Secure Group Key Agreement

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr. Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von

Michael Steiner

Gutachter:
Prof. Dr. Birgit Pfitzmann
Prof. Dr. Gene Tsudik

Dekan:
Prof. Dr. Rainer Schulze-Pillot-Ziemen

Einreichung:
20 Dezember, 2001

Promotions-Kolloquium:
15. März, 2002

Saarbrücken, 2002

Abstract

As a result of the increased popularity of group-oriented applications and protocols, group communication occurs in many different settings: from network multicasting to application layer tele- and video-conferencing. Regardless of the application environment, security services are necessary to provide communication privacy and integrity.

This thesis considers the problem of key management in a special class of groups, namely, dynamic peer groups. Key management, especially in a group setting, is the corner stone for all other security services. Dynamic peer groups require not only initial key agreement but also auxiliary key agreement operations such as member addition, member exclusion and group fusion. We discuss all group key agreement operations and present a concrete protocol suite, CLIQUES, which offers all of these operations. By providing the first formal model for group key establishment and investigating carefully the underlying cryptographic assumptions as well as their relations, we formally prove the security of a subset of the protocols based on the security of the Decisional Diffie-Hellman assumption; achieving as a side-effect the first provably secure group key agreement protocol.

Kurzzusammenfassung

Mit der Verbreitung offener Netze, insbesondere des Internets, fand auch die Gruppenkommunikation eine rasante Verbreitung. Eine Vielzahl heutiger Protokolle sind gruppen-orientiert: angefangen bei Multicast-Diensten in der Netzwerkschicht bis hin zu Videokonferenzsystemen auf der Anwendungsschicht. Alle diese Dienste haben Sicherheitsanforderungen wie Vertraulichkeit und Integrität zu erfüllen, die den Einsatz kryptographischer Techniken und die Verfügbarkeit gemeinsamer kryptographischen Schlüssel oft unumgänglich machen.

In der folgenden Doktorarbeit betrachte ich dieses grundlegendste Problem der Gruppenkommunikation, nämlich das Schlüsselmanagement, für dynamische Gruppen, die sogenannten "Dynamic Peer-Groups". Die Dynamik dieser Gruppen erfordert nicht nur initialen Schlüsselaustausch innerhalb einer Gruppe sondern auch sichere und effiziente Verfahren für die Aufnahme neuer und den Ausschluß alter Gruppenmitglieder. Ich diskutiere alle dafür notwendigen Dienste und präsentiere CLIQUES, eine Familie von Protokollen, die diese Dienste implementiert. Ich gebe erstmalig eine formale Definition für sicheres Gruppen-Schlüsselmanagement und beweise die Sicherheit der genannten Protokolle basierend auf einer kryptographischen Standardannahme, der "Decisional Diffie-Hellman" Annahme. Diese Sicherheitsbetrachtung wird durch eine detaillierte Untersuchung dieser Annahme und ihrer Relation zu verwandten Annahmen abgeschlossen.

Zusammenfassung

Der zunehmende Bedarf an gruppenorientierten Anwendungen und Protokollen hat in den letzten Jahren zu einer enormen Verbreitung der Gruppenkommunikation in verschiedensten Bereichen geführt: angefangen bei Multicast-Diensten in der Netzwerkschicht bis hin zu Videokonferenzsystemen auf der Anwendungsschicht. Die Gewährleistung von Sicherheitsgarantien wie Vertraulichkeit, Authentizität und Integrität sind dabei wichtige Eigenschaften von Gruppenkommunikation, vor allem um die notwendige Akzeptanz auf der Anwenderseite zu erreichen.

Während Peer-to-Peer Sicherheit ein relativ erwachsenes und gut erforschtes Gebiet ist, stellt die sichere Gruppenkommunikation noch immer eine ziemlich unerforschte Landschaft dar. Entgegen einer weit verbreiteten Fehleinschätzung, ist sichere Gruppenkommunikation keine triviale Erweiterung sicherer Zwei-Parteien-Kommunikation. Es gibt eine Vielzahl gravierender Unterschiede und sichere Gruppenkommunikation stellt noch immer zahlreiche Herausforderungen an die Forschungsgemeinde (vgl. Smith and Weingarten (1997) und Canetti et al. (1999)).

An dieser Stelle seien nur einige Unterschiede und Probleme erwähnt: Aufgrund ihrer zahlreichen und sehr unterschiedlichen Einsatzgebiete ist es sehr schwer eine allgemeingültige und konsistente Definition für Gruppen-Kommunikation zu finden. So haben beispielsweise Gruppen, die für eine Video-on-Demand Anwendung gebildet wurden, grundlegend andere Sicherheitsanforderungen als dynamische, spontan gebildete Peer-Gruppen in drahtlosen ad-hoc Netzwerken. Folglich werden Taxonomien und Klassifizierungskriterien benötigt, um Problemklassen und ihre Sicherheitsanforderungen zu identifizieren und zu definieren.[1] Noch wichtiger ist es die Sicherheit grundlegender Dienste, wie beispielsweise Authentifikation, formal zu definieren, da ohne fundierte und formale Sicherheitsdefinitionen, die Sicherheit der zugrundeliegenden Protokolle nicht rigoros bewiesen werden kann.

Ein zweiter Unterschied liegt in der größeren Bedeutung der Rechen- und Kommunikationskomplexität der Protokolle, da diese in direkter Abhängigkeit zur Anzahl der Teilnehmer steht. Desweiteren sind Topologie und Cha-

[1] Erste Schritte zur Charakterisierung sicherer Gruppenkommunikation wurden bereits in Hutchinson (1995) und Canetti et al. (1999) diskutiert, jedoch nur als sehr abstrakte und informelle Taxonomien.

rakteristik des Netzwerkes bedeutende Faktoren für das Design und die Auswahl geeigneter Protokolle.

Ein weiterer Unterschied liegt in der Dynamik der Gruppen: Zwei-Parteien-Kommunikation kann als ein diskretes Phänomen betrachtet werden: es beginnt, hat eine bestimmte Dauer und endet wieder. Gruppenkommunikation ist komplexer: die Gruppe wird gebildet, kann sich durch Ein- oder Austritt von Teilnehmern ändern und es gibt nicht notwendigerweise ein fest definiertes Ende. Diese Dynamik macht die Garantie von Sicherheitseigenschaften wesentlich aufwendiger und erschwert insbesondere auch das Schlüsselmanagement.

Die Lösung all dieser Fragen würde den Rahmen einer Doktorarbeit bei weitem sprengen. Daher betrachte ich in dieser Arbeit das grundlegendste Problem auf dem alle weiteren Sicherheitsmechanismen der Gruppenkommunikation aufbauen, nämlich das Schlüsselmanagement. Desweiteren beschränke ich mich auf eine spezielle Klasse von Gruppen, die Dynamischen Peer-Gruppen (DPG). DPGs sind Gruppen deren Mitglieder in symmetrischen Relationen zueinander stehen und daher als äquivalent bzw. gleichwertig behandelt werden. Insbesondere sind spezielle Rollen wie Gruppen-Koordinator nicht von vornherein fixiert, d.h. es gibt keine zentrale Instanz, die mehr Möglichkeiten als andere Gruppenmitglieder hat. Eine Zuweisung dieser speziellen Rollen sollte nur von der (möglicherweise variablen) Sicherheitsstrategie abhängen und unabhängig von dem Schlüsselmanagementprotokoll sein. Die Gruppenzugehörigkeit ist dynamisch, d.h. jeder Teilnehmer, insbesondere auch der aktuelle Gruppen-Koordinator, sollte sich prinzipiell einer Gruppe anschließen, oder diese auch verlassen können. Diese anspruchsvollen Eigenschaften heben DPGs von der zur Verteilung digitaler Multimediainhalte üblichen Multicast-Gruppen ab und machen sie zu einem interessanten Studienobjekt. DPGs sind in vielen Netzwerkschichten und Anwendungsgebieten üblich. Beispiele umfassen replizierte Server aller Bereiche (wie Datenbank-, Web- oder Zeitserver), Audio- und Videokonferenzsysteme, Battlefield-Netze oder kooperationsunterstützende Anwendungen aller Art. Im Gegensatz zu großen Multicast-Gruppen sind DPGs relativ klein. Größere Gruppen auf Peer-Basis sind sehr schwierig zu kontrollieren und werden daher meist hierarchisch organisiert. DPGs besitzen im allgemeinen auch ein many-to-many Kommunikationsmuster statt der üblichen one-to-many Kommunikation in großen hierarchischen Gruppen.

Überblick

Die Doktorarbeit ist wie folgt aufgebaut:

In den Kapiteln 1 und 2 gebe ich eine Einführung in die Thematik und analysiere notwendige Anforderungen an die Schlüsselverwaltung, um die Dynamik von DPGs zu unterstützen.

Die Basis für eine rigorose Sicherheitsanalyse lege ich in Kapitel 3, in

dem ich die notwendigen fundamentalen mathematischen Aspekte untersuche. Dabei betrachte ich insbesondere kryptographische Annahmen, die auf diskreten Logarithmen aufbauen, klassifiziere sie und diskutiere wichtige Eigenschaften und Parameter, deren Veränderung unterschiedliche Varianten dieser Annahmen implizieren. Zusätzlich beweise ich mehrere Relationen zwischen unterschiedlichen Annahmen, welche sich in späteren Sicherheitsbeweisen für die Protokolle zum Gruppen-Schlüsselaustausch als hilfreich erweisen werden. Insbesondere wird die Äquivalenz des Decisional Generalized Diffie-Hellman Problems und des Decisional Diffie-Hellman Problems konstruktiv bewiesen, indem eine effiziente Reduktion zwischen beiden Problemen in einer Vielzahl von Annahmenformulierungen angegeben wird. Desweiteren zeige ich, wie Bit-Strings aus Diffie-Hellman Schlüsseln erzeugt werden können, so daß diese Strings ununterscheidbar von gleichverteilten Strings sind.

Kapitel 4 zeigt CLIQUES, eine vollständige Familie von Protokollen zur Schlüsselverwaltung in Netzen mit authentischen Verbindungen. Diese umfaßt Protokolle zum initialen Schlüsselaustausch, zur Schlüsselerneuerung und zur Änderung von Gruppenzugehörigkeiten. Das Kapitel schließt mit einer Untersuchung der Eigenschaften und Effizienz dieser Protokolle sowie einigen Argumenten zum Beweis ihrer Sicherheit.

Diese Sicherheitsargumente entsprechen vom Formalitätsgrad her den Sicherheitsbeweisen existierender Gruppen-Schlüsselaustausch Protokolle. In Kapitel 5 gehe ich weit über dieses bisher übliche Maß an Formalität hinaus. Dazu definiere ich zunächst ein formales Modell für Gruppen-Schlüsselaustausch Protokolle und zeige einen detailierten und rigorosen Sicherheitsbeweis eines der CLIQUES Protokolle zum initialen Schlüsselaustausch. Insbesondere zeige ich, daß das Protokoll sogar gegen adaptive Angreifer unter der Decisional Diffie-Hellman Annahme sicher ist, wenn das Protokoll um eine Bestätigungsnachricht erweitert wird.

Die Arbeit schließt in Kapitel 6 mit einer Zusammenfassung der vorgestellten Ergebnisse und einem Ausblick auf offene Probleme und mögliche weitere Forschungsrichtungen.

Ergebnisse

Die Hauptresultate dieser Arbeit können wie folgt zusammengefaßt werden:

1. Die erste detaillierte Klassifizierung kryptographischer Annahmen basierend auf diskreten Logarithmen wird vorgestellt. Diese Klassifizierung erlaubt eine präzise und dennoch allgemeine Darstellung dieser Annahmen und liefert neuartige Einsichten in die Zusammenhänge zwischen diesen Annahmen. So wurden ausgehend von dieser Klassifizierung überraschende Ergebnisse hinsichtlich der Separierbarkeit von Annahmen in Abhängigkeit des zugrundeliegenden Wahrscheinlichkeitsraumes erzielt (Sadeghi and Steiner 2001).

2. Ein neues Problem, das Decisional Generalized Diffie-Hellman Problem, wird eingeführt und konstruktiv als äquivalent zum Decisional Diffie-Hellman Problem bewiesen, wobei der Beweis auch die konkrete Sicherheit, d.h., die genauen Reduktionskosten liefert. Das Problem bzw. die zugehörige Annahme ist nicht nur im Kontext von Diffie-Hellman-basierten Gruppen-Schlüsselaustausch Protokollen nützlich, sondern dient auch als Basis für die erste effiziente Konstruktion einer beweisbar sicheren Pseudo-Zufallfunktion (Naor and Reingold 1997).

3. CLIQUES, eine Familie flexibler Schlüsselmanagementprotokolle für dynamische Peer-Gruppen, wird eingeführt. Sie sind die ersten kollusionstoleranten[2] Protokolle, die keinen festen Gruppen-Koordinator voraussetzen. Die Protokolle sind optimal oder zumindest nahezu optimal bezüglich verschiedenster Leistungsmerkmale.

4. Die erste formale Definition von sicherem Gruppen-Schlüsselaustausch wird präsentiert. Ausgehend von dieser Definition wird die Sicherheit zweier effizienter Gruppen-Schlüsselaustausch Protokolle auf Netzwerken mit authentischen Verbindungen bewiesen. Dadurch wird eine wichtige Lücke in der Sicherheitsanalyse von Protokollen zum Gruppen-Schlüsselmanagement geschlossen und das Vertrauen in derartige Protokolle entsprechend erhöht. Ein weiterer Vorteil dieser Definition ist, daß sie die sichere modulare Kombination mit anderen Protokollen ermöglicht. Als Spezialfall liefert sie gleichzeitig auch die erste Definition für ein sicheres, modular kombinierbares Schlüsselaustausch Protokoll für den Zwei-Parteien-Fall.

Alle oben genannten Ergebnisse wurden bereits in Vorversionen veröffentlicht. Die erste Publikation ist Steiner, Tsudik, and Waidner (1996), welche die Grundlage für die Protokollfamilie CLIQUES legte, das Decisional Generalized Diffie-Hellman Problem einführte sowie einen ersten, nicht-konstruktiven Äquivalenzbeweis zwischen dem Generalized Diffie-Hellman Problem und dem Decisional Diffie-Hellman Problem enthielt. Die dynamischen Aspekte von Gruppen-Schlüsselaustausch wurden in Steiner, Tsudik, and Waidner (1998) beleuchtet. Diese Publikation führte auch die ersten Gruppen-Schlüsselaustausch-Protokolle ein, die kollusionstolerant sind und dynamische Gruppen-Koordinatore erlauben. Diese Papiere wurden zu einer erweiterten Journalversion (Steiner, Tsudik, and Waidner 2000) kombiniert. Die Untersuchung und Klassifizierung der kryptographischen Annahmen, wie in Kapitel 3 gezeigt, basiert auf Sadeghi and Steiner (2001, 2002). Das formale Modell von Group-Key-Agreement und die zugehörigen Beweise wurden in Pfitzmann, Steiner, and Waidner (2002) veröffentlicht.

[2] Kollusionen sind Koalitionen von unehrlichen Teilnehmern.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

This chapter gives an outline of the content of this thesis. In particular, it provides a summary of the major results: The first provably secure key agreement protocol for dynamic peer groups and a thorough study and classification of the underlying cryptographic assumptions.

AS a result of the increased popularity of group-oriented applications and protocols, group communication occurs in many different settings: from network layer multicasting to application layer tele- and video-conferencing. Regardless of the underlying environment, security services are necessary to provide authenticity, integrity and communication privacy.

While peer-to-peer security is a quite mature and well-developed field, secure group communication remains comparably unexplored. Contrary to a common initial impression, secure group communication is not a simple extension of secure two-party communication. There are important differences and many research challenges remain open as pointed out by Smith and Weingarten (1997) and Canetti, Garay, Itkis, Micciancio, Naor, and Pinkas (1999). In the following, I just mention a few.

First, there are a number of definitional problems, as group communication comes in various and fundamentally different forms. For example, groups as formed during a video-on-demand multicast expose quite different security requirements than requirements of dynamic peer groups in ad-hoc wireless networks. This means that we need taxonomies and classifications to characterize problem classes and their requirements.[1] However and even more importantly, basic services such as authentication need formal defi-

[1] Some initial steps towards such a characterization of group communication and security were already taken in the high-level and informal taxonomies of Hutchinson (1995) and Canetti et al. (1999).

nitions of security. Without such definitions we can never get a thorough confidence in proposed protocols as there is no way of rigorously and formally proving their security.

Secondly, protocol efficiency is of greater concern due to the direct relation of the number of participants with computation and communication complexity. Network topologies and characteristics are key issues for the design and selection of appropriate protocols.

A third difference is due to group dynamics. Two-party communication can be viewed as a discrete phenomenon: it starts, lasts for a while and ends. Group communication is more complicated: the groups starts, it might mutate (members leave and join) and there might not be a well-defined end. This complicates attendant security services, in particular for key management.

To tackle all these question would go far beyond the scope of a single thesis. In this work, I specifically focus on key management, the corner stone of the security services, and on **Dynamic Peer Groups (DPG)**. DPGs are groups where all members have a symmetric relationship and are treated equivalently. In particular, roles such as group controllership are not a priori fixed, i.e., there is no central authority with more power than other members, and assignment of such roles should be only a matter of (potentially variable) policy and orthogonal to the key management protocols. Furthermore, membership is highly dynamic, i.e., any member might join or leave, including a member who holds at that moment the role of a group controller. This makes DPGs an interesting object of study and separates them, e.g., from multicast groups used in multimedia distribution services. DPGs are common in many layers of the network protocol stack and many application areas of modern computing. Examples of DPGs include replicated servers (such as database, web, or time servers), audio and video conferencing applications, battlefield networks, and, more generally, collaborative applications of all kinds. In contrast to large multicast groups, DPGs are relatively small in size, on the order of a hundred members. Larger groups are harder to control on a peer basis and are typically organized in a hierarchy of some sort. DPGs typically assume a many-to-many communication pattern rather than one-to-many commonly found in larger, hierarchical groups.

1.1 Outline

The reminder of this thesis is as follows:

In Chapter 2, I discuss and analyze the requirements for key management in supporting the dynamics of DPGs.

Laying the ground for a rigorous analysis, I then investigate in Chapter 3 the mathematical foundations. I take a closer look at cryptographic assumptions based on discrete logarithms, and classify and discuss impor-

tant properties differentiating variants of such assumptions. Furthermore, I prove a tool box of relations among different assumptions. This tool box will be helpful in later proving the security of the group key agreement protocols. In particular, I investigate the relation of the Decisional Generalized Diffie-Hellman problem and the Decisional Diffie-Hellman problem. I show constructively that there is an efficient reduction equating the difficulty of the two problems in a variety of assumption formulations. Furthermore, I show how to derive bit strings from Diffie-Hellman keys such that these bit strings are computationally indistinguishable from uniformly chosen ones.

Chapter 4 presents CLIQUES, a complete family of protocols for key-management, namely, initial key agreement, key-renewal and membership change, in a model with authenticated links. I analyze properties and efficiency of these protocols and give arguments for their security.

The security arguments given in the previous section are not very formal. Nevertheless, they represent the practice of proving security for group key protocols in the past. In Chapter 5 I go beyond that. I define a formal model for group key agreement protocols and give a detailed and rigorous proof for one of the previously presented protocols, the initial key agreement. In particular, I show that under the Decisional Diffie-Hellman assumption and the addition of a confirmation flow the initial key agreement protocol is secure even in the presence of adaptive adversaries.

Finally in Chapter 6, I summarize the work and give an outlook on open problems and possible research directions.

1.2 Results

The major results of this thesis are as follows:

1. This thesis contains the first thorough classification of cryptographic assumptions related to discrete logarithms. This classification enables concise yet general assumption statements and gives novel insights into the relation of these assumptions, e.g., based on it Sadeghi and Steiner (2001) showed a surprising separability result.

2. A new problem, the Decisional Generalized Diffie-Hellman problem, is introduced and shown equivalent to the Decisional Diffie-Hellman problem with a constructive reduction giving the concrete security. This problem, or more precisely the related assumptions, is very useful in the context of Diffie-Hellman based group key agreement protocols. Additionally, it also serves as the basis of the first efficient construction of provably secure pseudo-random functions (Naor and Reingold 1997).

3. CLIQUES, a family of flexible key-management protocols for dynamic peer groups is presented. They are the first collusion-tolerant proto-

cols without the need for a fixed group controller. The protocols are optimal or close to optimal in a number of metrics.

4. The first formal definition of secure group key management is given. Based on this definition, the security of an efficient group key agreement protocol is proven for networks which provide authenticated links. This closes an important gap in the security analysis of group key management protocols and increases sharply the resulting confidence in such protocols. Furthermore, the definition allows the secure composition with other protocols and — when restricting the number of parties to two — gives even the first definition and provably secure key agreement protocol exhibiting such a property in the two-party case.

Most of above results were already published in preliminary form in a number of previous publications. The paper trail begins with Steiner, Tsudik, and Waidner (1996). This paper laid the ground to the protocol family CLIQUES, introduced the Decisional Generalized Diffie-Hellman problem and gave the first though non-constructive proof of the equivalence of the Decisional Diffie-Hellman problem and the Decisional Generalized Diffie-Hellman problem. The work was continued in Steiner, Tsudik, and Waidner (1998), a paper which discussed the dynamic aspects of group key agreements and proposed the first protocol family for group key management which is collusion-tolerant and allows dynamic group controllers. These two papers were combined into an extended journal version (Steiner, Tsudik, and Waidner 2000). The study and classification of assumptions presented in Chapter 3 is based on Sadeghi and Steiner (2001, 2002). Finally, the formal model and the corresponding proofs are published in Pfitzmann, Steiner, and Waidner (2002).

Acknowledgements

Naturally, my first thanks go to the coauthors of the papers which form the basis of my thesis and are mentioned above: Ahmad-Reza Sadeghi, Birgit Pfitzmann, Michael Waidner and Gene Tsudik. Furthermore, a lot of thanks also belong to the long list of all other coauthors of publications of mine and coworkers on projects such as SAMSON, SEMPER, *i*KP or CLIQUES. All your cooperation was very helpful for me in finding my way in research and, besides, was a lot of fun (How boring would research be if everything would have to be done alone :-).

For my scientific career three people have been most influential: From my early time with IBM Research, Gene Tsudik and Michael Waidner were great mentors of mine. While Gene showed me with lots of humor how much fun research can be and largely stimulated my curiosity on the many

facets of research, I am indebted to Michael for teaching me that theory and formality should not be overlooked and for convincing me to pursue a Ph.D. Finally, Birgit Pfitzmann provided me with the opportunity to do a Ph.D. in Saarbrücken and showed me how fascinating it can be if you look at the details. Without all of you this work would have certainly been impossible. Thanks a lot!

Special credits go to the colleagues with whom I had the pleasure to share an office for a longer time at IBM and in Saarbrücken: Ralf, thanks for sharing with me your enthusiasm for research and for turning ideas to realities; Asokan, thanks for being a good friend and a great and sincere collaborator (although your struggle to improve my English was probably not so successful. Well, the reader might judge this best herself while trying to digest this text ...); Ahmad, thanks for being a good friend as well as a fun person to be with, and for pushing me to be more goal-oriented (except that in respect to the goal "PhD" you were not always that successful. However, this might not be such a surprise since in this particular aspect your goal-orientation is also not completely perfect when applied to yourself...:-).

If this thesis (hopefully) ended up being understandable and not being garnished with too many errors, it is certainly due to the detailed comments, the careful proof reading and the help with the german part (swiss-german, my mother tongue, is in subtle ways different to "real" german as I had to learn the hard way round when I moved to Germany ...) of Ahmad, André,[2] Birgit and Gene. Thank you for your help!

Not to forget are all my colleagues in Saarbrücken: Ahmad, Alexander, Ammar, André, Chris, Matthias, Michael, Petra, and Sandra. Thank you for providing me with a nice and inspiring environment.

Furthermore, I like to express my gratitude towards my doctorate committee, namely to Reinhard Wilhelm, Birgit Pfitzmann, Gene Tsudik, and Stefan Funke.

Doing research is fun but does not directly pay the bills of your meals and your apartment. Therefore, I am very grateful for the financial support I received during my doctoral studies from the Graduiertenkolleg "Effizienz und Komplexität von Algorithmen und Rechenanlagen" and from the EU ITS project Maftia (Malicious- and Accidental-Fault Tolerance for Internet Applications).

Last but not least, I like to thank my parents who supported me through all my life and without whom I would certainly not stand where I stand today.

[2] By the way, André is author of a number of interesting publications on proof of ownership of digital content (Adelsbach 1999; Adelsbach et al. 2000; Adelsbach and Sadeghi 2001). Unfortunately, nobody seems to reference this work. I hope my citations will correct now this glaring injustice :-)

Chapter 2

Dimensions of Key Agreement

In this chapter, I investigate key management in the context of group communication. In particular, I introduce and define the required services such as initial key agreement (IKA) at the time of the group genesis and the auxiliary key agreement (AKA) services (key renewal, membership change) required later in the life time of a group. A special focus will be on dynamic peer groups and the question what environment can be expected and what properties are desired from key management protocols. I also discuss the particular metrics (e.g., for communication complexity) used in the sequel.

AUTHENTICATION and key establishment is the cornerstone of any secure communication. Without some form of authentication, all the other common security properties such as integrity or confidentiality do not make much sense.

Authentication is generally based on **long-term keys** which can be associated with identities. Note that the term "long-term key" is usually very broad and covers all forms of information which can be linked to identities. For example, it not only includes cryptographic keys such as DES (NIST 1999) or RSA (Rivest et al. 1978)[1] keys but also encompasses passwords and biometric information. However, in the sequel I will assume that cryptographic keys are the only form of long-term keys as usually done in the context of group key establishment. On the one hand, passwords require special treatment (Katz et al. 2001; Steiner et al. 2001) due to their low en-

[1] In the sequel of this thesis, I will primarily cite original literature, e.g., papers which introduced terms and concepts or contributed state-of-the-art protocols, and only few surveys. For general background information on cryptography, security and security engineering, I refer you to the books of Pfleeger (1997), Menezes et al. (1997) and Anderson (2001), respectively.

tropy and, therefore, are rarely used directly in the context of group key establishment, the only exception being Asokan and Ginzboorg (2000). On the other hand, biometric systems seem to be quite unsuitable for remote authentication and, to date, no viable protocol is known in the literature.

To associate identities with long-term keys, I will assume the existence of a **public-key infrastructure (PKI)** (Diffie and Hellman 1976; Kohnfelder 1978) which provides parties with some mechanisms for secure key registration and secure access to long-term keys of prospective peers. The issue of trust and PKIs will not be touched in this thesis.[2] I will assume that the PKI, or, more precisely, the involved *registration* and *certification authorities*, is unconditionally trusted to securely and reliably associate the correct identities and keys of entities. However, to minimize assumptions on the PKI and to match current practice, I will neither require that the certification authorities verify on registration that a public key pair is unique nor that the party registering a public key also knows the corresponding secret key. For example, an adversary will be able to register a public key of somebody else under his name. This scenario with a PKI also covers the case of pre-distributed pairwise shared long-term secret keys where each party is implicitly its own PKI. However, it does not directly apply to situations where trusted third-parties known as **key distribution centers** mediate session keys such as Kerberos (Medvinsky and Hur 1999; Kohl and Neuman 1993) and KryptoKnight (Janson et al. 1997; Molva et al. 1992).

Security properties — such as authenticity, integrity and confidentiality — are normally only meaningful when guaranteed during a complete **session** of closely related interactions over a communication channel. (Be it the transfer of a single e-mail between two companies which has to stay confidential or a long-standing connection between two servers which should guarantee the integrity of exchanged packets.) In most of these cases, there is a need for some temporary keys, e.g., an encryption key for a shared-key encryption scheme in the e-mail scenario or a key for a message authentication code in the second example. The goal of using temporary keys instead of using the long-term keys directly is threefold: (1) to limit the amount of cryptographic material available to cryptanalytic attacks; (2) to limit the exposure when keys are lost; and (3) to create independence between different and unrelated sessions. Furthermore, if our long-term keys are based on asymmetric cryptography, using session keys based on (faster) symmetric cryptography can bring a considerable gain in efficiency. The establishment of such temporary keys, usually called **session keys**, often involves interactive cryptographic protocols. These protocols should ensure that all the required security properties, such as the authenticity and freshness of the

[2] I refer you elsewhere (Ellison and Schneier 2000; Adams et al. 2000; Kohlas and Maurer 2000b; Kohlas and Maurer 2000a) for discussions on various aspects of this controversial topic .

resulting session key, are guaranteed. Such protocols are referred to as **key establishment protocols** and are the focus of this thesis.

As mentioned above, **authentication** is central to security. However, the term is very broad and can mean anything from access control, authentication of entities, data origin or keys to non-repudiation. The focus of this thesis is limited to authentication of (session) keys and I will define below in more detail what I mean by key authentication or, more precisely, (authenticated) key establishment. However, we first briefly digress on the subject of entity authentication as this term is often wrongly used as a synonym for authentication. This practice can lead to confusion when reasoning about protocols for entity authentication and (authenticated) key establishment.

A protocol providing **entity authentication** (often also referred to as **identification**) informally means that a party successfully engaging an other party in such a protocol can be assured of the other party's identity and its active presence during the protocol. If we consider potential applications of such a mechanism, it is clear that entity authentication cannot be seen in isolation and must be considered in a wider context. Mostly, entity authentication is required as part of a session of subsequent and separate actions over some form of channel and the authenticity has to extend over the complete lifetime of the session.[3] In cases where physical properties of the underlying channel, e.g., separate and tamper-resistant wires used exclusively to connect two secure access points, guarantee the integrity of a channel and its unique assignment to a particular session, an entity authentication protocol might be sufficient to ensure the authenticity over the lifetime of the session. However, one has to be very careful to make sure that apparent end-points of the channel really correspond to the authenticated party. Otherwise, there is a considerable danger that one falls prey to Mafia fraud (Bengio et al. 1991), a man-in-the-middle attack where the adversary transparently passes the protocol messages of the entity authentication protocol but modifies subsequent actions. For example, the (apparent) proximity of a user to her device, when performing mutual entity authentication, would seem to prevent adversaries from interfering. Nonetheless, this can be a completely false assumption (Pfitzmann et al. 1997; Asokan et al. 1999). In general, the identification of a peer and the securing of the communication channel are not orthogonal. For example, a web-banking application which separates the authentication of the client, e.g., based on passwords, from the securing of the channel, e.g., via SSL (Freier et al. 1996), might be vulnerable to man-in-the-middle-attacks (Steiner et al. 2001). Therefore, for situations with channels where the integrity and confidentiality of a session can only be guaranteed based on the establishment of a session-specific

[3] Two of the rare exceptions where entity authentication might make sense without an associated session are secure liveness check of servers and access control combined in an atomic action with the access itself, e.g., when authentication is required to access a protected room.

virtual channel secured by cryptography — as is the case for most applications where cryptographic authentication protocols might be deployed — an entity authentication protocol is of no use. The authentication has to be securely tied to the session keys, that is, we require an authenticated key establishment protocol.

2.1 Key Establishment in the Two-Party Case

Before getting into specific aspects of key establishment in group settings, we first overview key establishment in the classical two-party setting by introducing the necessary terminology and giving intuitive and informal definitions[4] for the different properties and requirements. (Some of these are adapted from Menezes, van Oorschot, and Vanstone (1997).)

2.1.1 Service and Security Properties

The basic service of a key establishment mechanism is clear: Two parties want to *establish a shared session key*. Less clear are the specific security properties which have to be provided by a protocol implementing such a service. Let us discuss the main properties in turn:

To achieve our goal of cleanly separating different sessions, the resulting session key has to be new and independent of other session keys. This property is usually called **key freshness**.

Furthermore, the session key should be known only to the involved parties. This aspect of **key secrecy** is often phrased as the inability of adversaries to learn the (complete) key. However, this formulation has its problems as it presupposes that the leakage of partial information—this would not be ruled out by such a definition!—on the session key has no effect. For example, consider an application which naively splits a session key in half into an encryption key and a key for a message authentication code. Above secrecy definition ensures that an attacker is not able to get both keys. However, it does not prevent that the attacker learns either one of them. This clearly defeats the security of such a system. Similarly, the direct use of the session key in a key establishment protocol message (as often done in the past, e.g., for key-confirmation flows) might violate the security of a higher-level protocol relying on the resulting session key: This message also can have a meaning in the higher-level protocol. To allow the arbitrary and modular composition of cryptographic protocols, we better do not make any assumptions on the usage pattern. Therefore, every single bit of the resulting session key should be unpredictable, a formulation which in the usual complexity-theoretic setting can be traced back to the **poly-**

[4] A formal treatment of n-party key agreement which will also cover the two-party case will be given later in Chapter 5.

nomial security (or **semantic security**, a slightly different formulation of an equivalent meaning and a more commonly used term) introduced by Goldwasser and Micali (1984).

Implicitly, the term "key secrecy" already includes some notion of authentication: We require that only the intended peer is able to learn the key (or any information thereof). This is called **implicit key authentication**. If the protocol confirms additionally the active and current participation of the peer in a particular session, we talk about **explicit key authentication**. This can be seen as a special form of entity authentication which provides additionally the establishment of a secure and coupled session key. Usually, this is achieved with a **key confirmation**, a protocol which shows evidence that the (same) session key is also possessed by the peer. While we usually cannot enforce **liveness**, i.e., guarantee a successful and timely termination of the protocol,[5] and key confirmation does not prevent a peer from crashing immediately afterwards, it can still form a useful basis in implementing robust and fail-safe applications. For example, honest parties might always write the application context, including the session key, to stable storage before sending a key-confirmation and make all efforts to recover such sessions after a crash; insofar, the key-confirmation would signal the successful and reliable establishment of the related session. Finally, we have to consider the reciprocity of the authentication. Usually, both parties want to authenticate each other including the common session context, e.g., they need an agreement on all information (explicitly or implicitly) visible at the service interface such as the particular session, both of their identities and the common key. This is called **mutual key authentication**. If authentication is one-sided, such as is typically the case for SSL-based web applications where only server authentication is used, we talk about **unilateral key authentication**. In the hierarchy of authentication specifications introduced by Lowe (1997), mutual key authentication corresponds to the level "Agreement" with the set of agreed data items defined as all protocol information visible to a service user. In particular, we do *not* require any agreement on protocols messages as implied by the level "Full Agreement" — or, for that matter, by intensional specifications (Roscoe 1996) — as this result in an over-specification. The security requirements should be defined based only on the service interface and not on the implementation, i.e., the protocol!

2.1.2 Adversary Model

So far, we discussed only useful security properties of a key establishment mechanism but did not mention **adversaries** trying to break these properties. Of course, to be able to reason about security, we also have to define

[5] Obviously a protocol should complete successfully when honest parties do not crash and the network faithfully forwards messages.

our adversary model. We are less interested here in the particular attacks an adversary might try — see Clark and Jacob (1997) for an extensive list of two-party key establishment protocols and related attacks — but rather in a generic categorization of bounds on their capabilities. There are two main aspects which we have to consider when defining the adversaries we are willing to cope with: their computational power and their access to the system and the underlying infrastructure. In this thesis, we will consider only adversaries whose computational power falls into the class of probabilistic polynomial-time algorithms. This is currently the most realistic complexity class which still allows for practical solutions.[6] The adversary will usually have access to the network and will be able to eavesdrop, insert, delete, replace and replay messages. Furthermore, the adversary may steal information, e.g., session or long-term keys, from honest parties and may even corrupt parties and learn their current state including their keys. Clearly, we cannot provide any security for a session where the session key is stolen or for a party from the point on where she is corrupted. However, depending on the impact of the loss of keys on other and past sessions, we can classify key agreement protocols as follows: If the compromise of long-term keys cannot result in the compromise of past session keys of a particular protocol, we say it offers **perfect forward secrecy (PFS)** (Günther 1990). Furthermore, if the compromise of session keys of a particular protocol allows (1) a passive adversary to compromise keys of other sessions, or (2) an active adversary to impersonate the identity of one of the protocol parties in other sessions, we say that this protocol is vulnerable to a **known-key attack (KKA)** (Yacobi and Shmuely 1990; Burmester 1994).

2.1.3 Types

A final distinction which one can make in a key establishment protocol is on who generates the key: In a **key transport** protocol, one party determines a session key and secretly sends it to the other party. In a **key agreement**[7] protocol, the session key is derived jointly by both parties as a function of information contributed by, or associated with, each of these, such that no party can predetermine the resulting value. This assures a party which contributed fresh information that an obtained session key is fresh even if the peer is dishonest.[8]

[6] Under some weak physical assumption it is in principle feasible to achieve secure key establishment also in information-theoretic settings (Maurer and Wolf 1999). However, there is still a long way to go before this becomes practical.

[7] The term "agreement" might sound a bit misleading here but is used for historical reasons. An agreement in the intuitive sense of a common understanding on the session context, e.g., identities and session key, is already required by mutual authentication and is orthogonal to the distinction in key transport and key agreement protocols.

[8] While the session key is guaranteed to be fresh, an adversary might nevertheless achieve some skew in the probability distribution of the session key, a distribution which

2.2 Key Establishment for Groups

2.2.1 Basic Service and Additional Security Properties

The basic service and security properties of a key establishment are roughly the same for the n-party case, i.e., groups, as they are for the two-party case described above. The main differences which have to be considered are as follows. On the one hand, due to the dynamic nature of a group, it is more difficult to reason about the honesty of parties. A party might be considered trusted in respect to its "legal" membership period. However, it also might try to misuse this time to prepare access to the group at other "illegal" membership periods, potentially even in *collusion* with other former group members. In fact, a number of group key establishment protocols actually fall prey to such *collusion attacks*, e.g., Briscoe (1999), Chang et al. (1999), and Caronni et al. (1999). On the other hand, the notion of key authentication has to be broadened. In the two-party case it is natural to always require that both parties agree on each others identities, at least for mutual key authentication.[9] This extends to the n-party case. Nonetheless, one can also imagine a weaker form of key authentication where group members are just assured that only legitimate group members are present without necessarily getting any knowledge on the actual group membership of a session. The former will be called **mutual group key authentication** whereas the latter will be called **simple group key authentication**. Simple group key authentication is the sufficient and only practical form of authentication in the case of large asymmetric groups where a static party controls access to the group and members do not know each other, e.g., in video-on-demand applications. However, in DPGs, where the roles of group members are symmetric and a common agreement on the group membership is essential, mutual group key authentication is more desirable and more natural than simple group key authentication. In groups, the verification of the authenticity does not always have to be **direct** as in the two-party case. It also can be **indirect** via some other group member(s). This requires additional trust assumptions in these intermediary group members. Nonetheless, these trust assumptions are quite natural as we do already trust insiders not to give away the common group key. Similarly, in extending explicit key authentication we have the option of requiring a confirmation which is either direct and pairwise or indirect (e.g., over a spanning tree on the membership graph.)

usually is uniform when peers are honest. To achieve the additional property that keys are always uniformly distributed even in the presence of dishonest parties, one would have to start from fair coin-tossing protocols (Blum 1982; Lindell 2001) and add the necessary key authentication and secrecy properties.

[9] Unilateral key authentication does not seem to have an equivalent in a group setting.

2.2.2 Types

We mentioned above that key establishment can be realized either as key transport or key agreement. In the following, we consider a special form of key agreement: If the individual contribution of each (honest) parties in a key agreement protocol remains computationally hidden after a protocol run even to any collusions of peers, we call such a protocol **contributory key agreement**. In these protocols, we can reuse the individual key contributions for subsequent key agreements. This is essential for DPGs, as can be seen below. A natural example of a contributory key agreement protocol for groups of two is the Diffie-Hellman key exchange (Diffie and Hellman 1976). A important advantage of contributory key agreement schemes is that they almost automatically yield perfect forward secrecy and resistance to active known-key attacks. Note that almost all group key transport protocols fail to provide at least one of perfect forward secrecy and resistance to known-key attacks.

If a group key agreement protocol assures additionally that a session key is shared by any two group members only if all members in the common view on the group membership did actively participate, we say it is a **complete group key agreement** (Hutchinson 1995; Ateniese et al. 2000). Implicitly, such a protocol provides mutual group key authentication and authentication is direct between any two group members.

2.2.3 Fault-Tolerance

Several group key agreement schemes have been proposed in the literature (Ingemarsson et al. 1982; Steer et al. 1990; Burmester and Desmedt 1995; Just 1994; Steiner et al. 1996; Just and Vaudenay 1996; Becker and Wille 1998), however, none have been widely deployed. In practice, group key establishment is typically done in a *centralized* manner (Harney and Muckenhirn 1997; Wong et al. 1998; Wallner et al. 1997): one dedicated party (typically, a group leader) chooses the group key and distributes it to all group members. This is actually *key transport* (often also called *key distribution* in such a context), not *key agreement*.

While the centralized approach works reasonably well for static groups or very large groups, it turns out that key agreement is superior for DPGs, i.e, flat (non-hierarchical) groups with dynamically changing membership.

A permanently fixed group leader is a potential performance bottleneck and a single point of failure. Some DPG environments (such as *ad hoc* wireless networks) are highly dynamic and no group member can be assumed to be present all the time. This is also the case in wired networks when high availability is required. Therefore, my view is that fault-tolerance (such as handling network partitions and other events) is best achieved by treating

all parties as peers. This is supported by the state-of-the-art in reliable group communication (see, for example, Birman (1996).)

Secure group key agreement protocols such as the CLIQUES family presented later are fault-tolerant in terms of *integrity* and *confidentiality*, e.g., the authenticity and secrecy of the key is ensured. To enhance fault-tolerance also in form of *availability* (or *liveness*), e.g., to prevent accidental denial of service, I suggest the use of some *reliable group communication system* which is resistant to fail-stop[10] failures and provides consistent, i.e., reliable and causally ordered, membership views and a corresponding multicast facility to all group members. A developer integrating a group key agreement protocol into an application will also benefit from the easier administration of group membership provided by such a group communication system.

While group key agreement protocols and group communication systems are a priori orthogonal, the integration of group key agreement and reliable group communication to form a secure group communication system raises a number of issues such as efficient handling of various cascading failures. Owing to the built-in flexibility of CLIQUES protocols, these issues can be resolved in an efficient and modular manner without interfering with the security properties discussed in this thesis. For further information, I refer you to some recent work (Amir et al. 2000; Agarwal et al. 2001) which reports on the integration of CLIQUES with the reliable group communication systems SPREAD (Amir and Stanton 1998) and Totem (Moser et al. 1996).

2.2.4 Management of Groups

There is no inherent reason to require a single group leader to make the decisions as to whom to add to, or exclude from, a group.[11] Ideally, decisions regarding **group admission control**, e.g., who can be added to or removed from a group and who can coordinate such operations, should be taken according to some local group policy (see, for example, Kim et al. (2002) for some discussion on this issue) and should be orthogonal to the actual key establishment protocol deployed. For instance, in some applications, each peer must be allowed to add new members and exclude members that it previously added. This **policy independence** cannot be easily implemented in centralized schemes, while the approach presented later supports it quite elegantly and efficiently: any party can initiate all membership change protocols.

[10] We are trusting legitimate group members. Therefore, assuming fail-stop and not byzantine behavior of group members seems appropriate and allows more efficient systems.

[11] One obvious issue with having a fixed group leader is how to handle its expulsion from the group. However, also environments with no hierarchy of trust are a poor match for centralized key transport. For example, consider a peer group composed of members in different, and perhaps competing, organizations or countries.

Although I argue in favor of distributed key agreement for DPGs, I also recognize the need for a central point of control for group membership operations such as adding and excluding members. This type of a role (group controller) serves only to coordinate and synchronize the membership operations and prevent chaos. However, the existence and assignment of this role is orthogonal to key establishment, can be changed at any time and is largely a matter of policy.

2.3 Handling the Dynamics of Groups

A key aspect of groups is their dynamic behavior as they evolve over time. This has to be reflected in a set of corresponding key establishment services, too. In the following, I distinguish between **Initial Key Agreement (IKA)**, a kind of group genesis, and **Auxiliary Key Agreement (AKA)**. AKA encompasses all operations that modify group membership, such as member addition and exclusion. Time periods separated by AKA operations will be called **epochs** whereas the term *session* is mostly used for the complete lifetime of a group. Nevertheless, for convenience I will talk about *session keys* even though the term *epoch keys* would be more correct.

2.3.1 Initial Key Agreement (IKA)

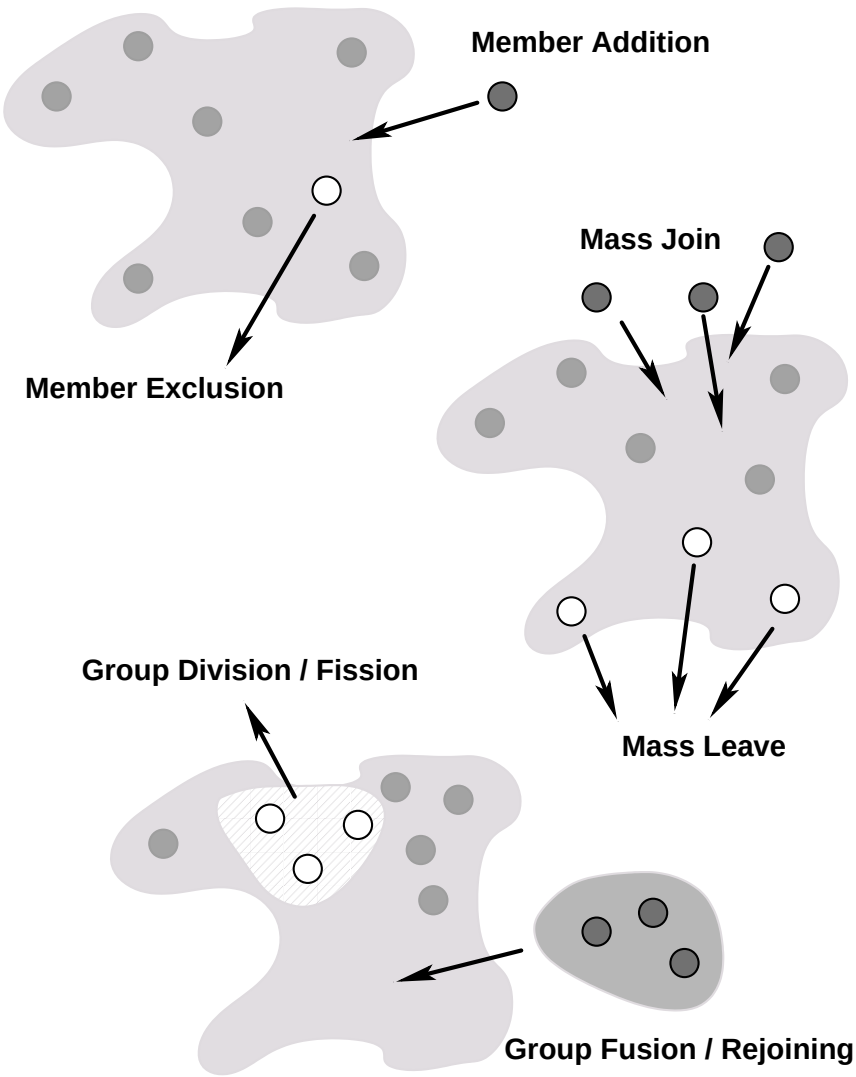
IKA takes place at the time of group *genesis*. On the one hand, this is the time when protocol overhead should be minimized since key agreement is a prerequisite for secure group communication. On the other hand, for highly dynamic groups, certain allowances can be made: for example, extra IKA overhead can be tolerated in exchange for lower AKA (subsequent key agreement operations) costs. However, note that it is the *security* of the IKA, not its overhead costs, that is the overriding concern.

Naturally, IKA requires contacting every prospective group member to obtain a key share from each member. Hence, it may be possible to coincide (or interleave) with the IKA other security services such as access control. However, care has to be taken that this does not interfere with the security of the key agreement protocols, in particular agreed keys should only be used when the protocol specification explicitly hands them back to higher layers.

2.3.2 Auxiliary Key Agreement (AKA) Operations

As mentioned above, initial group key agreement is only a part, albeit a major one, of the protocol suite needed to support secure communication in dynamic groups. In this section, I discuss other auxiliary group key operations and the attendant security issues. (See also Figure 2.1.)

Figure 2.1 AKA Operations



The security property crucial to all AKA operations is **key independence**. Informally, it encompasses the following two requirements closely related to PFS and in particular KKA:

- Old group keys used in past epochs must not be discovered by new group member(s). In other words, a group member must not have knowledge of keys used before it joined the group.
- New keys must remain out of reach of former group members, i.e., members excluded in past epochs.

