. Hertzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. In *Proc. of Crypto*, volume 576 of *Lecture Notes in Computer Science*, pages 44–61. Springer-Verlag, 1991.

[BJ87] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. In *ACM Transactions on Computer Systems*, volume 5(1), pages 47–76, February 1987.

[Ble98] D. Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1. In *Proc of Crypto*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, August 1998.

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *Proc of Third Algorithmic Number Theory Symposium (ANST)*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.

[BPR00] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Proc. of Eurocrypt*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

[BR93a] M. Bellare and P. Rogaway. Entity Authentification and Key Distribution. In *Proc. of Crypto*, volume 773 of *Lecture Notes in Computer Science*. Springer-Verlag, August 1993.

[BR93b] M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *Proc of the 1st ACM Conference*

on Computer and Communications Security, pages 62–73. ACM Press, 1993.

[BR95] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proc of the 27th ACM Symposium on the Theory of Computing (STOC)*, pages 57–66, 1995.

[BR96] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In *Proc of Eurocrypt*, volume 1070 of *Lecure Notes in Computer Science*, pages 399–416. Springer-Verlag, 1996.

[BW88] J. A. Buchmann and H. C. Williams. A Key-Exchange System Based on Imaginary Quadratic Fields. *Journal of Cryptology*, 2(2):107–118, 1988.

[BW98] K. Becker and U. Wille. Communication Complexity of Group Key Distribution (acm ccs). In *Proc of the 5th ACM Conference on Computer and Communications Security*, pages 1–6. ACM Press, November 1998.

[BWJM97] S. Blake-Wilson, D. Johnson, and A. Menezes. Key Agreement Protocols and their Security Analysis. In *Proc. of 6th IMA International Conference on Crypotography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer-Verlag, 1997.

[BWM98a] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. In *Proc. of the 5th Annual Workshop on Selected Areas in Cryptography (SAC)*, volume 1556 of *Lecure Notes in Computer Science*, pages 339–361. Springer-Verlag, 1998.

[BWM98b] S. Blake-Wilson and A. Menezes. Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques. In *Proc. of the 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 137–158. Springer-Verlag, 1998.

[CFIJ99] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. How to Forget a Secret. In *Symposium on Theoretical Computer Science (STACS)*, volume 1563 of *Lecture Notes in Computer Science*, pages 500–509. Springer-Verlag, 1999.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc of the 30th Symposium on the Theory of Computing (STOC)*, pages 209–218, 1998.

[CK74] V. G. Cerf and R. E. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):647–648, 1974.

[CKV01] G. V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys*, 33(4):pages 1–43, Dec 2001.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science, 1990.

[Cor00] J.-S. Coron. On the Exact Security of Full-Domain-Hash. In *Proc. of Crypto*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer-Verlag, 2000.

[CS98] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptative Chosen Ciphertext Attack. In *Proc of Crypto*, volume 1462 of *Lecure Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998.

[CS99] R. Cramer and V. Shoup. Signature Scheme based on the Strong RSA Assumption. In *Proc. of the 6th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 46–51. ACM Press, 1999.

[DH76] W. Diffie and M. Hellman. New Directions In Cryptography. In *IEEE Transactions on Information Theory*, volume IT-22(6), pages 644–654, November 1976.

[DR00] J. Daemen and V. Rijmen. The Rijndael Block Cipher. In *AES Proposal, NIST*, 2000.

[DR02] T. Dierks and E. Rescorla. The TLS Protocol Version 1.0. Internet Draft, September 2002.

[DvOW92] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchange. In *Designs, Codes and Cryptography*, volume 2, pages 107–125, 1992.

[FK98] I. Foster and C. Kesselman. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[FKK96] A. Freier, P. Karlton, and P. Kocher. *The SSL Protocol Version 3.0*, November 1996.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[FKTT98] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proc of the 5th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 83–92. ACM Press, 1998.

[FOPS01] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure under the RSA Assumption. In *Proc of Crypto*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, August 2001.

[GB01] S. Goldwasser and M. Bellare. Lecture Notes on Cryptography. Summer Course on Cryptography, Massachusetts Institute of Technology (MIT), August 2001. Available at http://www-cse.ucsd.edu/users/mihir/papers/gb.html.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):pages 792–807, October 1986.