Note that recent papers termed above two cases explicitly as *forward* and *backward access control* (Meadows, Syverson, and Cervesato 2001) or *forward* and *backward secrecy* (Kim, Perrig, and Tsudik 2001). This is useful to label some protocols which do not provide (full) key independence yet still some weaker form of it. However, in the sequel we will focus only on the combined key independence, the most desirable property.

While the requirement for key independence is fairly intuitive, we need to keep in mind that, in practice, it may be undesirable under certain circumstances. For example, a group conference can commence despite some of the intended participants running late. Upon their arrival, it might be best not to change the current group key so as to allow the tardy participant(s) to catch up.[12] In any case, this decision should be determined by policy local to a particular group.

Single Member Operations

The AKA operations involving single group members are **member addition** and **member exclusion**. The former is a seemingly simple procedure of admitting a new member to an existing group. We can assume that member addition is always multi-lateral or, at least, bilateral (i.e., it takes at least the group leader's and the new member's consent to take place.) Member exclusion is also relatively simple with the exception that it can be performed either unilaterally (by expulsion) or by mutual consent. In either case, the security implications of member exclusion are the same, i.e., an excluded member should not have access (knowledge) of any future key unless it is re-admitted to the group.

Subgroup Operations

Subgroup operations are group addition and group exclusion. Group addition, in turn, has multiple variants:

[12] Adding a new member without changing a group key is easy: the controller sends the new member the current key over an authentic and private channel providing PFS. Although the new member has not contributed to the group key, it can do so later by initiating a key refresh.

- **Mass join:** the case of multiple new members who have to be brought into an existing group and, moreover, these new members do not already form a group of their own.
- **Group fusion:** the case of two groups merging to form a super-group; perhaps only temporarily.
- **Group rejoining:** A special case of fusion where two groups merge which resulted from a previous division (see below).

Similarly, subgroup exclusion can also be thought of as having multiple flavors:

- **Mass leave:** multiple members must be excluded at the same time.
- **Group division:** a monolithic group needs to be broken up in smaller groups.
- **Group fission:** a previously merged group must be split apart.

Although the actual protocols for handling all subgroup operations may differ from those on single members, the salient security requirements (key independence) remain the same.

While group fission and group rejoining can be quite relevant in special scenarios, e.g., when network partitions require temporary subgroups, it requires keeping track of the various subgroups. As in most cases this might not be worth the bookkeeping effort I will not address them in the sequel. Furthermore, scenarios like aforementioned temporary network partition normally do not require changing the key — the decisions to change key epochs, and the related desire for key independence, are usually driven by application layer requirements and policies, and rarely by such network events.

Remark 2.1. In recent papers on group security the terminology has slightly changed, e.g., "addition" became "join", "exclusion" became "leave", "fusion" and "rejoining" are now often termed "merge", and "division" and "fission" are replaced by "partition". However, for historical reasons I kept the terminology as used in the original papers. ○

Group Key Refresh

For a variety of reasons it is often necessary to perform a routine key change operation independent of the group membership changes. This may include, for example, local policy that restricts the usage of a single key by time or by the amount of data that this key is used to encrypt or sign. To distinguish it from key changes due to membership changes, I refer to this operation in the sequel as **key refresh**.

2.4 Measures

Above I described the required key management services, i.e., IKA and AKA, and desirable properties such as PFS or policy independence. Clearly, we also have to consider the cost and performance of protocols to estimate their practicality, their scalability and their suitability to particular environments. There are primarily two aspects in measuring the cost: computation and communication.

It is impossible to estimate concrete **computational costs** as many cost-critical aspects are implementation-dependent and some primitives are difficult to compare. However, we should get a good base for comparison by identifying the expensive and time-critical operations, and by just considering them individually. Furthermore, computational capabilities might largely differ among the involved parties. However, in a DPG, all group members are equal and we can safely assume that they have similar computational capabilities. Therefore, my approach is to list for each protocol the number of *expensive operations*, on the one hand, summed up for individual group members and, on the other hand, as the sum of these operations on the critical path[13] of a protocol run. The former gives an estimate on the computational load on individual group members whereas the latter provides a lower bound on the duration of a protocol run.

Regarding **communication costs**, the impact of a protocol clearly depends on the topology and properties of the network and the group communication system used. The critical aspects are primarily latency and bandwidth. Unfortunately, they cannot be measured directly. My approach in the following will be to list for each protocol the number of *messages*, their *cumulative size*, and the number of *rounds*, i.e., the number of messages on the critical path in a protocol. Additionally, I will distinguish between *unicast* and *multicast* messages, i.e., messages from one group members to another one and from one group member to the rest of the group, respectively. From these numbers we can then derive an estimate on the concrete protocol latency and network load when the actual networking environment is known.

[13] The **critical path** denotes the sequence of all operations which have to be performed sequentially. Therefore, parallel operations are counted only once in computing the cost. The critical path corresponds to what is called elsewhere (Kim et al. 2000) the *serial operations*.

Chapter 3

Exploring the Mathematical Foundations

In this chapter, I investigate the mathematical foundations of the group key agreement protocols presented later. I take a closer look at cryptographic assumptions based on discrete logarithms. I classify and discuss important properties which significantly differentiate variants of such assumptions. Furthermore, I introduce the Decisional Generalized Diffie-Hellman problem and investigate its relation to the Decisional Diffie-Hellman problem. I prove a tool box of relations among these assumptions which will be helpful in proving the security of the group key agreement protocols introduced later.

MOST modern cryptographic systems rely on assumptions on the computational difficulty of some particular number-theoretic problem.[1] One well-known class of assumptions is related to the difficulty of computing discrete logarithms in cyclic groups (McCurley 1990). In this class a number of variants exists. The most prominent ones, besides **Discrete Logarithm (DL)**, are the computational and decisional **Diffie-Hellman (DH)** assumptions (Diffie and Hellman 1976; Brands 1994). Less known assumptions are **Matching Diffie-Hellman** (Frankel et al. 1996), **Square Exponent (SE)** (Maurer and Wolf 1996), and **Inverse Exponent (IE)** (Pfitzmann and Sadeghi 2000), an assumption closely related to the **Inverted-Additive Exponent (IAE)** Problem introduced by MacKenzie (2001)[2] and also implicitly required for the secu-

[1] The exceptions are information-theoretically secure systems and systems such as hash-functions or shared-key encryption relying on heuristic assumptions, e.g., the Random Oracle Model (Bellare and Rogaway 1993).

[2] Note that SE and IAE are originally called Squaring Diffie-Hellman (Wolf 1999) and Inverted-Additive Diffie-Hellman (MacKenzie 2001), respectively. They are renamed here for consistency and clarity reasons.

rity of the schemes proposed by Camenisch, Maurer, and Stadler (1996) and Davida, Frankel, Tsiounis, and Yung (1997). Further related assumptions mentioned in the sequel are **Generalized Diffie-Hellman (GDH)** (Shmuely 1985; Steiner et al. 1996) and the **Representation Problem (RP)** (Brands 1994). Several additional papers have studied relations among these assumptions, e.g., (Shoup 1997; Maurer and Wolf 1998a; Maurer and Wolf 1998b; Biham et al. 1999; Wolf 1999).

In the concrete formalizations of these assumptions, one has various degrees of freedom offered by parameters such as computational model, problem type (computational, decisional or matching) or success probability of the adversary. However, such aspects are often not precisely considered in the literature and consequences are simply overlooked. In this chapter, I address these aspects by identifying the parameters relevant to cryptographic assumptions. Based on this, I present a formal framework and a concise notation for defining DL-related assumptions. This enables us to precisely and systematically classify these assumptions.

Among the specified parameters, an interesting and, so far, overlooked parameter relevant to many cryptographic applications is the *granularity* of the probability space which underlies an assumption. Granularity defines what part of the underlying algebraic structure (i.e., algebraic group and generator) is part of the probability space and what is fixed in advance: For high granularity, an assumption has to hold for all groups and generators; for medium granularity, the choice of the generator is included in the probability space, and for low granularity, the probability is taken over both the choice of the group and the generator. Assumptions with lower granularity are weaker than those with higher granularity. Nonetheless, not all cryptographic settings can rely on the weaker variants: Only when the choice of the system parameters is guaranteed to be random, one can rely on a low-granularity assumption. For example, consider an anonymous payment system where the bank chooses the system parameters. To base the security of such a system a-priori on a low-granularity assumption would not be appropriate. A cheating bank might try to choose a weak group with trapdoors (easy problem instances) to violate the anonymity of the customer. Such a cheater strategy might be possible even if the low-granular assumption holds: The assumption would ensure that the overall number of easy problem instances is asymptotically negligible (in respect to the security parameter). Nonetheless, it would not rule out that there are infinitely many weak groups! However, if we choose the system parameters of the payment system through a random yet verifiable process we can resort to a weaker assumption with lower granularity. To my knowledge no paper on anonymous payment systems addresses this issue properly. Granularity was also overlooked in different contexts, e.g., Boneh (1998) ignores the fact that low-granular assumptions are not known to be random self-reducible and comes to a wrong conclusion regarding the correctness of a certain self-corrector.

The rest of this chapter is structured as follows: In the next section, I define the basic terminology. Section 3.2 introduces the classification of discrete-logarithm-based assumptions, and in Section 3.3 we see how this classification can be used to concisely yet precisely describe assumptions and relations among them. Section 3.4 briefly discusses the newly introduced "granularity" parameter and some results relevant in the context of this thesis. In Section 3.5, we take a closer look at the cornerstone of the following protocols, the Generalized Diffie-Hellman assumption. Finally, in Section 3.6, we see how we can derive a random bit-string from (Generalized) Diffie-Hellman keys.

3.1 Terminology

3.1.1 General Notational Conventions

By $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}$ and $(\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots)$ I denote the *set* and the *sequence* consisting of the elements a, b, c, ... By specifying a set as $\{f(v_1, \ldots, v_n) \mid \mathsf{pred}(v_1, \ldots, v_n)\}$ I mean the set of elements we get by evaluating the formula f with any instantiation of the n free variables $v_1, \ldots, v_n$ which fulfills the predicate pred, e.g., $\{(v, v^2) \mid v \in \mathbb{N}\}$ denotes the set of all tuples which contain a natural number and its square. Similarly, I define $(f(v_1, \ldots, v_n) \mid \mathsf{pred}(v_1, \ldots, v_n))$ to be the sequence of elements we get by evaluating the formula f with any instantiation of the n free variables $v_1, \ldots, v_n$ which fulfills the predicate pred. The elements are ordered according to some arbitrary but fixed order relation on the (instantiated) argument tuples $(v_1, \ldots, v_n)$. For example, $((v, v^2) \mid v \in \mathbb{N})$ denotes the infinite sequence of all tuples which contain a natural number and its square, and where the sequence is ordered, e.g., using the standard order $<$ on $\mathbb{N}$ and the value of v as the sort index.

The evaluation and following *assignment* of an expression expr to a variable v is denoted by $v \leftarrow \mathsf{expr}$. By $v \xleftarrow{\mathcal{R}} S$ I mean the assignment of a uniformly chosen random element from the set S to variable v. Similarly, $v \in_{\mathcal{R}} S$ denotes that v is a uniformly distributed random element from set S. Finally, by $\mathsf{t} := \mathsf{expr}$ I mean that by definition the term t is equal to expr.

Simple *random variables* are specified as $v \xleftarrow{\mathcal{R}} S$ as mentioned above. To specify more complicated random variables, I use the following notation: $(f(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n))$. By this I mean the random variable having a structure as defined by the formula f and a *probability space* as induced by binding the n free variables $v_1, \ldots, v_n$ via the assignment rule assign, e.g., $((v, v^2) :: v \xleftarrow{\mathcal{R}} \mathbb{Z}_n)$ denotes the random variable consisting of a tuple which contains an integer and its square where the integer is uniformly chosen from $\mathbb{Z}_n$. Similarly, $\{f(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n)\}$ defines an ensemble of random variables indexed by the free variables v_i which are left unspecified in the assignment rule assign and which have by definition

domain $\mathbb{N}$, e.g., $\{(v, v^k) :: v \xleftarrow{\mathcal{R}} \mathbb{Z}_n\}$ denotes the ensemble of random variables consisting of a tuple which contain an integer and its k-th power where the integer is uniformly chosen from $\mathbb{Z}_n$ and the natural number k is the index of the ensemble. Finally, let v be some arbitrary random variable or random variable ensemble. Then, $[v]$ denotes the set of all possible values of v.

To specify *probabilities*, I use the notation $\mathbf{Prob}[\mathsf{pred}(v_1, \ldots, v_n) :: \mathsf{assign}(v_1, \ldots, v_n)]$. This denotes the probability that the predicate pred holds when the probability is taken over a probability space defined by the formula assign on the n free variables v_i of the predicate pred. For example, $\mathbf{Prob}[v \equiv 0 \pmod 2 :: v \xleftarrow{\mathcal{R}} \mathbb{Z}_n]$ denotes the probability that a random element of $\mathbb{Z}_n$ is even.

For convenience, by log I always mean the logarithm to the base two, define $I_n := \{0,1\}^n$ as the set of all n-bit strings and 1^n as the bit string consisting of n 1's, i.e., n in unary encoding.

3.1.2 Asymptotics

Cryptographic assumptions are always expressed asymptotically in a **security parameter** $k \in \mathbb{N}$. To classify the asymptotic behavior of functions $\mathbb{N} \to \mathbb{R}^*$ (with $\mathbb{R}^*$ denoting the set of all non-negative real numbers) we require the following definitions.

We can extend ordinary relation operators $op \in \{<, \leq, =, >, \geq\}$ on elements of $\mathbb{R}^*$ to asymptotic relation operators op_∞ on functions f_1 and f_2 defined as above as follows:

$$f_1(k) \;\; op_\infty \;\; f_2(k) \;:=\; \exists k_0 \, \forall k > k_0 : f_1(k) \; op \; f_2(k).$$

The corresponding negation of the asymptotic relation operators is then denoted by $\not<_\infty$, $\not\leq_\infty$, $\neq_\infty$, $\not\geq_\infty$, and $\not>_\infty$, respectively.

For example, $f_1(k) <_\infty f_2(k)$ means that f_1 is asymptotically strictly smaller than f_2 and $f_1(k) \not\geq_\infty f_2(k)$ means that f_1 is not asymptotically larger or equal to f_2, i.e., for each k_0 there is a $k_1 > k_0$ such that $f_1(k_1) < f_2(k_1)$. However, note that the $f_1(k) \not\geq_\infty f_2(k)$ does not imply $f_1(k) <_\infty f_2(k)$!

Let $\mathsf{poly}(v)$ be the class of **univariate polynomials** with variable v and non-negative coefficients, i.e., $\mathsf{poly}(v) := \{\sum_{i=0}^{d} a_i v^i \mid d \in \mathbb{N}_0 \;\wedge\; a_i \in \mathbb{N}_0\}$. Furthermore, let $\mathsf{poly}(v_1, \ldots, v_n)$ be the class of **multivariate polynomials** with n variables v_j and non-negative coefficients, i.e., $\mathsf{poly}(v_1, \ldots, v_n) := \{\sum_{i=0}^{d} \sum_{j=1}^{|D_i|} a_{ij} \prod_{l=1}^{n} {v_l}^{d_{ijl}} \mid d \in \mathbb{N}_0 \wedge a_{ij} \in \mathbb{N}_0 \wedge (d_{ij1}, \ldots, d_{ijn}) \in D_i^n\}$ where $D_i^n := \{(d_l \mid l \in \{1, \ldots, n\}) \mid d_l \in \mathbb{N}_0 \wedge \sum_{l=1}^{n} d_l = i\}$. Based on this we can define the following useful classes of functions:

A **negligible** function $\epsilon(k)$ is a function where the inverse of any polynomial is asymptotically an upper bound, i.e., $\forall d > 0 \; \exists k_0 \; \forall k > k_0 : \epsilon(k) < 1/k^d$. I denote this by $\epsilon(k) <_\infty 1/\mathsf{poly}(k)$. If $\epsilon(k)$ cannot be upper bounded in such a way, I say $\epsilon(k)$ is **not negligible** and I denote this by $\epsilon(k) \not<_\infty 1/\mathsf{poly}(k)$.

A **non-negligible** function $f(k)$ is a function which asymptotically can be lower bounded by the inverse of some polynomial, i.e., $\exists d > 0\ \exists k_0\ \forall k > k_0 : f(k) \geq 1/k^d$. I denote this by $f(k) \geq_\infty 1/\mathsf{poly}(k)$.[3] If $f(k)$ cannot be lower bounded in such a way I say $f(k)$ is **not non-negligible** and denote this by $f(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

Non-negligible functions are — when seen as a class — closed under multivariate polynomial composition, i.e., $\forall n \in \mathbb{N}\ \forall i \in \{1, \ldots, n\}\ \forall p \in \mathsf{poly}(v_1, \ldots, v_n) \setminus \{0_{\mathsf{poly}}\}\ \forall f_i \geq_\infty 1/\mathsf{poly}(k) : \ p(f_1, \ldots, f_n) \geq_\infty 1/\mathsf{poly}(k)$ where 0_{poly} denotes the null polynomial. This holds also for negligible functions if there is no non-zero constant term in the polynomial, i.e., we select only elements from the class $\mathsf{poly}(v_1, \ldots, v_n)$ where a_{01} is zero. For not negligible and not non-negligible functions this holds solely for univariate polynomial composition. Finally, the addition (multiplication) of a non-negligible and a not negligible function is a non-negligible (not negligible) function. Similarly, the addition of a negligible and a not non-negligible function is a not non-negligible function. The multiplication of a negligible and a not non-negligible function is a not non-negligible function or even a negligible function if the not non-negligible function can be upper bounded by some polynomial.

3.1.3 Computational Model

The computational model is based on the class $\mathcal{TM}$ of probabilistic *Turing machines* on the binary alphabet $\{0, 1\}$. The **runtime** of a Turing machine M is measured by the number of simple Turing steps from the initial state with given inputs until the machine reaches a final state. This is denoted by $\mathrm{RunTime}(\mathsf{M}(\mathit{inputs}))$. The complexity of a Turing machine is expressed as a function of the bit-length of the inputs encoded on its input tape and defined as the maximum runtime for any input of a given bit-length. To make the definition of the probability spaces more explicit, I model a probabilistic Turing machine always as a deterministic machine with the random coins given as an explicit input $\mathcal{C}$ chosen from the *uniform distribution of infinite binary strings* $\mathcal{U}$. However, I do not consider the randomness when calculating the length of the inputs. The important class of **polynomial-time Turing machines** is the class of machines with polynomial complexity:

$$\begin{aligned}\{\mathsf{A} \ \mid \ & \mathsf{A} \in \mathcal{TM}; \\ & \forall d_1;\ \exists d_2;\ \forall k; \\ & \forall \mathit{inputs} \in \{0,1\}^{k^{d_1}};\ \forall \mathcal{C} \in \{0,1\}^\infty; \\ & :\ \mathrm{RunTime}(\mathsf{A}(\mathcal{C}, \mathit{inputs})) \ < \ k^{d_2}\}\end{aligned}$$

[3] Note that not negligible is *not* the same as non-negligible, there are functions which are neither negligible nor non-negligible!

When I use the term **efficient** in the context of algorithms or computation I mean a Turing machine with polynomial complexity. By a **hard problem** I mean the absence of any efficient algorithm (asymptotically) solving that problem.

In some situations, e.g., in a reduction, a machine M has access to some other machines $\mathcal{O}_1, \ldots, \mathcal{O}_n$ and can query them as **oracles**. I denote this by $\mathsf{M}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$. This means that the machine M can write the input tapes of all $\mathcal{O}_i$, run them on that input, and read the corresponding output tapes. However, M does not get access to the internal structure or state of the oracle.

3.1.4 Indistinguishability

Let two families of random variables $X := (X_k \mid k \in \mathbb{N})$ and $Y := (Y_k \mid k \in \mathbb{N})$ be defined over some discrete domain $\mathcal{D}$. They are said to be **computationally indistinguishable** iff there is no efficient distinguishing algorithm D which can distinguish the two asymptotically, i.e., $|\mathbf{Prob}[\mathsf{D}(1^k, X_k) = 1] - \mathbf{Prob}[\mathsf{D}(1^k, Y_k) = 1]|$ is a negligible function in k. This is denoted by $X \overset{c}{\approx} Y$. X and Y are **statistically indistinguishable** iff the **statistical difference** $\Delta_{(X,Y)}(k) := \sum_{d \in \mathcal{D}} |\mathbf{Prob}[X_k = d] - \mathbf{Prob}[Y_k = d]|$ is a negligible function. This is written as $X \overset{s}{\approx} Y$.

3.1.5 Algebraic Structures

The following terms are related to the algebraic structures underlying an assumption.

Finite cyclic group G: A group is an algebraic structure with a set G of **group elements** and a binary **group operation** $* : G \times G \to G$ such that the following conditions hold:

- the group operation is **associative**, i.e., $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$,
- there is an **identity element** $1 \in G$ such that $a * 1 = a = 1 * a$ for all $a \in G$, and
- for each $a \in G$ there is an **inverse** $a^{-1} \in G$ such that $a * a^{-1} = 1 = a^{-1} * a$.

The **group order** is the cardinality of the set G and is denoted by $|G|$.

In the following, I write group operations always multiplicatively by juxtaposition of group elements; Nonetheless, note that the following results apply — with the appropriate adaption of notation — also to additive groups such as elliptic curves. The **exponentiation** a^x for $a \in G$ and $x \in \mathbb{N}_0$ is then defined as usual as $\overbrace{a \cdots a}^{x \text{ times}}$. The **discrete logarithm** of a given $b \in G$ with

respect to a specified base $a \in G$ is the smallest $x \in \mathbb{N}_0$ such that $a^x = b$ or undefined if no such x exists. The **order of a group element** $b \in G$ is the least positive integer x such that $b^x = 1$ or ∞ if no such x exists.

A group G is **finite** if $|G|$ is finite. A group G is **cyclic** if there is a **generator** $g \in G$, such that $\forall b \in G\ \exists! x \in \mathbb{Z}_{|G|}\ :\ g^x = b$. The order of all elements in a finite cyclic group G divides $|G|$. In particular, there are exactly $\varphi(d)$ elements of order d (where d is any divisor of $|G|$). This means that there exactly $\varphi(|G|)$ elements of **maximal order**, i.e., generators.

All considered assumptions are based on finite cyclic groups. For brevity, however, I omit the "finite cyclic" in the sequel and refer to them simply as "groups".

For more information on the relevant abstract algebra I refer you to the book of Lidl and Niederreiter (1997).

Algorithmically, the following is noteworthy: Finding generators can be done efficiently when the factorization of $|G|$ is known; it is possible to perform exponentiations in $O(\log(|G|))$ group operations; and computing inverses can be done in $O(\log(|G|))$ group operations under the condition that $|G|$ is known. For the corresponding algorithms and further algorithms for abstract or concrete groups I refer you to the books of Bach and Shallit (1996) and Menezes, van Oorschot, and Vanstone (1997).

Structure instance SI:A tuple $(G, g_1, \ldots, g_n)$ containing a group G as first element followed by a sequence of one or more generators g_i. This represents the structure underlying a particular problem. We can assume that the structure instance SI (though not necessarily properties thereof such as the order or the factorization of the order) is publicly known.

As a convention I abbreviate g_1 to g if there is only a single generator associated with a given structure instance.

3.1.6 Problems

The following two terms characterize a particular problem underlying an assumption.

Problem family $\mathcal{P}$: A family of abstract relations indexed by their underlying structure instance SI. An example is the family of Diffie-Hellman problems which relate two (secret) numbers x and y, the two (public) values g^x and g^y, and the value g^{xy} where all exponentiations are computed using the generator g specified in SI. I define a problem family by explicitly describing its problem instances as shown in the next paragraph.

Problem instance PI: A list of concrete parameters fully describing a particular instance of a problem family, i.e., a description of the structure instance SI and a tuple $(priv, publ, sol)$ where $priv$ is the tuple of values kept

secret from adversaries, *publ* is the tuple of information publicly known on that problem and *sol* is the set of possible solutions[4] of that problem instance. When not explicitly stated, we can assume that *priv* consists always of elements from $\mathbb{Z}_{|G|}$, *publ* consists of elements from G, and *sol* is either a set of elements from $\mathbb{Z}_{|G|}$ or from G.

If we take the aforementioned Diffie-Hellman problem for subgroups of $\mathbb{Z}_p^*$ of order q with p and q prime as an example, a problem instance PI_{DH} is defined by a tuple

$$(((\mathbb{Z}_{p/q}^*, p, q), (g)), ((x, y), (g^x, g^y), \{(g^{xy})\}))$$

where $\mathbb{Z}_{p/q}^*$ denotes the parameterized description of the group and its operation, and p, q are the corresponding group parameters. (More details on the group description and parameter are given below when group samplers are introduced.)

This presentation achieves a certain uniformity of description and allows a generic definition of types of problems, i.e., whether it is a decisional or computational variant of a problem. While this might not be obvious right now, it should become clear at the latest in Section 3.2 below when I give the explicit definition of the different problem families with Parameter 1 and the precise definition of problem types with Parameter 2.

For convenience, I define PI^{SI}, PI^{publ}, PI^{priv} and PI^{sol} to be the projection of a problem instance PI to its structure instance, public, private and solution part, respectively. Picking up again above example, this means ${PI_{DH}}^{SI} := ((\mathbb{Z}_{p/q}^*, p, q), (g))$, ${PI_{DH}}^{priv} := (x, y)$, ${PI_{DH}}^{publ} := (g^x, g^y)$, and ${PI_{DH}}^{sol} := \{g^{xy}\}$, respectively.

3.1.7 Samplers

In the following, I describe different probabilistic polynomial-time algorithms I use to randomly select (sample) various parameters. Note that these samplers cannot be assumed to be publicly known, i.e., to sample from the corresponding domains adversaries have to construct their own sampling algorithms from publicly known information.

Group sampler $SG_{\mathcal{G}}$: A function which, when given a security parameter k as input, randomly selects a group G and returns a corresponding group index. I assume that a group sampler selects groups only of similar nature and type, i.e., there is a general description of a Turing machine which, based on a group index as parameter, implements at least the group operation and the equality test, and specifies how the group elements are represented.

[4] The solutions might not be unique, e.g., multiple solution tuples match a given public value in the case of the Representation Problem (See Section 3.2, Parameter 1).

An example are the groups pioneered by Schnorr (1991) in his identification and signature schemes and also used in the Digital Signature Standard (DSS) (National Institute of Standards and Technology (NIST) 2000), i.e., unique subgroups of $\mathbb{Z}_p^*$ of order q with p and q prime. The group index would be (p, q) and the description of the necessary algorithms would be taken, e.g., from Menezes et al. (1997). Note that, in this example, the group index allows the derivation of the group order and the factorization thereof. However, it cannot be assumed that the group index — the only information besides the description of the Turing machine which will be always publicly known about the group — allows to derive such knowledge on the group order in general.

The set of groups possibly returned by a group sampler, i.e., $[SG_{\mathcal{G}}]$, is called in the sequel a **group family** $\mathcal{G}$ and is required to be infinite. To make the specific group family $\mathcal{G}$ more explicit in the sampler I often label the sampler accordingly as $SG_{\mathcal{G}}$, e.g., for above example the sampler would be named $SG_{\mathbb{Z}^*_{p/q}}$.

Furthermore, the set of possible groups G returned by $SG_{\mathcal{G}}$ for a given fixed security parameter k, i.e., $[SG_{\mathcal{G}}(1^k)]$, is called **group siblings** $\mathcal{G}_{SG(k)}$. This represents the groups of a given family $\mathcal{G}$ with approximately the same "security". I assume that the group operation and equality test for the groups in $\mathcal{G}_{SG(k)}$ can be computed efficiently (in k); yet the underlying problem is supposedly asymptotically hard.

Slightly restricting the class of samplers, I require that the number of groups in $\mathcal{G}_{SG(k)}$ is super-polynomial in k and the order $|G|$ of all $G \in \mathcal{G}_{SG(k)}$ is approximately the same. In particular, I assume that the order can be bounded in the security parameter, i.e., $\exists d_1, d_2 > 0\ \forall k > 1\ \forall G \in \mathcal{G}_{SG(k)}$: $k^{d_1} \leq \log(|G|) \leq k^{d_2}$.[5] For Schnorr signatures, in the example given above, a group sampler might choose the random primes p and q with $|q| \approx 2k$ and $p = rq + 1$ for an integer r sufficiently large to make DL hard to compute in security parameter k. See Menezes et al. (1997) and Odlyzko (2000b) for the state-of-the-art algorithms for computing discrete logarithms and Lenstra and Verheul (2001) for a methodology on how to choose parameters (as a function of the security parameter k), illustrated concretely for group families such as $\mathbb{Z}_p^*$ or elliptic curves.

Generator sampler Sg: A function which, when given a description of a group G for a fixed group family, randomly selects a generator $g \in G$.

[5] This restriction is mainly for easier treatment in various reductions and is not a hindrance in practical applications: On the one hand, the upper bound is tight (larger groups cannot have efficient group operations). On the other hand, the common approach in choosing a safe group order, e.g., as proposed by Lenstra and Verheul (2001), will relate the group order closely to the negligible probability of guessing a random element correctly, and hence result in exponential order.

I assume that Sg has always access somehow, e.g., via an oracle, to the factorization of the group order. This information is required by the sampler as the group index might not be sufficient to find generators efficiently. This covers the situation where an honest party chooses the group as well as the generator but keeps the factorization of the group order secret. However, it also implies that the factorization of the order should in general be public when the adversary chooses the generators.

Note that the number of generators is $\varphi(|G|)$ and, due to requirements on group orders mentioned above, always super-polynomial in the security parameter k: Given the lower bound $\forall n \geq 5 \ : \ \varphi(n) > n/(6 \log(\log(n)))$ (Fact 2.102, Menezes et al. 1997) and our size restrictions on $|G|$ we have asymptotically the following relation: $\varphi(|G|)/|G| > 1/O(\log k) > 1/k$.

Problem instance sampler $SPI_{\mathcal{P}}$: A function indexed by a problem family $\mathcal{P}$ which, when given a description of a structure instance SI as input, randomly selects a problem instance PI. Similarly to Sg, I assume that $SPI_{\mathcal{P}}$ gets always access to the factorization of the group order. Furthermore, $SPI_{\mathcal{P}}$ gets also access to the discrete logarithms among the different generators in SI. This is required for some problem families, e.g., IE and RP(n).[6] In most cases and in all examples considered here, this corresponds to randomly selecting *priv* and deriving *publ* and *sol* from it. For example, a problem instance sampler SPI_{DH} for the Diffie-Hellman problem family would return a tuple $(SI, ((x, y), (g^x, g^y), \{(g^{xy})\}))$ with x and y randomly picked from $\mathbb{Z}_{|G|}$ and g taken from SI. When the specific problem family $\mathcal{P}$ is not relevant or clear from the context I abbreviate $SPI_{\mathcal{P}}$ to SPI.

Note that the running time of the samplers is always polynomially bounded in the security parameter k.[7]

If not stated explicitly we can always assume a uniform distribution of the sampled elements in the corresponding domains, as done in most cases of cryptographic applications. The rare exceptions are cases such as the c-DLSE assumption (Patel and Sundaram 1998; Gennaro 2000), an assumption on the difficulty of taking discrete logarithms when the random exponents are taken only from a small set, i.e., $\mathbb{Z}_{2^c}$ with $c = \omega(\log\log|G|)$ instead of $\mathbb{Z}_{|G|}$, or the Diffie-Hellman Indistinguishability (DHI) assumptions introduced by Canetti (1997). The difficulty of these assumptions is

[6] As a practical consequence, it means that for such problem families either this information has to be public, e.g., the group index should allow the derivation of the factorization of the order, or the group and generators are chosen by the same party which samples the problem instance.

[7] For SG this holds trivially as we required samplers to be polynomial-time in their inputs. The input of Sg are the outputs of a single call of a machine (SG) polynomially bounded by k and, therefore, can be polynomially upper bounded in k. As the class of polynomials is closed under polynomial composition this holds also for Sg and, using similar reasoning, also for SPI.

not necessarily their individual specification, e.g., c-DLSE could be defined by suitably restricting the domain of the *sol* part of a DL problem instance. The deeper problem is that proving relations among these and other assumptions seems to require very specific tools, e.g., for randomization and analysis of resulting success probabilities, and are difficult to generalize as desirable for a classification as presented here. However, it might be worthwhile to investigate in future work whether these cases can be addressed by treating the sampling probability distribution as an explicit parameter of the classification. To make this extension promising, one would have to first find a suitable categorization of sampling probability distributions which: (1) captures the assumptions currently not addressed, and (2) offers tools assisting in proving reductions in a generalizable fashion.

3.2 Classifying Discrete Log-Based Assumptions

In defining assumptions, a cryptographer has various degrees of freedom related to the concrete mathematical formulation of the assumption, e.g., what kind of attackers are considered or over what values the probability spaces are defined.

To shed some light in these degrees of freedom I classify intractability assumptions for problems related to DL and relevant to many cryptographic applications. I identify the following orthogonal parameters. Additionally, I give for each of these parameters in a corresponding sublist different values which can produce significantly different assumptions.

1. **Problem family**: The following problem families are useful (and often used) for cryptographic applications. As mentioned in Section 3.1.6 I define the problem family (or more precisely their problem instances) by a structure instance SI (described abstractly by G and g_i's) and a tuple $(priv, publ, sol)$:

   DL (Discrete Logarithm):

   $$PI_{\mathrm{DL}} \quad := \quad ((G, g), ((x), (g^x), \{(x)\})).$$

   DH (Diffie-Hellman):

   $$PI_{\mathrm{DH}} \quad := \quad ((G, g), ((x, y), (g^x, g^y), \{(g^{xy})\}))$$

   GDH(n) (Generalized Diffie-Hellman for $n \geq 2$):

   $$\begin{aligned} PI_{\mathrm{GDH}(n)} \quad := \quad & ((G, g), ((x_i | i \in \{1, \ldots, n\}), \\ & (g^{\prod_{i \in I} x_i} \mid I \subset \{1, \ldots, n\}), \ \{(g^{\prod_{i=1}^{n} x_i})\})), \end{aligned}$$

where n is a fixed parameter.[8]

SE (Square-Exponent):

$$PI_{\mathrm{SE}} \quad := \quad ((G,g),((x),(g^x),\{(g^{x^2})\})).$$

IE (Inverse-Exponent):

$$PI_{\mathrm{IE}} \quad := \quad ((G,g),((x),(g^x),\{(g^{x^{-1}})\})).$$

Note that for elements $x' \in \mathbb{Z}_{|G|} \setminus \mathbb{Z}^*_{|G|}$ the value x^{-1} is not defined. Therefore, ${PI_{\mathrm{IE}}}^{priv}$ $(= (x))$ has to contain an element of $\mathbb{Z}^*_{|G|}$, contrary to the previously mentioned problem families where *priv* consists of elements from $\mathbb{Z}_{|G|}$.

RP(n) (Representation Problem for $n \geq 2$):

$$PI_{\mathrm{RP}(n)} \quad := \quad ((G,g_1,\ldots,g_n),\ ((x_i \mid i \in \{1,\ldots,n\}),\ (\prod_{i=1}^{n} g_i^{x_i}),$$
$$\{(x'_i \mid i \in \{1,\ldots,n\}) \mid (x'_i \in \mathbb{Z}_{|G|}) \wedge (\prod_{i=1}^{n} g_i^{x'_i} = \prod_{i=1}^{n} g_i^{x_i})\})),$$

where n is a fixed parameter.[9]

IAE (Inverted Additive Exponent Problem):

$$PI_{\mathrm{IAE}} \quad := \quad ((G,g),((x,y),(g^{1/x},g^{1/y}),\{(g^{1/(x+y)})\})).$$

Similar to IE, ${PI_{\mathrm{IAE}}}^{priv}$ $(= (x,y))$ consists of elements from $\mathbb{Z}^*_{|G|}$. Additionally, it has to hold that $x + y \in \mathbb{Z}^*_{|G|}$.

2. **Problem type**: Each problem can be formulated in three variants.

C (Computational): For a given problem instance PI an algorithm A succeeds if and only if it can solve PI, i.e., $\mathsf{A}(\ldots, PI^{publ}) \in PI^{sol}$. For the Diffie-Hellman problem family this means that A gets g^x and g^y as input and the task is to compute g^{xy}.

There is a small twist in the meaning of $\mathsf{A}(\ldots, PI^{publ}) \in PI^{sol}$: As $|G|$ is not necessarily known, A might not be able to represent elements of $\mathbb{Z}_{|G|}$ required in the solution set uniquely in their "principal" representation as elements of $\{0,\ldots,|G|-1\}$. Therefore, we allow A in these cases to return elements of $\mathbb{Z}$ and we implicitly reduce them mod$|G|$.

[8] A slightly generalized form GDH($n(k)$) would allow n to be a function in k. However, this function can grow at most logarithmically (otherwise the tuple would be of super-polynomial size!)

[9] Similar to GDH(n) one can also define here a slightly generalized form RP($n(k)$). In this case, one can allow $n(k)$ to grow even polynomially.

D (Decisional): For a given problem instance PI_0, a random problem instance PI_1 chosen with the same structure instance using the corresponding problem instance sampler and a random bit b, the algorithm A succeeds if and only if it can decide whether a given solution chosen randomly from the solution set of one of the two problem instances corresponds to the given problem instance, i.e., $\mathsf{A}(\ldots, PI^{publ}, sol_c)) = b$ where $sol_c \xleftarrow{\mathcal{R}} {PI_b}^{sol}$.[10] For the Diffie-Hellman problem family this means that A gets g^x, g^y and g^c (where c is either xy or $x'y'$ for $x', y' \in_{\mathcal{R}} \mathbb{Z}_{|G|}$) as input and the task is to decide whether g^c is g^{xy} or not.

M (Matching): For two given problem instances PI_0 and PI_1 and a random bit b, the algorithm A succeeds if and only if it can correctly associate the given solutions with their corresponding problem instances, i.e., $\mathsf{A}(\ldots, {PI_0}^{publ}, {PI_1}^{publ}, sol_b, sol_{\bar{b}}) = b$ where $sol_0 \xleftarrow{\mathcal{R}} {PI_0}^{sol}$ and $sol_1 \xleftarrow{\mathcal{R}} {PI_1}^{sol}$. For the Diffie-Hellman problem family this means that A gets g^{x_0}, g^{y_0}, g^{x_1}, g^{y_1}, $g^{x_b y_b}$ and $g^{x_{\bar{b}} y_{\bar{b}}}$ as input and the task is to predict b.

Initially, only computational assumptions, which follow naturally from informal security requirements, were considered in cryptography. For example, a key exchange protocol should prevent the complete recovery of the key which is usually the solution part of an assumption. However, the later formalization of security requirements, in particular semantic security (Goldwasser and Micali 1984), requires often the indistinguishability of random variables. Taking again the example of a key exchange protocol, it was realized that if you do not want to make strong requirements on the particular use of exchanged keys but allow the modular and transparent composition of key exchange protocols with other protocols, e.g., for secure sessions, it is essential that an exchanged key is indistinguishable from random keys, i.e., not even partial information on the key is leaked. While this does not necessarily imply decisional assumptions, such assumptions might be indispensable for efficient systems: There is an efficient encryption scheme secure against adaptive adversaries under the Decisional Diffie-Hellman assumption (Cramer and Shoup 1998). Nonetheless, no system is known today which achieves the same security under a similar com-

[10] This definition differs subtly from most other definitions of decisional problems: Here the distribution of the challenge sol_c is for $b = 1$, i.e., the random "wrong" challenge, according to the distribution of sol induced by SPI whereas most others consider it to be a (uniformly chosen) random element of G. Taking DIE or DDH with groups where the order has small factors these distributions are quite different! Conceptually, the definition here seems more reasonable, e.g., in a key exchange protocol you distinguish a key from an arbitrary key, not an arbitrary random value. It also addresses nicely the case of samplers with non-uniform distributions.

putational assumption in the standard model.[11] Finally, the matching variant was introduced by Frankel, Tsiounis, and Yung (1996) where it showed to be a useful tool to construct fair off-line cash. Handschuh, Tsiounis, and Yung (1999) later showed that the matching and the decisional variants of Diffie-Hellman are equivalent, a proof which is adaptable also to other problem families.

3. **Group family**: Various group families are used in cryptographic applications. The following list contains some of the more common ones. For brevity I do not mention the specific parameter choice as a function of k. I refer you to, e.g., Lenstra and Verheul (2001), for concrete proposals:

   $\mathbb{Z}_p^*$**:** The multiplicative groups of integers modulo a prime p with group order $\varphi(p)$ having at least one large prime factor. The group index is p.

   $\mathbb{Z}_{p/q}^*$**:** The subgroups of $\mathbb{Z}_p^*$ of prime order q. The group index is the tuple (p, q).

   $\mathbb{Z}_n^*$**:** The multiplicative groups of integers modulo a product n of two (or more) large primes p and q with $p-1$ and $q-1$ containing at least one large prime factor. The group index is n.[12]

   $\mathbb{QR}_n^*$**:** The subgroups of $\mathbb{Z}_n^*$ formed by the quadratic residues with n product of two large safe[13] primes. The group index is n.

   $E_{a,b}/\mathbb{F}_p$**:** The elliptic curves over $\mathbb{F}_p$ with p and $|E_{a,b}|$ prime with group index (a, b, p).

   The concrete choice of a group family has significant practical impact on aspects such as computation or bandwidth efficiency or suitability for a particular hardware but discussing this goes beyond the scope of this document, namely comparing assumptions. In this scope, it is mostly sufficient to classify simple and abstract properties of the chosen family and the public knowledge about a given group. I established the following two general criteria:

   (a) The factorization of the group order contains

   lprim: large prime factors (at least one). Formally, it has to hold that (with $\mathbb{P}$ being the set of prime numbers):

   $$\forall d > 0\, \exists k_0\, \forall k > k_0\, \forall G \in \mathcal{G}_{SG(k)}\, \exists p \in \mathbb{P}\, \exists r \in \mathbb{N} \,:\, |G| = pr \wedge p > k^d,$$

[11] There are efficient schemes known in the random oracle model (Bellare and Rogaway 1993), e.g., OAEP (Bellare and Rogaway 1995a; Boneh 2001; Shoup 2001; Fujisaki et al. 2001). However, this model is strictly weaker than the standard model and has a number of caveats (Canetti et al. 1998).

[12] This means that the order of the group is secret if we assume factoring n is hard.

[13] A prime p is a safe prime when $p - 1 = 2p'$ and $p' \in \mathbb{P}$.

nsprim: no small prime factor. Formally, the following has to hold:

$$\forall d>0\,\exists k_0\,\forall k>k_0\,\forall G\in\mathcal{G}_{SG(k)}\,\nexists p\in\mathbb{P}\,\exists r\in\mathbb{N}\,:\,|G|=pr\wedge p<k^d,$$

prim: only a single and large prime factor.

Note that this is a strict hierarchy and later values imply earlier ones. There would also be an obvious fourth value, namely the order contains no large factor. However, in such cases no reasonable DL based assumption seems possible (Pohlig and Hellman 1978; Pollard 1978).

(b) The group order is publicly

$\overline{\text{o}}$: unknown,

o: known,

fct: known including its complete[14] factorization.

I assume any such public knowledge to be encoded in the description returned by a group sampler SG. Note that in practice the group order is never completely unknown: at least an efficiently computable upper bound $B(|G|)$ can always be derived, e.g., from the bit-length of the representation of group elements. This can be exploited, e.g., in achieving **random self-reducibility**[15] (Blum and Micali 1984) for DDH even in the case where the order is not known (Boneh 1998).

The cryptographic application will determine which of above properties hold, e.g., a verifiable group generation will quite likely result in a publicly known factorization.

Furthermore, note that the group families given above implicitly fix the properties of the group order factorization ($\mathbb{Z}_p^*$: lprim; $\mathbb{Z}_{p/q}^*$: prim; $\mathbb{Z}_n^*$: lprim; $\mathbb{QR}_n^*$: nsprim; $E_{a,b}/\mathbb{F}_p$: prim), and the public knowledge about it ($\mathbb{Z}_p^*$: o; $\mathbb{Z}_{p/q}^*$: fct; $\mathbb{Z}_n^*$: $\overline{\text{o}}$; $\mathbb{QR}_n^*$: $\overline{\text{o}}$; $E_{a,b}/\mathbb{F}_p$: fct).

4. **Computational capability of adversary**: Potential algorithms solving a problem have to be computationally limited for number-theoretic assumptions to be meaningful (otherwise we could never assume their nonexistence). Here, I only consider probabilistic polynomial-time algorithms (called **adversaries** in the following). The adversary can be of

[14] If the order is known then small prime factors can always be computed. Insofar the case here extends the knowledge about the factorization also to large prime factors.

[15] Informally, a problem is random self-reducible if solving *any* problem instance can be reduced to solving the problem on a *random* instance, i.e., when given an instance x we can efficiently randomize it to a random instance $x_{\mathcal{R}}$ and can efficiently derive (derandomize) the solution for x from the solution returned by an oracle call on $x_{\mathcal{R}}$.

u (Uniform complexity): There is a single probabilistic Turing machine A which for any given finite input returns a (not necessarily correct) answer in polynomial time in its input length. As the complexity of Turing machines is measured in the bit-length of the inputs the inputs should be neither negligible nor super-polynomial in the security parameter k, otherwise the algorithm might not be able to write out the complete desired output or might become too powerful. To address this issue one normally passes an additional input 1^k to A to lower bound the complexity and makes sure that the other inputs can be polynomially upper bounded in k. In all cases considered here, the inputs in the assumptions are already proportional to the security parameters, see remarks on the size of groups and on the runtime of samplers in Section 3.1.7. Therefore we can safely omit 1^k in the inputs of A.

n (Non-uniform complexity): There is an (infinite) family of Turing machines $(\mathsf{A}_k \mid k \in \mathbb{N})$ with description size and running time of A_k bounded by a polynomial in the security parameter k.[16] Equivalent alternatives are a (single) Turing Machine with polynomial running time and an additional (not necessarily computable) family of auxiliary inputs polynomially bounded by the security parameter, or families of circuits with the number of gates polynomially bounded by the security parameter,[17] respectively.

Uniform assumptions are (in many cases strictly) weaker than corresponding non-uniform assumptions as any uniform algorithm is also a non-uniform one. Furthermore, all uniform black-box reductions map to the non-uniform case (but not necessarily vice-versa!) and henceforth most uniform proofs should map to their non-uniform counterpart. This makes uniform assumptions preferable over non-uniform assumptions (e.g., honest users are normally uniform and weaker assumptions are always preferable over stronger ones). However, uniform assumptions also assume uniform adversaries which is a weaker adversary model than the model considering non-uniform adversaries. Furthermore, there are proofs which only work in a non-uniform model.

Further, potentially interesting yet currently ignored, attacker capabilities would be bounds on space instead of (or in addition) to time. Adaptive adversaries do not seem of concern for pure assumptions.

[16] The remarks on input length and runtime mentioned above for uniform complexity also apply here.

[17] In the case of circuits the bound on the running time automatically follows and does not have to be explicitly restricted.

Ideally, one would consider larger, i.e., less restricted, classes of adversaries than the strictly polynomial-time one following from the definition from Section 3.1.3. It would seem more natural, e.g., to require polynomial behavior only on inputs valid for a given assumption or to allow algorithms, e.g., Las Vegas algorithms, with no a-priori bound on the runtime.[18] Unfortunately, such classes are difficult to define properly and even harder to work with. However, as for each adversary of these classes, there seems to be a closely related (yet not necessarily black-box constructible) strictly polynomial-time adversary with similar success probability, this restriction seems of limited practical relevance.

5. **"Algebraic knowledge"**: A second parameter describing the adversary's computational capabilities relates to the adversary's knowledge on the group family. It can be one of the following:

   σ (Generic): This means that the adversary does not know anything about the structure (representation) of the underlying algebraic group. More precisely this means that all group elements are represented using an **encoding function** $\sigma(\cdot)$ drawn randomly from the set $\Sigma_{G,g}$ of bijective[19] functions $\mathbb{Z}_{|G|} \to G$. Group operations can only be performed via the addition and inversion[20] oracles $\sigma(x+y) \leftarrow \sigma_+(\sigma(x), \sigma(y))$ and $\sigma(-x) \leftarrow \sigma_-(x)$ respectively, which the adversary receives as a black box (Shoup 1997; Nechaev 1994) together with $\sigma(1)$, the generator.

   If I use σ in the following, I always mean the (not further specified) random encoding used for generic algorithms with a group G and generator g implied by the context. In particular, by A^σ I refer to a generic algorithm. To prevent clutter in the presentation, I do not explicitly encode group elements passed as inputs to such generic algorithms. However, they should all be considered suitable encoded with σ.