[GM84] S. Goldwasser and S. Micali. Probabilisitic Encryption. *Journal of Computer and System Sciences*, 28(2):pages 270–299, April 1984.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure against Adaptive Chosen-message Attacks. *SIAM Journal of Computing*, 17(2):pages 281–308, April 1988.

[Gri] Access Grid. Available at http://www-fp.mcs.anl.gov/fl/accessgrid.

[Gun89] C. G. Gunter. An Identity-Based Key Exchange Protocol. In *Proc. of Eurocrypt*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer-Verlag, 1989.

[HHM00] D. Hankerson, J. Hernandez, and A. Menezes. Software Implementation of Elliptic Curve Cryptography over Binary Fields. In *Proc of Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, August 2000.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A Pseudorandom Generator from any One-Way Function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[ITW82] I. Ingemarsson, D. Tang, and C. Wong. A Conference Key Distribution System. In *IEEE Transactions on Information Theory*, volume 28(5), pages 714–720, September 1982.

[Jou00] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Proc of the Algorithmic Number Theory Symposium (ANTS)*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2000.

[JQ97] M. Joye and J.J. Quisquater. On the Importance of Securing your Bins: The Garbage-Man-in-the-Middle Attack. In *Proc of the 4th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 135–141. ACM Press, 1997.

[JV96] M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. In *Proc. of Asiacrypt*, volume 1163 of *Lecture Notes in Computer Science*, pages 36–49. Springer-Verlag, 1996.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, Feburary 1997.

[KMMS98] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *Proc of the IEEE 31st Hawaii International Conference on System Sciences (HICSS)*, pages 317–326, Jan 1998.

[KMV00] N. Koblitz, A. Menezes, and S. Vanstone. The State of Elliptic Curve Cryptography. In *A Special Issue of Designs, Codes, and Cryptography: Towards a Quater-Century of Public-Key Cryptography*, volume 19, pages 103–193. Kluwer Academic Publishers, March 2000.

[Kob98] N. Koblitz. *Algebraic Aspects of Cryptography*, volume 3 of *Algorithms and Computation in Mathematics*, chapter 6, pages 117–154. Springer-Verlag, July 1998.

[KPT00] Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Group. In *Proc. of the 7th ACM Conference on Computer and Communications Security (ACM CCS)*, November 2000.

[KPT01] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proc. of International Federation for Information Processing (IFIP SEC)*, June 2001.

[MAMSA94] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended Virtual Synchrony. In *Proc of the 14th IEEE International Conference on Distributed Computing Systems (ICDS)*, pages 56–65, June 1994.

[Mea00] C. Meadows. Extending Formal Cryptographic Protocol Analysis Techniques for Group Protocols and Low-Level Cryptographic Primitives. In *Workshop on Issues in the Theory of Security (WITS)*, July 2000.

[MMR97] D. Malkhi, M. Merritt, and O. Rodeh. Secure Reliable Multicast Protocols in a WAN. In *Proc of the International Conference on Distributed Computing Systems (ICDCS)*, pages 87–94, May 1997.

[MMSA+96] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, pages 54–63, April 1996.

[MPH99] P. D. McDaniel, A. Prakash, and P. Honeyman. Antigone: A Flexible Framework for Secure Group Communication. In *USENIX Security Symposium*, pages 99–114, August 1999.

[MW00] U. Maurer and S. Wolf. The Diffie-Hellman Protocol. In *A Special Issue of Designs, Codes, and Cryptography: Towards a Quater-Century of Public-Key Cryptography*, volume 19, pages 77–171. Kluwer Academic Publishers, March 2000.

[MY99] A. Mayer and M. Yung. Secure Protocol Transformation via "Expansion" from Two-Party to Multi-Party. In *Proc of the ACM Conference on Computer and Communications Security (ACM CCS)*, pages 83–92. ACM Press, November 1999.

[NIS00] NIST. Advanced encryption standard, December 2000. `http://www.nist.gov/aes`.

[NR97] M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 458–467, 1997.

[Per99] A. Perrig. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC)*, July 1999.

[Poi01a] D. Pointcheval. Practical Security in Public-Key Cryptography. In *Proc of the 4th International Conference on Information Security and Cryptology (ICISC)*, volume 2288 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, Dec 2001.