   (marked by absence of σ) (Specific): In this case the adversary can also exploit special properties (e.g., the encoding) of the underlying group.

[18] However, we would have to restrict the considerations to polynomial time runs when measuring the success probability of adversaries.

[19] Others, e.g., Babai and Szemerédi (1984) and Boneh and Lipton (1996), considered the more general case where elements are not necessarily unique and there is a separate equality oracle. However, that model is too weak to cover some important algorithms, e.g., Pohlig and Hellman (1978), which are intuitively "generic". Furthermore, the impossibility results mentioned later still hold when transfered to the more general case.

[20] Computing inverses is usually efficient only when the group order is known. However, note that all impossibility results — the main use of generic adversaries — considered later hold naturally also without the inversion oracle.

This separation is interesting for the following reasons:

- Tight lower bounds on the complexity of some DL-based assumptions can lead to provably hard assumptions in the generic model (Shoup 1997; Maurer and Wolf 1998b). No such results are known in the standard model. However, similar to the random oracle model (Bellare and Rogaway 1993) the generic model is idealized and related pitfalls lure when used in a broader context than simple assumptions (Fischlin 2000).
- A number of algorithms computing discrete logarithms are generic in their nature. Two prominent ones are Pohlig-Hellman (1978) and Pollard-ρ (1978) paired with Shanks Baby-Step Giant-Step optimization. Furthermore, most reductions are generic.
- However, exploiting some structure in the group can lead to faster algorithms, e.g., for finite fields there is the class of index-calculus methods and in particular the generalized number field sieve (GNFS) (Gordon 1993b; Schirokauer 1993) with sub-exponential expected running time.
- Nonetheless, for many group families, e.g., elliptic curves, no specific algorithms are known which compute the discrete logarithms better than the generic algorithms mentioned above.

Note that a generic adversary can always be transformed to a specific adversary but not necessarily vice-versa. Therefore, a reduction between two generic assumptions is also a reduction between the specific counterparts of the two assumptions. However, proofs of the hardness of generic assumptions or the non-existence of relations among them do *not* imply their specific counterparts!

6. **"Granularity of probability space"**: Depending on what part of the structure instance is a-priori fixed (i.e., the assumption has to hold for all such parameters) or not (i.e., the parameters are part of the probability space underlying an assumption) we can distinguish among the following situations:

   l (Low-granular): The group family (e.g., prime order subgroups of $\mathbb{Z}_p^*$) is fixed but not the specific structure instance (e.g., parameters p, q and generators g_i for the example group family given above).

   m (Medium-granular): The group (e.g., p and q) but not the generators g_i are fixed.

   h (High-granular): The group as well as the generators g_i are fixed.

An assumption defines a family of probability spaces $\mathcal{D}_i$, where the index i is the tuple of k and, depending on granularity, group and generator, i.e., all parameters with an all-quantifier in the assumption statement. Each probability space $\mathcal{D}_i$ is defined over problem instances, random coins for the adversary, and, again depending on granularity, groups and generators. Note that for a given k there are always only polynomially many $\mathcal{D}_i$. In the sequel I use the term **probability space instance (PSI)** for a single probability space $\mathcal{D}_i$.

7. **Success probability**: This parameter gives an (asymptotic) upper bound on how large a success probability we tolerate from an adversary. The success probability is measured over the family of probability space instances $\mathcal{D}_i$. Violation of an assumption means that there exists an algorithm A whose success probability $\alpha(k)$ reaches or exceeds this bound for infinitely many k in respect to at least one of the corresponding probability space instances $\mathcal{D}_i$.

   The upper bound and the corresponding adversary can be classified in the following types:

   1 (Perfect): The strict upper bound on the success probability is 1. Therefore, a perfect adversary algorithm A with success probability $\alpha(k)$ has to solve the complete probability mass of infinitely many $\mathcal{D}_i$, i.e., $\alpha(k) \not<_\infty 1$.

   $(1-1/\mathsf{poly}(k))$ (Strong): The bound is defined by the error probability which has to be non-negligible. Therefore, a strong adversary algorithm A has to be successful for infinitely many $\mathcal{D}_i$ with overwhelming probability., i.e., if $\alpha(k)$ is the success probability of A then $1 - \alpha(k) \not\geq_\infty 1/\mathsf{poly}(k)$.

   ϵ (Invariant): The strict upper bound is a fixed and given constant $0 < \epsilon < 1$. Therefore, the success probability $\alpha(k)$ of an invariant adversary algorithm A has to be larger than ϵ for infinitely many $\mathcal{D}_i$, i.e., $\alpha(k) \not<_\infty \epsilon$.

   $1/\mathsf{poly}(k)$ (Weak): All non-negligible functions are upper bounds, i.e., only negligible success probabilities are tolerated. Therefore, a weak adversary algorithm A has to be successful with a not negligible fraction of the probability mass of $\mathcal{D}_i$ for infinitely many $\mathcal{D}_i$, i.e., if $\alpha(k)$ is the success probability of A then $\alpha(k) \not<_\infty 1/\mathsf{poly}(k)$.

   An assumption requiring the nonexistence of perfect adversaries corresponds to worst-case complexity, i.e., if the assumption holds then there are at least a few hard instances. However, what is a-priori required in most cases in cryptography is a stronger assumption requiring

the nonexistence of even weak adversaries, i.e., if the assumption holds then most problem instances are hard.

The classification given above is certainly not exhaustive. The exploration of new problem families, e.g., related to arbitrary multivariate functions in the exponents as investigated by Kiltz (2001), might require additional values for the existing parameters. This can be done without much impact on the classification itself and other results. However, the need for a new dimension such as adding probability distributions as a separate parameter (see Section 3.1.7) would be of much larger impact. Nevertheless, from the current experience, above classification seems quite satisfactory.

3.3 Defining Assumptions

Using the parameters and corresponding values defined in the previous section, we can define intractability assumptions in a compact and precise way.

The notation for a given assumption is

$$\$\text{s-}\$\text{t}\$\mathcal{P}^{\$\text{a}}(\text{c:}\$\text{c; g:}\$\text{g; f:}\$\mathcal{G})$$

where for each parameter there is a placeholder \$X which is instantiated by the labels corresponding to the value of that parameter in the given assumption. The placeholders and values (with $-$ denoting that this value can be absent in the notation and has the same meaning as a corresponding wild card) are as follows:

- \$s: The algorithm's success probability ($\$s \in \{1, (1-1/\mathsf{poly}(k)), \epsilon, 1/\mathsf{poly}(k)\}$).
- \$t: The problem type ($\$t \in \{\mathrm{C}, \mathrm{D}, \mathrm{M}\}$).
- $\$\mathcal{P}$: The problem family ($\$\mathcal{P} \in \{\mathrm{DL}, \mathrm{DH}, \mathrm{GDH}(n), \mathrm{SE}, \mathrm{IE}, \mathrm{RP}(n), \mathrm{IAE}\}$).
- \$a: The algebraic knowledge of the algorithm ($\$a \in \{\sigma, -\}$).
- \$c: The algorithm's complexity ($\$c \in \{u, n\}$).
- \$g: The granularity of the probability space ($\$g \in \{h, m, l\}$).
- $\$\mathcal{G}$: The group family ($\$\mathcal{G} \in \{\mathrm{lprim}, \mathrm{nsprim}, \mathrm{prim}, -\} \times \{\overline{\mathrm{o}}, \mathrm{o}, \mathrm{fct}, -\} \times \{\mathbb{Z}_p^*, \mathbb{Z}_{p/q}^*, \mathbb{Z}_n^*, \mathbb{QR}_n^*, E_{a,b}/\mathbb{F}_p, -\}$).[21]

[21] The parameters for $\mathcal{G}$ are not completely orthogonal in the sense that some combinations do not exist, e.g., $(\mathrm{prim}, \cdot, \mathbb{QR}_n^*)$, and some result in nonsensical assumptions, e.g., $(\cdot, \mathrm{fct}, \mathbb{Z}_n^*)$. Nonetheless, the assumptions still can be defined and insofar this is not really of concern here.

This is best illustrated in an example: The term

$$1/\mathsf{poly}(k)\text{-DDH}^{\sigma}(\text{c:u}; \text{g:h}; \text{f:prim})$$

denotes the decisional (D) Diffie-Hellman (DH) assumption in prime-order groups (f:prim) with weak success probability ($1/\mathsf{poly}(k)$), limited to generic algorithms (σ) of uniform complexity (c:u), and with high granularity (g:h).

To refer to classes of assumptions I use **wild cards** ($*$) and sets ($\{\cdots\}$) of parameter values, e.g.,

$$\{(1-1/\mathsf{poly}(k)),\epsilon,1/\mathsf{poly}(k)\}\text{-CDH}^{\sigma}(\text{c:u}; \text{g:h}; \text{f:}*)$$

denotes the class of computational (C) Diffie-Hellman (DH) assumptions with uniform complexity (c:u), limited to generic algorithms (σ), with high-granular probability space (g:h), with some error ($\{(1 - 1/\mathsf{poly}(k)), \epsilon, 1/\mathsf{poly}(k)\}$) and based on an arbitrary group family (f:$*$).

Let us turn now to the meaning of an assumption described by above notation: By stating that an assumption \$s-\$t$\$\mathcal{P}^{\$a}$(c:\$c; g:\$g; f:$\$\mathcal{G}$) holds, we believe that asymptotically no algorithm of complexity \$c and algebraic knowledge \$a can solve (random) problem instances of a problem family $\$\mathcal{P}$ with problem type \$t chosen from groups in $\$\mathcal{G}$ with sufficient (as specified by \$s) success probability where the probability space is defined according to granularity \$g.

The precise and formal definitions follow naturally and quite mechanically. In defining an assumption we always require a bound k_0 for the asymptotic behavior which says that beyond that bound no adversary will be successful. As further "ingredients" there are polynomials defined by their maximal degree d_1, d_2 and d_3 which bind the error probability, time and description of programs, respectively. Finally, we require a machine (or family thereof) A (A_i) trying to solve the problem, and various quantifiers specifying (using the various samplers) the required parameters for a problem instance PI to solve.

Finally, I denote the class of uniform complexity adversaries by $\mathcal{UPTM}$ and the corresponding class of generic adversaries by $\mathcal{UPTM}^{\sigma}$. The class of non-uniform complexity and generic non-uniform complexity adversaries is denoted similarly by $\mathcal{NPTM}$ and $\mathcal{NPTM}^{\sigma}$, respectively.

To illustrate the formal details of assumptions and to provide a feel for the various parameters I offer three sets of examples. In each set I vary one of the parameters, namely: (1) the computational complexity, (2) the less obvious and often overlooked granularity parameter, and (3) the success probability. The complete details on how to derive the formal assumption statement from the parameters can be found in Appendix A:

1. Weak computational DL assumptions in the generic model, a group order with at least one large prime factor and the two variants of

complexity measures (see Parameter 4). Remember that $PI_{\mathrm{DL}} := (SI, ((x), (g^x), \{(x)\}))$, ${PI_{\mathrm{DL}}}^{publ} := (g^x)$ and ${PI_{\mathrm{DL}}}^{sol} := \{(x)\}$. Further, let $SG_{\mathcal{G}}$ be a group sampler of some group family $\mathcal{G}$ where the groups have an order with at least one large prime factor.

(a) Assumption $1/\mathsf{poly}(k)$-CDL$^{\sigma}$(c:u; g:h; f:lprim), i.e., the uniform complexity variant:

$$
\begin{array}{l}
\forall \mathsf{A}^{\sigma} \in \mathcal{UPTM}^{\sigma}; \\
\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0; \\
\forall G \in [SG_{\mathcal{G}}(1^k)]; \\
\forall g \in [Sg(G)]; \\
SI \leftarrow (G, g); \\
\\
\mathbf{Prob}[\mathsf{A}^{\sigma}(\mathcal{C}, SI, {PI_{\mathrm{DL}}}^{publ}) \in {PI_{\mathrm{DL}}}^{sol} :: \\
\quad \sigma \xleftarrow{\mathcal{R}} \Sigma_{G,g}; \\
\quad PI_{\mathrm{DL}} \leftarrow SPI_{\mathrm{DL}}(SI); \\
\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U} \\
\quad] < 1/k^{d_1}.
\end{array}
$$

(b) Same setting as above except now with a non-uniform adversary ($1/\mathsf{poly}(k)$-CDL$^{\sigma}$(c:n; g:h; f:lprim)):

$$
\begin{array}{l}
\forall (\mathsf{A}_i^{\sigma} \mid i \in \mathbb{N}) \in \mathcal{NPTM}^{\sigma}; \\
\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0; \\
\forall G \in [SG_{\mathcal{G}}(1^k)]; \\
\forall g \in [Sg(G)]; \\
SI \leftarrow (G, g); \\
\\
\mathbf{Prob}[\mathsf{A}_k^{\sigma}(\mathcal{C}, SI, {PI_{\mathrm{DL}}}^{publ}) \in {PI_{\mathrm{DL}}}^{sol} :: \\
\quad \sigma \xleftarrow{\mathcal{R}} \Sigma_{G,g}; \\
\quad PI_{\mathrm{DL}} \leftarrow SPI_{\mathrm{DL}}(SI); \\
\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U} \\
\quad] < 1/k^{d_1}.
\end{array}
$$

2. Weak decisional DH assumption variants for prime order subgroups of $\mathbb{Z}_p^*$ with varying granularity. Recall that $PI_{\mathrm{DH}} := (SI, ((x, y), (g^x, g^y), \{(g^{xy})\}))$, ${PI_{DH}}^{publ} := (g^x, g^y)$ and ${PI_{DH}}^{sol} := \{(g^{xy})\}$.

(a) Assumption $1/\mathsf{poly}(k)$-DDH(c:u; g:h; f:$\mathbb{Z}_{p/q}^*$), i.e., with high granularity:

$\forall \mathsf{A} \in \mathcal{UPTM}$;
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0$;
$\forall G \in [SG_{\mathbb{Z}^*_{p/q}}(1^k)]$;
$\forall g \in [Sg(G)]$;
$SI \leftarrow (G, g)$;

$(|\ \mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, PI_{\mathrm{DH}/0}{}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
$b \xleftarrow{\mathcal{R}} \{0,1\}$;
$PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$sol_{\mathrm{DH}/c} \xleftarrow{\mathcal{R}} PI_{\mathrm{DH}/b}{}^{sol}$;
$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$]-1/2 \mid \cdot 2) \ < 1/k^{d_1}$.

(b) As above except now with medium granularity ($1/\mathsf{poly}(k)$-DDH(c:u; g:m; f:$\mathbb{Z}^*_{p/q}$)):

$\forall \mathsf{A} \in \mathcal{UPTM}$;
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0$;
$\forall G \in [SG_{\mathbb{Z}^*_{p/q}}(1^k)]$;

$(|\ \mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, PI_{\mathrm{DH}/0}{}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
$g \leftarrow Sg(G)$;
$SI \leftarrow (G, g)$;
$b \xleftarrow{\mathcal{R}} \{0,1\}$;
$PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$sol_{\mathrm{DH}/c} \xleftarrow{\mathcal{R}} PI_{\mathrm{DH}/b}{}^{sol}$;
$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$]-1/2 \mid \cdot 2) \ < 1/k^{d_1}$.

(c) As above except now with low granularity ($1/\mathsf{poly}(k)$-DDH(c:u; g:l; f:$\mathbb{Z}^*_{p/q}$)):

$\forall \mathsf{A} \in \mathcal{UPTM}$;
$\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0$;

$(|\,\mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, PI_{\mathrm{DH}/0}{}^{publ}, sol_{\mathrm{DH}/c}) = b ::$
$G \leftarrow SG_{\mathbb{Z}^*_{p/q}}(1^k)$;
$g \leftarrow Sg(G)$;
$SI \leftarrow (G, g)$;
$b \xleftarrow{\mathcal{R}} \{0, 1\}$;
$PI_{\mathrm{DH}/0} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$PI_{\mathrm{DH}/1} \leftarrow SPI_{\mathrm{DH}}(SI)$;
$sol_{\mathrm{DH}/c} \xleftarrow{\mathcal{R}} PI_{\mathrm{DH}/b}{}^{sol}$;
$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$]-1/2 \mid \cdot 2) \ < 1/k^{d_1}$.

3. Matching IE assumptions in $\mathbb{QR}^*_n$ with varying success probability. Recall that $PI_{\mathrm{IE}} := (SI, ((x), (g^x), \{(g^{x^{-1}})\}))$, $PI_{IE}{}^{publ} := (g^x)$ and $PI_{IE}{}^{sol} := \{(g^{x^{-1}})\}$.

   (a) Assumption $1/\mathsf{poly}(k)$-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$), i.e., the variant with weak success probability:

   $\forall \mathsf{A} \in \mathcal{UPTM}$;
   $\forall d_1 > 0;\ \exists k_0;\ \forall k > k_0$;
   $\forall G \in [SG_{\mathbb{QR}^*_n}(1^k)]$;
   $\forall g \in [Sg(G)]$;
   $SI \leftarrow (G, g)$;

   $(|\,\mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, PI_{\mathrm{IE}/0}{}^{publ}, PI_{\mathrm{IE}/1}{}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
   $b \xleftarrow{\mathcal{R}} \{0, 1\}$;
   $PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI)$;
   $PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI)$;
   $sol_{\mathrm{IE}/0} \xleftarrow{\mathcal{R}} PI_{\mathrm{DH}/0}{}^{sol}$;
   $sol_{\mathrm{IE}/1} \xleftarrow{\mathcal{R}} PI_{\mathrm{DH}/1}{}^{sol}$;
   $\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
   $]-1/2 \mid \cdot 2) \ < 1/k^{d_1}$.

   (b) Same setting as above except now with invariant success probability ϵ (ϵ-MIE(c:u; g:h; f:$\mathbb{QR}^*_n$)):

$\forall \mathsf{A} \in \mathcal{UPTM}$;
$\exists k_0;\ \forall k > k_0$;
$\forall G \in [SG_{\mathbb{QR}_n^*}(1^k)]$;
$\forall g \in [Sg(G)]$;
$SI \leftarrow (G, g)$;

$(|\ \mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
$b \xleftarrow{\mathcal{R}} \{0,1\}$;
$PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI)$;
$PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI)$;
$sol_{\mathrm{IE}/0} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/0}}^{sol}$;
$sol_{\mathrm{IE}/1} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/1}}^{sol}$;
$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$;

$]-1/2 \mid \cdot 2) \ < \epsilon$.

(c) Same setting as above except now with strong success probability $((1-1/\mathsf{poly}(k))$-MIE(c:u; g:h; f:$\mathbb{QR}_n^*$)):

$\forall \mathsf{A} \in \mathcal{UPTM}$;
$\exists d_1 > 0;\ \exists k_0;\ \forall k > k_0$;
$\forall G \in [SG_{\mathbb{QR}_n^*}(1^k)]$;
$\forall g \in [Sg(G)]$;
$SI \leftarrow (G, g)$;

$(|\ \mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::$
$b \xleftarrow{\mathcal{R}} \{0,1\}$;
$PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI)$;
$PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI)$;
$sol_{\mathrm{IE}/0} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/0}}^{sol}$;
$sol_{\mathrm{IE}/1} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/1}}^{sol}$;
$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$
$]-1/2 \mid \cdot 2) \ < (1 - 1/k^{d_1})$.

(d) Same setting as above except with no tolerated error, i.e., perfect success probability (1-MIE(c:u; g:h; f:$\mathbb{QR}_n^*$)):

$$
\begin{array}{l}
\forall \mathsf{A} \in \mathcal{UPTM};\\
\exists k_0;\ \forall k > k_0;\\
\forall G \in [SG_{\mathbb{QR}_n^*}(1^k)];\\
\forall g \in [Sg(G)];\\
SI \leftarrow (G, g);\\
\\
(|\,\mathbf{Prob}[\mathsf{A}(\mathcal{C}, SI, {PI_{\mathrm{IE}/0}}^{publ}, {PI_{\mathrm{IE}/1}}^{publ}, sol_{\mathrm{IE}/b}, sol_{\mathrm{IE}/\bar{b}}) = b ::\\
\quad b \xleftarrow{\mathcal{R}} \{0,1\};\\
\quad PI_{\mathrm{IE}/0} \leftarrow SPI_{\mathrm{IE}}(SI);\\
\quad PI_{\mathrm{IE}/1} \leftarrow SPI_{\mathrm{IE}}(SI);\\
\quad sol_{\mathrm{IE}/0} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/0}}^{sol};\\
\quad sol_{\mathrm{IE}/1} \xleftarrow{\mathcal{R}} {PI_{\mathrm{DH}/1}}^{sol};\\
\quad \mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}\\
\quad]-1/2\ |\cdot 2)\ < 1.
\end{array}
$$

To express relations among assumptions, I use the following operators where P and Q are assumptions as previously defined:

$P \implies Q$ means that if assumption P holds, so does assumption Q, i.e., P (Q) is a stronger (weaker) assumption than Q (P). Vice-versa, it also means that if there is a polynomially-bounded algorithm A_Q breaking assumption Q then there is also another polynomially-bounded algorithm A_P which breaks assumption P. Usually, this is shown in a **black-box reduction** where A_P, or more precisely $\mathsf{A}_P^{\mathsf{A}_Q}$, breaks assumption P with oracle access to A_Q. As a special case for invariant assumptions, I mean with $\epsilon\text{-}P \implies \epsilon\text{-}Q$ that it should hold that $\forall \epsilon' \in]0,1[\ \exists \epsilon'' \in]0,1[\ :\ \epsilon''\text{-}P \implies \epsilon'\text{-}Q$.

$P \iff Q$ means that $P \implies Q$ and $Q \implies P$, i.e., P and Q are assumptions of the same (polynomial) complexity.

$P \xRightarrow{\alpha' \geq f_\alpha(t,\alpha,|G|,\ldots);\ t' \leq f_t(t,\alpha,|G|,\ldots)} Q$ is used to specify the quality of the reduction, i.e., the concrete security. It means that if assumption Q can be broken in time t and with success probability α, we can break P in time t' and with success probability α' bounded by functions f_t and f_α, respectively. To measure time, I consider group operations and equality tests having unit-cost each and oracle calls having cost t. Obviously, the cost of group operations, the runtime and the success probability of the oracle, and the size of the groups are not constant but functions depending on the security parameter k, e.g., α should be written more precisely as $\alpha(k)$. However, for better readability I omit this and all asymptotic aspects in the presentation. For the identical reason, I also cautiously use the $O(\cdot)$ notation even if we slightly lose precision.

Let me illustrate this with the following result from Maurer and Wolf (1996):

$$\epsilon\text{-CDH}(\text{c:u}; \text{g:h}; \text{f:o}) \xRightarrow{\alpha' = \alpha^3;\ t' = 3t + O(\log(|G|)^2)} \epsilon\text{-CSE}(\text{c:u}; \text{g:h}; \text{f:o})$$

This means that with three calls to an oracle breaking ϵ-CSE(c:u; g:h; f:o) and additional $O(\log(|G|)^2)$ group operations we can achieve a success probability of at least α^3 in breaking ϵ-CDH(c:u; g:h; f:o) where t and α are the runtime and the success probability of the oracle, respectively.

For simple assumptions, above is interpreted without syntactical conditions on P and Q, i.e., they may be arbitrary assumptions. If a relation refers to assumption classes, i.e., they contain some parameters which are not fully specified and contain wild cards or sets, there is the following syntactical constraint: The parameters which are not fully specified have to be equal for both assumptions P and Q. The meaning is as follows: The relation P OP Q holds for any assumption P' and Q' we can instantiate from P and Q by fixing all not fully specified parameters to any matching value with the additional condition that these values are identical for P' and Q'. To give an example,

$$*\text{-CDH}^*(\text{c:}*; \text{g:}\{\text{h,m}\}; \text{f:o}) \implies *\text{-CSE}^*(\text{c:}*; \text{g:}\{\text{h,m}\}; \text{f:o})$$

illustrates that the result from Maurer and Wolf mentioned above can be generalized (Sadeghi and Steiner 2001) to high and medium granularity with arbitrary success probability, complexity and algebraic knowledge.

Furthermore, if I am referring to oracle-assumptions, i.e., assumptions where we give adversaries access to auxiliary oracles, I indicate it by listing the oracles at the end of the list in the assumption term. For example, the assumption $1/\mathsf{poly}(k)\text{-CDL}^{\sigma}(\text{c:u}; \text{g:h}; \text{f:lprim}; \mathcal{O}_{1\text{-CDL}(\text{c:u}; \text{g:h}; \text{f:lprim})})$ corresponds to the first assumption statement given in the example list above except that now the adversary also gets access to an oracle breaking the 1-CDL(c:u; g:h; f:lprim) assumption.

3.4 Granularity

It would go beyond the scope (and space) of this thesis to discuss all previously identified parameters; see Sadeghi and Steiner (2002) for more information. However, since this aspect was previously largely overlooked, I briefly focus on granularity, state its practical and theoretical relevance, and prove a theorem on the relation of assumptions with varying granularity required in the sequel.

The practical relevance of granularity was alluded to already in the introduction of this Chapter. As shown below, assumptions with lower granularity are weaker, and are as a consequence more desirable. However, which of the granularity variants is appropriate in cryptographic protocols depends on how and by whom the structure instance is chosen. Without presupposing special properties of this process, we are forced to use a high-granular assumption. Nonetheless, in the following situations we can resort to a less granular and, therefore, weaker assumption: The security requirements of the cryptographic system guarantee that it is in the best (and only) interest of the generating party of the structure instance to choose them properly; the structure instance is chosen by a mutually trusted third party; or the structure instance is chosen in a verifiable random process.[22] Also, at most in these cases we can reasonably assume a group family with the group order and its factorization to be hidden from the public and the adversary. As a consequence, it would seem strange to base a cryptographic system on a high-granularity assumption with unknown order factorization: either the system parameters are chosen by an honest party and we could resort to a weaker assumption with lower granularity, or the knowledge of the order and its factorization has to be assumed to be known to the adversary. Furthermore, care has to be taken for DL-related high- and medium-granularity assumptions in $\mathbb{Z}_p^*$ and its subgroups. Unless we further constrain the set of valid groups with (expensive) tests as outlined by Gordon (1993a), we require, for a given security parameter, considerably larger groups than for the low granular counterpart of the assumptions.

From a theoretical point of view, investigating granularity also uncovers some surprising results. Extending the results of Wolf (1999) to the problem family IE, Sadeghi and Steiner (2001) prove statements on relations between IE, DH and SE for both computational and decisional variants in the setting of Wolf (1999) which corresponds to the high-granular case. They then consider medium granularity (with other parameters unchanged) and show the impact: They prove that the decisional IE and SE assumptions are equivalent for medium granularity whereas this is provably not possible for their high-granular variants, at least not in the generic model. They also show that reductions between computational IE, SE and DH can offer much better concrete security for medium granularity than their high-granular analogues.

As informally mentioned above, assumptions with lower granularity are weaker than assumption of higher granularity. Formally, this is stated and proven in the following theorem:

[22] This can be done either through a joint generation using random coins (Cachin et al. 2000) or using heuristics such as the one used for DSS key generation (National Institute of Standards and Technology (NIST) 2000).

Theorem 3.1

$$*\text{-}**^{*}(\mathrm{c}{:}*;\mathrm{g}{:}\mathrm{h};\mathrm{f}{:}*) \implies *\text{-}**^{*}(\mathrm{c}{:}*;\mathrm{g}{:}\mathrm{m};\mathrm{f}{:}*) \implies *\text{-}**^{*}(\mathrm{c}{:}*;\mathrm{g}{:}\mathrm{l};\mathrm{f}{:}*)$$

□

Proof. Assume we are given an adversary A breaking a low-granularity assumption for some group and problem family, some problem type, computational complexity, arbitrary algebraic knowledge and success probability. Furthermore, we are given an input I corresponding to an assumption of high- or medium-granularity but otherwise identical parameters.

The reduction is simple: Just call A on this input I and return the result. To see that this achieves the desired attack on the medium- or high-granularity assumption, you have to note first that inputs to an adversary breaking a high- or medium-granularity assumption are also valid inputs to a low-granularity adversary. Therefore, this reduction is a legitimate attacker from a runtime perspective exactly in the case where the oracle itself is a legitimate attacker. Furthermore, the probability space instances defined by a high- or medium-granularity assumption always partition the probability space instances of a low-granularity assumption. Therefore, it it is clear that for a perfect adversary A the reduction breaks certainly the high- or medium granularity probability space instances which are part of the low-granularity probability space instances which A breaks. As there are by definition of A infinitely many such low-granularity probability space instances and for a given k there are only a finite number of probability space instances it automatically follows that for the perfect case the high- and medium granularity assumption is broken, too. By a counting argument this also easily extends to the case of strong, invariant and weak adversaries, i.e., at least some of the high- or medium granularity probability space instances which are part of the low-granularity probability space instances broken by A are broken with the necessary success probability as well.

By an identical argument it follows that a high-granularity assumption can be reduced to the corresponding medium-granularity assumption. This concludes the theorem. ■

Remark 3.1. Note that the inverse of above result, a low-granular assumption implies the corresponding high-granular one, does not hold in general: There are always super-polynomially many of the higher-granularity probability space instances contained in a given lower-granularity instance. Therefore, there might be situations where infinitely many high-granularity probability space instances — and henceforth the corresponding high-granularity assumption — are broken, yet they form only a negligible subset of the enclosing lower-granularity probability space instances and the low-granularity assumption can still hold.

However, if for a given granularity there exists a random self-reduction (Blum and Micali 1984), then the inverse reduction exists also from that granularity to all higher granularities. As random self-reductions are known for all mentioned problem families and problem types in their medium granularity variant, this equates the medium- and high-granularity cases. Unfortunately, no random self-reduction is yet known for low-granularity assumptions and achieving such "full" random self-reducibility seems very difficult in general (if not impossible) in number-theoretic settings (Boneh 2000) contrary to, e.g., lattice settings used by Ajtai and Dwork (1997). ○

3.5 Decisional Generalized Diffie-Hellman

The Decisional Generalized Diffie-Hellman Problem (DGDH(n)) was introduced by Steiner, Tsudik and Waidner (1996, 2000) and is a natural extension of the 2-party Decisional Diffie-Hellman assumption (DDH), first explored in a cryptographic context by Brands (1993), to an n-party case. The concrete form of the problem was already introduced in Section 3.2. However, for your convenience I shortly and informally repeat the problem statement: Given all **partial GDH keys** $\{g^{\prod_{\beta_i=1} x_i} \mid \beta \in I_n \backslash \{1^n\}\}$ and a value g^c, the task is to decide whether g^c is $g^{\prod x_i}$ or a random element of G. As we will see in Chapter 4, there is a large class of DH-based group-key protocols where the protocol flows consist of subsets of partial GDH keys. For these protocols, DGDH(n) is the natural assumption to base the security upon. However, DGDH(n) is not a standard assumption. Preferably, we could rely on a standard assumption such as DDH. DDH is used in many contexts (Boneh 1998) and assumed to hold for many cyclic groups, e.g., Shoup (1997) showed that no polynomial algorithm can solve DDH in the generic model if the group order contains only large prime factors. Luckily, Theorem 3.2 equates the two assumptions. The theorem is taken from Steiner, Tsudik, and Waidner (2000), adapted and generalized to the classification and notation introduced by Sadeghi and Steiner (2001) and explained in Section 3.2. Furthermore, the theorem is extended with the concrete security of the reduction:

Theorem 3.2

$$1/\mathsf{poly}(k)\text{-DDH}(\text{c:}*;\ \text{g:}*;\ \text{f:o}) \xRightarrow{\alpha'=\alpha/O(n);\ t'=t+O(2^n \log(|G|))} 1/\mathsf{poly}(k)\text{-DGDH}(n)(\text{c:}*;\ \text{g:}*;\ \text{f:o})$$

□

Before proving this theorem, let us first lower bound $\frac{\varphi(|G|)}{|G|}$, the proportion of group elements having maximal order, for group orders containing no small

prime factors.

Lemma 3.1 *Let $SG_{\mathcal{G}}$ be a group sampler generating a family $\mathcal{G}$ of groups whose orders contain no small prime factors. Let $\mathcal{G}_{SG(k)}$ be the corresponding group siblings. Furthermore, let $f : \mathbb{N} \mapsto \mathcal{G}$ be a function such that $f(k) \in \mathcal{G}_{SG(k)}$ and $\forall G' \in \mathcal{G}_{SG(k)}$ $\frac{\varphi(|G'|)}{|G'|} \geq \frac{\varphi(|f(k)|)}{|f(k)|}$, i.e., f selects for each security parameter k among the group siblings a group with maximal order. Then it follows*

$$1 - \frac{\varphi(|f(k)|)}{|f(k)|} <_\infty 1/\mathsf{poly}(k)$$

. □

Proof. Let $G := f(k)$, let $\prod_{i=1}^{m} p_i^{e_i} := |G|$ be the prime factorization of G's order, and let $p{:=}\min(p_1, \cdots, p_m)$ be the smallest prime factor of $|G|$. Then it follows that $|G| \geq p^m$ and $\log |G| \geq m \log p$ and thus $m \leq \log |G|/\log p \leq \log |G|$ for $\log p \geq 1$ (i.e., for $p \geq 2$). Moreover, as discussed in Section 3.1.7, we can assume that the group order can be upper bounded in the security parameter, i.e., $|G| \leq 2^{k^d}$ for $k > 1$ and some $d > 0$. It follows $m \leq \log |G| \leq k^d$. Hence we can write

$$\frac{\varphi(|G|)}{|G|} = \prod_{i=1}^{m}(1 - \frac{1}{p_i}) \geq (1 - \frac{1}{p})^m \geq (1 - \frac{1}{p})^{k^d}.$$

The group order $|G|$ is assumed to contain no small prime factor. It follows from the definition of the corresponding group families f:nsprim (see Section 3.2, Parameter 3) that for any real constant $c > 0$ there exists a k_0 such that for all $k > k_0$, $1/p < 1/k^c$ and thus

$$\frac{\varphi(|G|)}{|G|}) \geq (1 - \frac{1}{p})^{k^d} \geq_\infty (1 - \frac{1}{k^c})^{k^d}.$$

For $c > d$ and $k \in \mathbb{N}$ the relation $(1 - 1/k^c)^{k^d} \geq 1 - 1/k^{c-d}$ holds (see Sadeghi and Steiner (2002) for a proof of this relation). Since c is arbitrary, for all $c > d$ we have $c' := c - d > 0$ and thus for all $c' > 0$ the relations $\frac{\varphi(|G|)}{|G|} \geq_\infty 1 - 1/k^{c'}$ and $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/k^{c'}$ hold. It follows that for all $c' > 0$ there exists k_0 such that for all $k > k_0$ we have $1 - \frac{\varphi(|G|)}{|G|} < 1/k^{c'}$, i.e., $1 - \frac{\varphi(|G|)}{|G|} <_\infty 1/\mathsf{poly}(k)$. This completes the proof. ■

Equipped with this lemma, we are now ready to proceed with the proof of Theorem 3.2.

Proof. Let us address the theorem first for uniform-complexity and low-granularity. Assume there is a polynomial-time Turing machine $\mathsf{A}^{\mathrm{DGDH}(n)}$

breaking DGDH(n), i.e., $\mathsf{A}^{\mathrm{DGDH}(n)}$ distinguishes the following two distributions with not negligible success probability $\alpha_{\mathrm{DGDH}(n)}(k)$:

$$\begin{aligned} GDH^{(0)}_{k,n} &:= \{\, \{(\beta, g^{\prod_{\beta_i=1} x_i}) \mid \beta \in I_n \backslash \{1^n\}\} \cup \{(1^n, g^z)\} \\ &\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^n_{|G|}; z \leftarrow x_1 \cdots x_n\}, \\ GDH^{(1)}_{k,n} &:= \{\, \{(\beta, g^{\prod_{\beta_i=1} x_i}) \mid \beta \in I_n \backslash \{1^n\}\} \cup \{(1^n, g^z)\} \\ &\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^n_{|G|}; z \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}\}. \end{aligned}$$

The definition of the decisional problem type in Section 3.2 (see in particular Footnote 10) actually requires a slightly different and more involved random element for $GDH^{(1)}_{k,n}$: $g^{\prod z_i}$ for n random exponents $z_i \in \mathbb{Z}_{|G|}$ instead of g^z for $z \in \mathbb{Z}_{|G|}$ as mentioned here. However, above formulation is simpler to work with and makes the proof easier to understand. It is not difficult to see that the following proof can also be suitably adjusted to match the definition as required by Section 3.2. Also, note that the distributions g^z and $g^{\prod z_i}$ are statistically indistinguishable for the case where the group order has no small prime factor: In such cases, the proportion of elements in G which are relatively prime to $|G|$ is $\varphi|G|/|G|$. Therefore, for the considered group families $z_i \in_{\mathcal{R}} \mathbb{Z}_{|G|}$ is relatively prime with overwhelming probability (see Lemma 3.1.) Furthermore, g^{z_1} is almost certainly a generator and consequently $g^{z_1 z_2}$ a random element from G. Given that n is fixed it follows also that $g^{z_1 \cdots z_n}$ a random element from G with overwhelming probability.

We can prove the theorem by showing that we can construct a Turing machine $\mathsf{A}^{\mathrm{DDH}}$ with oracle access to $\mathsf{A}^{\mathrm{DGDH}(n)}$ which solves DDH, i.e., it distinguishes the following two distributions with not negligible success probability $\alpha_{\mathrm{DDH}}(k) \geq \alpha_{\mathrm{DGDH}(n)}(k)/(2(n-1)-1)$:

$$\begin{aligned} DDH^{(0)}_{k} &:= \{\, (g^{x_1}, g^{x_2}, g^z) \\ &\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (x_1, x_2) \xleftarrow{\mathcal{R}} \mathbb{Z}^2_{|G|}; z := x_1 x_2\}, \\ DDH^{(1)}_{k} &:= \{\, (g^{x_1}, g^{x_2}, g^z) \\ &\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (x_1, x_2) \xleftarrow{\mathcal{R}} \mathbb{Z}^2_{|G|}; z \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}\}. \end{aligned}$$

The proof is based on a **hybrid argument** (Goldwasser and Micali 1984; Goldreich 1998): We define a polynomial sequence of random variables, the **hybrids**, such that the extremes correspond to the views to distinguish, e.g., in our case $GDH^{(0)}_{k,n}$ and $GDH^{(1)}_{k,n}$, and each hybrid differs from its neighbors only by an instance of the distinguishing problem we like to reduce to, e.g., in our case $DDH^n_k(0)$ and $DDH^n_k(1)$. It follows that if there is an algorithm which can distinguish the two extremes with success probability $\alpha(k)$, the same algorithm also must distinguish at least one pair of neighboring hybrids with a success probability of at least $\alpha(k)/(m-1)$ where m is the number of hybrids.

The hybrid argument used in the following proof is slightly involved as it is inductive. It is based on the following observation:

Let $GDH_{k,n}(x_1, \ldots, x_n)$ be a DGDH(n) instance with secret exponents $x_1, \ldots, x_n$ and (implicitly) group G and generator g. Let $GDH^{\text{Key}}_{k,n}(x_1, \ldots, x_n)$ be the key from $GDH_{k,n}(x_1, \ldots, x_n)$, i.e., the element with label (first component) 1^n, and let $GDH^{\text{Public}}_{k,n}(x_1, \ldots, x_n)$ be the publicly known part, i.e. $GDH_{k,n}(x_1, \ldots, x_n) \setminus GDH^{\text{Key}}_{k,n}(x_1, \ldots, x_n)$. Then the following equality (ignoring a necessary but trivial adjustments of labels) holds for $2 < i \leq n$:

$$
\begin{aligned}
&GDH^{\text{Public}}_{k,i}(x_1, \ldots, x_i) = \\
&\quad GDH^{\text{Public}}_{k,i-1}(x_1, x_3, x_4, \ldots, x_i) \ \cup \ GDH^{\text{Key}}_{k,i-1}(x_1, x_3, x_4, \ldots, x_i) \ \cup \\
&\quad GDH^{\text{Public}}_{k,i-1}(x_2, x_3, x_4, \ldots, x_i) \ \cup \ GDH^{\text{Key}}_{k,i-1}(x_2, x_3, x_4, \ldots, x_i) \ \cup \\
&\quad GDH^{\text{Public}}_{k,i-1}(x_1 x_2, x_3, x_4, \ldots, x_i)
\end{aligned}
$$

This sorts the elements in three groups: the ones which may depend on x_1 but not on x_2, the ones which may depend on x_2 but not on x_1, and the ones which may depend on the product $x_1 x_2$ but not on x_1 and x_2 individually.

Using this observation, let us define the following four hybrids (again ignoring a necessary but trivial adjustments of labels):

$$
\begin{aligned}
A_n &:= GDH^{(1)}_{k,n} \\
&= \{GDH^{\text{Public}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_1x_2, x_3, \ldots, x_n) \cup (1^n, g^z) \\
&\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (z, x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^{n+1}_{|G|}\} \\
B_n &:= \{GDH^{\text{Public}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(c, x_3, \ldots, x_n) \cup (1^n, g^z) \\
&\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (c, z, x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^{n+2}_{|G|}\{ \\
C_n &:= \{GDH^{\text{Public}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(c, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(c, x_3, \ldots, x_n) \\
&\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (c, x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^{n+1}_{|G|}\} \\
D_n &:= \{GDH^{\text{Public}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_1, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_2, x_3, \ldots, x_n) \cup \\
&\quad GDH^{\text{Public}}_{k,n-1}(x_1x_2, x_3, \ldots, x_n) \cup GDH^{\text{Key}}_{k,n-1}(x_1x_2, x_3, \ldots, x_n) \\
&\quad :: G \leftarrow SG(1^k); g \leftarrow Sg(G); (x_1, \ldots, x_n) \xleftarrow{\mathcal{R}} \mathbb{Z}^{n}_{|G|}\} \\
&= GDH^{(0)}_{k,n}
\end{aligned}
$$

Note that A_n and B_n as well as C_n and D_n differ in essence only in a $DDH^{(0)}_k$ tuple $(g^{x_1}, g^{x_2}, g^{x_1x_2})$ versus a $DDH^{(1)}_k$ tuple (g^{x_1}, g^{x_2}, g^c). Finally, B_n and C_n differ only in a $GDH^{(1)}_{k,n-1}$ versus a $GDH^{(0)}_{k,n-1}$ tuple with exponents $c, x_3, \ldots, x_n$ and keys $g^{cx_3\cdots x_n}$ and g^z, respectively. Given that $GDH^{(b)}_{k,2}$ and $DDH^{(b)}_k$ are — ignoring the irrelevant syntactical differences — identical, the desired equivalence follows almost intuitively by induction on n.

Concretely, I construct A^{DDH} based on a recursive function $f(n, \text{ddh}) : \mathbb{N} \times G^3 \to 2^{(I_n \times G)}$ defined as follows:

Let an integer n and a DDH-tuple (g^{y_1}, g^{y_2}, g^c) be given as input to f. Then, $f(n, \text{ddh})$ returns one of $2(n-1)$ different hybrids where:

- all hybrids are structurally DGDH instances,
- the extremes correspond to $GDH^{(0)}_{k,n}$ and $GDH^{(1)}_{k,n}$, respectively,

- ddh is embedded with equal probability $1/(2(n-1)-1)$ in any two neighboring hybrids, and
- these neighboring hybrids differ exactly in ddh, i.e., depending whether ddh was a random or a real DDH tuple we land in one or the other hybrid.

More precisely, f performs the following steps: If $n = 2$, simply return the given DDH-tuple. Otherwise, we toss a coin and proceed as follows:

- With probability $1/(2(n-1)-1)$ we choose $x_3, \ldots, x_n$ and z randomly from $\mathbb{Z}_{|G|}$ and return the view

$$\begin{aligned} AB_n \;=\; & GDH^{\text{Public}}_{k,n-1}(y_1, x_3, \ldots, x_n) \;\cup\; GDH^{\text{Key}}_{k,n-1}(y_1, x_3, \ldots, x_n) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(y_2, x_3, \ldots, x_n) \;\cup\; GDH^{\text{Key}}_{k,n-1}(y_2, x_3, \ldots, x_n) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(c, x_3, \ldots, x_n) \;\cup\; (1^n, g^z). \end{aligned}$$

  Note that we do not need to know the exponents y_1 and y_2 to compute this view, all computations involving these values can be based on g^{y_1}, g^{y_2} and g^c. Furthermore, observe that depending on c being $y_1 y_2$ or a random value we get a view compatible with the distributions A_n and B_n, respectively.

- With probability $1/(2(n-1)-1)$ we choose $x_3, \ldots, x_n$ randomly from $\mathbb{Z}_{|G|}$ and return the view

$$\begin{aligned} CD_n \;=\; & GDH^{\text{Public}}_{k,n-1}(y_1, x_3, \ldots, x_n) \;\cup\; GDH^{\text{Key}}_{k,n-1}(y_1, x_3, \ldots, x_n) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(y_2, x_3, \ldots, x_n) \;\cup\; GDH^{\text{Key}}_{k,n-1}(y_2, x_3, \ldots, x_n) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(c, x_3, \ldots, x_n) \;\cup\; (1^n, g^{c x_3 \cdots x_n}). \end{aligned}$$

  Observe that depending on c being $y_1 y_2$ or a random value we get a view compatible with the distributions D_n and C_n, respectively.

- With probability $1 - 2/(2(n-1)-1)$ we call f recursively as $AD_{n-1}(x'_1 \ldots, x'_{n-1}) \leftarrow f(n-1, (g^{y_1}, g^{y_2}, g^c))$ to get a DGDH(n-1)-view. Then we choose randomly x_1 and x_2 from $\mathbb{Z}_{|G|}$ and return the view

$$\begin{aligned} BC_n \;=\; & GDH^{\text{Public}}_{k,n-1}(x_1, x'_2 \ldots, x'_{n-1}) \;\cup\; GDH^{\text{Key}}_{k,n-1}(x_1, x'_2 \ldots, x'_{n-1}) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(x_2, x'_2 \ldots, x'_{n-1}) \;\cup\; GDH^{\text{Key}}_{k,n-1}(x_2, x'_2 \ldots, x'_{n-1}) \;\cup \\ & GDH^{\text{Public}}_{k,n-1}(x'_1 \ldots, x'_{n-1}) \;\cup\; GDH^{\text{Key}}_{k,n-1}(x'_1 \ldots, x'_{n-1}). \end{aligned}$$

  If $AD_{n-1}(x'_1 \ldots, x'_{n-1})$ is an A_{n-1} or a D_{n-1} view, then this view is compatible with the distribution B_n or C_n, respectively.

This concludes the description of f.

The overall construction is now straightforward: A^{DDH} maps the given DDH-tuple to a DGDH(n)-tuple using $f(\cdot)$, calls $\mathsf{A}^{DGDH(n)}$ on this DGDH(n)-tuple, and returns the resulting bit. A final technicality is the fact that the correct and random DDH tuples are embedded in different "directions" in AB_n and CD_n, respectively. The interpretation of the result has to be adapted accordingly by remembering in $f(\cdot)$ whether we embedded the DDH tuple into AB_n or into CD_n, and by inverting the result from $\mathsf{A}^{DGDH(n)}$ in the former case. As the sum of distinguishing gaps between neighboring hybrids must be at least as much as the distinguishing gap between the extreme hybrids, above construction yields with the cost of a single oracle call $\mathsf{A}^{DGDH(n)}$ and $O(2^n)$ exponentiations a distinguishing success probability $\alpha_{DDH}(k) \geq \alpha_{DGDH(n)}(k)/(2(n-1)-1)$. For n constant or growing at most logarithmically in k, this results in a polynomial-time algorithm. Furthermore, the resulting success probability is not negligible as $\alpha_{DGDH(n)}(k)$ is by definition not negligible and the polynomial combination of a not negligible function with itself is again not negligible. This concludes the proof for uniform-complexity and low-granularity. Clearly, this reduction applies also to non-uniform adversaries, as uniform black-box reductions automatically yield non-uniform reductions. As nothing relies on properties of low granularity, e.g., no randomizations or assumptions on the the probability space instances, the reduction applies also to medium and high granularity. ■

Remark 3.2. The factor 2^n in the reduction cost gives a pretty bad efficiency but is unavoidable due to the size of a GDH(n) instance. However, in practice the number $\#pkey$ of partial keys visible to an adversary is small (usually, $O(n^2)$ in group key agreement protocols). By suitably ignoring partial keys which are not in the adversary's view, we can improve to a time complexity of at most $t + O(n\ \#pkey\ \log(|G|))$ with the same success probability. To achieve this we can add an additional input to the recursive function f which lists the indices of the desired partial keys. The number of exponentiations in a given recursion step corresponds to the size of this list. Furthermore, the size of the list passed to any further recursion is at most the size of the current list. As the index list has size $\#pkey$ initially and there are $n-2$ recursion steps we get a maximum number $O(n\ \#pkey)$ of exponentiations and the desired complexity. As a consequence, the theorem would hold even if n is a function polynomial in the security parameter k as long as $\#pkey$ can be bounded by a polynomial in k.

Remark 3.3. By sampling random elements from $\mathbb{Z}_{|G|}$ in the reduction we exploited that the group order is known. While the group order might not always be publicly known, there is always a publicly known upper bound $B(|G|)$ on the group order as mentioned in Section 3.2 during the discussion

of Parameter 3. If we now consider the two probability ensembles

$$X_k^* := \{g^{x^*} :: G \leftarrow SG(1^k) \ \wedge \ g \leftarrow Sg(G) \ \wedge \ x^* \xleftarrow{\mathcal{R}} \mathbb{Z}_{2^k B(|G|)}\}$$

and

$$X_k := \{g^x :: G \leftarrow SG(1^k) \ \wedge \ g \leftarrow Sg(G) \ \wedge \ x \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}\},$$

we can prove that they are statistically indistinguishable. First, observe that we compute in the exponents implicitly modulo $|G|$. Therefore, it is sufficient to consider the ensembles

$$Y_k^* := \{x^* \pmod{|G|} :: G \leftarrow SG(1^k) \ \wedge \ x^* \xleftarrow{\mathcal{R}} \mathbb{Z}_{2^k B(|G|)}\}$$

and

$$Y_k := \{x :: G \leftarrow SG(1^k) \ \wedge \ x \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}\}.$$

Investigating their statistical difference , we can derive the following inequalities:

$$\begin{aligned}
\Delta_{(Y^*,Y)}(k) \ := \ & \sum_{y \in \mathbb{Z}_{|G|}} |\mathbf{Prob}[Y_k^* = y] - \mathbf{Prob}[Y_k = y]| \\
= \ & \sum_{y \in \mathbb{Z}_{|G|}} |\mathbf{Prob}[Y_k^* = y] - \frac{1}{|G|}| \\
\leq \ & \sum_{y \in \mathbb{Z}_{|G|}} (\mathsf{max}_{y \in \mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y]) - \mathsf{min}_{y \in \mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y])) \\
= \ & |G| \ (\mathsf{max}_{y \in \mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y]) - \mathsf{min}_{y \in \mathbb{Z}_{|G|}}(\mathbf{Prob}[Y_k^* = y])) \\
= \ & |G| \ (\frac{\lceil 2^k B(|G|)/|G| \rceil}{2^k B(|G|)} - \frac{\lfloor 2^k B(|G|)/|G| \rfloor}{2^k B(|G|)}) \\
= \ & \frac{|G|}{2^k B(|G|)} \\
\leq \ & \frac{1}{2^k}
\end{aligned}$$

Clearly, from this it follows that Y and Y^* (and indirectly X and X^*) are statistically indistinguishable. The statistical indistinguishability holds also for suitably adjusted random ensembles covering exponentiations with multiple exponents $x_1, \ldots, x_n$ as statistical indistinguishability is closed under polynomial composition. Given that the behavior of the oracle machine $\mathsf{A}^{\mathrm{DGDH}(n)}$ cannot significantly differ on input distributions which are statistically indistinguishable from the correct ones — otherwise we would have a computational and, therefore, also statistical distinguisher — it is sufficient to sample random exponents from $\mathbb{Z}_{2^k B(|G|)}$ to make the reduction work

also for arbitrary group families.[23] This leads to the following more general theorem:

Theorem 3.3

$$1/\mathsf{poly}(k)\text{-DDH}(\text{c:}*;\ \text{g:}*;\ \text{f:}*) \xRightarrow{\alpha'=\alpha/O(n);\ t'=t+O(2^n(k+\log(B(|G|))))} 1/\mathsf{poly}(k)\text{-DGDH}(n)(\text{c:}*;\ \text{g:}*;\ \text{f:}*)$$

□

○

The previous relations considered only weak adversaries, i.e., relatively strong assumptions. However, we can weaken the assumptions by equating weak and strong adversaries and, therefore, by requiring only the nonexistence of oracles which have to solve virtually all problem instances.

Stadler (1996) and, independently, Naor and Reingold (1997) were the first to give a reduction from weak to strong DDH, i.e., a **self-corrector** for DDH. Their proof showed a *randomized reduction* (Blum and Micali 1984) based on the *random self-reducibility* of DDH for prime-order subgroups of $\mathbb{Z}_p^*$ with known order and high granularity. Boneh (1998) extended this result to general groups where the group order has no small prime factor and is publicly known only by an upper bound $B(|G|)$.

The following Lemma is an adaption of their work to the presented framework and medium granularity. The proof will also give a number of (necessary) details not discussed in above papers.

Lemma 3.2

$$(1-1/\mathsf{poly}(k))\text{-DDH}(\text{c:}*;\ \text{g:m};\ \text{f:nsprim}) \xRightarrow{\alpha'\geq 1-1/2^k;\ t'=(k/\alpha^2)(t+O(k+\log(B(|G|))))} 1/\mathsf{poly}(k)\text{-DDH}(\text{c:}*;\ \text{g:m};\ \text{f:nsprim})$$

□

Proof. Let a DDH instance $((G,g),(g^x,g^y,g^z))$ and an adversary A_{DDH} breaking medium granularity DDH with not negligible probability $\alpha(k)$ be given.

[23] A similar argument (but without proof) is given by Boneh (1998) for random self-reducing DDH with unknown order. He proposes to sample from $\mathbb{Z}_{B(|G|)^2}$. However, as in virtually all practical cases $B(|G|)$ is considerable larger than 2^k, this results in a much more expensive reduction. Let us consider the following (common) example: The computation is done in subgroups of $\mathbb{Z}_p^*$ with prime order q and an obvious upper bound on the group order is p. For concreteness, let us use the group parameters suggested by Lenstra and Verheul (2001) for security parameter $k = 80$, i.e., p and q having approximately 1460 and 142 bits, respectively. While my method requires exponentiation with exponents of 1540 bits, Boneh's method would require exponentiation with exponents of 2920 bits, i.e., a huge difference!

The first step is to *random self-reduce* the DDH instance: For this we choose elements a, a_1, a_2, a_3 from $\mathbb{Z}_{2^k B(|G|)}$ and compute $X \leftarrow (g^x)^{aa_1} g^{aa_2}$, $Y \leftarrow (g^y)^a g^{aa_3}$ and $Z \leftarrow (g^z)^{aa_1} (g^x)^{aa_1 a_3} (g^y)^{aa_2} g^{a_2 a_3}$. As G has no small prime factors g^a is a generator with overwhelming probability (this follows from Lemma 3.1.) If we now set $h := g^a$, $x' := a_1 x + a_2$, $y' := y + a_3$ and $z' := a_1 z + a_1 a_3 x + a_2 y + a_2 a_3$ then $X = h^{x'}$, $Y = h^{y'}$ and $Z = h^{z'}$. There are two cases to consider:

- If x, y, z is a valid DDH triple (in respect to g), i.e., $z = xy$, then X, Y, Z forms also a valid DDH triple (in respect to h) as $x'y' = z'$. Furthermore, the distribution h, X, Y, Z is statistically indistinguishable from a uniformly chosen generator in G and a corresponding random valid DDH triple due to Remark 3.3 and a_2 and a_3 acting as one-time pads.

- If x, y, z is a not a valid DDH triple (in respect to g), i.e., $z = xy + c$ for some non-zero $c \in \mathbb{Z}_{|G|}$, then Z can be written as $h^{x'y'} h^{a_1 c}$. As h^c is a generator with overwhelming probability $h^{a_1 c}$ is a one-time pad and the distribution h, X, Y, Z is statistically indistinguishable from a uniformly chosen generator in G and a corresponding random triple from G^3.

In the second step, we can use this random self-reducibility with standard *amplification* techniques to construct a machine which boosts with $O(k/\alpha(k)^2)$ oracle calls[24] the success probability to $1 - 1/2^k$:

In the first phase of the amplification, we approximate $\alpha(k)$ by some $\tilde{\alpha}$. This is achieved by repeatedly sampling two generators and a corresponding valid and invalid DDH triple, querying the oracle on both DDH instances and summing up the number of 1's returned in E_T and E_F, respectively. Let n be the number of rounds so far and $\tilde{\alpha} := |E_T - E_F|/n$. Further let p_T (p_F) be the probability of 1 returned in case of a valid (invalid) DDH triple. This loop is repeated until with overwhelming probability $\tilde{\alpha}/2 \leq \alpha(k) \leq 3\tilde{\alpha}/2$. For this we compute each round the Chernoff bound

$$\mathbf{Prob}[|\frac{\Sigma_{i=1}^n X_i}{n} - p| > \delta] < 2e^{-\frac{n\delta^2}{2p(1-p)}}$$

by setting δ to $\tilde{\alpha}/4$, $\Sigma_{i=1}^n X_i$ to E_T (E_F), and p to p_T (p_F) until $2e^{-\frac{n(\tilde{\alpha}/4)^2}{2p_T(1-p_T)}}$ is less than 2^{-k}.[25] To derive an upper bound on the number n of iterations

[24] None of Stadler (1996), Naor and Reingold (1997), or Boneh (1998) describe the details of the amplification. Boneh (1998) briefly sketches the technique and he as well as Naor and Reingold (1997) give numbers for the required oracle calls. However, while their numbers ($O(k^2/\alpha(k))$ and $O(k/\alpha(k))$, respectively) are better than the one given here, their papers lack any analysis on how they arrived at these numbers. Furthermore, it seems quite surprising that they could avoid the Chernoff bound and its δ^2 which almost certainly will result in the number of oracle calls being a function of $\alpha(k)^2$.

[25] This assumes that k is known. While this might not always be the case, we can always derive an upper bound from the inputs!

required by this phase we consider the worst case scenario in above configuration of the Chernoff bound given by $\alpha(k) = 3\tilde{\alpha}/2$, which minimizes δ, and $p = 0.5$, which maximizes $2e^{-\frac{n\delta^2}{2p(1-p)}}$. Then it holds that

$$n \leq \frac{k\ln(2) + \ln(2)}{2(\frac{\alpha(k)}{6})^2} = \frac{O(k)}{\alpha(k)^2}.$$

In the second phase of the amplification, we call the oracle n times — where n is same as the one computed above — on a random self-reduced version of the given DDH problem instance and sum up the number of 1's returned in $E_?$. If $|E_? - E_T| \leq \tilde{\alpha}/2$ we return 1, otherwise 0. It is easy to see — using the Chernoff bound — that we return the correct answer with probability at least $1 - 2^k$.

This approach does not directly lead to an algorithm which is polynomial time according to my definition in Section 3.1.3: The first phase of the amplification is guaranteed to be polynomial only for the (by definition infinitely many) k_j's where the success probability $\alpha(k)$ of the given weak adversary can be lower bounded by the inverse of some polynomial $p(\cdot)$, but not necessarily for the other k_j's. However, let us define a family of algorithms indexed by a polynomial $p_i(\cdot)$ which perform above self-correction but abort the first phase of the amplification when more than $k\ p_i(k)^2$ steps are performed. Clearly, all elements of this family have a runtime of $O(k\,p_i(k)^2(t + O(k + \log(B(|G|)))$ and, therefore, are strictly polynomial time. Furthermore, there are elements of this family, namely all elements where $p_i(\cdots)$ is asymptotically larger than the bounding polynomial $p(\cdot)$ of the adversary's success probability, which fulfill the criteria of a strong adversary. In particular, they do this for exactly the same k_j's where the criteria is fulfilled for the given weak adversary. As this holds for both uniform and non-uniform adversaries, the Lemma follows. However, note that this is only an existential argument. The algorithms are not constructive as none of the success probability $\alpha(k)$, the bounding polynomial $p(\cdot)$ or the related points k_j's are either a-priori known or can be approximated by querying the oracle in strict polynomial time! ■

Remark 3.4. The random self-reducibility holds only for group families where the group order contains no small prime factor. However, if the group order is known, we can extend the result to group families with arbitrary order and achieve slightly improved efficiency, i.e.,

Lemma 3.3

$$(1-1/\mathsf{poly}(k))\text{-DDH}(\text{c:}*;\text{g:m};\text{f:o}) \xRightarrow{\alpha' \geq 1-1/2^k;\ t'=(k/\alpha^2)(t+O(\log(|G|)))} 1/\mathsf{poly}(k)\text{-DDH}(\text{c:}*;\text{g:m};\text{f:o}).$$

□

Proof. By Lemma 3.2 this holds for group families where the group order contains no small prime factor. The improved efficiency stems from the public knowledge of the group order which allows us cheaper randomizations. For the the remaining group families, this lemma holds as for all such families the group order contains by definition at least one small prime factor. Due to this there is a trivial polynomial-time statistical test based on the order of the group elements. Therefore, no such DDH assumption can hold for these group families and the implication follows trivially. ■

Remark 3.5. As it is easy to adapt above random self-reducibility to high-granularity — just omit the randomization of the generator with a — above self-corrector works also for high granularity. Unfortunately — and opposite to what is implicitly claimed by Boneh (1998) — above self-corrector does not directly extend to low granularity as the "classical" random self-reducibility mentioned above does not apply to the low granularity case and no other approach of amplifying low-granularity oracles is known so far. ○

Combining Lemma 3.3 with Theorems 3.1 and 3.2 immediately yields the following corollary which serves as the basis of the security of the protocols presented later:

Corollary 3.1

$$(1-1/\mathsf{poly}(k))\text{-DDH}(\text{c:}*;\ \text{g:m};\ \text{f:o}) \xrightarrow{\alpha' \geq 1-1/2^k;\ t'=(O(n^2k/\alpha^2)(t+O(2^n \log(|G|)))} 1/\mathsf{poly}(k)\text{-DGDH}(n)(\text{c:}*;\ \text{g:l};\ \text{f:o})$$

□

Remark 3.6. As there is no low-granularity self-corrector (see Remark 3.5) we can rely on a strong assumption only in their medium or high granularity variant. However, note that the requirement of increased group size in medium granularity due to weak groups (Gordon 1993a) (see Section 3.4) does not apply to the protocols proposed later as the group choice is guaranteed to be random. ○

We can also combine the previous results with the following Theorem by Shoup (1997) that DDH is provably hard in the generic model:

Theorem 3.4

$$\textit{true} \implies 1/\mathsf{poly}(k)\text{-DDH}^{\sigma}(\text{c:}*;\ \text{g:h};\ \text{f:nsprim})$$

□

This trivially leads to the following corollary:

Corollary 3.2

$$\mathit{true} \implies 1/\mathsf{poly}(k)\text{-DGDH}(n)^{\sigma}(\text{c:}*;\text{g:l};\text{f:nsprim})$$

□

This raises our confidence that under a suitable choice of the algebraic group, namely that the group order does not contain any small primes, this is a good assumption to base the security of a protocol upon.

Further confidence can also be drawn from the following results: Boneh and Venkatesan (1996), Gonzalez Vasco and Shparlinski (2000), Shparlinski (2000) and Boneh and Shparlinski (2001) investigate the bit-security of DH and narrow the gap between the decisional and the computational variant; Canetti et al. (2000) show desirable statistical properties of DDH triples; and Coppersmith and Shparlinski (2000) prove the difficulty of approximating DH and DL by polynomials or algebraic functions.

3.6 Key Derivation

From an abstraction point of view, we would like that keys returned from a key-exchange are random k-bit strings rather than protocol-dependent keys of special form and properties.

Therefore, there must be a way to derive a random bit-string from a Generalized-Diffie-Hellman key. This can be achieved with the help of **(pairwise independent) universal hash functions (UHF)** (Carter and Wegman 1979). A (pairwise independent) universal hash function family *UHF* is defined as follows:

Definition 3.1 Let $\mathit{UHF} := (\{\mathsf{h}_i : \{0,1\}^{n(k)} \to \{0,1\}^{m_n(k)} | i \in \mathbb{Z}_{2^{l_n(k)}}\} \mid k \in \mathbb{N})$ be a family of function ensembles with n, m_n and l_n being functions mapping natural numbers to natural numbers. Then *UHF* is a (pairwise independent) universal hash function family if

$$\mathbf{Prob}[(\mathsf{h}_Y(x) = a) \wedge (\mathsf{h}_Y(x') = a') :: Y \xleftarrow{\mathcal{R}} \{0,1\}^{l_n(k)}] = 2^{-2m_n(k)}$$

for all $k \in \mathbb{N}$, $x \in \{0,1\}^{n(k)}$, $x' \in \{0,1\}^{n(k)} \setminus \{x\}$, and for all $a, a' \in \{0,1\}^{m_n(k)}$. ◇

To derive a bit string from a Generalized-Diffie-Hellman key we take the following two steps:

First, we construct a suitable universal hash function family $\mathit{UHF}_{G,k}$ from groups to bit strings: We take an arbitrary family of injective polynomial-time mappings[26] $(F_k : G(k) \to \{0,1\}^{n_k} \mid k \in \mathbb{N})$ with $G(k)$

[26] Such a mapping must trivially exist as we compute ultimately on bit strings and, therefore, as we have to represent group elements as bit strings.

the union of all group elements of the group siblings $\mathcal{G}_{SG(k)}$ and for some n_k. Then we compose it element-wise for each $k \in \mathbb{N}$ with an arbitrary universal hash function family[27] for which $m_n(k) = k$ and $\forall k \in \mathbb{N} \ : \ n(k) \geq n_k$ holds. As the probability statement for a universal hash function has to hold for all pairs x, x' (i.e., the probability space is only over the function indices, not the elements from the domain) this property is retained by this composition due to the injective nature of F_k.

Secondly, we choose a random element h of $UHF_{G,k}$ and apply it on the Generalized-Diffie-Hellman key to derive a k-bit string.

The security of this approach is shown in the following lemma:

Lemma 3.4

$$1/\mathsf{poly}(k)\text{-DGDH}(n)(\text{c:}*;\text{g:}*;\text{f:nsprim}) \ \wedge \ \forall G \in \mathcal{G}_{SG(k)} : |G| \geq 2^{3k}$$
$$\xRightarrow{\alpha' \geq \alpha - 2^{-k};\ t'=t}$$
$$(\mathsf{h}(g^{\prod x_i}), GDH_{k,n}^{Public}(x_1, \ldots, x_n), \overline{\mathsf{h}}) \ \overset{c}{\approx} \ (K, GDH_{k,n}^{Public}(x_1, \ldots, x_n), \overline{\mathsf{h}})$$

where $\mathsf{h} \in_{\mathcal{R}} UHF_{G,k}$, $\overline{\mathsf{h}}$ *denotes a description of the function* h, $(x_i) \in_{\mathcal{R}} \mathbb{Z}_{|G|}^n$, *and* $K \in_{\mathcal{R}} \{0,1\}^k$. □

Before proving this lemma, let me introduce a definition from information theory: The **Renyi entropy** (of order two) $R(X)$ of a random variable X on some discrete domain S is defined as $-log(\sum_{x \in S} \mathbf{Prob}[X = x]^2)$. Furthermore, we require the following lemma from Håstad, Impagliazzo, Levin, and Luby (1999):[28]

Lemma 3.5 (Entropy Smoothing Lemma) *Let* $n(k)$, $m_n(k)$, $e_n(k)$ *and* $l_n(k)$ *be functions* $\mathbb{N} \to \mathbb{N}$ *with the constraints* $m_n(k) \leq m_n(k) + 2e_n(k) \leq n(k)$. *Let* $UHF := (\{\mathsf{h}_i : \{0,1\}^{n(k)} \to \{0,1\}^{m_n(k)} | i \in \mathbb{Z}_{2^{l_n(k)}}\} \mid k \in \mathbb{N})$ *be a family of universal hash function ensembles. Furthermore, let* X *be a family of random variables indexed by* $k \in \mathbb{N}$ *and defined on domain* $\{0,1\}^{n(k)}$ *with arbitrary distribution and* $R(X_k)$ *being at least* $m_n(k) + 2e_n(k)$. *Finally, let* Y *and* Z *be two families of random variables with uniform distribution on domain* $\{0,1\}^{l_n(k)}$ *and* $\{0,1\}^{m_n(k)}$, *respectively. Then it holds that*

$$\Delta_{(<\mathsf{h}_Y(X), Y>, <Z, Y>)}(k) \leq 2^{-(e_n(k)+1)}$$

where $< X, Y >$ *denotes the concatenation of the random variables* X *and* Y. □

Based on this we can prove Lemma 3.4 as follows:
Proof. Let $GDH_{k,n} \leftarrow GDH_{k,n}^{(0)}$, $\mathsf{h} \xleftarrow{\mathcal{R}} UHF_{G,k}$, $z \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$, and $K \xleftarrow{\mathcal{R}} \{0,1\}^k$

[27] For constructions of universal hash functions which are efficient and appropriate in this context see, e.g., (Schweinberger and Shoup 2000).

[28] The lemma is slightly extended from its original formulation (Håstad et al. 1999) to cover the asymptotic environment as required in our context.

be families of random variables (implicitly) indexed by k. Furthermore, let us refer to G and g as the random variables defining the underlying structure instance implicitly induced by $GDH_{k,n}$.

Given that g^z is a uniformly distributed random element, the Renyi entropy of it is $\log(|G|)$. Hence, we can set $e_n(k) := (R(X_k) - m_n(k))/2 = k$ (note that $m_n(k) = k$ by construction of $UHF_{G,k}$ and $R(X_k) = \log(|G|) \geq 3k$ by the corresponding precondition of Lemma 3.4). Applying lemma 3.5 we derive that the statistical difference of $(\mathsf{h}(g^z), \overline{\mathsf{h}})$ and $(K, \overline{\mathsf{h}})$ is at most $2^{-(k+1)}$ and, therefore, negligible. Furthermore, given that g^z is independent of $GDH_{k,n}^{\text{Public}}$ and that statistical indistinguishability implies computational indistinguishability it also holds that

$$(GDH_{k,n}^{\text{Public}}, \mathsf{h}(g^z), \overline{\mathsf{h}}) \overset{c}{\approx} (GDH_{k,n}^{\text{Public}}, K, \overline{\mathsf{h}}).$$

Furthermore, by Theorem 3.2 and the statistical indistinguishability of $(g^z :: z \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{|G|})$ and $(g^{\prod z_i} :: (z_1, \ldots, z_n) \overset{\mathcal{R}}{\leftarrow} \mathbb{Z}_{|G|}{}^n)$ for groups with no small prime factor (see proof of Theorem 3.2) it has to hold that

$$(GDH_{k,n}^{\text{Public}}, \mathsf{h}(GDH_{k,n}^{\text{Key}}), \overline{\mathsf{h}}) \overset{c}{\approx} (GDH_{k,n}^{\text{Public}}, \mathsf{h}(g^z), \overline{\mathsf{h}})$$

and by transitivity

$$(GDH_{k,n}^{\text{Public}}, \mathsf{h}(GDH_{k,n}^{\text{Key}}), \overline{\mathsf{h}}) \overset{c}{\approx} (GDH_{k,n}^{\text{Public}}, K, \overline{\mathsf{h}}),$$

our desired result. ∎

Remark 3.7. A slightly better variant of key derivation could be based on Shoup's hedge (Shoup 2000): Compute the key as $\mathsf{h}(g^{x_1,\ldots,x_n}) \oplus \mathcal{H}(g^{x_1,\ldots,x_n})$ where $\mathcal{H}$ is a random oracle. It follows that in addition to the security in the standard model based on DGDH(n) the derived key is also secure in the random oracle model (Bellare and Rogaway 1993) based on CGDH(n).

Unfortunately, there is no known reduction from CDH to CGDH(n). The best we can do is to self-correct for medium and high granularity a weak CGDH(n) oracle to a corresponding strong oracle deploying the techniques developed to self-correct CDH (Maurer and Wolf 1996; Shoup 1997).[29] The weakest possible assumption, $(1-1/\mathsf{poly}(k))$-CGDH(n)(c:$*$; g:m; f:nsprim), is rather non-standard and is certain to hold only in the random oracle model. This model requires "magical" properties not realizable in general (Canetti, Goldreich, and Halevi 1998). Therefore, the hedge seems to provide only limited benefit when considering general group families.

However, a noticeable exception are the multiplicative groups of integers modulo a product n of two large primes p and q with $p = q = 3 \pmod 4$,

[29] Self-correcting $*$-CGDH(n)(c:$*$; g:$*$; f:$*$) is non-trivial as to amplify in the naive way would require solving $*$-DGDH(n)(c:$*$; g:$*$; f:$*$)!

i.e., p and q are Blum integers, and $p-1$ and $q-1$ contain no small prime factor. In such groups, we can reduce the (well-known and often-used) factoring problem to the computational variant of GDH(n) (Shmuely 1985; Biham, Boneh, and Reingold 1999) and both the decisional variant of DH and the factoring problem – and consequently decisional and computational GDH(n)— are assumed to be hard. Therefore, it certainly seems to be a good idea to apply above hedge when such groups are used and the factorization of the group order is guaranteed to be secret. ○

Chapter 4

CLIQUES

In this Chapter, I present CLIQUES, a complete family of protocols for key management in dynamic peer groups, namely, initial key agreement, key refresh and membership change. I analyze properties and efficiency of these protocols and give arguments for their security. The protocols assume a model with authenticated channels and are secure under the Decisional Diffie-Hellman assumption.

EQUIPPED with the requirements and desirable properties of group key agreement as well as the necessary mathematical foundations presented in the previous two chapters, we are now ready to look at concrete group key agreement protocols. In this chapter, I present **CLIQUES**, a complete family of protocols for key management in dynamic peer groups, namely, initial key agreement, key refresh and membership change, i.e., single-member and subgroup operations for joining and leaving a group.

For all protocols we assume a model where all communication channels are *authenticated* but *not private*. This means that a receiver of a message can be sure of the identity of the originator and the integrity of that message. Therefore, an adversary may not, in any way, directly interfere with it. However, an adversary still can eavesdrop on arbitrary communication between honest parties. He also can misbehave when directly involved in a protocol run. Finally, he can (potentially adaptively) corrupt honest parties to cheat disguised under their identity. The assumption that channels are authenticated is rarely realistic in practice. However, adapting[1] the *compiler* techniques from Bellare, Canetti, and Krawczyk (1998) to the PKI model presented in Chapter 2, it is possible to automatically construct

[1]The PKI model considered here is weaker (and more realistic) than the one implicitly defined in Bellare et al. (1998). This difference requires that the MT-authenticators from Bellare et al. (1998) need to include *both* involved identities and not only as done in their original form.

Figure 4.1 Notational conventions used throughout Chapter 4

n	number of protocol participants (group members)
i, j, r, m, d, c	indices of group members
M_i	i-th group member; $i \in \{1, \ldots, n\}$
M_*	all group members
G	cyclic algebraic group
$\|G\|$	order of G (must not contain small prime factors)
g	exponentiation base; generator of G
$x_i, \widehat{x}_i$	secret exponents $\in_{\mathcal{R}} \mathbb{Z}_{\|G\|}$ generated by M_i
$\prod(S)$	product of all elements in sequence S
K_n	group key shared among n members

protocols also secure in unauthenticated networks. This allows us to obtain a very modular and clean approach to the design of secure protocols.

The organization of the remainder of this chapter is as follows. In Section 4.1 I define a class of protocols that I call natural extensions of the two-party Diffie-Hellman key exchange and prove the security of all protocols in this class in a network with authenticated channels, provided the two-party *Decisional Diffie-Hellman problem* is hard. This result allows us to craft a number of efficient protocols without having to be concerned about their individual security. In particular in Section 4.2, I present two new protocols, each optimal with respect to certain aspects of protocol efficiency. Subsequently in Section 4.3, we consider a number of different scenarios of group membership changes and introduce protocols which enable addition and exclusion of group members as well as refreshing of the keys. Altogether, the protocols described below form a complete key management suite geared specifically for DPGs. However, it should be noted from the outset that related policy questions such as access control decisions are not treated: They are, due to the policy independence of the proposed protocols, orthogonal issues. In Section 4.4 I compare the work presented here with related work and conclude in Section 4.5.

4.1 Generic n-Party Diffie-Hellman Key Agreement

The **Diffie-Hellman key exchange protocol** (Diffie and Hellman 1976), depicted in Figure 4.2 (see also Figure 4.1 for some notational conventions used in this chapter), is the basis for most key agreement protocols in the two-party case. Furthermore, under the Decisional Diffie-Hellman assumption this protocol is secure in a model with authenticated channels

Figure 4.2 Two-party Diffie-Hellman key-exchange

M_1		M_2
$x_1 \xleftarrow{\mathcal{R}} \mathbb{Z}_{\lvert G\rvert}$		
	$\xrightarrow{g^{x_1}}$	
		$x_2 \xleftarrow{\mathcal{R}} \mathbb{Z}_{\lvert G\rvert}$
	$\xleftarrow{g^{x_2}}$	
$K_2 \leftarrow (g^{x_2})^{x_1}$		$K_2 \leftarrow (g^{x_1})^{x_2}$

(Bellare et al. 1998).[2]

All key agreement protocols presented later belong to a large class of protocols which can be seen as natural extensions of two-party Diffie-Hellman key exchange to the n-party case.

Scheme 4.1 (Class of natural n-party extensions of the Diffie-Hellman key exchange)

Let there be n participating group members[3] $M_1, \ldots, M_n$. As in the two-party case, all participants agree a priori on a cyclic group G. Let g be a generator of G.

For each key exchange, each member, M_i, chooses randomly a value $x_i \in \mathbb{Z}_{|G|}$. The group key will be $K_n = g^{x_1 \cdots x_n}$. In the two-party case, K_2 is computed by exchanging g^{x_1} and g^{x_2}, and computing $K_2 = (g^{x_1})^{x_2} = (g^{x_2})^{x_1}$. To solve the n-party case, a certain subset of the partial GDH keys $(g^{\prod_{\beta_i=1} x_i} \mid \beta \in I_n \backslash \{1^n\})$ is exchanged between the participants (and, therefore, exposed to the adversary). This set has to include for all i the value $g^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_n}$. If M_i receives that value, it can easily compute K_n as $(g^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_n})^{x_i}$.

Furthermore, we require that all protocols of this class have following properties:

1. All communication is over *authenticated channels*.
2. The *system parameters* are generated by a *trusted party* Gen using some *generation algorithm* genG. In particular, genG determines an

[2] More precisely, the protocol is secure against static adversaries. Further precaution has to be taken to be secure against (strong) adaptive attacks (Shoup 1999).

[3] Note that the notation M_i and the corresponding index i of protocol participants are not "real" identifiers. They are only aliases which give some ordering among the protocol participants and can be used to synchronize and coordinate the protocol. The ordering is arbitrary and specific to a single protocol run only. In particular, it does not presuppose any fixed and static ordering among protocol participants.

(algebraic) group G based on a (trusted) group sampler $SG_{\mathcal{G}}$ for a group family $\mathcal{G}$ where no group G has a group order $|G|$ containing any small prime factors. Furthermore, genG also fixes a generator g using a generator sampler Sg. Finally, the trusted party Gen distributes the system parameters, including the group order $|G|$ and its factorization, reliably to all potential group members.

3. The protocol ensures that no flow ever contains $g^{x_1 \cdots x_n}$, the group key K_n, or values derived from it. Furthermore, each member M_i keeps the secret exponent x_i securely and uses it solely to compute, as necessary, partial GDH keys and the group key.

4. The protocol ensures that each message contains *identifiers* indicating the particular *group*, *session*, and corresponding *group membership view* of the sender. Furthermore, messages have to be *typed*, e.g., to uniquely determine their exact position in the protocol. For AKA operations (see Section 4.3), we additionally require an identifier to the particular *AKA epoch*.

5. All participants verify that received messages have the *proper format* and contain valid[4] elements of G of *maximal order*, i.e., generators, and reject any other message.

◇

As we see in the following theorem, all protocols in this class have the same security properties.

Theorem 4.1 *All protocols in the class of natural n-party extensions of the Diffie-Hellman key exchange are secure authenticated key-agreements protocols assuming that the assumption* $(1-1/\mathsf{poly}(k))$-DDH(c:$*$; g:m; f:fct,nsprim) *holds. In particular, the protocols are contributory and ensure semantic security and freshness of the group key. They also provide implicit and mutual group key authentication. Furthermore, the protocols provide PFS and are resistant to KKA.* □

Proof (sketch).[5] It is clear that the security of protocols in this class is closely related to the Generalized Diffie-Hellman problem which we investigated in Section 3.5. More precisely, let us look at the different properties mentioned in the theorem in turn:

Due to property 3, the protocol does not leak information on the group key other than some related partial GDH keys. Therefore, the *key secrecy* depends solely on a suitable variant of the Generalized Diffie-Hellman

[4] We assume that, in addition to the group operations mentioned in Section 3.1.5, group membership tests can be performed efficiently.

[5] Note that the following security argumentation matches past practice in proving security for group key protocols. However, to better contrast it to the formal proof in Chapter 5 I call the argumentation here only a proof sketch.

assumption. Property 2 allows us to rely on a low-granularity assumption. Therefore, the validity of the low-granular DGDH assumption with weak success probability, i.e., $1/\mathsf{poly}(k)$-DGDH(n)(c:$*$; g:l; f:fct,nsprim), is sufficient to guarantee key secrecy.[6] Note that by relying on a decisional assumption we ensure the semantic security of the session key.[7] Corollary 3.2 tells us that this assumption is true in the generic model. The assumption is not necessarily true in the specific (non-generic) model. However, we can weaken this assumption further: Taking Lemma 3.1, we require only that the medium-granular, strong DDH assumption, i.e., $(1-1/\mathsf{poly}(k))$-DDH(c:$*$; g:m; f:fct,nsprim), holds.

Due to Properties 1 and 4, all (honest) participants who successfully terminate the protocol for a given session will agree on a common session key and a common group membership view. Together with the key secrecy discussed above, we achieve *implicit mutual group key authentication*.

The protocols are *contributory key-agreement protocols* due to Properties 4 and 5, and the difficulty of taking discrete logarithms which is implied by above DDH assumption. The difficulty of taking discrete logarithms is required to make it hard to recover the secret contribution of individual members and to ensure that the protocol is contributory. Property 5 is required to counter attacks, e.g., a small subgroups attack (Lim and Lee 1997). Such an attack could violate *key freshness* in the presence of dishonest insiders, a property required by a key-agreement protocol, as the key could become predictable and would not be fresh. Note that the restriction that transmitted group elements must have maximal order is of no harm even as protocol participants choose their exponents uniformly from $\mathbb{Z}_G$. The probability that a random element of G does *not* have maximal order is $1-\varphi(|G|)/|G|$ and is negligible for the groups considered here (see Lemma 3.1.) Therefore, no protocol failures should happen in practice due to an honest participant choosing "bad" exponents such that some transmitted partial GDH keys do not have maximal order.

A priori we do not have any long-term keys. However, implementing authenticated channels with the compiler techniques mentioned above involves long-term keys. Reasonably assuming that the choice of the long-term keys is independent from the choice of the secret exponents, an adversary gaining access to these long-term keys cannot learn anything new about past messages; we already assume the content of these messages to be known. Furthermore, sessions terminated before the exposure of the long-term key

[6] This makes it clear why I required the restriction on the group family $\mathcal{G}$: No DGDH assumption can hold if there are groups for which the group order contains some small prime factors.

[7] Using a key derivation based on cryptographic hash functions we could resort to a (weaker) computational assumption at the cost of having to resort to the (very strong) random oracle model. See Remark 3.7 for an approach which combines the benefits of the standard and the random oracle model.

were protected by the compiler which prevented active attacks on them. Therefore, an adversary can attack past sessions only passively. As the adversary does not learn anything new, above argumentation regarding the key secrecy still holds. Furthermore, all other security properties mentioned in the theorem are inherently immune to passive attacks. This means that the protocols also guarantees *PFS*.

Finally, if we consider different protocol runs, it is clear that, due to the contributory nature of the protocols and the random choice of secret contributions by honest participants, the keys of different sessions are independent. Therefore, the loss of a key of one session does not endanger any other sessions and we get *security against KKA*. ■

Remark 4.1. We also could allow an arbitrary group member to generate the system parameters. However, since initially none of them could be trusted, we would have to resort to a high-granularity assumption and all group members would have to verify that the system parameters are in fact members of the desired group family. See Section 3.4 for more information on this issue and other alternatives, e.g., a joint generation.

Remark 4.2. Note that the protocols in above class yield only implicit group key authentication since not all group members will necessarily be convinced about the active presence of all other group members.

However, if we extend above protocols such that after the successful establishment of the key each member notifies all other members about this fact, we can achieve *explicit group key authentication*. If everybody contacts everybody, we get in addition *complete group key agreement*. However, if this is not required, the notification can be indirect and is probably best performed in two round: first, everybody sends a message to a dedicated member; second and after the receipt of all these messages, the dedicated member broadcasts this event to all group members. Of course, the communication has to be over authenticated channels (Property 1). However, besides the identification information required to fulfill Property 4, the notification messages can be "empty".

Such a strategy implicitly also provides *key confirmation* even though the notification messages are not directly linked to the key! Key confirmation is usually defined only vaguely and informally, e.g., as "evidence that a key is possessed by some party" (Menezes et al. 1997). The only reasonably formal characterization is a *proof of knowledge* (Feige et al. 1987; Tompa and Woll 1987; Bellare and Goldreich 1993). In a key confirmation protocol, the prover is implicitly trusted by the verifier. If the authenticity of messages from the prover is guaranteed, e.g., as in our case due to the authenticated channels, a simpler variant, namely a **proof of knowledge with honest provers**, is sufficient. This basically means that the *knowledge extractor*, who has access to the prover's machine, will know the

"program" of the prover and understand its internal state. In our case, we can trivially construct a knowledge extractor from the notification protocol. The receipt of a notification message of some particular group member shows that this party, the prover, has computed the group key and we can just read it from the provers working tape. Note that in the light of the discussion on the semantic security of session keys, we require a zero-knowledge (Goldwasser et al. 1989) variant of a proof of knowledge. Nonetheless, this is clearly fulfilled in our case as no information on the key is contained in the notification flows. (By contrast, many "classical" key confirmation protocols often violate the semantic security of session keys by the inappropriate use of these keys in the protocol flows!)

Remark 4.3. For proper group key authentication and session association, the proof relies crucially on Property 4, the inclusion of identifiers for the group, session, and corresponding group membership view in each message. In a naive implementation these identifiers would grow linearly in the number of sessions and the size of the group, and might become rather big. However, by using collision-resistant hash functions we can securely *compress identifiers* and reduce this overhead to be essentially constant (the growth of the hash-function's output length required by an increasing security parameter should be irrelevant in practice). ○

Hereafter, the above result allows us to construct a number of specific protocols belonging to the class of natural n-party extension of DH without worrying too much about their individual security. For clarity, I omit in the remainder of this chapter identifiers contained in messages and the tests performed by receivers as required by Properties 4 and 5. However, when implementing the protocols in practice, they are clearly necessary and also correspondingly made explicit later in the formal treatment in Chapter 5. Furthermore, I assume that the system setup is performed consistent with Property 2 and communication is over authenticated channels (Property 1). The remaining Property 3 (no leakage on the group key other than the partial GDH keys) should be obvious in the following protocols.

4.2 CLIQUES: Initial Key Agreement

The cornerstone of the CLIQUES protocol suite is formed by two IKA protocols called IKA.1 and IKA.2. (They were referred to as GDH.2 and GDH.3, respectively, in Steiner, Tsudik, and Waidner (1996).)

4.2.1 IKA.1

The first IKA protocol (IKA.1) is depicted in Figure 4.3 and illustrated in Figure 4.4 by an example protocol run for a group with four members. It consists of an upflow and a downflow stage.

Figure 4.3 Group Key Agreement: IKA.1

M_i $\qquad\qquad\qquad\qquad$ M_{i+1}

$$\xrightarrow{(g^{\prod(x_m|m\in\{1,\ldots,i\}\setminus\{j\})}|\ j\in\{1,\ldots,i\}),\ g^{\prod(x_m|m\in\{1,\ldots,i\})}}$$

Stage 1 (Upflow): round i; $i \in \{1, \ldots, n-1\}$

M_* $\qquad\qquad\qquad\qquad$ M_n

$$\xleftarrow{(g^{\prod(x_m|m\in\{1,\ldots,n\}\setminus\{j\})}|\ j\in\{1,\ldots,n\})}$$

Stage 2 (Broadcast): round n

Figure 4.4 Example of IKA.1. (The dotted line denotes a broadcast. The g in the first message could be omitted, but allows a more unified description.)

M_1 $\qquad$ M_2 $\qquad$ M_3 $\qquad$ M_4

$x_1 \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xrightarrow{g,\ g^{x_1}}$$

$x_2 \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xrightarrow{g^{x_2},\ g^{x_1},\ g^{x_1x_2}}$$

$x_3 \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xrightarrow{g^{x_2x_3},\ g^{x_1x_3},\ g^{x_1x_2},\ g^{x_1x_2x_3}}$$

$x_4 \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xleftarrow{g^{x_2x_3x_4},\ g^{x_1x_3x_4},\ g^{x_1x_2x_4},\ g^{x_1x_2x_3}}$$

$K_4 \leftarrow (g^{x_2x_3x_4})^{x_1}$ $\qquad$ $K_4 \leftarrow (g^{x_1x_3x_4})^{x_2}$ $\qquad$ $K_4 \leftarrow (g^{x_1x_2x_4})^{x_3}$ $\qquad$ $K_4 \leftarrow (g^{x_1x_2x_3})^{x_4}$

The purpose of the upflow stage is to collect contributions from all group members, one per round. In round i ($i \in \{1, \ldots, n-1\}$), M_i unicasts M_{i+1} a collection of $i+1$ values. Of these, i are intermediate and one is *cardinal*. The cardinal value CRD_i is simply the generator raised to all secret exponents generated so far:

$$\mathrm{CRD}_i := g^{\prod(x_m | m \in \{1,\ldots,i\})}$$

Let $\mathrm{INT}_{i,j}$ denote the j-th intermediate value in round i. It is always of the following form (i.e., CRD_i with the j-th exponent missing):

$$\mathrm{INT}_{i,j} := g^{\prod(x_m | m \in \{1,\ldots,i\} \setminus \{j\})} \qquad \text{for } j \in \{1, \ldots, i\}$$

M_i's computations upon the receipt of the upflow message can now be described as follows:

1. generate private exponent $x_i \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$
2. set $\mathrm{INT}_{i,j} = (\mathrm{INT}_{i-1,j})^{x_i}$ for all $j \in \{1, \ldots, i-1\}$
3. set $\mathrm{INT}_{i,i} = \mathrm{CRD}_{i-1}$
4. set $\mathrm{CRD}_i = (\mathrm{CRD}_{i-1})^{x_i}$

In total, M_i composes i intermediate values (each with $(i-1)$ exponents) and a cardinal value containing i exponents.

In round $(n-1)$, when the upflow reaches M_n, the cardinal value becomes $g^{x_1 \cdots x_{n-1}}$. M_n is thus the first group member to compute the key K_n. Also, as the final part of the upflow stage, M_n computes the last batch of intermediate values. In the second stage M_n broadcasts the intermediate values to all group members.

The highest-indexed group member M_n plays a special role by having to broadcast the last round of intermediate values. However, this special role *does not* afford M_n any added rights or privileges. The reason IKA.1 broadcasts the last flow, instead of unicasting $n-1$ shares individually (potentially saving some bandwidth), will become apparent later in Section 4.3 when I discuss AKA operations: This allows us to achieve *policy independence* on group controllership. Furthermore and depending on the underlying group communication system (see Section 2.2.3), a broadcast can give us natural synchronization and causal ordering of the termination of the protocol.

We now consider the performance characteristics of IKA.1 based on the measures discussed in Section 2.4. The computation of exponentiations in G is by far the dominant computational cost factor. Therefore, we can take into account the number of required exponentiations as the only measure for the *computational cost*. As the size of the messages depends on the particular choice of the algebraic group G, its encoding and the additional overhead of group and session identifiers, I will not be able to give concrete *communication costs*. The overhead can always be kept constant (see Remark 4.3)

in the number of group members and the elements of G, i.e., the partial GDH keys, are the only non-linear size aspect of a message. Therefore, I will just count the number of transmitted elements of G to measure the cumulative message size. From this, you can then easily derive the concrete bandwidth requirement once the concrete parameters, such as the group G and its encoding, are known. This results in the following measures for IKA.1:

rounds	n
unicast messages	$n-1$
broadcast messages	1
cumulative message size	$\frac{n(n+3)}{2}-1$
exponentiations per M_i	$(i+1)$ for $i<n$, n for $i=n$
exponentiations on critical path	$\frac{n(n+1)}{2}$

Some remarks on these characteristics:

The number of required messages, n, is optimal in a network model which provides broadcasts. IKA.1 is also optimal in that respect in a network model which does not provide broadcasts (and in which case we can implement the broadcast as $n-1$ unicasts and we require a total $2(n-1)$ unicast messages.) For the proof of these properties I refer you to Becker and Wille (1998) who systematically analyze the communication complexity of contributory group key agreement protocols.

The computational cost of the critical path is quadratic in the number of participants. This is certainly a potential problem for the scalability of IKA.1 to large groups. However, I argued that DPGs are relatively small so the negative effect should be limited. Furthermore, in the case where this cost dominates the overall duration of the protocol, e.g., delays due to networking are much smaller, and becomes problematic, we can apply the following optimization: Instead of having each group member perform all exponentiations and accumulate the corresponding results before sending the complete message, we can interleave the computation and the communication in a *pipeline* fashion, i.e., forward the individual partial GDH keys of a message as soon as they are computed. This will optimize the critical path and cut down the cost to $2n-1$ exponentiations, i.e., linear cost! Of course, pipelining increases the number of messages and corresponding communication costs and, potentially, this outweighs that gain. The optimal strategy might be to pipeline with coarser granularity, i.e., several partial GDH keys per message instead of one only, and to choose the granularity according to the ratio of computation and communication costs.

4.2.2 IKA.2

In certain environments, it is crucial to minimize the amount of computation performed by each group member. This is particularly the case in large

Figure 4.5 Group Key Agreement: IKA.2

M_i $\qquad\qquad\qquad\qquad$ M_{i+1}

$$\xrightarrow{\quad g^{\prod(x_m \mid m \in \{1,\ldots,i\})} \quad}$$

Stage 1 (Upflow): Round i; $i \in \{1, \ldots, n-2\}$

M_* $\qquad\qquad\qquad\qquad$ M_{n-1}

$$\xleftarrow{\quad g^{\prod(x_m \mid m \in \{1,\ldots,n-1\})} \quad}$$

Stage 2 (Broadcast): Round $n-1$

M_i $\qquad\qquad\qquad\qquad$ M_n

$$\xrightarrow{\quad g^{\frac{\prod(x_m \mid m \in \{1,\ldots,n-1\})}{x_i}} \quad}$$

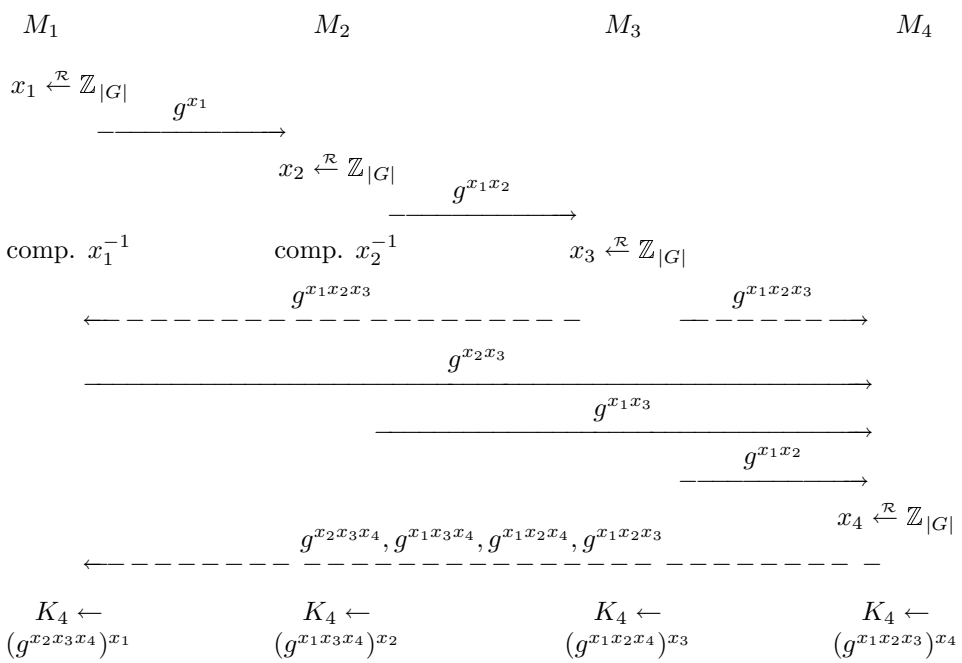
Stage 3 (Response): Round n

M_* $\qquad\qquad\qquad\qquad$ M_n

$$\xleftarrow{\quad (g^{\frac{\prod(x_m \mid m \in \{1,\ldots,n\})}{x_j}} \mid j \in \{1, \ldots, n\}) \quad}$$

Stage 4 (Broadcast): Round $n+1$

Figure 4.6 Example of IKA.2



groups or groups involving low-power entities such as smart cards or PDAs. Since IKA.1 requires a total of $(i+1)$ exponentiations of every M_i, the computational burden increases as the group size grows. The same is true for message sizes.