[Poi01b] D. Pointcheval. Secure Designs for Public-Key Cryptography based on the Discrete Logarithm. *To appear in Discrete Applied Mathematics*, Elsevier Science, 2001.

[PQ01a] O. Pereira and J. J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *Proc of the 14-th IEEE Computer Security Foundations Workshop*, pages 73–81. IEEE Computer Society Press, June 2001.

[PQ01b] O. Pereira and J. J. Quisquater. A Security Analysis of the Cliques Protocols Suites: 1st. In *Proc of the International Federation for Information Processing (IFIP Sec)*, pages 151–166. Kluwer Publishers, June 2001.

[PS00] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *J. of Cryptology*, 13(3):361–396, 2000.

[PSW98] E. R. Palmer, S. W. Smith, and S. Weingart. Using a Hight-Performance, Programmable Secure Coprocessor. In *Proc of Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 73–89. Springer-Verlag, 1998.

[RBH+98] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev. Ensemble security. Technical Report TR98-1703, Cornell, Sept 1998.

[Rei94] M. K. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proc of the 1st ACM Conference on Computer and Commmunications Security (ACM CCS)*. ACM Press, Nov 1994.

[Rei96] M. K. Reiter. Secure Group Membership Protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, January 1996.

[Res01] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001. See also the OpenSSL Project at http://www.openssl.org.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sho97] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Proc. of Eurocrypt*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.

[Sho99] V. Shoup. On Formal Models for Secure Key Exchange. Technical report, IBM Research Report RZ3120, 1999.

[Sho01] V. Shoup. OAEP Reconsidered. In *Proc of Crypto*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259. Springer-Verlag, August 2001.

[SL01] M. Stam and A. K. Lenstra. Speeding-up XTR. In *Proc of Asiacrypt*, volume 2248, pages 125–143. Lecture Notes in Computer Science, Springer-Verlag, Dec 2001.

[SNS98] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open networks systems. In *Usenix Winter Conference*, pages 191–202, Jan 1998.

[SPW02] M. Steiner, Birgit Pfitzmann, and Michael Waidner. A formal model for multi-party group key agreement. Technical report, Research Report RZ 3383 IBM Research, 2002.

[SR96] V. Shoup and A. Rubin. Session-Key Distribution using Smart Cards. In *Eurocrypt '96*, Lecture Notes in Computer Science, pages 321–331. Springer-Verlag, 1996.

[SSDW88] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A Secure Audio Teleconference System. In *Proc. of Crypto*, volume 403 of *Lectures Notes in Computer Science*, pages 520–528. Springer-Verlag, 1988.

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Groups. In *Proc of the 3rd ACM Conference on Computer and Communications Security (ACM CCS)*, pages 31–37. ACM Press, March 1996.

[STW00] M. Steiner, G. Tsudik, and W. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, August 2000.

[TJM+99] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based Access Control for Widely Distributed Resources. In *Usenix Security Symposium*, pages 215–227, August 1999.

[Tze00] Wen-Guey Tzeng. A Practical and Secure Fault-Tolerant Conference-Key Agreement Protocol. In *Proc. of Public-Key Cryptography (PKC)*, volume 1751 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, January 2000.

[U. 94] U. S. National Institute of Standards and Technology. *Federal Information Processing Standards Publication 140-1: Security Requirements for Cryptographic Modules*, Jan 1994.

[VKCD99] R. Vitenberg, I. Keidar, G. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. Technical Report MIT-LCS-TR-790, MIT, Sep 1999.

[vO93] P. C. van Oorschot. Extending Cryptographic Logics of Belief to Key Agreement. In *Proc. of the 1st ACM Conference on Computer and Communications Security (ACM CCS)*, pages 232–243. ACM Press, 1993.

[VW97] K. Vedder and F. Weikmann. Smart Cards Requirements, Properties, and Applications. In *State of the Art in Applied Cryptography*, volume 1528 of *Lecture Notes in Computer Science*, pages 307–331. Springer-Verlag, 1997.

[Wei00] S. H. Weingart. Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses. In *Proc of Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 302–317. Springer-Verlag, 2000.

[Zim95] P. Zimmerman. *The Official PGP User's Guide*. The MIT Press, 95.