In order to address these concerns, I present a very different protocol, IKA.2 (see Figure 4.5). IKA.2 consists of four stages. In the first stage IKA.2 collects contributions from all group members similar to the upflow stage in IKA.1. After processing the upflow message M_{n-1} obtains $g^{\prod(x_m|m\in\{1,\ldots,n-1\})}$ and broadcasts this value in the second stage to all other participants. At this time, every M_i $(i \neq n)$ factors out its own exponent and forwards the result to M_n. In the final stage, M_n collects all inputs from the previous stage, raises every one of them to the power of x_n and broadcasts the resulting $n-1$ values to the rest of the group. Every M_i now has a value of the form $g^{\prod(x_m|m\in\{1,\ldots,n\}\setminus\{i\})}$ and can easily generate the intended group key K_n. IKA.2 is illustrated in Figure 4.6 by an example protocol run for a group with four members.

Note that factoring out x_i requires computing its inverse — $x_i^{-1} \pmod{|G|}$. This is always possible if the group order is known and we choose the group G as a group of prime order. In the groups mentioned above, namely groups where the group order does not contain any small prime factor, not all elements of $\mathbb{Z}_{|G|}$ do have an inverse. However, the

probability to pick such a non-invertible element is negligible (this follows from Lemma 3.1) and, therefore, not a problem.

The performance characteristics of IKA.2 are summarized in the following table:

rounds	$n+1$
unicast messages	$2n-3$
broadcast messages	2
cumulative message size	$3n-2$
exponentiations[8] per M_i	4 for $i \in \{1,\ldots,n-2\}$, 2 for $i=(n-1)$, n for $i=n$
exponentiations on critical path	$2n+1$

IKA.2 has two appealing features:

- Constant message sizes and close to optimal cumulative message size minimize the network bandwidth requirements. A lower bound on the cumulative message size is $2(n-1)$ as can easily be seen from a similar argumentation as used in achieving the lower bounds on the number of messages in (Becker and Wille 1998). No other contributory group key agreement protocol is known yet to reach that lower bound or even improve over IKA.2.

- Constant (and small) number of exponentiations for each M_i (except for M_n with n exponentiations required) limit computation requirements. The total number of exponentiations $(5n-6)$ is only a constant factor away from being optimal; clearly, there have to be at least $2n$ exponentiations.[9]

One notable drawback of IKA.2 is that, in Stage 3 (n-th round), $n-1$ unicast messages are sent to M_n. This might lead to congestion at M_n.

4.3 CLIQUES: Auxiliary Key Agreement

Both IKA protocols operate in two phases: a gathering phase whereby M_n collects contributions from all participants to compute $(g^{\frac{x_1 \cdots x_n}{x_i}} | i \in$

[8]The computation of x_i^{-1} (in $\mathbb{Z}^*_{|G|}$) in Stage 2 is counted as an exponentiation (in G). The costs are not necessarily identical but the cost of the latter is certainly an upper bound to the cost of the former. Furthermore, note that the computation of the inverse is not on the critical path as it can already be done in parallel to stage 1 and 2. However, factoring out the exponent needs, besides the computation of the inverse, an additional exponentiation which *is* on the critical path.

[9]In a contributory agreement protocol, each participant has to contribute the own secret key share — in our case, using an exponentiation — at least once to provide the required input for the key computation of other parties and a second time to derive the actual key.

$\{1, \ldots, n\}$) and a final broadcast phase. The following AKA operations take advantage of the keying information (i.e., partial keys) collected in the gathering phase of the most recent IKA protocol run. This information is incrementally updated and re-distributed to the new incarnation of the group. In particular, any member who caches the most recent message of the final broadcast round can initiate an AKA operation. Any member can take over the role of group controller at no cost and whenever the situation requires it, e.g., when the former group controller abruptly disappears due to a crash or network partition. This way, these protocols achieve complete *policy independence*.

Since the final broadcast phase is exactly the same for both IKA.1 and IKA.2 we note that the AKA operations described below work with both IKA protocols. This results in the flexibility to choose an IKA protocol that suits a particular DPG setting.

In the following, we look first at the concrete protocols for the different AKA operations. Afterwards in Section 4.3.7, we will investigate the security of these protocols.

4.3.1 Member Addition

The member addition protocol is shown in Figure 4.7 and illustrated by an example in Figure 4.8. As mentioned above I assumed that the current group controller M_c ($c \in \{1, \ldots, n\}$) remembers the contents of the broadcast message that was sent in the last round in the IKA protocol of Figure 4.3.[10]

In effect, M_c extends Stage 1 of IKA.1 by one round: it generates a new and random exponent $\widehat{x_c}$ and creates a new upflow message. $\widehat{x_c}x_c$ is used in place of x_c to prevent the new member and outsiders from learning the old group key. The broadcast in the second round is then identical in its structure[10] to the final broadcast flow in the IKA protocols and allows all group members to compute the new group key:

$$K_{new} = g^{\widehat{x_c} \prod(x_m | m \in \{1,\ldots,n+1\})}.$$

Additionally, M_c replaces x_c by $\widehat{x_c}x_c \bmod |G|$ as its own contribution for further AKA operations. This is the reason for not blinding partial GDH keys which do not contain M_c's old contribution x_c, i.e., $g^{\prod(x_m | m \in \{1,\ldots,n\} \setminus \{c\})}$. While blinding all partial GDH keys would have simplified the protocol description in Figure 4.7, we save with the current protocol one exponentiation and protect past session keys even when (the new) x_c would be lost later.

The performance characteristics of the member addition protocol are summarized in the following table:

[10] This is only the case for the very first member addition; subsequent member additions as well as other AKA operations require the current controller to save the most recent broadcast message from the AKA operation of the preceding epoch.

Figure 4.7 Member Addition (The new member is M_{n+1})

M_c $\qquad\qquad\qquad\qquad$ M_{n+1}

$$\xrightarrow{\begin{array}{c}(g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n\}\setminus\{j\})}\mid j\in\{1,\ldots,n\}\setminus\{c\}),\\ g^{\prod(x_m|m\in\{1,\ldots,n\}\setminus\{c\})},\ g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n\})}\end{array}}$$

Upflow: round 1 ($x_c \leftarrow x_c\widehat{x_c}$)

M_* $\qquad\qquad\qquad\qquad$ M_{n+1}

$$\xleftarrow{\begin{array}{c}(g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n+1\}\setminus\{j\})}\mid j\in\{1,\ldots,n+1\}\setminus\{c\}),\\ g^{\prod(x_m|m\in\{1,\ldots,n+1\}\setminus\{c\})}\end{array}}$$

Broadcast: round 2

To prevent too much clutter in the presentation of this figure, I list in the flows the partial GDH keys containing $\widehat{x_c}x_c$ before the corresponding (single) partial GDH key which does not contain this exponent. However, I assume that here as well as for the AKA protocols presented later the partial GDH keys are sorted according to the same order relation on members as in the corresponding IKA flows.

Figure 4.8 Example of member addition. M_2 is the current group controller, K_{old} is $g^{x_1x_2x_3}$ and M_4 is the new member.

M_1 $\qquad$ M_2 $\qquad$ M_3 $\qquad$ M_4

$\widehat{x_2} \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xrightarrow{g^{\widehat{x_2}x_2x_3},\, g^{x_1x_3},\, g^{x_1\widehat{x_2}x_2},\, g^{x_1\widehat{x_2}x_2x_3}}$$

$x_4 \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$

$$\xleftarrow{g^{\widehat{x_2}x_2x_3x_4},\, g^{x_1x_3x_4},\, g^{x_1\widehat{x_2}x_2x_4},\, g^{x_1\widehat{x_2}x_2x_3}}$$

M_1	M_2	M_3	M_4
$K_{new} \leftarrow (g^{\widehat{x_2}x_2x_3x_4})^{x_1}$	$K_{new} \leftarrow (g^{x_1x_3x_4})^{\widehat{x_2}x_2}$ $x_2 \leftarrow \widehat{x_2}x_2$	$K_{new} \leftarrow (g^{x_1\widehat{x_2}x_2x_4})^{x_3}$	$K_{new} \leftarrow (g^{x_1\widehat{x_2}x_2x_3})^{x_4}$

rounds	2
unicast messages	1
broadcast messages	1
cumulative message size	$2(n+1)$
exponentiations per M_i	1 for $i \in \{1, \ldots, n\} \setminus \{c\}$, $n+1$ for $i \in \{c, n+1\}$
exponentiations on critical path	$2n+1$

The number of rounds and messages are clearly optimal. The total number of exponentiations $(3n+1)$ is close to optimal for protocols from the class of n-party extensions of DH: $2(n+1)$ exponentiations are inevitable as the new member has to contribute his share, and all members have to compute the new key.

Note that the computational cost of the critical path can be reduced to $n+1$ if M_c precomputes[11] his message in anticipation of a membership addition or other AKA operations (as will become clear later, the group controller always performs the same computation as the first step in all the AKA operations.)

4.3.2 Mass Join

Distinct from both member and group addition is the issue of *mass* join. Mass join is necessary in cases when multiple new members need to be brought into an existing group.

It is, of course, always possible to add multiple members by consecutive runs of a single-member addition protocol. However, this would be inefficient since, for each new member, every existing member would have to compute a new group key only to *throw it away* thereafter. To be more specific, if n' new members were to be added in this fashion, the cost would be:

- $2n'$ rounds.
- Included in the above are n' rounds of broadcast.
- n' exponentiations by every "old" group member.

The overhead is clearly very high.

A better approach is to *chain* the member addition protocol as shown in Figure 4.9. The idea is to capitalize on the fact that multiple, but disparate, new members need to join the group and chain a sequence of upflow messages to traverse all new members in a certain order. This allows us to incur only one broadcast round and postpone it until the very last step, i.e., the last new member being *mass-joined* performs the broadcast. The savings, compared

[11] Pipelining could reduce the costs here only to $n+2$ exponentiations and would not bring any gain in addition to precomputation. Thus, pipelining would not merit its additional cost and complication.

Figure 4.9 Mass Join (The new members are M_{n+1} to $M_{n+n'}$)

M_{n+i} $\qquad\qquad\qquad\qquad$ M_{n+i+1}

$$(g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n+i-1\}\setminus\{j\})} \mid j \in \{1,\ldots,n+i-1\}\setminus\{c\}),$$
$$g^{\prod(x_m|m\in\{1,\ldots,n+i-1\}\setminus\{c\})},\ g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n+i-1\})}$$

$$\longrightarrow$$

Upflow: round i $(1 \le i \le n')$
(if $(i=1)$ { $M_{n+i-1} := M_c$; $x_c \leftarrow x_c\widehat{x_c}$ })

M_* $\qquad\qquad\qquad\qquad$ $M_{n+n'}$

$$(g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n+n'\}\setminus\{j\})} \mid j \in \{1,\ldots,n+n'\}\setminus\{c\}),$$
$$g^{\prod(x_m|m\in\{1,\ldots,n+n'\}\setminus\{c\})}$$

$$\longleftarrow$$

Broadcast: round $n'+1$

with the naive approach, amount to $n'-1$ broadcast rounds. The cost of adding n' new members is summarized as follows:

rounds	$n'+1$
unicast messages	n'
broadcast messages	1
cumulative message size	$({n'}^2+2nn'+3n'+2n)/2$
exponentiations per M_i	1 for $i \in \{1,\ldots,n\}\setminus\{c\}$,
	$(i+1)$ for $i \in \{n+1,\ldots n+n'\}$
	$(n+2)$ for $i=c$
exponentiations on critical path	$({n'}^2+2nn'+n'+2n)/2$

4.3.3 Group Fusion

Group fusion, as defined in Section 2.3.2, occurs whenever two groups merge to form a super-group. The only real difference with respect to mass join is that group fusion assumes preexisting relationships within both groups. Thus, if we ignore the preexisting relationships we can treat group fusion as either:

(1) Special case of mass join as in Figure 4.9, or

(2) Creation of a new super-group via a fresh IKA, e.g., IKA.1 (Figure 4.3) or IKA.2 (Figure 4.5).

Unfortunately, in both cases the resulting protocols are quite costly, in particular, in their round complexity. The obvious question is whether we could exploit the preexisting relationships within the two (sub-)groups, i.e., the two sets of partial GDH keys already distributed among the members of these groups, to gain efficiency. However, there does not seem to be any way to reasonably combine partial GDH keys corresponding to two different groups without leaving the class of natural n-party extension of DH (and losing the security properties shown in the Theorems 4.1 and 4.2.) Therefore, we can exploit the existing relationships within *at most one* (as done in Case (1) above) but not of *both* groups simultaneously. This leaves us with above two solutions and the decision whether to use (1) or (2) would be heuristic- or policy-driven on a case-by-case basis.

Tree-Based Group Fusion

Leaving the class of natural n-party extension of DH, more efficient, or at least more elegant, solutions geared specifically towards group fusion are possible. I briefly sketch one possible approach to group fusion below.

The idea is to use a technique fashioned after the one developed by Becker and Wille (1998) for initial key agreement. In brief, suppose that two groups $\mathcal{M}_1$ and $\mathcal{M}_2$ currently using group keys K_1 and K_2, respectively, would like to form a super-group. To do so, the two groups exchange their respective key residues: g^{K_1} and g^{K_2} and compute a new super-group key $K_{12} = g^{K_1 K_2}$. The actual exchange can be undertaken by the group controllers. Note that this type of fusion is very fast since it can in principle be accomplished in one round of broadcast.

However, there is one glaring problem with above protocol: It does not provide semantic security for old keys as g^{K_1} and g^{K_2} are public. One probably can solve this problem by using $h(K_n)$, instead of K_n, as session key with h being a random oracle (Bellare and Rogaway 1993). However, in this case we are leaving the standard model and we can achieve only weaker security results.

Furthermore, if we consider subsequent AKA operations it becomes clear that combining natural n-party extension of DH such as CLIQUES with this tree-based approach does not match well. Reverting to the original group structure is easy since each group can simply fall back to using K_1 and K_2 at any time thus effectively reversing the fusion. However, any other group split seems to require two complete and inefficient IKA operations and confirms the decision to use above mentioned approaches.

Figure 4.10 Member Exclusion (The excluded member is M_d)

M_c $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ M_*

$$\xrightarrow{\begin{array}{c}(g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n\}\setminus\{j\})} \mid j\in\{1,\ldots,n\}\setminus\{c,d\}),\\ g^{\prod(x_m|m\in\{1,\ldots,n\}\setminus\{c\})}\end{array}}$$

Broadcast: round 1 ($x_c \leftarrow x_c\widehat{x_c}$)

4.3.4 Member Exclusion

The member exclusion protocol is illustrated in Figure 4.10. In it, M_c effectively "re-runs" the last round of the IKA: As in member addition, it generates a new exponent $\widehat{x_c}$ and constructs a new broadcast message — with $\widehat{x_c}x_c$ instead of x_c — using the most recently received broadcast message. (Note that the last broadcast message can be from an IKA or any AKA, depending which was the latest to take place.) M_c then broadcasts the message to the remaining members of the group. The private exponents of the other group members remain unchanged.

Let M_d be the member to be excluded from the group. We assume, for the moment, that $d \neq c$. Since the following sub-key:

$$g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n\}\setminus\{d\})}$$

is conspicuously *absent* from the set of broadcasted sub-keys, the newly excluded M_d is unable to compute the new group key:

$$K_{new} = g^{\widehat{x_c}\prod(x_m|m\in\{1,\ldots,n\})}.$$

A notable side-effect is that the excluded member's contribution x_d is still factored into the new key. Nonetheless, this in no way undermines the secrecy of the new key. In the event that the current group controller M_c has to be excluded, any other M_i can assume its role, assuming it stored the last broadcast message.

The cost of excluding a member is summarized as follows:

rounds	1
unicast messages	0
broadcast messages	1
cumulative message size	$n-1$
exponentiations per M_i	1 for $i \in \{1,\ldots,n\}\setminus\{c,d\}$, $(n-1)$ for $i=c$
exponentiations on critical path	$n-1$

Note that the use of precomputation can cut the cost of the critical path to a single exponentiation! Furthermore, the number of rounds, messages and

exponentiations is optimal. This holds as we are required to add a new exponent which has above computation and communication as a consequence. Note that the idea of directly reusing the partial GDH key from the old session key, which just contains the exponents of the current members, does not work despite its appeal of potentially not requiring any communication at all: The excluded member can always compute this value from the old session key and his own contribution.

4.3.5 Subgroup Exclusion

In most cases, subgroup exclusion is even simpler than single member exclusion. The protocol for *mass leave* is almost identical to that in Figure 4.10. The only difference is that the group controller computes and broadcasts fewer sub-keys; only those which correspond to the remaining members. Therefore, the cost of the protocol, when compared to the cost of member exclusion tabulated above, is even slightly cheaper (we can replace in the table above all terms -1 by $-n'$ where n' is the number of excluded members.)

A slightly different scenario is that of *group division* when a monolithic group needs to be split into two or more smaller groups. The obvious way of addressing this is to select for each of the subgroups a subgroup controller which runs the group exclusion protocol within its subgroup by broadcasting only those sub-keys corresponding to subgroup members.

In contrast to its counterpart (group fusion), I argue that *group fission* does not warrant any special treatment, i.e., a mechanism distinct from those illustrated thus far. The chief reason is that, in this case, the obvious solution works perfectly well.

4.3.6 Key Refresh

There are two main reasons for the group key refresh operation:

- limit exposure due to loss of group session keys, or
- limit the amount of ciphertext available to cryptanalysis for a given group session key.

This makes it important for the key refresh protocol not to violate key independence. (For example, this rules out using a straight-forward method of generating a new key as a result of applying a one-way hash function to the old key.) Additionally, note that the loss of a member's key share (x_i) can result in the disclosure of all the session keys to which the member has contributed with this share. Therefore, not only session keys, but also the individual key shares must be refreshed periodically.

Figure 4.11 Key Refresh

M_r $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ M_*

$$\xrightarrow{(g^{\widehat{x_r}\prod(x_m|m\in\{1,\ldots,n\}\setminus\{j\})}|\ j\in\{1,\ldots,n\})\setminus\{r\},\ g^{\prod(x_m|m\in\{1,\ldots,n+1\}\setminus\{r\})}}$$

Broadcast: round 1 ($x_r \leftarrow x_r\widehat{x_r}$)

This leads to the following key refresh protocol: The member M_r which is the least recent to have refreshed its key share[12] generates a new share (exponent) $\widehat{x_r}$ and "re-runs" the broadcast round as shown in Figure 4.11. All members then compute as usual the refreshed group key: $K_{new} = g^{\widehat{x_r}\prod(x_m|m\in\{1,\ldots,n\})}$.

This procedure guarantees key independence between different session keys and, due to the least-recently-refreshed policy, limits the damage of leaked key share to at most n epochs. We also note that this one-round protocol encourages precomputation and can be piggy-backed easily and at almost no cost on a group broadcast which is a likely operation assuming that the established group key is used to protect intra group communication. The cost of a key refresh, which is clearly optimal in all respects, is summarized as follows:

rounds	1
unicast messages	0
broadcast messages	1
cumulative message size	n
exponentiations per M_i	1 for $i \in \{1,\ldots,n\}\setminus\{r\}$,
	n for $i = r$
exponentiations on critical path	n

4.3.7 Security Considerations for AKA Operations

The security of the AKA operations is shown in the following theorem:

Theorem 4.2 *The CLIQUES AKA protocols are secure authenticated key-agreement protocols assuming that the assumption* $(1-1/\mathsf{poly}(k))$-DDH(c:$*$; g:m; f:fct,nsprim) *holds. In particular, they are contributory and ensure semantic security as well as freshness of*

[12] Of course, other policies on the choice of M_r are possible, too.

the group key. Additionally, they provide implicit and mutual group key authentication. Furthermore, the protocols provide key independence, PFS and are resistant to KKA. □

Proof (sketch). In order to demonstrate the security of the AKA protocols, we need to consider a snapshot in a life of a group, i.e., the lifespan and security of a particular short-term key.

The following sets are defined:

- $C = \{M_1, \ldots, M_c\}$ denotes all *current* group members with current key shares $x_1, \ldots, x_c$.
- $P = \{M_{c+1}, \ldots, M_p\}$ denotes all *past* (excluded before) group members with last key shares $x_{c+1}, \ldots, x_p$.
- $F = \{M_{p+1}, \ldots, M_f\}$ denotes all *future* (subsequently added) group members with $x_{p+1}, \ldots, x_f$ as their first contributed key shares.

Note that the term *future* is used relative to the specific session key.

The main security property we have to investigate is key independence. *Key secrecy* is then immediately implied by key independence. The remaining security properties follow from the various properties required by natural n-party extensions of DH (see Scheme 4.1) based on the same argumentation as used in Theorem 4.1. However, some remarks on *key freshness* are appropriate: key freshness can be deduced by the trust in the current group controller to refresh his key share and the freshness of the current epoch. The freshness of the current epoch in turn can be deduced from the secure linking of the epoch history (Property 4 of Scheme 4.1) and the freshness assurance obtained in the initial key agreement.

The issue at hand for *key independence* is the ability of all past and future members to compute the current key:

$$K = g^{x_1 \cdots x_c x_{c+1} \cdots x_p}.$$

To simplify our discussion, I collapse all members of P and F into a single powerful adversary (Eve). (This is especially fitting since P and F are not necessarily disjoint.) The result is that Eve $= P \cup F$ and she possesses $(x_j \mid M_j \in$ Eve$)$. Furthermore, we also can collapse conceptually all current members into a single entity as they are inherently trusted for this particular session and, therefore, behave honestly. Finally and without loss of generality, we can assume that M_c was group controller for both the operation leading to the current and to the following state.[13]

Let us first consider the case where Eve *attacks* only *passively*, i.e., in periods of legal membership in the group she follows the protocol to the

[13] Both group controllers must be in the current group and, therefore, are by definition honest. The fact that the current group controller could be excluded on the following round does not change this.

letter and otherwise she just eavesdrops. We can thus rewrite the key as:

$$K = g^{B(\prod(\mathcal{E}))}$$

where B is a *constant* known to Eve, and $\mathcal{E} = (x_1, \ldots, x_{c-1}, x_c)$ are the secret exponents (contributions) of current group members. Note that the group controller's current exponent x_c is independent from both its past exponent $x_c^{'} = x_c/\widehat{x_c}^{'}$ and its future exponent $x_c^{''} = x_c * \widehat{x_c}^{''}$. This holds as the blinding factors $\widehat{x_c}^{'}$ and $\widehat{x_c}^{''}$ were both chosen randomly and the multiplication in $\mathbb{Z}^*_{|G|}$ forms for the used groups G a statistically indistinguishable one-time pad (this follows from Lemma 3.1.)

In Eve's view, the only expressions containing x_c are in the upflows and the broadcast round of either the member addition or member exclusion protocol leading to the current key. This can be upper-bounded by:

$$\{g^{B\frac{x_1 \cdots x_{c-1} x_c}{\prod(x_i | M_i \in I)}} \mid I \subset C \ \wedge \ I \neq \{\} \ \} \quad .$$

If we assume that Eve can invert B (and if this assumption is wrong, Eve's task is certainly not easier), Eve can factor out B in all values above and Eve's view is equivalent to

$$\{g^{\frac{x_1 \cdots x_{c-1} x_c}{\prod(x_i | M_i \in I)}} \mid I \subset C \ \wedge \ I \neq \{\} \ \} \quad .$$

However, this corresponds exactly to the view of some protocol belonging to the class of natural n-party extensions of DH and, using the same argumentation as in Theorem 4.1, it follows that the secrecy of the key is guaranteed in this case.

Let us now consider an Eve which tries *active attacks*. Due to the properties of authenticated channels, Eve cannot affect the current session and can only gain advantage over a passive adversary by trying to "plant" an attack during her memberships in past epochs by not following the protocol. Assume now that any group member proceeds with a membership change only when the previous epoch terminated successfully, i.e., an agreement on a common key, and all receivers in the current epoch performed the tests required by Property 5 of Scheme 4.1, i.e., they verified that all partial keys contained in a message are indeed elements of G and are of maximal order. Then it is clear that the current key K has still the structure mentioned above and no attacks such as the small subgroup attack from Lim and Lee (1997) are possible. Furthermore, due to the fact that the exponent x_c is random and the group order has no small prime factors, the key K will be statistically indistinguishable from a random group element (this follows from Lemma 3.1). Therefore, the key secrecy is also maintained in this case.

Similarly to the IKA case, the inclusion of all required identifiers should also prevent any attack on AKA protocols in respect to key authentication. Furthermore, the case discussed above clearly also covers the case of the loss of past session keys and, therefore, the resulting protocol is also secure against KKA. PFS will be retained with similar reasoning as for IKA. ■

Remark 4.4. The key secrecy is maintained even in the presence of active attacks of dishonest excluded members in past epochs. However, there will be a priori a common agreement on the group key in the current epoch only if there was already one in the previous epoch. A dishonest excluded member could always, in particular as group controller, have disrupted the protocol in the past epoch so that no common key was shared then. To counter this (if this is a real concern) we would have to add some key confirmation flows. Unfortunately, in this case the simple approach of Remark 4.2 is not sufficient as even with an honest prover we are not sure now that he knows the same key as the verifier. However, if we use the following technique to efficiently implement the notification protocol from Remark 4.2 over unauthenticated channels, we also implicitly verify a common agreement on the session key in the current epoch: For this technique, we use the computed GDH key not as the session key but only as "meta-key". Using the meta-key (solely) as a seed to a *pseudo-random number generator* (Blum et al. 1986; Gennaro 2000), we compute the "real" session key and the notification messages by partitioning the output of the pseudo-random number generator into $m + 1$ chunks (where m is the number of required notification messages) of length proportional to the security parameter. The properties of the pseudo-random number generator guarantee, on the one hand, the unpredictability of the confirmation messages, while still tightly associating them to the sender's meta-key, and, on the other hand, the independence of the confirmation messages to the session key such that semantic security is not endangered. Furthermore, the key agreement protocol guarantees that the meta-key is uniquely associated with the session, epoch and corresponding group views. This means that we can also satisfy Property 4 of Scheme 4.1 without including explicit identifiers. ◦

4.4 Related Work

This section puts CLIQUES in context with related work. Primarily, the comparison is with other contributory key agreement protocols. However, at the end of this section I broaden the scope and briefly consider other group establishment protocols, e.g., key transport, as well.

4.4.1 Contributory Key Agreement

The earliest attempt to provide contributory key agreement and to extend DH to groups is due to Ingemarsson, Tang, and Wong (1982). The protocol in Figure 4.12 (called ING) requires synchronous startup and executes in $(n - 1)$ rounds. The members must be arranged in a logical ring. In a given round, every participant raises the previously-received intermediate key value to the power of its own exponent and forwards the result to the

Figure 4.12 ING Protocol

$$M_i \xrightarrow{g^{x_{(i-l+1) mod\, n} \cdots\, x_i}} M_{(i+1) mod\ n}$$

Round l; $l \in \{1, \ldots, n-1\}$

next participant. After $(n-1)$ rounds every group member computes the same key K_n.

We note that this protocol falls into the class of natural n-party extensions to DH as defined in Scheme 4.1 (assuming the protocol is suitably enriched with the properties mentioned in Scheme 4.1). It is, thus, suitable for use as an IKA protocol. However, the protocol is considerably less efficient in terms of communication than CLIQUES while having the same computational complexity than IKA.1. Furthermore, the limited amount of partial GDH keys, in particular such which contain the contribution of most group members, accumulated at the end of the protocol by any group member makes it difficult to use ING as a foundation for efficient auxiliary key agreement protocols.

Another DH extension geared towards teleconferencing was proposed by Steer, Strawczynski, Diffie, and Wiener (1990). This protocol (referred to as STR) requires all members to have broadcasting facilities and takes n rounds to complete. In some ways, STR is similar to IKA.1. Both take the same number of rounds and involve asymmetric operation. Also, both accumulate keying material by traversing group members one per round. However, the group key in STR has a very different structure:

$$K_n \;=\; g^{x_n g^{x_{n-1} g^{\cdots x_3 g^{x_1 x_2}}}} .$$

Therefore, STR does not fall into class of natural n-party extensions of DH and we cannot apply Theorem 4.1 to prove its security. To get a reasonable degree of security, e.g., semantic security in the standard model based on a common assumption such as DDH, it seems this requires groups G where the order does not contain any small factors and where there is a bijective mapping f from G to $\mathbb{Z}_{|G|}$ to transform keys to appropriately distributed secret exponents. However, the mapping $f(x) := x \pmod{|G|}$, as implicitly defined by STR, is certainly not bijective. While there is an efficient mapping for all prime-order subgroups of $\mathbb{Z}_p^*$ where p is a safe prime (Chaum 1991), it is not clear if such efficient mappings exist also for the other groups applicable to natural n-party extensions of DH. Hence, the exponentiations in the standard CLIQUES protocols could be considerably faster, e.g., by

the use of elliptic curves or subgroups of $\mathbb{Z}_p^*$ with much smaller order such as the ones used in DSS, than exponentiations in a secure version of STR. Steer et al. (1990) do not consider AKA operations. However, see below for some work which extends STR (IKA) with corresponding auxiliary operations.

One notable result is due to Burmester and Desmedt (1995). They construct a very efficient protocol (BD) which executes in only three rounds:

1. Each M_i generates its random exponent x_i and broadcasts $z_i = g^{x_i}$.
2. Each M_i computes and broadcasts $X_i = (z_{i+1}/z_{i-1})^{x_i}$.
3. Each M_i can now compute[14] the following group key:
$$K_n = z_{i-1}^{nx_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} \bmod p.$$

The key defined by BD is different from all protocols discussed thus far, namely $K_n = g^{x_1x_2+x_2x_3+\cdots+x_nx_1}$. Nonetheless, the protocol is proven secure provided the DH problem is intractable. However, they prove only the difficulty of a complete break, i.e., the recovery of the complete session key. It is not clear if this proof can be extended, at least in the standard model and not in the random oracle model, to semantic security as required in most practical applications.

Some important assumptions underly the BD protocol:

1. The ability of each M_i to broadcast to the rest of the group.
2. The ability to of each M_i to receive $n-1$ messages in a single round.
3. The ability of the system to handle n simultaneous broadcasts.

While the BD (IKA) protocol is efficient, I claim that it is also not well-suited for dynamic groups. On the one hand, above assumptions, in particular assumption 3, are quite strong and easily lead to congestion in the network. Of course, one could serialize the simultaneous broadcasts but then the resulting round complexity would exceed the CLIQUES protocols roughly by a factor of two and the main benefit of BD would be lost. On the other hand, we also have to consider the AKA operations for BD. While addition looks trivial at first sight, closer inspection reveals that all group members have to refresh their shares to prevent leaking too much information or serve as exponentiation oracles. This means that in fact AKA operation get as expensive in terms of communication and computation as the BD IKA, in fact, the only reasonable choice is to use BD IKA as-is for AKA protocols. In practice DPGs tend to start only with a very small number of initial members (if not even a single one) and grow mostly through AKA operations. Therefore, IKA operations are far less relevant than AKA operations.

[14] All indexes are modulo n.

Thus, the cost savings of BD IKA when compared to IKA.1 and IKA.2 are very quickly amortized and exceeded by the costs of their much less efficient AKA operations. In addition, Burmester and Desmedt (1995) proposed a variant of their protocol targeted at unauthenticated networks. However, as shown by Just and Vaudenay (1996) there is a (repairable) problem with the key authentication in this variant.

Becker and Wille (1998) systematically analyze the communication complexity of initial key agreement for contributory group key agreement protocols. They prove lower bounds for various measures and, e.g., confirm that IKA.1 is optimal in respect to the number of messages. Additionally, they describe a novel protocol, 2^d-octopus, which reaches the lower bound for simple rounds[15] ($d = \lceil \log_2 n \rceil$). Their main idea is to arrange the parties on a d-dimensional hypercube, i.e., each party is connected to d other parties. The protocol proceeds through d rounds, $1 \ldots d$. In the j-th round, each participant performs a two-party DH with its peer on the j-th dimension, using the key of the $j-1$-th round as its secret exponent. The exponents of the 0-th rounds are chosen at random by each party. For illustration purposes I show the resulting key for a group of 8 parties:

$$K_8 \;=\; g^{(g^{(x_1 x_2)} g^{(x_3 x_4)})} g^{(g^{(x_5 x_6)} g^{(x_7 x_8)})}.$$

While adding new members and in particular groups is easy with 2^d-octopus, it fails completely in terms of member exclusion. Splitting the group on the d-th dimension into two halves seems the only efficient exclusion procedure.

More recently, Kim, Perrig, and Tsudik (2000) presented a protocol suite, TGDH, using tree-based keys similar to the 2^d-octopus protocol. By basing *all* IKA and AKA protocols on key trees, these protocols overcome the problem on splitting groups mentioned in Section 4.3.3 in the sketch of tree-based group fusion protocol. TGDH improves the efficiency of join and merge when compared to the equivalent CLIQUES operations. Regarding computation costs, TGDH cuts down the critical path to $O(\log(n))$ exponentiations. The use of precomputation and pipelining as well as the potentially cheaper exponentiation — TGDH faces the same limitation on the choice of the algebraic group as STR — can narrow the gap for the CLIQUES protocols. Nonetheless, due to the logarithmic growth factor, TGDH will eventually exceed CLIQUES in efficiency as groups get large. Regarding communication costs, TGDH provides a considerably more round-efficient merge operation than the merge-by-mass-join method of CLIQUES in the case when both of the merging groups are larger than $O(\log(n))$. However, all these benefits are somewhat comprised by the fact that the security argument relies on the random oracle model.

The same authors later reconsider in Kim et al. (2001) STR as a basis for AKA operations. While the computational costs are inferior to TGDH and comparable to CLIQUES when considering all mentioned optimizations

[15] Simple rounds are rounds where each member sends and receives at most one message.

— join and merge will be cheaper and exclusion will be more expensive — the protocols improve the communication cost of all AKA operations to a constant number of rounds. If we assume Moore's Law to hold on, exponentiations will become cheaper and cheaper over time[16] and, eventually, the cost of latency, which is lower bounded by the speed of light, will dominate the cost of computation in determining the runtime of the discussed protocols. Therefore, the STR-based protocols proposed in Kim et al. (2001) seem to be the currently most efficient group key agreement protocol suite when one does not require: (1) a formal security proof in the standard model — the security argument in Kim et al. (2001) relies on the fact that their protocols is a special case of TGDH which was proven informally and in the random oracle model only — and (2) the flexibility in the choice of the algebraic group — the issue of the bijective mapping mentioned for STR and TGDH also applies here — provided by CLIQUES.

Finally, Tzeng (2000) and Tzeng and Tzeng (2000) propose contributory key agreement schemes based on some form of verifiable secret sharing. However, it does not seem that the schemes do have any performance advantages over CLIQUES. Furthermore, the protocols do not achieve semantic security and their claim that their protocols provide fair (unbiased) session keys seems wrong as the protocols are clearly susceptible to problems such as the ones identified by Gennaro, Jarecki, Krawczyk, and Rabin (1999) unless we assume an (unrealistic) synchronous model with no rushing adversaries.

4.4.2 Key Transport

The focus in my work was on contributory key agreement, not key transport. As discussed in Chapter 2 contributory key agreement has a number of advantages over (centralized) key transport. However, there is one main drawback with contributory schemes. Due to the contributory nature and perfect key independence, the natural n-party extension of DH inevitably require exponentiations linear in the number of participants for AKA operations; of course, this does not scale well to very large groups. This is not a fundamental problem for DPGs as they tend to be reasonably small (< 100). Furthermore, as mentioned above the importance of the computational cost will probably vanish over time when compared to costs due to latency.

However, in situations where the security, fault-tolerance and flexibility requirements are less stringent and scalability and computation efficiency is the main issue, key distribution protocols might be more favorable.

Early key transport proposals (Harney and Muckenhirn 1997; Gong 1997) were all based on a fixed group controller and did not

[16] While we do have to increase the size of the underlying algebraic groups with the increase of the available computational resources, the required increase in size is only roughly logarithmically in the gain of computational power even when considering additional factors such as algorithmic progress (Odlyzko 2000a; Lenstra and Verheul 2001).

address scalability or dynamics in group membership to a large extent. Subsequent work (Ballardie 1996; Mittra 1997) addressed scalability by splitting up the group into a hierarchy of subgroups controlled by subgroup controllers. These protocols improve overall efficiency but their support for the dynamics of group is either rather limited or has costly side effects, e.g., Iolus (Mittra 1997) requires intermediary subgroup controllers to relay all messages and perform key translation.

Tree-based group rekeying systems, commonly called Logical Key Hierarchy (LKH), independently proposed by Wallner, Harder, and Agee (1997) and Wong, Gouda, and Lam (1998), achieve all AKA operations in 2 rounds and bring down the communication and storage costs down to $O(\log(n))$. Optimized variants (McGrew and Sherman 1998; Canetti, Garay, Itkis, Micciancio, Naor, and Pinkas 1999) reduce the communication overhead by half and their security can be proven using standard cryptographic assumptions. Due to their communication and computation efficiency, these protocols scale very well to large groups. Their main drawback is their reliance on a fixed group controller. Caronni, Waldvogel, Sun, Weiler, and Plattner (1999) overcome this by distributing the role of group controller over all members. Unfortunately, as they note themselves their protocols are vulnerable to collusions by excluded members. Another approach to increase safety of the tree-based group rekeying schemes is described in Rodeh, Birman, and Dolev (2002). Finally, further smaller optimizations for LKH protocols, e.g., applying the idea from Setia, Koussih, and Jajodia (2000) to bundle rekey operations in periodic operations, are presented by Perrig, Song, and Tygar (2001).

4.4.3 Other

Further related work we can find in the context of distributed and fault-tolerant computing (Birman 1996; Reiter et al. 1994). Protocol suites and toolkits such as Rampart (Reiter 1996; Reiter 1994) aim at achieving high fault-tolerance, even in the presence of malicious (i.e., byzantine) faults inside a group. This level of fault-tolerance and the underlying model of virtual synchronous process groups might be required for securely and reliably replicating services (Reiter and Birman 1994) of great importance. However, these systems are very expensive as they rely on reliable and atomic multicasts secure against byzantine faults, e.g., Cachin et al. (2001).

4.5 Summary

In summary, this chapter presented the CLIQUES protocol family for IKA and AKA operations based on the Diffie-Hellman key exchange. The protocols match virtually all requirements identified in Chapter 2 and achieve secure key agreement in the context of dynamic peer groups. The protocols

are very flexible and, except for the group merge operation, quite efficient. It remains an open question whether one can find more efficient group merge operations in the class of natural n-party extension of DH (or prove there non-existence.)

However and more importantly, while the argumentation for the security of the protocols represent the practice of proving security for group key protocols in the past, the proofs are not very formal. This aspect is the focus of the remaining investigations and brought to more formal foundations in the next chapter.

Chapter 5

Formal Model and Proofs

In this chapter, I put the security argumentation of the previous chapter into a formal setting. To achieve this, I define a general formal model for group key establishment protocols. I then give a detailed and rigorous proof for one of the previously presented protocols, the initial key agreement IKA.1. In particular, I show that under the Decisional Diffie-Hellman assumption and the addition of a confirmation flow this protocol is secure even in the presence of strong adaptive adversaries.

KEY-ESTABLISHMENT protocols have a long history of new protocols improving over past work in various aspects such as efficiency, features or security. However, this history is also paved with numerous flaws in many protocols which got only discovered later. Most of these flaws are due to an ad-hoc security analysis and due to overlooking various attacks. Building the protocol with systematic design (Bird et al. 1993) and following prudent design and engineering principles (Anderson and Needham 1995; Abadi and Needham 1996) can greatly reduce this risk. However, only a sound underlying formal model and rigorous security proofs can give real assurance of security.[1]

This was recognized in early stages and lead to work on the formalization of cryptographic protocols and key establishment in particular. Most of this work can be traced back to a model introduced by Dolev and Yao (1983).

[1] Obviously, not only the security of the protocol but also many other aspects are critical for the overall security: The correctness of the requirement analysis and the specifications, the robustness of the implementation and its faithfulness to the specifications, the appropriateness of the deployment (configuration), the security of the (operating) systems, the appropriate education of users, ... So one might argue (Schneier 1999) that provable security does not really matter as most security breaches in practice are not directly related to flaws in the protocols themselves. However, there are still a considerable number of attacks which would never have occurred with appropriate security proofs and it seems only prudent to strive for the best achievable security for *each* of these *orthogonal* aspects.

The fundamental idea of the Dolev-Yao model is to assume perfect cryptography (e.g., the encryption $E(m)$ of a message m hides unconditionally all information on m) and to abstract it with a term algebra with cancellation rules (e.g., the decryption of an encryption leads again to the original message: $D(E(m)) = m$). Various approaches based on this idea were explored: ad-hoc constructions (Millen et al. 1987; Meadows 1992; Meadows 1996), belief logics (Burrows et al. 1990; Gong et al. 1990; Syverson and van Oorschot 1994), explorations of finite-state models (Lowe 1996) or inductive proofs in predicate or process calculi (Kemmerer 1989; Lowe 1996; Abadi and Gordon 1997; Bolignano 1996; Paulson 1997). They allow for various trade-offs between ease-of-use, efficiency and completeness. See Gritzalis et al. (1999) and Millen (1998) for an overview of these techniques.

Dolev-Yao's way of abstracting cryptography is appealing by presenting a simple and discrete model with no need to reason about number-theory and complexity-theoretic (probabilistic) settings. Unfortunately, an attacker can also try to exploit the low-level "ingredients" of the cryptographic primitives and their interference with the high-level protocol. As shown by Pfitzmann, Schunter, and Waidner (2000) we cannot rely on the classical security definitions used in the cryptographer community, e.g., semantic security or security against chosen-ciphertext attacks for encryptions. It is possible to concoct protocols which are secure in the Dolev-Yao model and, nonetheless, realizations with primitives provably secure in the above-mentioned cryptographic sense can still lead to a completely insecure protocol. Work to bridge this gap and to define robust cryptographic definitions or primitives which securely realize the Dolev-Yao abstraction is still in a premature state, e.g., only limited additional properties such as homomorphic or multiplicative properties (Even, Goldreich, and Shamir 1986; Pereira and Quisquater 2000) or weaker (non-adaptive and passive) attackers (Abadi and Rogaway 2002) were considered.

Only few researchers have worked on formalizing authentication and key-exchange protocols with no cryptographic abstractions. This work was pioneered by Bellare and Rogaway (1994, 1995b) for shared-key cryptography and extended by Blake-Wilson and Menezes (1998) to public-key cryptography. Shoup (1999) pointed out serious (yet salvageable) problems and limitations in the Bellare-Rogaway model[2] and, extending prior work by Bellare, Canetti, and Krawczyk (1998), proposed a model based on the ideal-host paradigm. The ideal-host paradigm allows to clearly layer protocols, e.g., to build secure sessions on top of a key exchange protocol. The model of Shoup can be considered as the cur-

[2]Most notably, the Bellare-Rogaway model captures adaptive adversaries only after suitably extending the model with perfect forward secrecy (Shoup 1999, Section 15.5 & 15.6) and there is no composition theorem to allow the use of session keys in an arbitrary context.

rent state-of-the-art and has been applied also to variations, such as authenticated key-exchange relying only on passwords as long-term secrets (Boyko, MacKenzie, and Patel 2000). Nonetheless, the model of Shoup is still relatively ad-hoc and lacks the underpinning of a clear and formal (meta-)model of communication, computation, and adversaries for general reactive protocols such as the model from Pfitzmann and Waidner (2001).

Aspects of group communication are so far mainly neglected. Only little past work on formalizing group key establishment protocols exists and it is either limited in scope (Mayer and Yung 1999) (key distribution only, no key agreement and no consideration of group dynamics) or still work-in-progress (Meadows 2000; Pereira and Quisquater 2001); the latter two also suffer from the aforementioned fundamental problems of the Dolev-Yao model. Independent of the following work, Bresson, Chevassut, Pointcheval, and Quisquater (2001) proposed very recently a formal definition of initial key agreement based on the formalization tradition of Bellare and Rogaway (1994) and prove the security of protocols very similar to the ones given here. This work was also extended to auxiliary protocols in Bresson, Chevassut, and Pointcheval (2001). (See below for a short comparison of this approach with the one chosen here.) Finally, the formal specification of some requirements for a concrete group key establishment protocol suite is proposed in Meadows, Syverson, and Cervesato (2001).

In the following, I give a precise definition of group key establishment in the simulatability-based model of Pfitzmann and Waidner (2001): Essentially, I specify an **ideal system** for group key establishment where a single, non-corruptible party TH, called **trusted host**, serves all parties. Whenever a group wants to establish a new key, TH chooses a random key and distributes it to all group members, provided they are all non-corrupted. Depending on when a member becomes corrupted, TH gives the random key to the adversary A or lets A even choose the keys for the non-corrupted parties. I assume an asynchronous network completely controlled by the adversary. The definition of the ideal system covers most informal *security notions* discussed in Section 2.1 and 2.2 like key authentication and forward secrecy. It also covers *auxiliary protocols*. Furthermore, these properties persist under arbitrary composition with higher-level protocols (Pfitzmann and Waidner 2001). A **real system** for group key establishment is a system where parties have to agree on a key without the help of such a "magic" non-corruptible trusted party. It is considered secure if whatever happens to the honest users in this real system, with some adversary A, could happen to the same honest users in the ideal system, with some other adversary A'.

This form of specification is quite natural and intuitive. Furthermore, the robustness of the specification under arbitrary composition allows us to tolerate any (potentially unexpected) use of session keys and, e.g., makes

it quite natural to specify and design modular secure channels for groups.[3] These desirable properties are the major distinctions of the specification style used here when compared with the more ad-hoc manner[4] of specifications following the tradition of Bellare and Rogaway (1994).

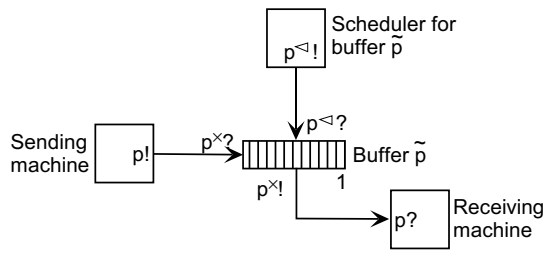
Above translates also into advantages of the model of group key establishment presented here over the model of Bresson et al. (2001). Further advantages are (1) the generality of the model which is applicable to arbitrary group key agreement and distribution protocols (and not limited to Diffie-Hellman-based protocols only), (2) the tolerance of stronger adaptive adversaries which on corruption receive *all* state (and not only long-term keys as assumed in Bresson et al. (2001)), and (3) the security of auxiliary operations which provides security also against misbehaving excluded members, i.e., insiders to a particular group session history (whereas Bresson et al. (2001) consider only security against outsiders). The protocols proven secure in Bresson et al. (2001) are quite similar to the protocols proven here. In particular, they are also based on the CLIQUES IKA (IKA.1) and AKA protocols. They mainly differ in providing — by appropriate use of signatures — security directly in unauthenticated networks instead of using a modular approach with compilers (see Chapter 4) as chosen here.

The organization of the remainder of this chapter is as follows. In Section 5.1, I briefly recapitulate the model of Pfitzmann and Waidner (2001). In Section 5.2, I give the details of the formal model, i.e., the ideal system with a trusted host, for secure authenticated group key establishment and discuss the different properties. Subsequently in Section 5.3, I formalize the protocol IKA.1 presented in Section 4.2.1 and I also derive a second protocol to handle adaptive corruptions. Equipped with these definitions, I analyze the security of the two proposed concrete protocols in Section 5.4. In particular, I prove them secure against static and adaptive adversaries, respectively. In this process, I also investigate the concrete security of the interactive version of the DGDH problem discussed in Section 3.5.

[3]For session-keys used in the implementation of a secure channel a slightly weaker definition might be sufficient (Shoup 1999; Canetti and Krawczyk 2001a). However, I believe that the entirety of properties offered by a group key establishment protocol is simpler to capture in a trusted-host style definition, in particular when one considers the design of surrounding group-oriented applications other than secure channels.

[4]One might well argue that the problems and limitations in the Bellare-Rogaway model which required various changes and modifications (Blake-Wilson et al. 1997; Blake-Wilson and Menezes 1999; Shoup 1999; Bellare et al. 2000) are due to the ad-hoc manner of specification.

Figure 5.1 Ports and buffers



5.1 Basic Definitions and Notation

Our definitions and proofs are based on the notion of standard cryptographic systems with adaptive corruptions as defined in Pfitzmann and Waidner (2001), Section 3.2. We briefly recapitulate this model, omitting all details not needed here.

5.1.1 System Model and Simulatability

The systems are parametrized with a **security parameter**, $k \in \mathbb{N}$, and depend on the number of participants, $n \in \mathbb{N}$. Let $\mathcal{M} := \{1, \ldots, n\}$.

The main component of a system is a set of **protocol machines**, $\{\mathsf{M}_1, \ldots, \mathsf{M}_n, \mathsf{Gen}\}$ for real systems and $\{\mathsf{TH}\}$ for ideal systems. Intuitively, M_u serves user u. Machine Gen is incorruptible; it is used for reliably generating and distributing initial parameters used by all machines (in our case a cyclic group and generator for the Diffie-Hellman setting and a corresponding universal hash function to map GDH keys to bit strings).

The machines are *probabilistic state-transition machines* (where the state-transition functions are realized by arbitrary probabilistic *Turing machines*.) Each machine can communicate with other machines via ports. **Output (input) ports** are written as $\mathsf{p}!$ ($\mathsf{p}?$), and $\mathsf{Ports}(\mathsf{M})$ denotes the set of all ports of a machine M. Messages are transported from $\mathsf{p}!$ to $\mathsf{p}?$ over a connection represented by a **buffer** machine $\widetilde{\mathsf{p}}$. A buffer $\widetilde{\mathsf{p}}$ stores all messages received from $\mathsf{p}!$ at $\mathsf{p}^{\times}?$ and waits for inputs on its **clock port** $\mathsf{p}^{\triangleleft}?$. Each input $i \in \mathbb{N}$ triggers $\widetilde{\mathsf{p}}$ to put the i-th stored message on $\mathsf{p}^{\times}!$ (or no message if it contains less than i messages) to be forwarded to $\mathsf{p}?$. Ports and buffers are illustrated in Figure 5.1.

A **structure** is a pair (M, S), where M is a set of machines and S, the **specified ports**, is a subset of the free ports[5] of the union of M and all the buffer machines needed for connections used or clocked by machines in M.

[5] **Free ports** of a set of machines are all input (output) ports $\mathsf{p}?$ ($\mathsf{p}!$) where the corresponding output (input) port $\mathsf{p}!$ ($\mathsf{p}?$) is not associated to any machine in the set.

S models the service interfaces offered or required by M. The remaining free ports will be available to the adversary and model unavoidable or tolerable adversary control and information flow to and from the adversary. This is often required — even in an ideal system — to achieve realistic models without further unwanted restriction, e.g., for a practical key establishment protocol there is normally no harm when the adversary learns who runs the protocol with whom. Nonetheless, without modeling this information flow in a trusted host, a faithful implementation of that trusted host would have to be based on a (costly) anonymous network.

A structure describes (known) components and their interaction with the (unknown) environment. However, to obtain a whole runnable system we have to specify the environment, too. Therefore, the structure (M, S) is complemented to a **configuration** by adding an arbitrary **user machine** H, which abstracts higher-layer protocols and ultimately the end user, and an **adversary machine** A. H connects to ports in S and A to the rest, and they may interact. We will describe the specified ports not directly but by their *complements*, S^c, i.e., by listing the ports that H should have. Finally, a **system** *Sys* is a set of structures.

The machines in a configuration are scheduled sequentially: In principle only buffers have clock input ports, like $\mathsf{p}^{\triangleleft}?$ for buffer $\widetilde{\mathsf{p}}$. The currently active machine M_s can schedule any buffer $\widetilde{\mathsf{p}}$ for which it owns $\mathsf{p}^{\triangleleft}!$, and if $\widetilde{\mathsf{p}}$ can actually deliver a message, this schedules the receiving machine M_r. If M_s tries to schedule multiple buffers at a time then only one is taken, and if no buffer is scheduled (or the scheduled one cannot deliver a message) then a designated **master scheduler** is scheduled; usually, the adversary A plays that role. A configuration is a runnable system, i.e., one gets a probability space of runs and views of individual machines in these runs.

Simulatability essentially means that whatever can happen to certain users in the real system can also happen to the same users in the ideal system: for each configuration $(M, S, \mathsf{H}, \mathsf{A})$ there is a configuration $(\{\mathsf{TH}\}, S, \mathsf{H}, \mathsf{A}')$ such that the views of H in the two configurations are indistinguishable (Yao 1982). Simulatability is abbreviated by "$\geq_{\mathsf{sec}}$." As by definition only good things can happen in the ideal system, simulatability guarantees that no bad things can happen in the real world.

5.1.2 Standard Cryptographic Systems

In a **standard cryptographic system** with static adversaries, *Sys* is a set of structures $(M_{\mathcal{H}}, S_{\mathcal{H}})$, one for each set $\mathcal{H} \subset \mathcal{M}$ of non-corrupted users. The structures $(M_{\mathcal{H}}, S_{\mathcal{H}})$ are derived from an intended structure (M^*, S^*), where $M^* = \{\mathsf{M}_1^*, \ldots, \mathsf{M}_n^*\}$, $S^{*c} = \{\mathsf{in}_u!, \mathsf{in}_u{}^{\triangleleft}!, \mathsf{out}_u? \mid u \in \mathcal{M}\}$ and $\{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!\} \subseteq \mathsf{Ports}(\mathsf{M}_u^*)$. Each $S_{\mathcal{H}}$ is the subset of S^* where

u only ranges over $\mathcal{H}$.[6] The derivation depends on a **channel model**: Each connection (i.e., buffer) of (M^*, S^*) is labeled as "**secure**" (private and authentic), "**authenticated**" (only authentic), or "**insecure**" (neither authentic or private.) In the derivation all output ports of authenticated connections are duplicated; thus A connects to them and can read all messages. All insecure connections are routed through A, i.e., the ports are renamed so that both become free and thus connected to A. The reliability of a connection is implicitly determined by the definition of specified ports: If the clock output port corresponding to a buffer is a specified port, we have a **reliable**, otherwise an **unreliable** channel.

For adaptive adversaries,[7] the derivation makes some additional modifications: The specified ports are extended[8] by ports $\{\mathsf{corrupt}_u!, \mathsf{corrupt}_u{}^{\triangleleft}! \mid u \in \mathcal{M}\}$ used for corruption requests.[9] Furthermore, each M_u gets three additional ports $\mathsf{corIn}_u?$, $\mathsf{corOut}_u!$ and $\mathsf{corOut}_u{}^{\triangleleft}!$ for communication with A after corruption: If M_u receives (do) on $\mathsf{corrupt}_u?$ in state σ it encodes σ in some standard way and outputs (state, σ) at $\mathsf{corOut}_u!$ (i.e., reveals everything it knows to A). From then on it is taken over by A and operates in **transparent mode**: Whenever M_u receives an input m on a port $\mathsf{p}? \neq \mathsf{corIn}_u?$, it outputs (p, m) at $\mathsf{corOut}_u!$. Whenever it receives an input (p, m) on $\mathsf{corIn}_u?$ for which $\mathsf{p}!$ is a port of M_u, it outputs m at that port. Over time any subset of $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ can become corrupted.[10]

5.1.3 Notation

Variables are written in italics (like *var*), constants and algorithm identifiers in straight font (like const and algo), and sets of users in calligraphic font (like $\mathcal{M}$). For a set $set \subseteq \mathbb{N}$ and $i \leq |set|$, let $set[i]$ denote the i-th element with respect to the standard order $<$ on $\mathbb{N}$ and $\mathsf{idx}(set, elem)$ the index of *elem* in *set* if present and -1 otherwise, i.e., $\mathsf{idx}(set, set[i]) = i$ for $i \in \{1, \ldots, |set|\}$.

Machines are specified by defining their state variables and transitions. The variable *state* of M_i is written as $\mathsf{M}_i.state$, or, if clear from the context such as in a transition rule, as *state* only. To simplify notation, we allow arrays that range over an infinite index set, like $(a_i)_{i \in \mathbb{N}}$, but always initialize them everywhere with the same value (e.g., undef for "undefined"). Thus, they can be efficiently represented by polynomial-time machines.

[6] Consequently, for each set $\mathcal{H}$ one trusted host $\mathsf{TH}_{\mathcal{H}}$ is defined.

[7] For a more concise presentation and without loss of generality, I slightly deviate from Pfitzmann and Waidner (2001): I use a separate structure for each set $\mathcal{H} \subset \mathcal{M}$ also for the adaptive case even though a single structure for $\mathcal{M}$ would have sufficed.

[8] If those names are already occupied they can be renamed arbitrarily.

[9] Those must be made via specified ports as the service will change at the corresponding ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ also in the ideal system.

[10] In terms of Pfitzmann and Waidner (2001): our adversary structure is $\mathcal{ACC} = 2^{\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}}$.

Transitions are described using a simple language similar to the one proposed in Garland and Lynch (2000). Most of the notation should be clear without further explanations. Each transition starts with "**transition** $\mathsf{p}?(m)$" where $\mathsf{p}?$ is an input port and m an abstract message, i.e., a message template with free variables. Optional parameters in m are denoted by $[\ldots]$ and their presence can be queried using the predicate $\mathsf{present}(\cdot)$. An "**enabled if:** *cond*" (where *cond* is an arbitrary boolean expression on machine-internal state variables) specifies the condition under which the transition is enabled. If the (optional) **enabled if:** is absent, the transition is always enabled. When a message msg arrives at a port $\mathsf{p}?$ and all transitions on this port are disabled, the message is silently (and at no computational cost[11] for the corresponding machine) discarded. Otherwise, we first increment the **message counter** $\mathsf{p}?.cntr$ associated with the given input port $\mathsf{p}?$. This counter keeps track of the number of activations on a port (and indirectly the computational cost of a machine) and is initialized to zero. If the message msg matches the template m of any enabled transition on this port, the corresponding transition is executed. Without loss of generality, we further require from the specification that at any given time at most one enabled transition matches any given message. The final states of a machine are implicitly defined as the situations when no transition is enabled anymore.

5.2 Ideal System for Group Key Establishment

The following scheme specifies the trusted host for an ideal system for group key establishment.

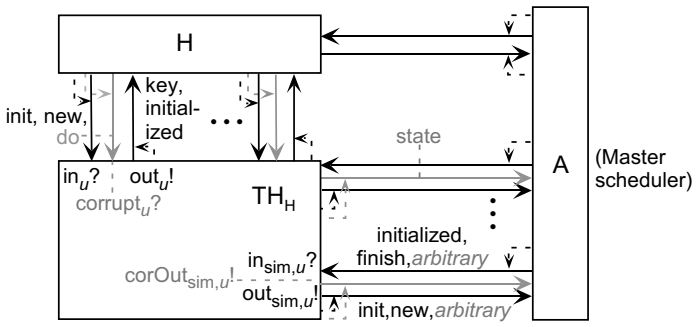
Scheme 5.1 (Ideal System for Group Key Establishment $Sys^{\mathsf{gke,ideal}}_{n,tb,ct}$)
Let $n, tb \in \mathbb{N}$ and $ct \in \{\mathsf{static}, \mathsf{adaptive}\}$ be given, and let $\mathcal{M} := \{1, \ldots, n\}$. Here, n denotes the number of intended participants, tb a bound on the number of activations per port — this is required to make TH polynomial — and ct the type of corruptions to be tolerated. An ideal system for secure group key establishment is then defined as

$$Sys^{\mathsf{gke,ideal}}_{n,tb,ct} = \{(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}.$$

Here $\mathcal{H}$ denotes the set of a priori uncorrupted participants. Let $\mathcal{A} := \mathcal{M} \setminus \mathcal{H}$.

[11] Recall that the condition *cond* of **enabled if:** depends only on machine-internal state variables. This allows the computation to be done on state-changes and requires no computation on message arrival. For example, if the condition also would depend on the message, a real-time evaluation (and, hence, computation costs) would be required on message arrival. This would make such a construct unsuitable for the context discussed here, i.e., specifying how ports can be disabled such that messages on these ports do not incur any computational costs. Nevertheless, as such broader conditions are useful in other cases, there is a second similar construct "**ignore if:** *cond*" where *cond* may also depend on variables of the message.

Figure 5.2 Trusted host and its message types. Parts related to adaptive adversaries are in gray. Dashed lines indicate who schedules a connection.



An overview of $\mathsf{TH}_{\mathcal{H}}$ is given in Figure 5.2. The ports of $\mathsf{TH}_{\mathcal{H}}$ are $\{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!, \mathsf{corrupt}_u?, \mathsf{in}_{\mathsf{sim},u}?, \mathsf{out}_{\mathsf{sim},u}!, \mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!, \mathsf{corOut}_{\mathsf{sim},u}!, \mathsf{corOut}_{\mathsf{sim},u}{}^{\triangleleft}! \mid u \in \mathcal{H}\}$. The specified ports are as described in Section 5.1.2 for standard cryptographic systems, i.e., $S^{*\,c}_{\mathcal{H}} = \{\mathsf{in}_u!, \mathsf{in}_u{}^{\triangleleft}!, \mathsf{out}_u? \mid u \in \mathcal{H}\}$ and $S^{c}_{\mathcal{H}} = S^{*\,c}_{\mathcal{H}} \cup \{\mathsf{corrupt}_u!, \mathsf{corrupt}_u{}^{\triangleleft}! \mid u \in \mathcal{H}\}$.

The message formats exchanged over the ports are shown in Table 5.1. Common parameters are as follows: $u \in \mathcal{M}$ identifies a user, $grp \subseteq \mathcal{M}$ is the set of group members, $sid \in \mathcal{SID}$ is a session identifier (relative to a group grp), and $key \in \{0,1\}^k$ is an exchanged session key. The domain of session identifiers, $\mathcal{SID}$, can be arbitrary as long as the representations of elements can be polynomially bounded in k (otherwise resulting machines might not be polynomial anymore.)

The state of $\mathsf{TH}_{\mathcal{H}}$ is given by the variables shown in Table 5.2. The state-transition function is defined by the following rules; the message types are also summarized in Figure 5.2.

Initialization. Assume an honest $u \in \mathcal{M}$, i.e., one with $u \in \mathcal{H}$ and $\mathsf{TH}_{\mathcal{H}}.state_{u,u} \neq \mathsf{corrupted}$. H triggers initialization of u by entering init at $\mathsf{in}_u?$. In a real system, M_u will now set system parameters, generate long-term keys, etc., and possibly send public keys to other machines. In the ideal system $\mathsf{TH}_{\mathcal{H}}$ just records that u has triggered initialization by the state wait. Any subsequent input init is ignored. The adversary immediately learns that u is initializing. (In most real systems initialization requires interaction with other machines, which is visible to the adversary.)

transition $\mathsf{in}_u?$ (init)

enabled if: $(state_{u,u} = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;

$state_{u,u} \leftarrow \mathsf{wait}$;

output: $\mathsf{out}_{\mathsf{sim},u}!$ (init), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);

Table 5.1 The message types and parameters handled by $\mathsf{TH}_{\mathcal{H}}$

Port	Type	Parameters	Meaning
At specified ports $S_{\mathcal{H}}$ to user $u \in \mathcal{H}$			
$\mathsf{in}_u?$	init		Initialize user u.
$\mathsf{out}_u!$	initialized	v	User v initialized from user u's point of view.
$\mathsf{in}_u?$	new	sid,grp,[sid',grp']	Initialize a new session, extending a previous one if optional parameters are present.
$\mathsf{out}_u!$	key	sid,grp,key	Return newly agreed key.
$\mathsf{corrupt}_u?$	do		Corrupt user u!
$\mathsf{out}_u!$	*arbitrary*	*arbitrary*	Possible outputs after corruptions
At adversary ports			
$\mathsf{out}_{\mathsf{sim},u}!$	init		User u is initializing.
$\mathsf{in}_{\mathsf{sim},u}?$	initialized	$v \in \mathcal{M}$	User u should consider user v as initialized.
$\mathsf{out}_{\mathsf{sim},u}!$	new	sid,grp,[sid',grp']	User u has initialized a new session.
$\mathsf{in}_{\mathsf{sim},u}?$	finish	sid,grp,[$key_{u,sim}$]	Complete session for user u. If present and allowed, assign $key_{u,sim}$ to user u.
$\mathsf{corOut}_{\mathsf{sim},u}!$	state	*state*	State of corrupted party.
$\mathsf{out}_{\mathsf{sim},u}!$	*arbitrary*	*arbitrary*	Corrupted party u sent a message.
$\mathsf{in}_{\mathsf{sim},u}?$	*arbitrary*	*arbitrary*	Send message to (corrupted) party u.

Table 5.2 Variables in $\mathsf{TH}_{\mathcal{H}}$

Name	Domain	Meaning	Init.
$(state_{u,v})_{u,v\in\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by user u	undef
$(ses_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{init}, \mathsf{finished}\}$	State of sessions as seen by user u	undef
$(key_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}\}$	Session keys still in negotiation	undef
$(prev_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$(sid' \in \mathcal{SID}, grp' \subseteq \mathcal{M})$	Dependency graph of sessions	$(0, \{\})$
$(p?.cntr)_{p\in\{\mathsf{in}_u,\mathsf{corrupt}_u,\mathsf{in}_{\mathsf{sim},u} \mid u\in\mathcal{H}\}}$	$\mathbb{N}$	Activation counters	0

end transition

By entering $(\mathsf{initialized}, v)$ at $\mathsf{in}_{\mathsf{sim},u}?$ the adversary triggers that an honest user u learns that user v, potentially u itself, has been initialized. Note that this can happen even before u has been initialized itself.

This transition is only enabled when user u is not corrupted and the port's transition bound is not exceeded. The first condition is necessary to disambiguate between this ("honest"-mode) transition and transparent mode after a corruption, i.e., the last two transitions below. The second condition helps making the machine polynomial-time. Both conditions are also part of the enable condition of all other "honest"-mode transitions.

transition $\mathsf{in}_{\mathsf{sim},u}?$ $(\mathsf{initialized}, v)$
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_{\mathsf{sim},u}?.cntr < tb)$;
  ignore if: $((state_{v,v} = \mathsf{undef}) \wedge (v \notin \mathcal{A})) \ \vee \ ((u = v) \wedge (state_{u,u} \neq \mathsf{wait}))$;
  $state_{u,v} \leftarrow \mathsf{init}$;
  output: $\mathsf{out}_u!$ $(\mathsf{initialized}, v)$, $\mathsf{out}_u{}^{\triangleleft}!$ (1);
end transition

Group key establishment. To start a group key establishment for user u, H enters $(\mathsf{new}, sid, grp, [sid', grp'])$ at $\mathsf{in}_u?$. User u has to be a mem-

ber of the intended group and has to believe that all group members are initialized. Furthermore, the pair (sid, grp) has to be fresh, i.e., never used by u before ($\mathsf{TH}_{\mathcal{H}}.ses_{u,sid,grp} = \mathsf{undef}$); otherwise the command is ignored. The optional parameter (sid', grp') points to a previous group key establishment to which the current one is auxiliary. If (sid', grp') is present, it is required that either $u \notin grp'$ (i.e., this member is added), or the old establishment has terminated and the previous group key was delivered ($\mathsf{TH}_{\mathcal{H}}.ses_{u,sid',grp'} = \mathsf{finished}$). The pair (sid', grp') is recorded in $\mathsf{TH}_{\mathcal{H}}.prev_{u,sid,grp}$. In the real system, M_u would now start the protocol. $\mathsf{TH}_{\mathcal{H}}$ just records this fact ($\mathsf{TH}_{\mathcal{H}}.ses_{u,sid,grp} \leftarrow \mathsf{init}$). The adversary immediately learns over port $\mathsf{out}_{\mathsf{sim},u}$? that u has started an establishment with parameters $sid, grp, [sid', grp']$.

transition in_u? (new, $sid, grp, [sid', grp']$)
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  ignore if: $(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_{u,v} \neq \mathsf{init}) \vee$
    $(ses_{u,sid,grp} \neq \mathsf{undef}) \vee$
    $(\mathsf{present}(sid', grp') \wedge (u \in grp') \wedge (ses_{u,sid',grp'} \neq \mathsf{finished}))$;
  $ses_{u,sid,grp} \leftarrow \mathsf{init}$;
  if $\mathsf{present}(sid', grp')$ **then**
    $prev_{u,sid,grp} \leftarrow (sid', grp')$;
  end if;
  output: $\mathsf{out}_{\mathsf{sim},u}!$ (new, $sid, grp, [sid', grp']$), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
end transition

The adversary decides to finish the protocol for u by entering (finish, sid, $grp, [key_{u,sim}]$) at $\mathsf{in}_{\mathsf{sim},u}$?. This input is allowed only once for each honest $u \in grp$. Its effect depends on the presence of dishonest users in grp:

- If a group member is dishonest (a priori not in $\mathcal{H}$ or adaptively corrupted) then the adversary can propose[12] a key which $\mathsf{TH}_{\mathcal{H}}$ takes and stores in $\mathsf{TH}_{\mathcal{H}}.key_{u,sid,grp}$. Thus, we do not require anything in this case.

- The same happens if two honest group members do not agree on the details of the previous group epoch. This consistency condition is very weak; e.g., we do not require that the old group was non-corrupted, or

[12]It is essential that passing a key is optional. Otherwise, no protocol providing PFS could be proven secure against adaptive corruptions: Consider a key establishment among two honest users u and v such that u finishes the protocol first and then gets corrupted before v can finish. Such a situation is unavoidable in our asynchronous systems. Since in the real world u and v would have agreed on a common key (u was corrupted only after the session establishment!), the simulator has to model this also in the ideal world. However, this cannot be simulated as we cannot provide $\mathsf{TH}_{\mathcal{H}}$ with the correct key to finish v's session: u's key was generated secretly by $\mathsf{TH}_{\mathcal{H}}$ and not leaked on corruption (it was previously deleted inside $\mathsf{TH}_{\mathcal{H}}$ to make PFS possible.)

that all non-corrupted members obtained the same key. Thus, some protocols for auxiliary key establishment might satisfy only accordingly restricted definitions. However, most of these protocols should be adaptable for the current model by adding explicit key-confirmation. Note that protocols secure against adaptive adversaries most likely require (implicitly) such a key-confirmation phase anyway.

- Otherwise, the system will produce a good key, i.e., one chosen randomly from $\{0,1\}^k$. Thus if u is the first group member for which the adversary inputs "finish" (i.e., $\mathsf{TH}_{\mathcal{H}}.ses_{v,sid,grp} \neq \mathsf{finished}$ for all $v \in grp$), then $\mathsf{TH}_{\mathcal{H}}$ selects a good key and stores it for all group members v in $\mathsf{TH}_{\mathcal{H}}.key_{v,sid,grp}$.

The selected key $\mathsf{TH}_{\mathcal{H}}.key_{u,sid,grp}$ is output to u, deleted internally ($\mathsf{TH}_{\mathcal{H}}.key_{u,sid,grp} \leftarrow \mathsf{undef}$) (this models forward secrecy), and the key establishment is finished for u ($\mathsf{TH}_{\mathcal{H}}.ses_{u,sid,grp} \leftarrow \mathsf{finished}$).

transition $\mathsf{in}_{\mathsf{sim},u}?$ (finish, $sid, grp, [key_{u,sim}]$)
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_{\mathsf{sim},u}?.cntr < tb)$;
  ignore if: $(ses_{u,sid,grp} \neq \mathsf{init})$;
  if $\mathsf{present}(key_{u,sim}) \wedge$
    $((\exists v \in grp : state_{v,v} = \mathsf{corrupted} \vee v \in \mathcal{A}) \vee$
    $(\exists v_0, v_1 \in grp : (ses_{v_0,sid,grp} \neq \mathsf{undef}) \wedge (ses_{v_1,sid,grp} \neq \mathsf{undef}) \wedge$
    $(prev_{v_0,sid,grp} \neq prev_{v_1,sid,grp})))$ **then**
    # Corrupted or inconsistent session so ...
    $key_{u,sid,grp} \leftarrow key_{u,sim}$;*# ... use session key provided by adversary*
  else if $(\forall\, v \in grp : ses_{v,sid,grp} \neq \mathsf{finished})$ **then**
    # First to finish (ideal) session
    $key \xleftarrow{\mathcal{R}} \{0,1\}^k$; *# Generate new (random) session key ...*
    for all $v \in grp$ **do**
      $key_{v,sid,grp} \leftarrow key$; *# ... and assign it to all parties*
    end for;
  end if;
  output: $\mathsf{out}_u!$ (key, $sid, grp, key_{u,sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);*# Give key to user ...*
  $key_{u,sid,grp} \leftarrow \mathsf{undef}$; *# ... and delete it locally to enable forward secrecy*
  $ses_{u,sid,grp} \leftarrow \mathsf{finished}$;
end transition

Corruptions. Corruptions are handled as sketched in Section 5.1.2. A priori, the users in $\mathcal{H}$ are uncorrupted. If $ct = \mathsf{static}$, any inputs on port $\mathsf{corrupt}_u?$ are ignored. If $ct = \mathsf{adaptive}$ then H can corrupt user $u \in \mathcal{H}$ at any time by entering do at $\mathsf{corrupt}_u?$. (We do not pose any limitation on the number of users that can be corrupted.) In this case, $\mathsf{TH}_{\mathcal{H}}$ extracts

all data corresponding to u with a call to $\mathsf{encode\_state}(u)$ and sends them to A. More precisely, $\mathsf{encode\_state}(u)$ maps to $(\{(u, v, state_{u,v}) \mid v \in \mathcal{M}\}, \{(sid, grp, ses_{u,sid,grp}, key_{u,sid,grp}, prev_{u,sid,grp}) \mid sid \in \mathcal{SID} \wedge grp \subseteq \mathcal{M} \wedge ses_{u,sid,grp} \neq \mathsf{undef}\})$.

The main part are all group keys that are already determined but not yet output to u (and thus not deleted). $\mathsf{TH}_{\mathcal{H}}$ records u's corruption ($\mathsf{TH}_{\mathcal{H}}.state_{u,u} \leftarrow \mathsf{corrupted}$), and from now on operates in transparent mode in respects to ports in_u? (routed to $\mathsf{out}_{\mathsf{sim},u}$!) and $\mathsf{in}_{\mathsf{sim},u}$? (routed to out_u!). Note that the transparent mode of the trusted host is slightly different to the transparent mode of standard systems as described in Section 5.1.2. For $\mathsf{TH}_{\mathcal{H}}$, the messages should *not* contain any port indicator: on the one hand, it is always implicitly clear from which input port a message comes or to which output port it has to go, and, on the other hand, explicit port indicators would make the construction of simulators difficult if not impossible.

transition $\mathsf{corrupt}_u$? (do)
  enabled if: $(ct = \mathsf{adaptive} \wedge state_{u,u} \neq \mathsf{corrupted})$;
  $state_{u,u} \leftarrow \mathsf{corrupted}$;
  output: $\mathsf{corOut}_{\mathsf{sim},u}$! (state, $\mathsf{encode\_state}(u)$), $\mathsf{corOut}_{\mathsf{sim},u}{}^{\triangleleft}$! (1);
end transition

transition in_u? (any_msg)
  enabled if: $(state_{u,u} = \mathsf{corrupted})$; # *Transparent mode*
  output: $\mathsf{out}_{\mathsf{sim},u}$! ($any\_msg$), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}$! (1);
end transition

transition $\mathsf{in}_{\mathsf{sim},u}$? ($any\_msg$)
  enabled if: $(state_{u,u} = \mathsf{corrupted})$; # *Transparent mode*
  output: out_u! (any_msg), $\mathsf{out}_u{}^{\triangleleft}$! (1);
end transition

$\diamond$

Let us briefly discuss, why the ideal system defined by Scheme 5.1 matches the notion and properties of a secure group key establishment as informally introduced in Chapter 2. This match as well as the preservation of integrity and confidentiality properties by simulation-based proofs allows us to deduce from a proof $Sys^{\mathsf{gke,real}} \geq_{\mathsf{sec}} Sys^{\mathsf{gke,ideal}}_{n,tb,ct}$ (with $Sys^{\mathsf{gke,real}}$ any real-world protocol) that $Sys^{\mathsf{gke,real}}$ inherits all properties from the ideal system and, therefore, is a secure group key establishment protocol. There are three questions to answer on the model: (1) does it provide an appropriate service, (2) does it capture necessary security properties, and (3) does it support the required group dynamics?

Service. It is obvious that the model provides the service "establishment of a common session key." Furthermore, the provided service is as general as possible. To capture all types of key establishment protocols, e.g., (centralized) key transport protocol as well as (distributed) key agreement protocols, the service is independent of particularities of protocols. In particular, it provides a uniformly distributed bit string as key which is the most general abstraction of a key. This is in sharp contrast, e.g., to the model provided by Bresson, Chevassut, Pointcheval, and Quisquater (2001) which is highly customized towards Diffie-Hellman-based key agreement protocols.

Security Properties. The primary security property to consider is *key secrecy*. For uncorrupted sessions — we cannot expect any secrecy for corrupted sessions — the session key is generated randomly and secretly by $\mathsf{TH}_{\mathcal{H}}$. Furthermore, the adversary will not learn any information on the key other than what is leaked by the users of the key-establishment protocol.[13] This is the strongest secrecy requirement imaginable and also implies the *semantic security* of the session key. The *freshness* of the group key is guaranteed as well since $\mathsf{TH}_{\mathcal{H}}$ generates the session keys randomly and independently from each other.

Except for corruptions, $\mathsf{TH}_{\mathcal{H}}$ returns a session key only to legitimate members of a group. Therefore, the ideal system provides *implicit key authentication*. Additionally, the ideal system ensures that all honest group members successfully establishing an uncorrupted session agree on the same key and know the involved identities. This holds for the following reasons: (1) $\mathsf{TH}_{\mathcal{H}}$ enforces the uniqueness of a session as identified by the pair (sid, grp), (2) this identifier implies a common agreement on the group membership of a session, and (3) $\mathsf{TH}_{\mathcal{H}}$ provides all parties with the same key. As a consequence, the ideal system also offers *mutual group key authentication*. Note that the ideal system does not ensure *explicit group key authentication* or guarantee *complete group key agreement*. However, this can be easily achieved with following change in **transition** $\mathsf{in}_{\mathsf{sim},u}?$ (finish, $sid, grp, [key_{u,sim}]$): replace the condition **ignore if:** $(ses_{u,sid,grp} \neq \mathsf{init})$ by **ignore if:** $(\nexists v \in grp : state_{v,v} = \mathsf{corrupted} \vee v \in \mathcal{A}) \wedge (\exists v \in grp : (ses_{v,sid,grp} = \mathsf{undef}))$, i.e., ensure that for uncorrupted sessions a finish message is handled only when everybody has initialized the session. In fact, as will be easy to verify, the protocol-variant presented later which is proven secure against adaptive adversaries turn out to be secure also in such a restricted model and, therefore, is a complete group key agreement protocol offering explicit group key authentication.

The ideal system captures both *PFS* and *KKA*. PFS is addressed by allowing participants to be corrupted: This leaks as part of the state, on the

[13] This leakage is modeled as flows from H to A and is unavoidable when we allow arbitrary modular composition with other protocols.

one hand, all their long-term keys as well as the keys and state of ongoing session but, on the other hand, no session keys from completed sessions. The possibility of KKA is inherent in the model as H can leak arbitrary information to A.

The model does not cover the special properties of (*contributory*) *key agreement* protocols, e.g., the guarantee of key freshness even in sessions with dishonest group members. While these properties are very useful in achieving flexible group key establishment protocols for dynamic peer groups, their security value per se is of only secondary importance and often not required. Therefore, these aspects are not captured in the main model for the sake of a broader model, i.e., one which captures key establishment in general. If desired, however, the model could be extended accordingly, e.g., by adding a restriction on the freshness of the key passed by the simulator in finish.

Dynamic groups. If we omit all the optional arguments $[sid', grp']$ in Scheme 5.1 we obtain the basic notion of group key establishment. In terms of Section 2.2, this corresponds to *initial key agreement*. As mentioned in that section, we also need to transform one or more existing groups $(sid_1, grp_1), (sid_2, grp_2), \ldots$ into a new group (sid, grp), i.e., we require AKA operations.

A group can grow by adding a subset grp_2 to a group (sid_1, grp_1) via input $(\mathsf{new}, sid, grp_1 \cup grp_2, sid_1, grp_1)$. If $|grp_2| = 1$, we have *member addition*, otherwise *mass join*. Note, however, that the current ideal system cannot directly express the transformation of two groups (sid_1, grp_1) and (sid_2, grp_2) into $(sid, grp \leftarrow grp_1 \cup grp_2)$, i.e., *group fusion*.

A group can also shrink by *excluding a member* u from a group (sid', grp') via input $(\mathsf{new}, sid, grp' \setminus \{u\}, sid', grp')$. In similar ways, we can also perform *mass leave* and *group division*.

Finally, note that the model ensures for all AKA operations *key independence* since $\mathsf{TH}_{\mathcal{H}}$ generates independent and random session keys. As there are no constraints on the membership in the new group grp' related to the previous group grp, we also obtain *policy independence*.

Intuitively, member exclusion is a problematic operation: If the to-be-excluded group member u was corrupted in a previous epoch (sid', grp'), we do not have any guarantee about the outcome of that epoch, the resulting keys might be arbitrary[14] and unlikely to be of much help for forming a new group. However, in the case of adaptive adversaries, corruption of u might have happened only *after* the formation of (sid', grp'). Therefore, the remaining members might benefit from reusing the consistent result from

[14]Note that for such corrupted epochs neither a successful explicit group key confirmation nor the (apparent) correct functioning of services depending on the group key guarantee consistency!

$(\mathsf{new}, sid', grp')$. Of course, protocols have to deal with potential inconsistencies of prior sessions, e.g., by adding explicit key-confirmation as previously mentioned when describing the transition new. Furthermore, even in the case of excluding statically corrupted group members, one should keep in mind that corruption does not necessarily mean destructive interference with the protocol. Therefore, an (optimistic) approach of AKA protocols makes sense. If implied checks indicate inconsistency of the prior epoch,[15] the protocol can always resort to an IKA.

In general it seems more difficult to prove security of AKA protocols in a model with adaptive corruptions, and actually we can prove our AKA protocols in the static model only. The reason is that, on the one hand, in order to utilize the result of previous protocol runs the machines have to store some information from those runs. On the other hand, we require that if a group (sid', grp') was honest at the time all users completed the protocol then the secrets for that run will never be given to the adversary, even if all members of grp' are corrupted afterwards. This would require a forward-secure state at group members, a property currently not provided by any group key protocol.[16] This is a useful property per se, but also practically needed in simulatability proofs: The information A has observed fully determines the correct key key' for run (sid', grp') (e.g., A sees all g^{x_i}, which determine the correct key $key' = g^{x_1 \cdots x_n}$). If no member of grp' is corrupted then $\mathsf{TH}_{\mathcal{H}}$ outputs a random key key'' instead of key'. Now assume A corrupts some $u \in grp'$. If the state of M_u contains enough secrets to let A check whether a certain key is correct, we are in trouble: we must consider the case where H gets all information from A to do this test. In the real system the key received from the system will pass this test, while in the ideal system this will most likely not be the case. Thus the views of H will be different. This problem is typically avoided by deleting all information from all M_u regarding (sid', grp') that would allow to test correctness before any user outputs the key. However, this more or less seems to exclude efficient AKA protocols.

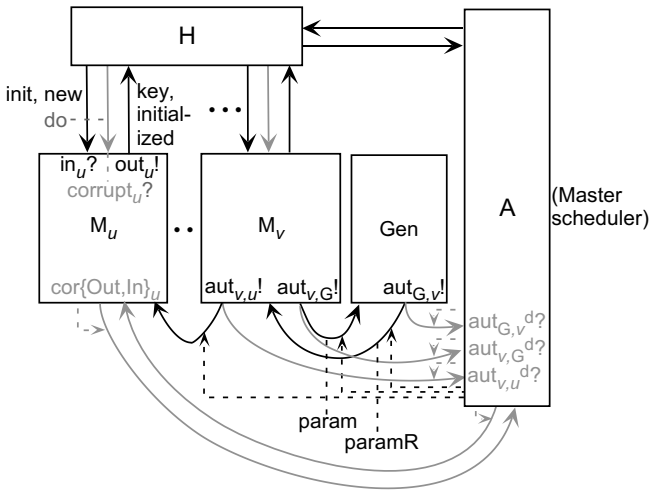
5.3 Real System for Group Key Establishment

We now consider the security of concrete group key establishment or, more precisely, group key agreement protocols. While some group key agreement protocols from the literature turn out to be secure in a simulatability sense, none does so against adaptive corruptions. We show how to extend them

[15] Such detection of inconsistencies might actually also serve as an additional deterrence for users to misbehave.

[16] Note that Bresson et al. (2001) prove the security of their AKA protocol only against *weakly* adaptive adversaries which do *not* get session-specific state. Due to that they do not have to solve above-mentioned problem.

Figure 5.3 Sketch of the real system. Derived parts are shown in gray. Scheduling is shown only for newly introduced ports.



to achieve adaptive security. Both of the following protocols presuppose authenticated connections.

As the basis of the real system, we take the protocol *IKA.1* presented in Section 4.2.1. (IKA.2 and any other protocol belonging to the family of natural DH extensions should work in exactly the same way.)

For the non-adaptive case ($ct = \mathsf{static}$) the protocol is identical to IKA.1 from Section 4.2.1 with two exceptions: (1) we explicitly use identifiers in messages and perform tests on their receipt as outlined in Section 4.1, and (2) instead of taking the Group Diffie-Hellman key directly, we derive a key using a universal hash-function h similar to Shoup (1999): This is required to get uniformly distributed random bit-strings as keys as mandated by the model, i.e., the ideal system.

For adaptive security ($ct = \mathsf{adaptive}$), we ensure that all secrets have been erased before the first key is output (following Shoup (1999) for the 2-party case). As long as we use the authenticated channels only, without additional signatures, this means a synchronization based on confirmation messages between all pairs of participants.

Scheme 5.2 (Real System for Group Key Establishment $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$)

Let $n \in \mathbb{N}$ be the number of intended participants and $\mathcal{M} := \{1, \ldots, n\}$. Similar to the trusted host, we parameterize the protocol with tb, the maximum number of transitions per port, and $ct \in \{\mathsf{adaptive}, \mathsf{static}\}$ depending on whether it has to deal with adaptive adversaries or not. The system $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$ — see Figure 5.3 for an overview — is defined by the following intended structure (M^*, S^*) and channel model. The actual system is derived

as a standard cryptographic system as defined in Section 5.1.2.

The specified ports S^* are the same as in the ideal system, i.e., those connecting user machines M to H in Figure 5.3. The intended machines are $M^* = \{\mathsf{M}_u^* \mid u \in \mathcal{M}\} \cup \{\mathsf{Gen}\}$. Their ports are $\mathsf{ports}(\mathsf{M}_u^*) := \{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{out}_u{}^{\lhd}!\} \cup \{\mathsf{aut}_{v,u}?, \mathsf{aut}_{u,v}! \mid v \in \{\mathsf{G}\} \cup \mathcal{M} \setminus \{u\}\}$ and $\mathsf{ports}(\mathsf{Gen}) := \{\mathsf{aut}_{u,\mathsf{G}}?, \mathsf{aut}_{\mathsf{G},u}! \mid u \in \mathcal{M}\}$. All system-internal connections are labeled "authenticated". (Connections to H are secure.)

The machine Gen generates and distributes the system parameters. These parameters are generated using the *generation algorithm* genG. On input 1^k, this algorithm outputs a tuple (G, g, h) where G is a suitable cyclic group of order $|G|$,[17] g a generator of G and h a random element of a family $UHF_{G,k}$ of universal hash functions (Carter and Wegman 1979) with domain G and range $\{0,1\}^k$. Suitable means that the group operations are efficiently computable, $|G| \geq 2^{3k}$ and the Decisional Diffie-Hellman problem is assumed to be hard. For example, according to Lemma 3.4, $|G|$ should not contain any small prime factors. (See Chapter 3, in particular Sections 3.5 and 3.6, for more information on the Decisional Diffie-Hellman problem and universal hash functions.)

The machine Gen is incorruptible, i.e., it always correct. It contains variables $state \in \{\mathsf{undef}, \mathsf{init}\}$ and (G, g, h). Its single state-transition function is:

transition $\mathsf{aut}_{u,\mathsf{G}}?$ (param)
  enabled if: $(\mathsf{aut}_{u,\mathsf{G}}?.cntr < tb)$;
  if $(state = \mathsf{undef})$ **then**
    $(G, g, \mathsf{h}) \leftarrow \mathsf{genG}(1^k)$;
    $state \leftarrow \mathsf{init}$;
  end if
  output: $\mathsf{aut}_{\mathsf{G},u}!$ (paramR, G, g, h);
end transition

A machine M_u^* implements the group key establishment service for the corresponding user u. It contains the variables shown in Table 5.3. Its state-transition function is shown below.

transition $\mathsf{in}_u?$ (init) # *Trigger initialization*
  enabled if: $(state_u = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  $state_u \leftarrow \mathsf{wait}$;
  output: $\mathsf{aut}_{u,\mathsf{G}}!$ (param);
end transition

[17] The group order $|G|$ and its factorization is assumed to be public. However, for simplicity this is not explicitly coded it into genG's return.

Table 5.3 Variables in M_u^*

Name	Domain	Meaning	Init.
$(state_v)_{v\in\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by M_u^*.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(ses_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{upflow}, \mathsf{downflow}, \mathsf{confirm}, \mathsf{finished}\}$	State of a (potential) session.	undef
$(\mathcal{C}_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathcal{I} \mid \mathcal{I} \subseteq \mathcal{M}\}$	Records received session confirmations	$\emptyset$
$(key_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}\}$	Group key of a session.	undef
$(x_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\mathbb{Z}_{\lvert G\rvert} \cup \{\mathsf{undef}\}$	Individual secret key of a session.	undef
$(\mathsf{aut}_{v,u}?.cntr)_{v\in\{\mathsf{G}\}\cup\mathcal{H}\setminus\{u\}}$	$\mathbb{N}$	Activation counters	0

transition $\mathsf{aut}_{\mathsf{G},u}?$ (paramR, G', g', h') # *Get system parameters*
  enabled if: $(state_u = \mathsf{wait})$;
  $state_u \leftarrow \mathsf{init}$;
  $(G, g, \mathsf{h}) \leftarrow (G', g', \mathsf{h}')$;
  output: $\mathsf{out}_u!$ (initialized, u); $\mathsf{out}_u{}^{\triangleleft}!$ (1);
  for all $v \in \mathcal{M} \setminus \{u\}$ **do**
    output: $\mathsf{aut}_{u,v}!$ (initialized);
  end for
end transition

transition $\mathsf{aut}_{v,u}?$ (initialized) # *Notification for other machines*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  $state_v \leftarrow \mathsf{init}$;
  output: $\mathsf{out}_u!$ (initialized, v), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_u?$ (new, sid, grp) # *Start new session*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  ignore if:
    $(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_v \neq \mathsf{init}) \vee (ses_{sid,grp} \neq \mathsf{undef})$;
  $x_{sid,grp} \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$;

$ses_{sid,grp} \leftarrow \mathsf{upflow}$;
if $(u = grp[1])$ **then** # *u is the first member*
$m'_1 \leftarrow g$;
$m'_2 \leftarrow g^{x_{sid,grp}}$
output: $\mathsf{aut}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
$ses_{sid,grp} \leftarrow \mathsf{downflow}$;
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (up, sid, grp, msg) # *Upflow message arrives*
enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(ses_{sid,grp} \neq \mathsf{upflow}) \vee (v \neq grp[\mathsf{idx}(grp,u)-1]) \vee$
$(msg$ is not $(m_1, \ldots, m_{\mathsf{idx}(grp,u)})$ with $m_i \in G$ having maximal order);
$i \leftarrow \mathsf{idx}(grp, u)$; # *u's position in the group*
$m'_1 \leftarrow m_i$;
for $1 \leq j \leq \mathsf{min}(i, |grp| - 1)$ **do**
$m'_{j+1} \leftarrow m_j^{x_{sid,grp}}$
end for
if $(i < |grp|)$ **then**
output: $\mathsf{aut}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
$ses_{sid,grp} \leftarrow \mathsf{downflow}$;
else # $i = |grp|$, *i.e., u is the last member*
$key_{sid,grp} \leftarrow \mathsf{h}((m_{|grp|})^{x_{sid,grp}})$;
if $(ct = \mathsf{static})$ **then** # *For the static case we are done*
$ses_{sid,grp} \leftarrow \mathsf{finished}$;
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
else # *For the adaptive case wait first for the confirmation flows*
$ses_{sid,grp} \leftarrow \mathsf{confirm}$;
$\mathcal{C}_{sid,grp} \leftarrow \{u\}$;
$x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
end if
for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" to the group members*
output: $\mathsf{aut}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
end for
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (down, sid, grp, msg) # *Downflow message arrives*
enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(ses_{sid,grp} \neq \mathsf{downflow}) \vee (v \neq grp[|grp|]) \vee$
$(msg$ is not $(m_1, \ldots, m_{|grp|})$ with $m_i \in G$ having maximal order);
$i \leftarrow \mathsf{idx}(grp, u)$; # *u's position in the group*
$key_{sid,grp} \leftarrow \mathsf{h}((m_{|grp|+1-i})^{x_{sid,grp}})$;
if $(ct = \mathsf{static})$ **then** # *For the static case we are done*

$ses_{sid,grp} = \mathsf{finished}$;
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
else # *For the adaptive case, start confirmation*
$ses_{sid,grp} \leftarrow \mathsf{confirm}$;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{u, v\}$;
$x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" confirmation to group members*
output: $\mathsf{aut}_{u,v}!$ ($\mathsf{confirm}$, sid, grp);
end for
if ($\mathcal{C}_{sid,grp} = grp$) **then** # *We got* down *after all* confirm ...
$ses_{sid,grp} = \mathsf{finished}$; # *...so we are done: Give key to user ...*
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow \mathsf{undef}$; # *...and delete it locally*
end if
end if
end transition

transition $\mathsf{aut}_{v,u}?$ ($\mathsf{confirm}$, sid, grp) # *Confirmation message arrives*
enabled if: $(ct = \mathsf{adaptive}) \wedge (state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(v \notin grp \setminus \mathcal{C}_{sid,grp}) \vee (ses_{sid,grp} \notin \{\mathsf{downflow}, \mathsf{confirm}\})$;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{v\}$;
if $(\mathcal{C}_{sid,grp} = grp) \wedge (ses_{sid,grp} = \mathsf{confirm})$ **then** # *All* confirm *received ...*
$ses_{sid,grp} \leftarrow \mathsf{finished}$; # *...so we are done: Give key to user ...*
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow \mathsf{undef}$; # *...and delete it locally*
end if
end transition

◇

The derivation of the actual system from the intended structure is now made as defined in Section 5.1.2. For example, the ports $\mathsf{aut}_{u,v}!$ are duplicated and passed to the adversary on port $\mathsf{aut}^{\mathsf{d}}_{u,v}!$. Similarly, a corruption switches a machine into transparent mode. The corresponding complete specification of the real system can be found in Appendix B.

Remark: In Chapter 4, I argued that a nice feature of CLIQUES is the provision of *policy independence*, e.g., it is not enshrined in the protocol who is the group controller. Above modeling now forces implicitly a particular policy, i.e., the use of the standard order $\leq$ in $\mathbb{N}$. However, this is only to keep the formalization of the protocol simple. It should be clear from the following proof that an arbitrary epoch-specific total order on group members (which could even be constructed "on-the-fly") is sufficient.

5.4 Security of Real System

Theorem 5.1 (Security of Scheme 5.2) *For all* $n \in \mathbb{N}$ *and* $ct \in \{\mathsf{static}, \mathsf{adaptive}\}$

$$Sys^{\mathsf{gke,ika1}}_{n,tb,ct} \geq_{\mathsf{sec}} Sys^{\mathsf{gke,ideal}}_{n,tb,ct}$$

□

We prove Theorem 5.1 in several steps:

First, we define an interactive version of the n-party Diffie-Hellman decision problem, and show that it is hard provided the ordinary Diffie-Hellman decision problem is hard. We do this by defining two (computationally indistinguishable) machines, $\mathsf{GDH}^{(0)}_{n,mxkey}$ and $\mathsf{GDH}^{(1)}_{n,mxkey}$. The former computes keys as in the real protocol while the latter is idealized: It works like $\mathsf{GDH}^{(0)}_{n,mxkey}$, but instead of producing the correct key as $\mathsf{h}(g^{x_1 \ldots x_n})$ it produces some random bit string of the appropriate length.

Next, we rewrite the real system such that all partial Diffie-Hellman keys of all machines M_u are computed by a hypothetical joint submachine $\mathsf{GDH}^{(0)}_{n,mxkey}$. Thus, we separate the computational indistinguishability aspects from others like state keeping (e.g., to show the sufficiency of confirmation messages in handling adaptive adversaries.) By the composition theorem from Pfitzmann and Waidner (2001), we can replace this submachine by $\mathsf{GDH}^{(1)}_{n,mxkey}$.

Finally, we show that the resulting system is perfectly indistinguishable from the trusted host together with a suitable simulator.

5.4.1 Interactive Generalized Diffie-Hellman Problem

As mentioned in the introduction of this section, our goal is to abstract the computation of keys and, indirectly, the underlying number-theoretic problem in a clean way. This is achieved with the following machine and its two modes of operation determined by the parameter b:

Scheme 5.3 (Generalized Diffie-Hellman Machine $\mathsf{GDH}^{(b)}_{n,mxkey}$)

The machines $\mathsf{GDH}^{(b)}_{n,mxkey}$, for $b \in \{0,1\}$, are constructed as follows: n is the maximum number of members in any session, $mxkey$ is the maximum number of sessions. $\mathsf{GDH}^{(b)}_{n,mxkey}$ has ports $\{\mathsf{in}_{\mathsf{gdh}}?, \mathsf{out}_{\mathsf{gdh}}!, \mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!\}$, where in each transition triggered at $\mathsf{in}_{\mathsf{gdh}}?$ exactly one output is sent to $\mathsf{out}_{\mathsf{gdh}}!$ which is immediately scheduled. As a convention we will call such self-clocked request-reply pairs **remote procedure calls (RPC)** and replies to message type mt will always have message type mtR.

A machine $\mathsf{GDH}^{(b)}_{n,mxkey}$ handles the messages shown in Table 5.4 and contains the variables shown in Table 5.5. The state transition functions are defined in following rules:

Table 5.4 The message types and parameters handled by $\mathsf{GDH}^{(b)}_{n,mxkey}$.

Port	Type	Parameters	Meaning
$\mathsf{in}_{\mathsf{gdh}}?$	init	—	Get system parameters
$\mathsf{out}_{\mathsf{gdh}}!$	initR	G, g, h	Reply to above
$\mathsf{in}_{\mathsf{gdh}}?$	$\mathsf{getView}$	n'	Get GDH partial keys of a new session
$\mathsf{out}_{\mathsf{gdh}}!$	$\mathsf{getViewR}$	$i, \{(\beta, g^{\prod_{\beta_j=1} x_{i,j}}) \mid \beta \in I_{n_i} \setminus \{1^{n_i}\}\}$	Reply to above, i is the session reference identifier
$\mathsf{in}_{\mathsf{gdh}}?$	getKey	i	Get key of session i
$\mathsf{out}_{\mathsf{gdh}}!$	$\mathsf{getKeyR}$	z_i	Reply to above
$\mathsf{in}_{\mathsf{gdh}}?$	$\mathsf{getSecret}$	i	Get secret exponents of session i
$\mathsf{out}_{\mathsf{gdh}}!$	$\mathsf{getSecretR}$	$(x_{i,1}, \ldots, x_{i,n_i})$	Reply to above

Table 5.5 Variables in $\mathsf{GDH}^{(b)}_{n,mxkey}$

Name	Domain	Meaning	Init.
(G, g, h)	Range of $\mathsf{genG}(1^k)$	System parameters	
i	$\mathbb{N}$	Session counter	0
$(c_i)_{i \in \mathbb{N}}$	$\{\mathsf{undef}, \mathsf{init}, \mathsf{finished}, \mathsf{corrupted}\}$	Session status	undef
$(n_i)_{i \in \mathbb{N}}$	$\mathbb{N}$	Number of session participants	
$(x_{i,j})_{i,j \in \mathbb{N}}$	$\mathbb{Z}_{\lvert G \rvert}$	Secret exponents	
$(z_i)_{i \in \mathbb{N}}$	G	Session keys	
$\mathsf{in}_{\mathsf{gdh}}?.cntr$	$\mathbb{N}$	Activation counter	0

transition $\mathsf{in}_{\mathsf{gdh}}?$ (init)
  enabled if: $(i = 0)$;
  $(G, g, \mathsf{h}) \xleftarrow{\mathcal{R}} \mathsf{genG}(1^k)$;
  $i \leftarrow 1$;
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (initR, G, g, h), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdh}}?$ (getView, n')
  enabled if: $(1 \leq i \leq mxkey)$; # *Initialized & maxima not exceeded*
  ignore if: $\neg(2 \leq n' \leq n)$; # *Illegal number of participants*
  $c_i \leftarrow \mathsf{init}$;
  $n_i \leftarrow n'$;
  $(x_{i,1}, \ldots, x_{i,n_i}) \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}^{n_i}$;
  if $b = 0$ **then** # *Depending on type of machine ...*
    $z_i \leftarrow \mathsf{h}(g^{x_{i,1} \cdots x_{i,n_i}})$; # *... set real key ...*
  else
    $z_i \xleftarrow{\mathcal{R}} \{0,1\}^k$; # *... or random key.*
  end if
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getViewR, $i, \{(\beta, g^{\prod_{\beta_j=1} x_{i,j}}) \mid \beta \in I_{n_i} \setminus \{1^{n_i}\}\}$), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
  $i \leftarrow i + 1$;
end transition

transition $\mathsf{in}_{\mathsf{gdh}}?$ (getKey, i)
  ignore if: $(c_i \neq \mathsf{init})$; # *Session not yet initialized or already terminated*
  $c_i \leftarrow \mathsf{finished}$;
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getKeyR, z_i), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdh}}?$ (getSecret, i)
  ignore if: $(c_i \neq \mathsf{init})$; # *Session not yet initialized or already terminated*
  $c_i \leftarrow \mathsf{corrupted}$;
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getSecretR, $(x_{i,1}, \ldots, x_{i,n_i})$), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition

$\diamond$

Let me briefly motivate the transitions. The meaning of the init message should be clear: It causes the initialization of the machine and the generation of the system parameters. Using a getView message, a caller can then instantiate a particular instance of a GDH problem and retrieve all corresponding partial GDH keys. We will use this later to generate the messages exchanged in a session of the key establishment protocol. The purpose of getKey is to provide a key corresponding to the partial GDH keys returned by getView. Depending on the bit b, this will result in the correctly derived

key or an independent random bit-string of the appropriate length, respectively. Therefore, we can satisfy our goal of decoupling the actual session key from the messages in a key establishment session by setting $b = 1$. However, in sessions with dishonest group members, e.g., due to a corruption, this strategy will not work. In these cases, the protocol messages might contain elements of the group G other than the partial GDH keys. Even worse, we also cannot use the "fake" session key provided by getKey. The dishonest members, i.e., the adversary, can correctly derive the "real" session key from the GDH partial keys and the secret exponents. Therefore, the adversary would immediately detect the difference. This explains the existence of $\mathsf{getSecret}$. It provides us with all secret exponents and allows us to also handle corrupted sessions. Finally, note that for each session only either $\mathsf{getSecret}$ or getKey can be called successfully!

As we will show in the following lemma, views from the two machines $\mathsf{GDH}^{(b)}_{n,mxkey}$ are indistinguishable if the DGDH(n) assumption (and indirectly the DDH assumption) holds. Note that this does not immediately follow from the DGDH(n) assumption: The interactivity, in particular corruptions (modeled by calls to $\mathsf{getSecret}$), requires special attention.

Lemma 5.1 For any $n \geq 2$ and *mxkey* and any polynomial-time machine A it holds that

$$(1-1/\mathsf{poly}(k))\text{-DDH}(\text{c:}*;\text{g:m};\text{f:fct,nsprim}) \xRightarrow{\alpha' \geq 1-1/2^k;\ t' \leq (t+O(mxkey\, 2^n k^3))(O(n^2 k)/\alpha^2)} view^{(0)} \overset{c}{\approx} view^{(1)}$$

where $view^{(b)}$ denotes the view of A while interacting with $\mathsf{GDH}^{(b)}_{n,mxkey}$. □

Proof. Assume that there is an interactive machine $\mathsf{D_A}$ which can distinguish $view^{(0)}$ from $view^{(1)}$ with non-negligible advantage $\delta := \mathbf{Prob}[\mathsf{D_A}(view^{(b)}) = b :: b \xleftarrow{\mathcal{R}} \{0,1\}] - 0.5$.

Without loss of generality, we can assume that A always uses $n' = n$: We can always transform outputs for n into outputs for an $n' < n$ by virtually combining $x_{n'}, x_{n'+1}, \ldots, x_n$ into a single value $\prod_{j=n'}^{n} x_j$, i.e., we delete from $\{(\beta, g^{\prod_{\beta_j=1} x_{i,j}} \mid \beta \in I_{n'} \setminus \{1^{n'}\}\})$ all pairs where not all values β_j for $j = n', \ldots, n$ are equal, and for the remaining ones we replace β by $\beta_1 \ldots \beta_{n'}$. In the output generated on input $\mathsf{getSecret}$, we replace $x_{n'}$ by $\prod_{j=n'}^{n} x_j$ and omit all x_i with $i > n'$. It is easy to see that everything is consistent and correctly distributed ($\prod_{j=n'}^{n} x_j$ is statistically indistinguishable from a uniformly chosen $x \in \mathbb{Z}_{|G|}$; this follows from Lemma 3.1.)

Now the lemma follows from a *hybrid argument*: Let us define $mxkey+1$ hybrid machines $\mathsf{GDH}^{\{i\}}_{n,mxkey}$. The machine $\mathsf{GDH}^{\{i\}}_{n,mxkey}$ is built and behaves like $\mathsf{GDH}^{(1)}_{n,mxkey}$ but flips the bit $\mathsf{GDH}^{\{i\}}_{n,mxkey}.b$ to 0 before handling

the i-th $\mathsf{getView}$ request. Clearly, the extreme hybrids $\mathsf{GDH}^{\{1\}}_{n,mxkey}$ and $\mathsf{GDH}^{\{mxkey+1\}}_{n,mxkey}$ are identical to $\mathsf{GDH}^{(0)}_{n,mxkey}$ and $\mathsf{GDH}^{(1)}_{n,mxkey}$, respectively. Let δ_i be $\mathsf{D_A}$'s advantage of distinguishing $\mathsf{GDH}^{\{i\}}_{n,mxkey}$ from $\mathsf{GDH}^{\{i+1\}}_{n,mxkey}$.

Using A and $\mathsf{D_A}$ as a subroutine we can now construct a distinguisher D which distinguishes $GDH^{(0)}_{k,n}$ from $GDH^{(1)}_{k,n}$ (see the proof of Theorem 3.2 for the exact definition of these ensembles of random variables): Given a sample $GDH_{k,n} \leftarrow GDH^{(b)}_{k,n}$, D first picks $c \xleftarrow{\mathcal{R}} \{1,\ldots,mxkey\}$. Then it starts and interacts with A behaving like $\mathsf{GDH}^{\{c\}}_{n,mxkey}$ with the following exceptions:[18] When it receives an init query, it replaces G and g returned by $\mathsf{genG}(1^k)$ with the group and generator associated with $GDH_{k,n}$; in the c-th $\mathsf{getView}$ query it answers with $(\mathsf{getViewR}, c, GDH^{\text{Public}}_{k,n})$; on valid (i.e., $c_c \neq \mathsf{init}$) input (getKey, c) it returns $(\mathsf{getKeyR}, c, \mathsf{h}(GDH^{\text{Key}}_{k,n}))$; and on valid input $(\mathsf{getSecret}, c)$ it simply gives up (it cannot correctly answer that request), outputs bit $b' \xleftarrow{\mathcal{R}} \{0,1\}$ and halts. Finally, when A terminates with view $view_{\mathsf{A}}$ it outputs $b' \leftarrow \mathsf{D_A}(view_{\mathsf{A}})$ and halts.

Let $\mathsf{D}^{\{i\}}$ denote D with c chosen as i. Further, let bad_i be the event that a valid input $(\mathsf{getSecret}, i)$ occured, i.e., the event which makes $\mathsf{D}^{\{i\}}$ give up. Note that the distribution of G, g, h and exponents of DGDH-tuples produced by $\mathsf{D}^{\{i\}}$ is identical to the equivalent distribution in $GDH^{(b)}_{k,n}$ due to the well-behavior of genG. Therefore, if bad_i does not happen then $\mathsf{D}^{\{i\}}$ behaves exactly like A interacting with $\mathsf{GDH}^{\{c+b\}}_{n,mxkey}$.

Let the probability of D in guessing b correctly be written as

$$\begin{aligned}&\mathbf{Prob}[b' = b] = \\ &\sum_{i=1}^{mxkey} \mathbf{Prob}[c = i] \;\; (\mathbf{Prob}[b' = b | \mathsf{bad}_i \wedge c = i]\mathbf{Prob}[\mathsf{bad}_i] + \\ &\qquad\qquad\qquad\qquad \mathbf{Prob}[b' = b | \neg\mathsf{bad}_i \wedge c = i]\mathbf{Prob}[\neg\mathsf{bad}_i]).\end{aligned}$$

As $\mathsf{D}^{\{i\}}$ simulates A's environment perfectly up to a possible occurrence of bad_i, the probability of bad_i is the same for $\mathsf{D}^{\{i\}}$ as for views of A when operating in reality. Additionally, views of A from the i-th and the $i+1$-th hybrids conditioned on the occurrence of bad_i are identical in reality (without giving up) because the only difference, z_i, is not output. So $\mathsf{D_A}$ has to guess (as does $\mathsf{D}^{\{i\}}$), i.e.,

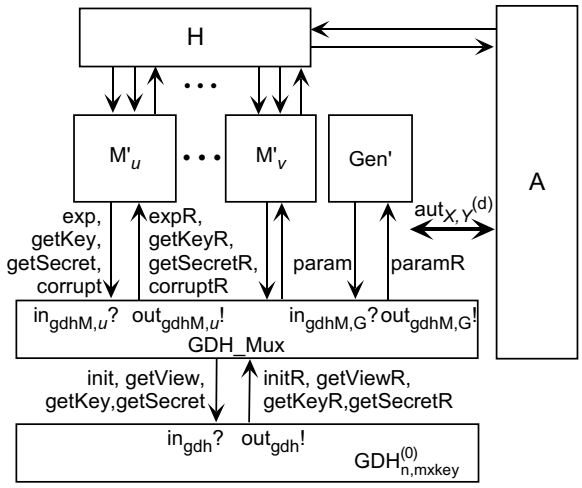
$$\mathbf{Prob}[\mathsf{D_A}(view_{\mathsf{GDH}^{\{i+b\}}_{n,mxkey}}) = b | \mathsf{bad}_i] = 0.5 = \mathbf{Prob}[\mathsf{D}^{\{i\}}(GDH_{k,n}) = b | \mathsf{bad}_i].$$

If bad_i does not occur, then $\mathsf{D}^{\{i\}}$ perfectly simulates $\mathsf{GDH}^{\{i+b\}}_{n,mxkey}$ so

$$\mathbf{Prob}[\mathsf{D_A}(view_{\mathsf{GDH}^{\{i+b\}}_{n,mxkey}}) = b | \neg\mathsf{bad}_i] = \mathbf{Prob}[\mathsf{D}^{\{i\}}(GDH_{k,n}) = b | \neg\mathsf{bad}_i].$$

[18] Note that the changes apply only for cases where the **require:** condition is fulfilled, otherwise the requests are rejected as usual.

Figure 5.4 Semi-real system. (Clocking of new components GDH_Mux and $\mathsf{GDH}^{(0)}_{n,mxkey}$ is RPC-style.)



By combining the previous two equations it follows that

$$\mathbf{Prob}[\mathsf{D_A}(view_{\mathsf{GDH}^{\{i+b\}}_{n,mxkey}}) = b] = \mathbf{Prob}[\mathsf{D}^{\{i\}}(GDH_{k,n}) = b]$$

and by this and the first equation it has to hold that

$$\begin{aligned}\mathbf{Prob}[b' = b] &= \frac{1}{mxkey}\sum_{i=1}^{mxkey}\mathbf{Prob}[\mathsf{D_A}(view_{\mathsf{GDH}^{\{i+b\}}_{n,mxkey}}) = b] \\ &= 1/2 + \frac{1}{mxkey}\sum_{i=1}^{mxkey}\delta_i.\end{aligned}$$

Using the equality $\sum_{i=1}^{mxkey}\delta_i = \delta$ and the hypothesis that the advantage δ of $\mathsf{D_A}$ is non-negligible, leads to an immediate contradiction of the $1/\mathsf{poly}(k)$-DGDH(n)(c:$*$; g:l; f:fct,nsprim) assumption. The lemma then follows immediately from this contradiction and the Lemmas 3.1 and 3.2. ■

5.4.2 Real System Rewritten with Interactive Diffie-Hellman Machine

We now rewrite the real system so that it uses $\mathsf{GDH}^{(0)}_{n,mxkey}$. We do this via a multiplexer GDH_Mux which maps group names, indices u, etc., of the individual modified machines M'_u to the simple sequence numbers of $\mathsf{GDH}^{(0)}_{n,mxkey}$, and distributes the parts of views to the machines as they

need them. Essentially, this rewriting shows that the real system only uses Diffie-Hellman keys in the proper way captured in $\mathsf{GDH}^{(0)}_{n,mxkey}$, i.e., never outputting both a key and a secret, and that active attacks (where the machines raise adversary-chosen elements to secret powers) do not make a difference. The situation is summarized in Figure 5.4. More precisely, the system is defined as follows:

Scheme 5.4 (Semi-real system $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$)

The structures of the semi-real system $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$ contain machines M'_u for all $u \in \mathcal{H}$, Gen', $\mathsf{GDH\_Mux}$, and $\mathsf{GDH}^{(0)}_{n,mxkey}$, where $mxkey$ can be upper bounded according to the runtime of M'_u, i.e., tb, as $n * tb$.[19]

M'_u and Gen' are identical to the corresponding M_u and Gen from scheme $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$ except that all operations on Diffie-Hellman keys are offloaded to $\mathsf{GDH\_Mux}$ (see Figure 5.7 for the message interface of $\mathsf{GDH\_Mux}$ towards these machines). Gen' gets additional ports $\{\mathsf{in}_{\mathsf{gdhM,G}}!, \mathsf{in}_{\mathsf{gdhM,G}}{}^{\triangleleft}!, \mathsf{out}_{\mathsf{gdhM,G}}?\}$. It uses them to get the system parameters by replacing the call to genG with a remote procedure call to param at $\mathsf{GDH\_Mux}$. M'_u has the same variables as M_u. They also have the same meaning except that the domain of $\mathsf{M}'_u.x_{sid,grp}$ is extended with a distinct value exists and the domain of $\mathsf{M}'_u.key_{sid,grp}$ by G. M'_u has additional ports $\{\mathsf{in}_{\mathsf{gdhM},u}!, \mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!, \mathsf{out}_{\mathsf{gdhM},u}?\}$ to communicate with $\mathsf{GDH\_Mux}$ via remote procedure calls. The cryptographic actions are changed as defined in Table 5.6. Additionally, on input $\mathsf{corrupt}_u?$ (do), M'_u first outputs $\mathsf{in}_{\mathsf{gdhM},u}!$ (corrupt) and waits for the response $\mathsf{corruptR}$. (And after the corruption, the forwarding only refers to the original ports of M_u.) The corresponding complete specification of Gen' and M'_u can be found in Appendix B.

$\mathsf{GDH\_Mux}$ has ports $\{\mathsf{in}_{\mathsf{gdh}}!, \mathsf{out}_{\mathsf{gdh}}?, \mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!\} \cup \{\mathsf{in}_{\mathsf{gdhM},u}?, \mathsf{out}_{\mathsf{gdhM},u}!, \mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}! \mid u \in \mathcal{M} \cup \{\mathsf{G}\}\}$. At its "upper" ports, it handles the message types shown in Table 5.7. All of them are of the remote procedure call type, i.e., responses are immediately scheduled. The $\mathsf{GDH\_Mux}$ de-multiplexes requests to and from $\mathsf{GDH}^{(0)}_{n,mxkey}$ and shields $\mathsf{GDH}^{(0)}_{n,mxkey}$ from illegal requests, i.e., $\mathsf{GDH}^{(0)}_{n,mxkey}$ is asked at most one of $\mathsf{getSecret}$ and getKey for a given session, and handles corruptions. In the **require:**-clauses we collect the pre-conditions under which $\mathsf{GDH\_Mux}$ will get the desired correct answers from $\mathsf{GDH}^{(0)}_{n,mxkey}$; we will show below that they are always fulfilled in the overall semi-real system.

The variables of $\mathsf{GDH\_Mux}$ are shown in Table 5.8. Below follows the state transition functions of $\mathsf{GDH\_Mux}$. Note that requests to $\mathsf{in}_{\mathsf{gdh}}?$ are

[19] This bound is of course overly conservative in practice. To get a considerably improved concrete security without much loss of generality, one could parameterize the model with additional bounds on the number of new requests and on the maximum size of a group. The changes throughout the model and proof would be cumbersome yet straightforward.

Table 5.6 Changed elementary actions in the semi-real machines M'_u

Elementary action	Replaced by
$x_{sid,grp} \xleftarrow{\mathcal{R}} \mathbb{Z}_{\lvert G\rvert}$	$x_{sid,grp} \leftarrow \mathsf{exists}$.
$m^* \leftarrow m^{x_{sid,grp}}$	Output $\mathsf{in}_{\mathsf{gdhM},u}!$ (exp, sid, grp, m) and use the answer as m^*.
$key_{sid,grp} \leftarrow \mathsf{h}(m^{x_{sid,grp}})$	$key_{sid,grp} \leftarrow m$, i.e., delay key computation.
Output $key_{sid,grp}$ (when passing key to H)	Output $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), i.e., perform delayed key computation, and use the answer as $key_{sid,grp}$.
Output $key_{sid,grp}$ (during corruption)	If $key_{sid,grp} \neq \mathsf{undef}$ (key computed but not yet erased) output $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$) and use the answer as $key_{sid,grp}$.
Output $x_{sid,grp}$ (during corruption)	If $x_{sid,grp} = \mathsf{exists}$ (secret generated but not yet erased) output $\mathsf{in}_{\mathsf{gdhM},u}!$ ($\mathsf{getSecret}$, sid, grp) and use the answer as $x_{sid,grp}$.

Table 5.7 Messages at "Upper" ports of $\mathsf{GDH\_Mux}$ where u ranges over $\mathcal{M}$.

Port	Type	Parameters	Meaning
$\mathsf{in}_{\mathsf{gdhM},\mathsf{G}}?$	param	—	Get system parameters
$\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}!$	paramR	G, g, h	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	$\mathsf{corrupt}$	—	Corruption
$\mathsf{out}_{\mathsf{gdhM},u}!$	$\mathsf{corruptR}$	—	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	exp	sid, grp, γ	Exponentiate γ with secret for u in this session. Limited to the computation of partial keys!
$\mathsf{out}_{\mathsf{gdhM},u}!$	expR	γ^{x_u}	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	getKey	sid, grp, γ	Get derived key matching final partial key γ
$\mathsf{out}_{\mathsf{gdhM},u}!$	$\mathsf{getKeyR}$	K	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	$\mathsf{getSecret}$	sid, grp	Get secret of this session (to hand it over during corruption)
$\mathsf{out}_{\mathsf{gdhM},u}!$	$\mathsf{getSecretR}$	x_u	Reply to above

Table 5.8 Variables in GDH_Mux

Variables	Domain	Meaning	Init.
$(i_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\mathbb{N}$	Index used for this session with $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(corr_u)_{u \in \mathcal{M}}$	$\{\mathsf{true}, \mathsf{false}\}$	Corrupted machine?	true iff $u \in \mathcal{M} \setminus \mathcal{H}$
$(ses_{u,sid,grp})_{u \in \mathcal{M}, sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{\mathsf{undef}, \mathsf{finished}, \mathsf{corrupted}\}$	Session status related to u	undef
$(key_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}\}$	Session key from $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(view_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	View of a session	undef
$(secrets_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	Secrets of a session	undef
$(\mathsf{in}_{\mathsf{gdhM},u}?.cntr)_{u \in \in \mathcal{M} \cup \{\mathsf{G}\}\}}$ $\mathsf{out}_{\mathsf{gdh}}?.cntr$	$\mathbb{N}$	Activation counters	0

remote procedure calls immediately answered by $\mathsf{GDH}^{(b)}_{n,mxkey}$. Therefore, we do not define special wait-states where GDH_Mux waits for these answers, but treat them within the surrounding transitions. We further assume that the corresponding input port $\mathsf{out_{gdh}}$? is enabled only for a single outstanding reply. For an n-bit string β and $1 \leq i \leq n$, let $\mathsf{bit}(\beta, i)$ denote the i-th bit in β and $\mathsf{setbit}(\beta, i)$ denote that the i-th bit in β is set to one. Furthermore, let "$\beta :: \mathsf{predicate}(\beta)$" means "all β such that predicate predicate holds".

transition $\mathsf{in_{gdhM,G}}$? (param)
  output: $\mathsf{in_{gdh}}$! (init), $\mathsf{in_{gdh}}^{\triangleleft}$! (1);
  input: $\mathsf{out_{gdh}}$? (initR, G, g, h);
  output: $\mathsf{out_{gdhM,G}}$! (paramR, G, g, h), $\mathsf{out_{gdhM,G}}^{\triangleleft}$! (1);
end transition

transition $\mathsf{in_{gdhM,}}_u$? (exp, sid, grp, γ)
  require: $(u \in grp) \ \wedge \ ((i_{sid,grp} = \mathsf{undef})$
    $\vee \ ((\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted}) \wedge (key_{sid,grp} = \mathsf{undef}))$
    $\vee \ ((\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted}) \wedge (\exists \beta : (\beta, \gamma) \in view_{sid,grp} \wedge$
    $\mathsf{bit}(\beta, \mathsf{idx}(grp, u)) = 0 \wedge \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) \neq 1^{|grp|})));$
    # A legitimate caller and either session is completely undefined or ses-
    # sion is corrupted but key is not yet divulged or session is uncorrupted
    # and query is for one of "our" partial keys.
  if $(i_{sid,grp} = \mathsf{undef})$ **then** *# New session*
    output: $\mathsf{in_{gdh}}$! (getView, $|grp|$), $\mathsf{in_{gdh}}^{\triangleleft}$! (1);
    input: $\mathsf{out_{gdh}}$? (getViewR, $i, view$);
    $i_{sid,grp} \leftarrow i$; $view_{sid,grp} \leftarrow view$
    for all $(v :: corr_v = \mathsf{true})$ **do** $ses_{v,sid,grp} \leftarrow \mathsf{corrupted}$; **end for**
  end if
  if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** *# Session uncorrupted*
    $\beta' \leftarrow \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) :: (\beta, \gamma) \in view_{sid,grp}$; *# Index of exponentiation*
    output: $\mathsf{out_{gdhM,}}_u$! (expR, $\gamma' :: (\beta', \gamma') \in view_{sid,grp}$), $\mathsf{out_{gdhM,}}_u{}^{\triangleleft}$! (1);
  else *# Group contains a corrupted participant*
    if $(secrets_{sid,grp} = \mathsf{undef})$ **then** *# Secrets not yet known*
      output: $\mathsf{in_{gdh}}$! (getSecret, $i_{sid,grp}$), $\mathsf{in_{gdh}}^{\triangleleft}$! (1);
      input: $\mathsf{out_{gdh}}$? (getSecretR, $secrets$);
      $secrets_{sid,grp} \leftarrow secrets$;
    end if
    output: $\mathsf{out_{gdhM,}}_u$! (expR, $\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}}$); $\mathsf{out_{gdhM,}}_u{}^{\triangleleft}$! (1);
  end if
end transition

transition $\mathsf{in_{gdhM,}}_u$? (getKey, sid, grp, γ)

require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}) \wedge$
$(((\exists \beta : (\beta, \gamma) \in view_{sid,grp}) \wedge (\mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) = 1^{|grp|})) \vee$
$((key_{sid,grp} = \mathsf{undef}) \wedge (\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted})))$;
 # A legitimate caller of an initialized but unfinished session either ask-
 # ing for a correct key or being corrupted without somebody having
 # asked for the ideal key before
if $key_{sid,grp} \neq \mathsf{undef}$ **then** *# (Ideal) key already defined...*
 # ...so just return this key
$ses_{u,sid,grp} \leftarrow \mathsf{finished}$;
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
else *# (Ideal) key does not yet exist and ...*
if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** *# ...uncorrupted session*
output: $\mathsf{in}_{\mathsf{gdh}}!$ (getKey, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdh}}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
$ses_{u,sid,grp} \leftarrow \mathsf{finished}$; *# Mark only uncorrupted sessions as finished!*
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
else *# Group contains corrupted participants and (ideal) key undefined*
if $(secrets_{sid,grp} = \mathsf{undef})$ **then** *# Secrets not yet known*
output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, $secrets$);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $\mathsf{h}(\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}})$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end if
end if
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (corrupt)
$corr_u \leftarrow \mathsf{true}$;
for all $(sid, grp :: (u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}))$
do
$ses_{u,sid,grp} \leftarrow \mathsf{corrupted}$; *# Mark only locally unfinished sessions*
end for
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (corruptR), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (getSecret, sid, grp)
require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} = \mathsf{corrupted}) \wedge$
$(key_{sid,grp} = \mathsf{undef})$;
 # A legitimate caller of a started session and we are corrupted but the
 # key has not been exposed
if $(secrets_{sid,grp} = \mathsf{undef})$ **then** *# Secrets not yet known*
output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);

input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, *secrets*);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getSecretR, $secrets_{sid,grp,\mathsf{idx}(grp,u)}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition ◇

The following lemma shows that we safely replace the real system by the semi-real system.

Lemma 5.2

$$Sys^{\mathsf{gke,ika1}}_{n,tb,ct} \geq_{\mathsf{sec}} Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$$

□

Proof. Our goal is to show that the input-output behavior of the two systems is identical.

The biggest difference, of course, is the different number of machines in both systems. However, the existence of the sub-machines GDH_Mux and $\mathsf{GDH}^{(0)}_{n,mxkey}$ is hidden. The self-clocking and the use of secure connections for remote procedure calls in $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$ ensures that the system control the scheduling for the whole duration of information flows through (honest) machines from H to A (and vice versa) and makes these flows externally visible as single atomic actions identical to $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$. This is also not violated by corruptions since the transparent mode does not leak any information on the existence of sub-machines.

Furthermore, it is easy to see that we mainly have to focus on the deterministic aspects. The only probabilistic actions of honest machines are the generation of the parameters and of the secret exponents, and they are chosen in both systems randomly as well as independently from the same distribution. The fact that the exponents are chosen in $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$ by a submachine and also not at the same points in time as in $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$ does not matter. As argued above, the submachine is hidden. Additionally, the behavior of honest machines does not directly depend on these random choices. Due to this and the following argumentation on the deterministic behavior, externally visible events which are causally related to the generation of secret exponents are consistent with their corresponding events in $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$.

To see that the deterministic behavior in $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$ is consistent with $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$, you should first observe that the external interface including **enabled if:** and **ignore if:** conditions is identical in both systems by construction. The next and most crucial step is to understand the **require:**-clauses in GDH_Mux. They ensure that, independent of the behavior of a calling M'_u:

1. $\mathsf{GDH}^{(0)}_{n,mxkey}$ is consistently called, e.g., for each session at most one of getSecret and getKey is sent to $\mathsf{GDH}^{(0)}_{n,mxkey}$; and
2. all partial GDH keys and session keys returned to a caller of getKey and exp are consistent with the provided γ's and previously delivered related values.[20]

However, these condition as well as the behavior of GDH_Mux should also not be too strict. They certainly have to ensure that:

3. calls by an uncorrupted M'_u, in particular to getKey, do not block on a **require:** condition;
4. GDH_Mux provides an ideal key, i.e., one retrieved via getKey from $\mathsf{GDH}^{(0)}_{n,mxkey}$, for sessions where no group member is corrupted at the point of the first getKey;[21] and
5. "corrupted" keys, i.e., keys where the provided γ does not match the expected value, are always computed correctly using exponentiations to the given base γ.

If these conditions are fulfilled, clearly, an uncorrupted M'_u performs (in conjunction with GDH_Mux) the same state updates and behaves (as visible externally) identical to the corresponding M_u. This holds also for corruptions since GDH_Mux provides the necessary information contained by M_u but lacking in M'_u, i.e., exponents or keys which are not yet deleted.

This leaves us, finally, with the task of verifying that all of above conditions are fulfilled by GDH_Mux and $\mathsf{GDH}^{(0)}_{n,mxkey}$. Foremost, observe that $\mathsf{GDH\_Mux}.i_{sid,grp}$ uniquely relates sessions from M'_u (using the parameters (sid, grp)) with GDH instances provided by $\mathsf{GDH}^{(0)}_{n,mxkey}$ and identified by i. Furthermore, the tests $(u \in grp)$ ensure that only legitimate users of session are serviced. Let us address now the different conditions in turn.

Condition 1: The validity of this condition holds for the following reasons. Since honest machines always call Gen′ before calling GDH_Mux, $\mathsf{GDH}^{(0)}_{n,mxkey}$ is appropriately initialized before GDH_Mux calls it. Additionally, GDH_Mux requests GDH instances correctly on demand. Furthermore, a call to getKey is remembered in $key_{sid,grp}$. This caching as well as the similar caching of secret exponents ensures that $\mathsf{GDH}^{(0)}_{n,mxkey}$ is asked only once per session for

[20] This does not necessarily mean that the session key must be identical to the key correctly derived from the GDH key corresponding to γ. It only requires that everybody asking for the session key and providing the same γ for a particular session will receive the same session key. This is not important here but will be crucial when constructing the simulator.

[21] Again, this is not directly relevant here but crucial when constructing the simulator.

either of them. Furthermore, the conditions ($key_{sid,grp}$ = undef) and the protocol flow guarantee that getSecret is never called after a call to getKey. Similarly, getSecret is only called for corrupted sessions, a case in which getKey is never called (note that sessions cannot be "uncorrupted").

Condition 2: This condition is trivially true since: (1) $\mathsf{GDH}^{(0)}_{n,mxkey}$ computes all keys based on real GDH partial keys and the correct key derivation, and (2) GDH_Mux tests for "incorrect" γ's, which cannot be found in the set of partial GDH keys, and computes the required value itself. (Note that this can only happen in case of a corruption and therefore calling getSecret is OK.)

Condition 3: Regarding this condition, note that honest machines M'_u pass always properly formated parameters. We first show that GDH_Mux will not block on any **require:** condition for uncorrupted sessions. exp is called by each machine at least once before getKey is called a single time. Furthermore, the parameters are always correct and consistent with the GDH partial keys obtained by $\mathsf{GDH}^{(0)}_{n,mxkey}$ due to the honesty of machines and by construction of the protocol. This ensures that all exponentiations can be served from the GDH partial keys and, on calls to getKey, the session is initialized but not terminated.

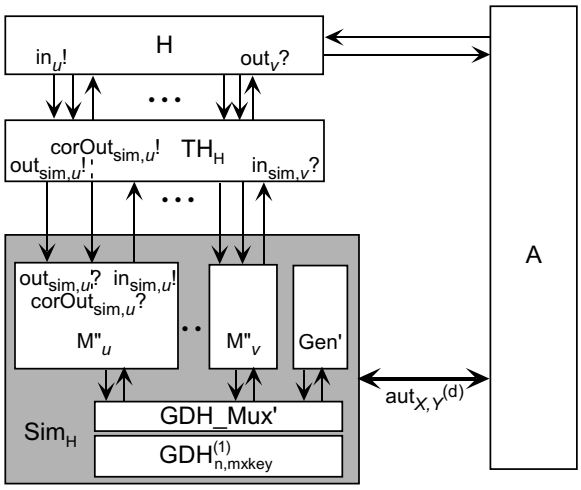
Similarly, for sessions where some group members are corrupted beforehand, e.g., due to static corruptions, $\mathsf{GDH}^{(0)}_{n,mxkey}$ is never asked for getKey. Therefore, $key_{sid,grp}$ remains undefined and exponentiations and key derivations do not block when the base γ does not match the partial GDH keys. This covers the case of static corruption (ct = static).

To cover the case of adaptive corruptions (ct = adaptive), it is sufficient to consider the following scenario: an uncorrupted group starts a session and later during the session a group member M'_u gets corrupted. First, note that dishonest machines never call GDH_Mux after encode_state(). Then, observe that, due to the confirmation flows, at the point of the first call to getKey no other member will call exp anymore for the same session. Let us now consider the following two possible cases:

- In the first case, the session gets first corrupted *before* the first call to getKey. In this case, GDH_Mux marks the session as corrupted and can safely retrieve the secret exponents $\mathsf{GDH}^{(0)}_{n,mxkey}$ by calling getSecret and serve (potentially inconsistent) queries to exp, getKey, and getSecret. (Note that getSecret or getKey might be called during corruption of M'_u or subsequent corruptions of other machines.)

- In the second case, the session gets corrupted only *after* the first call to getKey. Then, all exponents got previously deleted in all (then honest) machines M'_u. Furthermore, M'_u will call neither exp nor getSecret

Figure 5.5 Simulator



anymore. Since $key_{sid,grp}$ is cached and since, due to the confirmation flows, there exists an agreement among all machines on the γ required as input to getKey, GDH_Mux can serve all subsequent getKey queries.

Conditions 4 and 5: The fulfillment of these conditions should be immediately clear from the **require:** condition for getKey messages and the corresponding ways to derive the key. ■

5.4.3 Replacing $\mathsf{GDH}^{(0)}_{n,mxkey}$ by $\mathsf{GDH}^{(1)}_{n,mxkey}$

In the next step, we replace $\mathsf{GDH}^{(0)}_{n,mxkey}$ by $\mathsf{GDH}^{(1)}_{n,mxkey}$. The rest of the system remains as in Figure 5.4. We call the resulting semi-ideal system $Sys^{\mathsf{gke,ika1,si}}_{n,tb,ct}$. The composition theorem from Pfitzmann and Waidner (2001) and Lemma 5.1 immediately imply the following result:

Lemma 5.3

$$Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct} \geq_{\mathsf{sec}} Sys^{\mathsf{gke,ika1,si}}_{n,tb,ct}$$

□

5.4.4 Security of the System with $\mathsf{GDH}^{(1)}_{n,mxkey}$ with Respect to the Ideal System

We now define as a final step the simulator as a variant of the previous system.

Scheme 5.5 (Simulator for Scheme 5.2)

The overall structure of the simulator $\mathsf{Sim}_{\mathcal{H}}$ is shown in Figure 5.5.

The submachine Gen' of $\mathsf{Sim}_{\mathcal{H}}$ is identical to its counterpart in the semi-real and semi-ideal systems. Each submachine M''_u of $\mathsf{Sim}_{\mathcal{H}}$ has the same ports as its semi-real counterpart M'_u, except that its ports are connected to $\mathsf{TH}_{\mathcal{H}}$ and correspondingly renamed, i.e., $\mathsf{in}_u?$ becomes $\mathsf{out}_{\mathsf{sim},u}?$, $\mathsf{out}_u!$ becomes $\mathsf{in}_{\mathsf{sim},u}!$, and $\mathsf{corrupt}_u?$ becomes $\mathsf{corOut}_u?$ for all $u \in \mathcal{M}$. Furthermore, the domain of the variable $key_{sid,grp}$ is extended to the value ideal, a value which is distinct from $\{0,1\}^k$, G and undef and has an empty transport encoding. M''_u also has the same state-transition function as M'_u except for this renaming and the following changes:

- the message type key is everywhere replaced by finish. Note that a message of type finish with a parameter ideal as third parameter will result, due to above mentioned encoding properties, in a two-parameter message only (allowing $\mathsf{TH}_{\mathcal{H}}$ to choose an ideal key).
- M''_u expects a message $(\mathsf{state}, \mathit{state})$ instead of (do) on port $\mathsf{corOut}_{\mathsf{sim},u}?$. This corruption message is also passed to $\mathsf{in}_{\mathsf{gdhM},u}!$.

Submachine $\mathsf{GDH\_Mux}'$ is identical to $\mathsf{GDH\_Mux}$ except

- The domain of the variable $key_{sid,grp}$ is extended to the value ideal.
- Instead of calling getKey to $\mathsf{GDH}^{(1)}_{n,mxkey}$ in transition getKey it defines $key_{sid,grp}$ always as ideal: This will result in a finish message with no key and allow $\mathsf{TH}_{\mathcal{H}}$ to choose the key as desired in the absence of corrupted parties. (Note that due to the program logic no call to getKey from $\mathsf{encode\_state}()$ during a corruption will ever return ideal, so no adversary will be confused by an unexpected value ideal.)
- It expects the (ideal) state *state* of the corrupted party as a parameter of message $\mathsf{corrupt}$, extracts all session keys from *state* and assigns them to the corresponding variable $\mathsf{GDH\_Mux}'.key_{sid,grp}$.

The corresponding complete specification can be found in Appendix B. ◇

As the following lemma shows, the semi-ideal system is at least as secure as the ideal system.

Lemma 5.4

$$Sys^{\mathsf{gke,ika1,si}}_{n,tb,ct} \geq_{\mathsf{sec}} Sys^{\mathsf{gke,ideal}}_{n,tb,ct}$$

□

Proof. The proof of this lemma is quite similar to the proof of Lemma 5.2. The difference in the structure among the two systems is hidden for the

same reasons given in that lemma. Similar arguments hold regarding the probabilistic aspects, except that now the ideal key is generated by $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{GDH}^{(1)}_{n,mxkey}$, respectively. This leaves the deterministic aspects.

The same argumentation from Lemma 5.2 ensures also here that the sub-machines M''_u of $\mathsf{Sim}_{\mathcal{H}}$ interoperate consistently with $\mathsf{GDH\_Mux}'$, and $\mathsf{GDH}^{(1)}_{n,mxkey}$. The main question to answer is whether the interposition of $\mathsf{TH}_{\mathcal{H}}$ does not result in observable differences of behavior. It is easy to verify, that the messages exchanged on the connections between $\mathsf{TH}_{\mathcal{H}}$ and uncorrupted sub-machines in $\mathsf{Sim}_{\mathcal{H}}$ match the required message format. For corrupted sub-machines, the logic of the specification ensures that the corresponding "virtual sub-machine" in $\mathsf{TH}_{\mathcal{H}}$ is switched to transparent mode at the same time, too. Furthermore, it should be clear that $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$ keep session-specific state and the corruption status of users in lock-step. This means for most cases, $\mathsf{TH}_{\mathcal{H}}$ will safely route messages forth and back between M''_u and the corresponding user u. The only real question is whether a finish is accepted by $\mathsf{TH}_{\mathcal{H}}$ and results in appropriate assignment of session keys. However, this is ensured mainly due to the fulfillment of the Conditions 2, 4 and 5 mentioned and shown to hold in the proof of Lemma 5.2. For sessions which are already corrupted *before* the first getKey occurs, the "real" session key derived from the GDH keys are passed by $\mathsf{GDH\_Mux}'$ to M''_u. Furthermore, as the session is corrupted $\mathsf{TH}_{\mathcal{H}}$ will accept this key in a finish message. For sessions which get corrupted only *after* the first call to getKey, we are forced to finish the session with an ideal key. This works for following reasons: (1) no traces of the "real" session key exist anymore, (2) the corresponding session key returned by $\mathsf{GDH\_Mux}'$ to M''_u on a call to getKey will have the value ideal, and (3) the sending of a finish message with key ideal results, as noted in the description of the simulator, in a finish message with no third parameter as required by $\mathsf{TH}_{\mathcal{H}}$ to accept that session and to generate the concrete ideal key itself.

■

Proof. (of Theorem 5.1) The result follows immediately from Lemmas 5.2, 5.3 and 5.4, and the fact that "$\geq_{\mathsf{sec}}$" is transitive (Pfitzmann and Waidner 2001). Applying Remark 3.2 to Theorem 5.1 (the number *numpkey* of different partial keys visible to an adversary in Scheme 5.2 is $(n(n-1)/2)-1$) and observing that only Lemma 5.3 involves computational security, we achieve the following overall concrete security: Given a distinguisher which breaks Scheme 5.2 in time t and with success probability ϵ, we can break $(1-1/\mathsf{poly}(k))$-DDH(c:$*$; g:m; f:fct,nsprim) with a time complexity of at most $(t + O(mxkey\ n^3\ k^3))(O(n^2k)/\epsilon^2)$ and with overwhelming success probability. ■

Chapter 6

Conclusion and Outlook

In this final chapter I summarize my thesis. Furthermore, I give an outlook on open problems and possible research directions.

IN this thesis, I investigated the problem of key establishment in dynamic peer groups. Specifically, I considered the different services — namely, initial key agreement, key refresh and membership change operations — and the various required and desirable properties thereof. I presented CLIQUES, a family of protocols which provide all the services mentioned above and which are optimal or close to optimal in a number of metrics. The main drawback of the protocols are their relatively large round complexity for group merge operations. This deficiency is overcome in the STR protocols proposed in Kim et al. (2001) although at the cost of a considerably less rigorous proof and a reliance on the random oracle model (Bellare and Rogaway 1993). It is an interesting open question whether we can prove the STR protocols (or variations thereof) in the standard model while retaining the good round complexity.

By providing the first formal model of the required group key establishment services and a thorough investigation in the underlying cryptographic assumptions, I achieved the first formal proofs of group key establishment protocols. In particular, I proved that two variants of the initial key agreement operation of CLIQUES are secure against static and adaptive adversaries, respectively. These proofs hold only in a network with authenticated channels. However, using the compiler techniques from Bellare, Canetti, and Krawczyk (1998) it is possible to automatically construct protocols also secure in unauthenticated networks. This way we get a very modular and clean approach to the design of secure protocols. One drawback with this approach is that the resulting protocols are not optimal in their performance, i.e., the compiler adds a certain overhead of additional

messages which do not seem necessary. More efficient implementations based directly on digital signatures seem achievable as well, e.g., by applying optimizations similar to Canetti and Krawczyk (2001b) to the initial agreement and using the techniques from Bresson et al. (2001) for the authentication of auxiliary protocols, but require a corresponding careful analysis. Clearly, there are ample opportunities for future research.

The formal model also covers the auxiliary protocols. I only showed informally that CLIQUES AKA protocols are secure in the static model. However, the formal proof can be done by applying the same techniques as used in the proof of the CLIQUES IKA protocol. However, it seems to be an open research problem to find (efficient) AKA protocols which are secure against strong-adaptive[1] adversaries. The model is general enough to also cover key transport protocols. Therefore, it would be worthwhile to investigate the formal security of state-of-the-art group key transport protocol such as the tree-based scheme from Canetti et al. (1999) to further validate the model and to also get some provably secure group key transport protocols.

One of the advantages of the proposed formal model is the composition theorem which is part of the underlying computation and communication meta-model. An obvious application of the model for group key establishment and the composition theorem is in the modular definition of group services which rely on a group key exchange such as secure group communication. Proceeding on this way there is hope that one day there will be a modular and complete group communication *system* which is *provably secure in its entirety*, a thought which currently is beyond any imagination using past approaches. However, besides the provision of the necessary models and protocols for the other services of a group communication system there is one other gap to bridge. Reality is only (but inevitably) partially captured by the computation and communication meta-model. Therefore, one also has to carefully identify and analyze the remaining abstractions in this model, e.g., the absence of time or certain implicit properties of communication, and to consider how these abstractions can be securely implemented based on real hardware, operating systems and programming languages. For first steps in this directions I refer you to Adelsbach and Steiner (2002).

Finally, the classification of cryptographic assumptions related to discrete logarithms has its independent merits: It can serve as the basis of a standardization of results related to such assumptions and encourages their generalization to the most broadest case possible. Ideally, this could culminate in a large tool box which covers all known results and which supports cryptographic protocol designers in finding an assumption which is directly

[1] In a model with a weaker corruption model (Bresson, Chevassut, and Pointcheval 2001) where on corruption only long-term keys, but no short-term information such as random exponents, are leaked, CLIQUES AKA can withstand adaptive adversaries.

appropriate for their cryptographic applications, and in, potentially automatically, deriving the weakest possible equivalent assumption.

Bibliography

Abadi, Martín and Andrew D. Gordon. 1997. "Reasoning about cryptographic protocols in the spi calculus." *CONCUR'97: Concurrency Theory*, Volume 1243 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 59–73.

Abadi, Martín and Roger Needham. 1996. "Prudent Engineering Practice for Cryptographic Protocols." *IEEE Transactions on Software Engineering* 22 (1): 6–15 (January).

Abadi, Martín and Phillip Rogaway. 2002. "Reconciling two views of cryptography (The computational soundness of formal encryption)." *Journal of Cryptology* 15 (2): 103–127.

Adams, Carlisle, Mike Burmester, Yvo Desmedt, Mike Reiter, and Philip Zimmermann. 2000, November. "Which PKI (Public Key Infrastructure) is the right one?" Edited by Sushil Jajodia, *Proceedings of the 7th ACM Conference on Computer and Communications Security*. Athens, Greece: ACM Press, 98–101. Panel session.

Adelsbach, André. 1999, November. "Urheberschaftsbeweise mittels Watermarking und Registrierung." Diplomarbeit, Fachbereich 14, Informatik, der Universität des Saarlandes, Saarbrücken.

Adelsbach, André and Ahmad-Reza Sadeghi. 2001. "Zero-Knowledge Watermark Detection and Proof of Ownership." Edited by Ira S. Moskowitz, *Information Hiding—4th International Workshop, IHW 2001*, Volume 2137 of *Lecture Notes in Computer Science*. Pittsburgh, PA, USA: Springer-Verlag, Berlin Germany, 273–288.

Adelsbach, André and Michael Steiner (Editors). 2002, February. "Crypographic Semantics for Algebraic Model." Deliverable D08, EU Project IST-1999-11583 Malicious- and Accidental-Fault Tolerance for Internet Applications (MAFTIA).

Adelsbach, André, Birgit Pfitzmann, and Ahmad-Reza Sadeghi. 2000, October. "Proving Ownership of Digital Content." Edited by Andreas Pfitzmann, *Information Hiding—3rd International Workshop, IH'99*, Volume 1768 of *Lecture Notes in Computer Science*. Dresden, Germany: Springer-Verlag, Berlin Germany, 126–141.

Agarwal, Deborah A., Olivier Chevassut, Mary R. Thompson, and Gene Tsudik. 2001, July. "An Integrated Solution for Secure Group Communication in Wide-Area Networks." *2001 IEEE Symposium on Computers and Communications.*

Ajtai, Miklós and Cynthia Dwork. 1997, May. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence." *Proceedings of the 29th Annual Symposium on Theory Of Computing (STOC).* El Paso, TX, USA: ACM Press, 284–293.

Amir, Yair and Jonathan Stanton. 1998. "The Spread Wide Area Group Communication System." Technical report CNDS 98-4, The Center for Networking and Distributed Systems, John Hopkins University.

Amir, Yair, Giuseppe Ateniese, Damian Hasse, Yongdae Kim, Cristina Nita-Rotaru, Theo Schlossnagle, John Schultz, Jonathan Stanton, and Gene Tsudik. 2000, April. "Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments." *20th International Conference on Distributed Computing Systems (ICDCS'2000).* IEEE Computer Society Press.

Anderson, Ross J. 2001. *Security Engineering — A Guide to Building Dependable Distributed Systems.* John Wiley & Sons.

Anderson, Ross and Roger Needham. 1995. "Robustness principles for public key protocols." Edited by Don Coppersmith, *Advances in Cryptology – CRYPTO '95*, Volume 963 of *Lecture Notes in Computer Science.* International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Asokan, N. and Philip Ginzboorg. 2000. "Key-Agreement in Ad-hoc Networks." *Computer Communications* 23 (17): 1627–1637 (November).

Asokan, N., Hervé Debar, Michael Steiner, and Michael Waidner. 1999. "Authenticating Public Terminals." *Computer Networks* 31 (8): 861–870 (May).

Ateniese, Giuseppe, Michael Steiner, and Gene Tsudik. 2000. "New Multiparty Authentication Services and Key Agreement Protocols." *IEEE Journal on Selected Areas in Communications* 18 (4): 628–639 (April).

Babai, Laszlo and Endre Szemerédi. 1984. "On the complexity of matrix group problems." *Proceedings of the 25th Symposium on Foundations of Computer Science (FOCS).* IEEE Computer Society Press, 229–240.

Bach, Eric and Jeffrey Shallit. 1996. *Algorithmic Number Theory — Efficient Algorithms.* Volume I. Cambridge, USA: MIT Press. ISBN: 0-262-02405-5.

Ballardie, Anthony. 1996, May. "Scalable Multicast Key Distribution." Internet request for comment RFC 1949, Internet Engineering Task Force.

Becker, Klaus and Uta Wille. 1998, November. "Communication Complexity of Group Key Distribution." *Proceedings of the 5th ACM Conference on Computer and Communications Security*. San Francisco, California: ACM Press, 1–6.

Bellare, Mihir and Oded Goldreich. 1993. "On Defining Proofs of Knowledge." Edited by E.F. Brickell, *Advances in Cryptology – CRYPTO '92*, Volume 740 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Bellare, Mihir and Phillip Rogaway. 1993, November. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols." Edited by Victoria Ashby, *Proceedings of the 1st ACM Conference on Computer and Communications Security*. Fairfax, Virginia: ACM Press, 62–73.

———. 1994. "Entity Authentication and Key Distribution." Edited by Douglas R. Stinson, *Advances in Cryptology – CRYPTO '93*, Volume 773 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 232–249.

———. 1995a. "Optimal Asymmetric Encryption — How to encrypt with RSA." Edited by A. De Santis, *Advances in Cryptology – EUROCRYPT '94*, Volume 950 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 92–111. Final (revised) version appeared November 19, 1995. Available from http://www-cse.ucsd.edu/users/mihir/papers/oaep.html.

———. 1995b, May. "Provably Secure Session Key Distribution — The Three Party Case." *Proceedings of the 27th Annual Symposium on Theory of Computing (STOC)*. ACM Press, 57–66.

Bellare, Mihir, Ran Canetti, and Hugo Krawczyk. 1998, May. "A modular approach to the design and analysis of authentication and key exchange protocols." *Proceedings of the 30th Annual Symposium on Theory Of Computing (STOC)*. Dallas, TX, USA: ACM Press, 419–428.

Bellare, Mihir, David Pointcheval, and Phillip Rogaway. 2000. "Authenticated Key Exchange Secure Against Dictionary Attacks." In Preneel 2000, 139–155. Appeared also as Cryptology ePrint Archive Report 2000/014, 28 April, 2000.

Bengio, Samy, Gilles Brassard, Yvo G. Desmedt, Claude Goutier, and Jean-Jacques Quisquater. 1991. "Secure Implementation of Identification Systems." *Journal of Cryptology* 4 (3): 175–183.

Biham, Eli, Dan Boneh, and Omer Reingold. 1999. "Breaking Generalized Diffie-Hellman modulo a composite is no easier than factoring."

Information Processing Letters 70:83–87. Also appeared in Theory of Cryptography Library, Record 97-14, 1997.

Bird, Ray, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. 1993. "Systematic design of a family of attack-resistant authentication protocols." *IEEE Journal on Selected Areas in Communications* 11 (5): 679–693 (June).

Birman, Kenneth. 1996. *Building secure and reliable network applications.* Manning Publications Co. ISBN 1-884777-29-5.

Blake-Wilson, Simon and Alfred Menezes. 1998. "Entity Authentication and Authenticated Key Transport Protocols Emplyoing Asymmetric Techniques." Edited by Bruce Christianson, Bruno Crispo, Mark Lomas, and Michael Roe, *Security Protocols—5th International Workshop*, Volume 1361 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Germany, 137–158.

———. 1999. "Authenticated Diffie-Hellman key agreement protocols." *Fifth Annual Workshop on Selected Areas in Cryptography (SAC '98)*, Volume 1556 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Germany, 339–361.

Blake-Wilson, Simon, Don Johnson, and Alfred Menezes. 1997, December. "Key agreement protocols and their security analysis." *Cryptography and condig (IMA'97)*, Volume 1355 of *Lecture Notes in Computer Science.* 30–45.

Blum, Manuel. 1982. "Coin Flipping by Telephone: A Protocol for Solving Impossible Problems." *Proceedings of the 24th IEEE Computer Conference.* 133–137. See also *ACM SIGACT News*, Vol. 15, No. 1, 1983.

Blum, Manuel and Silvio Micali. 1984. "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits." *SIAM Journal on Computing* 13 (4): 850–864 (November).

Blum, Lenore, Manuel Blum, and Michael Shub. 1986. "A Simple Unpredictable Pseudo-Random Number Generator." *SIAM Journal on Computing* 15 (2): 364–383 (May).

Bolignano, Dominique. 1996, March. "An Approach to the Formal Verification of Cryptographic Protocols." Edited by Clifford Neuman, *Proceedings of the 3rd ACM Conference on Computer and Communications Security.* New Delhi, India: ACM Press, 106–118.

Boneh, Dan. 1998. "The Decision Diffie-Hellman problem." *Third International Algorithmic Number Theory Symposium (ANTS-III)*, Volume 1423 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Germany, 48–63.

———. 2000, October. Personal Communication.

———. 2001. "Simplified OAEP for the RSA and Rabin functions." In Kilian 2001, 275–291.

Boneh, Dan and Richard J. Lipton. 1996. "Algorithms for black box fields and their application to cryptography." In Koblitz 1996, 283–297.

Boneh, Dan and Igor Shparlinski. 2001. "Hard Core Bits for the Elliptic Curve Diffie-Hellman Secret." In Kilian 2001, 201–212.

Boneh, Dan and Ramarathnam Venkatesan. 1996. "Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes." In Koblitz 1996, 129–142.

Boyko, Victor, Philip MacKenzie, and Sarvar Patel. 2000. "Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman." In Preneel 2000, 156–171.

Brands, Stefan. 1993, March. "An Efficient Off-line Electronic Cash System Based On The Representation Problem." Technical Report CS-R9323, Centrum voor Wiskunde en Informatica.

———. 1994. "Untraceable Off-line Cash in Wallet with Observers." Edited by Douglas R. Stinson, *Advances in Cryptology – CRYPTO '93*, Volume 773 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 302–318.

Bresson, Emmanuel, Olivier Chevassut, and David Pointcheval. 2001. "Provably authenticated group Diffie-Hellman key exchange — the dynamic case." Edited by Colin Boyd, *Advances in Cryptology – ASIACRYPT '2001*, Lecture Notes in Computer Science. International Association for Cryptologic Research Gold Coast, Australia: Springer-Verlag, Berlin Germany, 290–309.

Bresson, Emmanuel, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. 2001, November. "Provably Authenticated Group Diffie-Hellman Key Exchange." Edited by Pierangela Samarati, *Proceedings of the 8th ACM Conference on Computer and Communications Security*. Philadelphia, PA, USA: ACM Press, 255–264.

Briscoe, Bob. 1999, November. "MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences." *First International Workshop on Networked Group Communication*. 301–320.

Burmester, Mike. 1994. "On the risk of opening distributed keys." Edited by Yvo G. Desmedt, *Advances in Cryptology – CRYPTO '94*, Volume 839 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 308–317.

Burmester, Mike and Yvo Desmedt. 1995. "A Secure and Efficient Conference Key Distribution System." Edited by A. De Santis, *Advances*

in Cryptology – EUROCRYPT '94, Volume 950 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 275–286. Final version of proceedings.

Burrows, Michael, Martín Abadi, and Roger Needham. 1990. "A Logic of Authentication." *ACM Transactions on Computer Systems* 8 (1): 18–36 (February).

Cachin, Christian, Klaus Kursawe, and Victor Shoup. 2000, July. "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography." *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*. ACM Portland, Oregon. Full version appeared as Cryptology ePrint Archive Report 2000/034 (2000/7/7).

Cachin, Christian, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. "Secure and Efficient Asynchronous Broadcast Protocols." In Kilian 2001, 524–541.

Camenisch, Jan, Ueli Maurer, and Markus Stadler. 1996, September. "Digital Payment Systems with Passive Anonymity-Revoking Trustees." Edited by E. Bertino, H. Kurth, G. Martella, and E. Montolivo, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS)*, Volume 1146 of *Lecture Notes in Computer Science*. Rome, Italy: Springer-Verlag, Berlin Germany, 33–43.

Canetti, Ran. 1997. "Towards realizing random oracles: Hash functions that hide all partial information." Edited by Burton S. Kaliski, Jr., *Advances in Cryptology – CRYPTO '97*, Volume 1294 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 455–469.

Canetti, Ran and Hugo Krawczyk. 2001a, May. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels." Report 2001/040, Cryptology ePrint Archive.

———. 2001b. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels." In Pfitzmann 2001, 451–472.

Canetti, Ran, John Friedlander, Sergei Konyagin, Michael Larsen, Daniel Lieman, and Igor Shparlinski. 2000. "On the statistical properties of Diffie-Hellman distributions." *Israel Journal of Mathemathics* 120 (March): 23–46.

Canetti, Ran, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. 1999, March. "Multicast Security: A Taxonomy and Some Efficient Constructions." *INFOCOMM'99*. 708–716.

Canetti, Ran, Oded Goldreich, and Shai Halevi. 1998, May. "The Random Oracle Methodology, Revisited." *Proceedings of the 30th Annual*

Symposium on Theory Of Computing (STOC). Dallas, TX, USA: ACM Press, 209–218.

Caronni, Germano, Marcel Waldvogel, Dan Sun, Nathalie Weiler, and Berhardt Plattner. 1999. "The VersaKey Framework: Versatile Group Key Management." *IEEE Journal on Selected Areas in Communications* 17 (9): 1614–1631 (September).

Carter, L. Lawrence and Mark N. Wegman. 1979. "Universal Classes of Hash Functions." *Journal of Computer and System Sciences* 18:143–154.

Chang, Isabella, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. 1999, March. "Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques." *Proceedings IEEE Infocomm'99*, Volume 2. 689–698.

Chaum, David. 1991. "Zero-Knowledge Undeniable Signatures." Edited by I.B. Damgard, *Advances in Cryptology – EUROCRYPT '90*, Volume 473 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 458–464.

Clark, John A. and Jeremy L. Jacob. 1997, November. "A survey of authentication protocol literature." Version 1.0, University of York, Department of Computer Science.

Coppersmith, Don and Igor Shparlinski. 2000. "On Polynomial Approximation of the Discrete Logarithm and the Diffie-Hellman Mapping." *Journal of Cryptology* 13 (3): 339–360 (March).

Cramer, Ronald and Victor Shoup. 1998. "A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack." Edited by Hugo Krawczyk, *Advances in Cryptology – CRYPTO '98*, Volume 1462 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 13–25.

Davida, George, Yair Frankel, Yiannis Tsiounis, and Moti Yung. 1997, February. "Anonymity Control in E-Cash Systems." *Proceedings of the First Conference on Financial Cryptography (FC '97)*, Volume 1318 of *Lecture Notes in Computer Science*. International Financial Cryptography Association (IFCA) Anguilla, British West Indies: Springer-Verlag, Berlin Germany, 1–16.

Diffie, Whitfield and Martin Hellman. 1976. "New Directions in Cryptography." *IEEE Transactions on Information Theory* IT-22 (6): 644–654 (November).

Dolev, Danny and Andrew C. Yao. 1983. "On the Security of Public Key

Protocols." *IEEE Transactions on Information Theory* 29 (2): 198–208.

Ellison, Carl and Bruce Schneier. 2000. "Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure." *Computer Security Journal* 16 (1): 1–7.

Even, Shimon, Oded Goldreich, and Adi Shamir. 1986. "On the security of Ping-Pong Protocols when implemented using RSA." Edited by Hugh C. Williams, *Advances in Cryptology – CRYPTO '85*, Volume 218 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 58–72.

Feige, Uriel, Amos Fiat, and Adi Shamir. 1987, May. "Zero-Knowledge Proofs of Identity." *Proceedings of the 19th Annual Symposium on Theory of Computing (STOC)*. New York, NY USA: ACM Press, 210–217.

Fischlin, Marc. 2000. "A Note on Security Proofs in the Generic Model." In Okamoto 2000, 458–469.

Frankel, Yair, Yiannis Tsiounis, and Moti Yung. 1996. ""Indirect Discourse Proofs": Achieving Fair Off-Line Cash (FOLC)." Edited by K. Kim and T. Matsumoto, *Advances in Cryptology – ASIACRYPT '96*, Volume 1163 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 286–300.

Freier, Alan O., Philip Kariton, and Paul C. Kocher. 1996. "The SSL Protocol: Version 3.0." Internet draft, Netscape Communications.

Fujisaki, Eiichiro, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. 2001. "RSA—OAEP is secure under the RSA Assumption." In Kilian 2001, 260–274.

Garland, Stephen J. and Nancy A. Lynch. 2000. "Using I/O Automata for Developing Distributed Systems." In *Foundations of Component-Based Systems*, edited by Gary T. Leavens and Murali Sitaraman, 285–312. Cambridge University Press.

Gennaro, Rosario. 2000. "An Improved Pseudo-random Generator Based on Discrete Log." Edited by Mihir Bellare, *Advances in Cryptology – CRYPTO '2000*, Volume 1880 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 469–481.

Gennaro, Rosario, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems." Edited by Jacques Stern, *Advances in Cryptology – EUROCRYPT '99*, Volume 1599 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 295–310.

Goldreich, Oded. 1998, February. Foundations of Cryptography (Fragments of a Book). Manuscript. Version 2.03, available from http://www.wisdom.weizmann.ac.il/~oded/frag.html.

Goldwasser, Shafi and Silvio Micali. 1984. "Probabilistic Encryption." *Journal of Computer Security* 28:270–299.

Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. 1989. "The Knowledge Complexity of Interactive Proof Systems." *SIAM Journal on Computing* 18 (1): 186–208.

Gong, Li. 1997. "Enclaves: Enabling Secure Collaboration over the Internet." *IEEE Journal on Selected Areas in Communications*, pp. 567–575.

Gong, Li, Roger Needham, and Raphael Yahalom. 1990, May. "Reasoning about Belief in Cryptographic Protocols." *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 234–248.

Gonzalez Vasco, Maria I. and Igor Shparlinski. 2000. "On the security of Diffie-Hellman bits." *Workshop on Cryptography and Computational Number Theory Singapore 1999*. Birkhäuser.

Gordon, Daniel M. 1993a. "Designing and Detecting Trapdoors for Discrete Log Cryptosystems." Edited by E.F. Brickell, *Advances in Cryptology – CRYPTO '92*, Volume 740 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 66–75.

——— 1993b. "Discrete logarithms in GF(p) using the number field sieve." *SIAM Journal on Discrete Mathematics* 6 (1): 124–138.

Gritzalis, Stefanos, Diomidis Spinellis, and Panagiotis Georgiadis. 1999. "Security Protocols over Open Networks and Distributed Systems: Formal Methods for their Analysis, Design, and Verification." *Computer Communications* 22 (8): 695–707 (May).

Günther, C.G. 1990. "An identity-based key-exchange protocol." Edited by J-J. Quisquater and J. Vandewalle, *Advances in Cryptology – EUROCRYPT '89*, Volume 434 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 29–37.

Handschuh, Helena, Yiannis Tsiounis, and Moti Yung. 1999, March. "Decision oracles are equivalent to matching oracles." Edited by H. Imai and Z. Zheng, *International Workshop on Practice and Theory in Public Key Cryptography '99 (PKC '99)*, Volume 1560 of *Lecture Notes in Computer Science*. Kamakura, Japan: Springer-Verlag, Berlin Germany.

Harney, Hugh and Carl Muckenhirn. 1997, July. "Group Key Management Protocol (GKMP) Architecture." Internet request for comment RFC 2094, Internet Engineering Task Force.

Håstad, Johan, Russell Impagliazzo, Leonid Levin, and Michael Luby. 1999. "A Pseudorandom generator from any one-way function." *SIAM Journal on Computing* 28 (4): 1364–1396. A preliminary version appeared in 21rst STOC, 1989.

Hutchinson, Andrew. 1995. "Group Security in Distributed Systems." Ph.D. diss., Philosophische Fakultät II der Universität Zürich, Zurich, Switzerland.

Ingemarsson, Ingemar, Donald T. Tang, and C. K. Wong. 1982. "A Conference Key Distribution System." *IEEE Transactions on Information Theory* 28 (5): 714–720 (September).

Janson, Phil, Gene Tsudik, and Moti Yung. 1997, April. "Scalability and Flexibility in Authentication Services: The KryptoKnight Approach." *IEEE INFOCOM'97*. Tokyo, Japan.

Just, Michael K. 1994, August. "Methods of Multi-Party Cryptographic Key Establishment." Master's thesis, Carleton University, Computer Science Department, Caleton University, Ottawa, Ontario.

Just, Mike and Serge Vaudenay. 1996. "Authenticated Multi-Party Key Agreement." Edited by K. Kim and T. Matsumoto, *Advances in Cryptology – ASIACRYPT '96*, Volume 1163 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 36–49.

Katz, Jonathan, Rafail Ostrovsky, and Moti Yung. 2001. "Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords." In Pfitzmann 2001, 473–492.

Kemmerer, Richard A. 1989. "Analyzing Encryption Protocols Using Formal Verification Techniques." *IEEE Journal on Selected Areas in Communications* 7 (4): 448–457 (May).

Kilian, Joe, ed. 2001. *Advances in Cryptology – CRYPTO '2001*. Volume 2139 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Kiltz, Eike. 2001. "A Tool Box of Cryptographic Functions related to the Diffie-Hellman Function." *Advances in Cryptology – INDOCRYPT '2001*, Volume 2247 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 339–350.

Kim, Yongdae, Daniele Mazzochi, and Gene Tsudik. 2002, May. Admission Control in Peer Groups. http://www-users.cs.umn.edu/~kyd/publications.html.

Kim, Yongdae, Adrian Perrig, and Gene Tsudik. 2000, November. "Simple and fault-tolerant key agreement for dynamic collaborative groups." Edited by Sushil Jajodia, *Proceedings of the 7th ACM Conference on Computer and Communications Security*. Athens, Greece: ACM Press, 235–244.

———. 2001. "Communication-Efficient Group Key Agreement." *Information Systems Security, Proceedings of the 17th International Information Security Conference IFIP SEC'01*.

Koblitz, Neal, ed. 1996. *Advances in Cryptology – CRYPTO '96*. Volume 1109 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Kohl, John T. and B. Clifford Neuman. 1993. "The Kerberos Network Authentication Service (V5)." Internet request for comment RFC 1510, Internet Engineering Task Force.

Kohlas, Reto and Ueli Maurer. 2000a, January. "Confidence Valuation in a Public-Key Infrastructure based on Uncertain Evidence." Edited by H. Imai and Y Zheng, *International Workshop on Practice and Theory in Public Key Cryptography '2000 (PKC '2000)*, Volume 1751 of *Lecture Notes in Computer Science*. Melbourne, Australia: Springer-Verlag, Berlin Germany, 93–112.

———. 2000b. "Reasoning About Public-Key Certification: On Bindings Between Entities and Public Keys." *IEEE Journal on Selected Areas in Communications* 18 (4): 551–560 (April).

Kohnfelder, Loren M. 1978. "Towards a practical public-key cryptosystem." B.Sc thesis, MIT Departement of Electrical Engineering.

Lenstra, Arjen K. and Eric R. Verheul. 2001. "Selecting Cryptographic Key Sizes." *Journal of Cryptology* 14 (4): 255–293.

Lidl, Rudolf and Harald Niederreiter. 1997, January. *Finite Fields*. Second edition. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Lim, Chae Hoon and Pil Joong Lee. 1997. "A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup." Edited by Burton S. Kaliski, Jr., *Advances in Cryptology – CRYPTO '97*, Volume 1294 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 249–263.

Lindell, Yehuda. 2001. "Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation." In Kilian 2001, 171–189.

Lowe, Gavin. 1996. "Breaking and fixing the Needham-Schroeder public-key protocol using FDR." *Tools and Algorithms for the Construction*

and Analysis of Systems (TACAS), Volume 1055 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 147–166.

———. 1997. "A Hierarchy of Authentication Specifications." *10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 31–43.

MacKenzie, Philip. 2001, July. "On the Security of the SPEKE Password-Authenticated Key Exchange Protocol." Report 2001/057, Cryptology ePrint Archive.

Maurer, Ueli M. and Stefan Wolf. 1996. "Diffie-Hellman Oracles." In Koblitz 1996, 268–282.

———. 1998a, August. "Diffie-Hellman, Decision Diffie-Hellman, and Discrete Logarithms." *IEEE Symposium on Information Theory*. Cambridge, USA, 327.

———. 1998b. "Lower bounds on generic algorithms in groups." Edited by Kaisa Nyberg, *Advances in Cryptology – EUROCRYPT '98*, Volume 1403 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 72–84.

———. 1999. "Unconditionally Secure Key Agreement and the Intrinsic Conditional Information." *IEEE Transactions on Information Theory* 45 (2): 499–514.

Mayer, Alain and Moti Yung. 1999, November. "Secure Protocol Transformation via "Expansion": From Two-party to Multi-party." Edited by Gene Tsudik, *Proceedings of the 6th ACM Conference on Computer and Communications Security*. Singapore: ACM Press, 83–92.

McCurley, Kevin S. 1990. "The Discrete Logarithm Problem." Edited by Carl Pomerance, *Cryptology and Computational Number Theory*, Volume 42 of *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society Providence, 49–74.

McGrew, David A. and Alan T. Sherman. 1998, May. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. Manuscript.

Meadows, Catherine. 1992. "Applying Formal Methods to the Analysis of a Key Management Protocol." *Journal of Computer Security* 1 (1): 5–35.

———. 1996. "The NRL Protocol Analyzer: An Overview." *Journal of Logic Programming* 26 (2): 113–131.

———. 2000, July. "Extending Formal Cryptographic Protocol Analysis Techniques for Group Protocols and Low-Level Cryptographic Primitives." Edited by Pierpaolo Degano, *Workshop on Issues in the Theory of Security (WITS'00)*.

University of Geneva, Switzerland. Electronic proceedings: http://www.dsi.unive.it/IFIPWG1_7/WITS2000/programme-new.html.

Meadows, Cathering, Paul Syverson, and Iliano Cervesato. 2001, November. "Formalizing GDOI Group Key Management Requirements in NPATRL." Edited by Pierangela Samarati, *Proceedings of the 8th ACM Conference on Computer and Communications Security*. Philadelphia, PA, USA: ACM Press, 235–244.

Medvinsky, Ari and Matthew Hur. 1999, October. "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)." Internet request for comment RFC 2712, Internet Engineering Task Force.

Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone. 1997. *Handbook of Applied Cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press. ISBN 0-8493-8523-7.

Millen, Jonathan K. 1998, November. Authentication Protocol Verification and Analysis. Tutorial given at ACM Computer and Communication Security symposium. Slides available from http://www2.csl.sri.com/~millen/ccs5tut98/index.htm.

Millen, Jonathan K., Sidney C. Clark, and Sheryl B. Freedman. 1987. "The Interrogator: Protocol Security Analysis." *IEEE Transactions on Software Engineering* 13 (2): 274–288 (February).

Mittra, S. 1997, September. "Iolus: A Framework for Scalable Secure Multicasting." *ACM SIGCOMM'97*. 277–288.

Molva, Refik, Gene Tsudik, Els Van Herreweghen, and Stefano Zatti. 1992, November. "*KryptoKnight* Authentication and Key Distribution System." Edited by Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, *Proceedings of the Second European Symposium on Research in Computer Security (ESORICS)*, Volume 648 of *Lecture Notes in Computer Science*. Toulouse, France: Springer-Verlag, Berlin Germany, 155–174.

Moser, L. E., P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. 1996. "Totem: A Fault-Tolerant Multicast Group Communication System." *Communications of the ACM* 39 (4): 54–63 (April).

Naor, Moni and Omer Reingold. 1997. "Number-Theoretic Constructions of Efficient Pseudo-Random Functions." *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 458–467.

National Institute of Standards and Technology (NIST). 1999, October. Data Encryption Standard (DES). Federal Information Processing Standards Publication (FIPS PUB) 46-3.

———. 2000, January. The Digital Signature Standard (DSS). Federal Information Processing Standards Publication (FIPS PUB) 186-2. updated 2001-10-05.

Nechaev, V. I. 1994. "Complexity of a determinate algorithm for the discrete logarithm." *Mathematical Notes* 55 (2): 165–172. Translated from Matematicheskie Zametki, 55(2):91–101, 1994.

Odlyzko, Andrew. 2000a. "Cryptographic Abundance and Pervasive Computing." *iMP Magazine*, June.

———. 2000b. "Discrete logarithms: The past and the future." *Designs, Codes and Cryptography* 19:129–145.

Okamoto, T., ed. 2000. *Advances in Cryptology – ASIACRYPT '2000*. Volume 1976 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Kyoto, Japan: Springer-Verlag, Berlin Germany.

Patel, Sarvar and Ganapathy S. Sundaram. 1998. "An Efficient Discrete Log Pseudo Random Generator." Edited by Hugo Krawczyk, *Advances in Cryptology – CRYPTO '98*, Volume 1462 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 304–317.

Paulson, Lawrence C. 1997. "Proving Properties of Security Protocols by Induction." *10th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 70–83.

Pereira, Olivier and Jean-Jacques Quisquater. 2000, July. "On the Perfect Encryption Assumption." Edited by Pierpaolo Degano, *Workshop on Issues in the Theory of Security (WITS'00)*. University of Geneva, Switzerland. Electronic proceedings: http://www.dsi.unive.it/IFIPWG1_7/WITS2000/programme-new.html.

———. 2001, June. "Security Analysis of the Cliques Protocols Suites." *14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press.

Perrig, Adrian, Dawn Song, and Doug Tygar. 2001, May. "ELK, a New Protocol for Efficient Large-Group Key Distribution." *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 247–262.

Pfitzmann, Birgit, ed. 2001. *Advances in Cryptology – EUROCRYPT '2001*. Volume 2045 of *Lecture Notes in Computer Science*. Innsbruck, Austria: Springer-Verlag, Berlin Germany.

Pfitzmann, Birgit and Ahmad-Reza Sadeghi. 2000. "Anonymous Fingerprinting with Direct Non-Repudiation." In Okamoto 2000, 401–414.

Pfitzmann, Birgit and Michael Waidner. 2001, May. "A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission." *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 184–200.

Pfitzmann, Andreas, Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. 1997. "Trusting Mobile User Devices and Security Modules." *IEEE Computer* 30 (2): 61–68 (February).

Pfitzmann, Birgit, Matthias Schunter, and Michael Waidner. 2000. "Cryptographic Security of Reactive Systems." *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 32. Workshop on Secure Architectures and Information Flow, Royal Holloway, University of London, December 1 - 3, 1999.

Pfitzmann, Birgit, Michael Steiner, and Michael Waidner. 2002. "A Formal Model for Multi-party Group Key Agreement." Technical Report RZ 3383 (# 93419), IBM Research.

Pfleeger, Charles. 1997. *Security in computing*. 2nd Edition. Prentice Hall International.

Pohlig, Stephen C. and Martin E. Hellman. 1978. "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance." *IEEE Transactions on Information Theory* 24:106–110.

Pollard, J. M. 1978. "Monte Carlo methods for index computation mod p." *Mathematics of Computation* 32:918–924.

Preneel, Bart, ed. 2000. *Advances in Cryptology – EUROCRYPT '2000*. Volume 1807 of *Lecture Notes in Computer Science*. Brugge, Belgium: Springer-Verlag, Berlin Germany.

Reiter, Michael K. 1994, May. "A Secure Group Membership Protocol." *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 176–189.

——— 1996. "Distributing Trust with the Rampart Toolkit." *Communications of the ACM* 39 (4): 71–74 (April).

Reiter, Michael K. and Kenneth P. Birman. 1994. "How to Securely Replicate Services." *ACM Transactions on Programming Languages and Systems* 16 (3): 986–1009 (May).

Reiter, Michael, Kenneth Birman, and Robbert van Renesse. 1994. "A Security Architecture for Fault-Tolerant Systems." *ACM Transactions on Computer Systems* 12 (4): 340–371 (November).

Rivest, Ron L., Adi Shamir, and Leonard M. Adleman. 1978. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM* 21 (2): 120–126 (February).

Rodeh, Ohad, Kenneth P. Birman, and Danny Dolev. 2002. "Using AVL trees for fault-tolerant group key management." *International Journal of Information Security* 1 (2): 84–99.

Roscoe, A. W. 1996, June. "Intensional specifications of security protocols." *9th IEEE Computer Security Foundations Workshop*. Kenmare, Co. Kerry, Ireland: IEEE Computer Society Press, 28–38.

Sadeghi, Ahmad-Reza and Michael Steiner. 2001. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference." In Pfitzmann 2001, 243–260.

———. 2002, August. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference." Report 2002/126, Cryptology ePrint Archive.

Schirokauer, Oliver. 1993. "Discrete logarithms and local units." *Philosophical Transactions of the Royal Society of London* A 345:409–423.

Schneier, Bruce. 1999, February. Snake Oil. Crypto-Gram Newsletter. http://www.counterpane.com/crypto-gram-9902.html.

Schnorr, Claus P. 1991. "Efficient Signature Generation by Smart Cards." *Journal of Cryptology* 4 (3): 161–174.

Schweinberger, Thomas and Victor Shoup. 2000, March. "ACE: The Advanced Cryptographic Engine." Technical Report, IBM Research. Submission to IEEE P1363a (http://grouper.ieee.org/groups/1363/).

Setia, Sanjeev, Samir Koussih, and Sushil Jajodia. 2000, May. "Kronos: A Scalable Group Re-keying Approach for Secure Multicast." *Proceedings of the IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 215–228.

Shmuely, Zahava. 1985, February. "Composite Diffie-Hellman Public-Key Generating Systems are Hard to Break." Computer science technical report 356, Israel Institute of Technology (Technion).

Shoup, Victor. 1997. "Lower Bounds for Discrete Logarithms and Related Problems." Edited by Walter Fumy, *Advances in Cryptology – EUROCRYPT '97*, Volume 1233 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 256–266.

———. 1999, April. "On Formal Models for Secure Key Exchange." Research report RZ 3120 (#93166), IBM Research. A

revised version 4, dated November 15, 1999, is available from http://www.shoup.net/papers/.

———. 2000. "Using Hash Functions as a Hedge against Chosen Ciphertext Attacks." In Preneel 2000, 275–288.

———. 2001. "OAEP Reconsidered." In Kilian 2001, 239–259.

Shparlinski, Igor E. 2000, May. "Security of Polynomial Transformations of the Diffie–Hellman Key." Report 2000/023, Cryptology ePrint Archive.

Smith, Jean E. and Fred W. Weingarten, eds. 1997, May. *Research Challenges for the Next Generation Internet*. Computing Research Association. Report from the Workshop on Research Directions for the Next Generation Internet.

Stadler, Markus. 1996. "Publicly Verifiable Secret Sharing." Edited by Ueli Maurer, *Advances in Cryptology – EUROCRYPT '96*, Volume 1070 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany, 190–199.

Steer, David G., Leo Strawczynski, Whitfield Diffie, and Michael J. Wiener. 1990. "A Secure Audio Teleconference System." Edited by Shafi Goldwasser, *Advances in Cryptology – CRYPTO '88*, Volume 403 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 520–528.

Steiner, Michael, Peter Buhler, Thomas Eirich, and Michael Waidner. 2001. "Secure Password-Based Cipher Suite for TLS." *ACM Transactions on Information and System Security* 4 (2): 134–157 (May).

Steiner, Michael, Gene Tsudik, and Michael Waidner. 1996, March. "Diffie-Hellman Key Distribution Extended to Groups." Edited by Clifford Neuman, *Proceedings of the 3rd ACM Conference on Computer and Communications Security*. New Delhi, India: ACM Press, 31–37. Appeared as revised and extended journal version as (Steiner, Tsudik, and Waidner 2000).

———. 1998, May. "CLIQUES: A New Approach to Group Key Agreement." *18th International Conference on Distributed Computing Systems (ICDCS'98)*. Amsterdam: IEEE Computer Society Press, 380–387. Appeared as heavily revised and extended journal version in (Steiner, Tsudik, and Waidner 2000).

———. 2000. "Key Agreement in Dynamic Peer Groups." *IEEE Transactions on Parallel and Distributed Systems* 11 (8): 769–780 (August).

Syverson, Paul and Paul C. van Oorschot. 1994, May. "On Unifying Some Cryptographic Protocol Logics." *Proceedings of the IEEE Symposium*

on Research in Security and Privacy. IEEE Computer Society, Technical Committee on Security and Privacy Oakland, CA: IEEE Computer Society Press, 14–28.

Tompa, Martin and Heather Woll. 1987. "Random self-reducibility and zero knowledge proofs of possession of information." *Proceedings of the 28th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 472–482.

Tzeng, Wen-Guey. 2000, January. "A Practical and Secure-Fault-Tolerant Conference-Key Agreement Protocol." Edited by H. Imai and Y Zheng, *International Workshop on Practice and Theory in Public Key Cryptography '2000 (PKC '2000)*, Volume 1751 of *Lecture Notes in Computer Science*. Melbourne, Australia: Springer-Verlag, Berlin Germany, 1–13.

Tzeng, Wen-Guey and Zhi-Jia Tzeng. 2000. "Round-Efficient Conference-Key Agreement Protocols with Provable Security." In Okamoto 2000, 614–628.

Wallner, Debbby M., Eric J. Harder, and Ryan C. Agee. 1997, June. Key Management for Multicast: Issues and Architecture. Internet-Draft draft-wallner-key-arch-00.txt.

Wolf, Stefan. 1999. "Information-Theoretically and Computionally Secure Key Agreement in Cryptography." Ph.D. diss., ETH Zürich.

Wong, Chung Kei, Mohamed G. Gouda, and Simon S. Lam. 1998. "Secure Group Communications Using Key Graphs." *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*. 68–79. Appeared in ACM SIGCOMM Computer Communication Review, Vol. 28, No. 4 (Oct. 1998).

Yacobi, Yacov and Zahava Shmuely. 1990. "On key distribution systems." Edited by Giles Brassard, *Advances in Cryptology – CRYPTO '89*, Volume 435 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 344–355.

Yao, Andrew C. 1982. "Theory and Applications of Trapdoor Functions." *Proceedings of the 23rd Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 80–91.

Index

Symbols

A

B

C

D

E

G

H

I

R

S

T

U

W

Appendix A

Deriving Formal Assumptions from the Parameters

The "mechanics" of deriving the formal assumption statement from its short form \$s-\$t$\$\mathcal{P}^{\$a}$(c:\$c; g:\$g; f:$\$\mathcal{G}$) — as described in Section 3.3 the \$X's are placeholders of the parameters defined in Section 3.2 — is as follows:

1. **Group and problem family:** Just fix the group, generator and problem instance sampler $SG_{\mathcal{G}}$, Sg, and $SPI_{\mathcal{P}}$ corresponding to group family $\$\mathcal{G}$ and problem family $\$\mathcal{P}$, respectively. In the context of generic relations, $\$\mathcal{G}$ does normally not fix a particular group family and sampler but gives just some specific constraints on group families, e.g., groups with large prime factors indicated by "lprim". In such a case $SG_{\mathcal{G}}$ denotes an arbitrary sampler for an arbitrary group family fulfilling the given constraints on the group family and the constraints on samplers given in Section 3.1.7.[1]

2. **Problem type:** Prepare the assumption formula $\$\mathbf{F}$ as the probability statement $\$\mathbf{P}$ defined as "**Prob**[". $\$\mathbf{P}_{\text{pred}}$." :: ". $\$\mathbf{P}_{\text{def}}$."]". The . denotes the string-concatenation operator and the variables $\$\mathbf{P}_{\text{pred}}$ and $\$\mathbf{P}_{\text{def}}$ are the probability predicate and the probability space instance definition, respectively. They are defined depending on the problem type \$t as follows (where $SPI_{\mathcal{P}}$ is the problem sampler fixed in item 1 above and where the source of SI is explained in item 3 below):

   - \$t = C: Initialize $\$\mathbf{P}_{\text{def}}$ to "$PI \leftarrow SPI_{\mathcal{P}}(SI)$;" (the problem instance to solve) and add "$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$;" (the random coins for the adversary) to it. Define $\$\mathbf{P}_{\text{pred}}$ as "$\mathsf{A}(\mathcal{C}, SI, PI^{publ}) \in PI^{sol}$".

[1] In practice, only the later application of this relation using specific assumptions implied by a cryptographic systems will determine the concrete choices of group family and sampler.

- \$t = D: Initialize $\$\mathbf{P}_{\text{def}}$ to the concatenation of "$b \xleftarrow{\mathcal{R}} \{0,1\}$;" (the random bit used as challenge), "$PI_0 \leftarrow SPI_{\mathcal{P}}(SI)$;" and "$PI_1 \leftarrow SPI_{\mathcal{P}}(SI)$;" (the real problem instance and an auxiliary problem instance for the random public part), "$sol_c \xleftarrow{\mathcal{R}} {PI_b}^{sol}$;" (one possible solution), and "$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$;". $\$\mathbf{P}_{\text{pred}}$ is defined as "$\mathsf{A}(\mathcal{C}, SI, PI^{publ}, sol_c) = b$". Additionally, the probability statement $\$\mathbf{P}$ is normalized to "$2 \cdot |\mathbf{Prob}[\$\mathbf{P}_{\text{pred}} :: \$\mathbf{P}_{\text{def}}] - 0.5|$".
- \$t = M: Initialize $\$\mathbf{P}_{\text{def}}$ to the concatenation of "$b \xleftarrow{\mathcal{R}} \{0,1\}$;" (the random bit used as challenge), "$PI_0 \leftarrow SPI_{\mathcal{P}}(SI)$;" and "$PI_1 \leftarrow SPI_{\mathcal{P}}(SI)$;" (the two problem instances to match), "$sol_0 \xleftarrow{\mathcal{R}} {PI_0}^{sol}$" and "$sol_1 \xleftarrow{\mathcal{R}} {PI_1}^{sol}$" (two corresponding solutions), and "$\mathcal{C} \xleftarrow{\mathcal{R}} \mathcal{U}$;". $\$\mathbf{P}_{\text{pred}}$ is defined as "$\mathsf{A}(\mathcal{C}, SI, {PI_0}^{publ}, {PI_1}^{publ}, sol_b, sol_{\bar{b}}) = b$". Additionally, the probability statement $\$\mathbf{P}$ is normalized as above to "$2 \cdot |\mathbf{Prob}[\$\mathbf{P}_{\text{pred}} :: \$\mathbf{P}_{\text{def}}] - 0.5|$".

3. **Granularity:** Depending on the granularity value \$g do the following (where $SG_{\mathcal{G}}$ and Sg are the group and generator sampler fixed in item 1):

   - \$g = l: Prepend "$G \leftarrow SG_{\mathcal{G}}(1^k)$;", "$g_i \leftarrow Sg(G)$;" (for as many $i \in \mathbb{N}$ as required by the problem family, e.g., one generator for DL and n generators for RP(n)), and "$SI \leftarrow (G, g_1, \ldots)$;" to $\$\mathbf{P}_{\text{def}}$.
   - \$g = m: Prepend "$\forall G \in [SG_{\mathcal{G}}(1^k)]$; to $\$\mathbf{F}$. Prepend "$g \leftarrow Sg(G)$;" and "$SI \leftarrow (G, g_1, \ldots)$;" to $\$\mathbf{P}_{\text{def}}$.
   - \$g = h: Prepend "$\forall G \in [SG_{\mathcal{G}}(1^k)]$;", "$\forall g_i \in [Sg(G)]$;", and "$SI \leftarrow (G, g_1, \ldots)$;" to $\$\mathbf{F}$.

4. **Computational complexity and algebraic knowledge:** Depending on the computational complexity \$c do the following:

   - \$c = u: Prefix $\$\mathbf{F}$ with "$\forall \mathsf{A} \in \mathcal{UPTM}$;", "$\exists k_0$;", and "$\forall k > k_0$;".
   - \$c = n: Prefix $\$\mathbf{F}$ with "$\forall(\mathsf{A}_i \mid i \in \mathbb{N}) \in \mathcal{NPTM}$;", "$\exists k_0$;", and "$\forall k > k_0$;". In $\$\mathbf{P}_{\text{pred}}$ replace "A" by "A_k".

   If the considered assumption is in the generic model ($\$a = \sigma$) then replace everywhere "A", $\mathcal{UPTM}$ and $\mathcal{NPTM}$ by "A^σ", $\mathcal{UPTM}^\sigma$ and $\mathcal{NPTM}^\sigma$, respectively. Furthermore, append "$\sigma \xleftarrow{\mathcal{R}} \Sigma_{G,g}$;" (the choice of the random encoding function) to $\$\mathbf{P}_{\text{def}}$.

5. **Success probability:** Depending on the success probability \$s do the following to finish the formal assumption statement:

   - \$s = 1: Append "< 1" to $\$\mathbf{F}$, i.e., immediately after $\$\mathbf{P}$.

- $\$s = (1-1/\mathsf{poly}(k))$: Append "$\exists d_1$;" immediately after the all-quantifier on adversary algorithms in $\$\mathbf{F}$. Append "$< (1-1/k^{d_1})$" to $\$\mathbf{F}$.
- $\$s = \epsilon$: Append "$< \epsilon$" to $\$\mathbf{F}$.
- $\$s = 1/\mathsf{poly}(k)$: Append "$\forall d_1$;" immediately after the all-quantifier on adversary algorithms in $\$\mathbf{F}$. Append "$< 1/k^{d_1}$" to $\$\mathbf{F}$.

Evaluating $\$\mathbf{F}$ by expanding the variables , i.e., $\$\mathbf{P}$, $\$\mathbf{P}_{\text{pred}}$ and $\$\mathbf{P}_{\text{def}}$, and applying the string-concatenation operator gives now the desired precise formal assumption statement.

Appendix B

Detailed Specification of Models and Protocols

This appendix contains the complete and detailed specification of the machines defined in Chapter 5. In particular, it contains explicitly the structures derived from the intended structures in the main text as described in Section 5.1.2 and thus the full details of their behavior during and after corruption. Furthermore, it explicitly spells out all machines in the semi-real system and the simulator whereas the main text described a number of them only implicitly by giving the differences to previously defined machines.

Scheme 5.1 (Ideal System for Group Key Establishment $Sys^{\mathsf{gke,ideal}}_{n,tb,ct}$)
An overview of the ideal host $\mathsf{TH}_{\mathcal{H}}$, the connectivity and exchanged messages is given in Figure B.1. The message types and parameters are described in the Tables B.1. The variables of $\mathsf{TH}_{\mathcal{H}}$ are described in the Table B.2. The transitions of $\mathsf{TH}_{\mathcal{H}}$ are defined as follows:

transition $\mathsf{in}_u?$ (init)
  enabled if: $(state_{u,u} = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  $state_{u,u} \leftarrow \mathsf{wait}$;
  output: $\mathsf{out}_{\mathsf{sim},u}!$ (init), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{sim},u}?$ (initialized, v)
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_{\mathsf{sim},u}?.cntr < tb)$;
  ignore if: $((state_{v,v} = \mathsf{undef}) \wedge (v \notin \mathcal{A})) \ \vee \ ((u = v) \wedge (state_{u,u} \neq \mathsf{wait}))$;
  $state_{u,v} \leftarrow \mathsf{init}$;
  output: $\mathsf{out}_u!$ (initialized, v), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
end transition

Table B.1 The message types and parameters handled by $\mathsf{TH}_{\mathcal{H}}$

Port	Type	Parameters	Meaning
At specified ports $S_{\mathcal{H}}$ to user $u \in \mathcal{H}$			
$\mathsf{in}_u?$	init		Initialize user u.
$\mathsf{out}_u!$	initialized	v	User v initialized from user u's point of view.
$\mathsf{in}_u?$	new	sid,grp,[sid',grp']	Initialize a new session, extending a previous one if optional parameters are present.
$\mathsf{out}_u!$	key	sid,grp,key	Return newly agreed key.
$\mathsf{corrupt}_u?$	do		Corrupt user u!
$\mathsf{out}_u!$	*arbitrary*	*arbitrary*	Possible outputs after corruptions
At adversary ports			
$\mathsf{out}_{\mathsf{sim},u}!$	init		User u is initializing.
$\mathsf{in}_{\mathsf{sim},u}?$	initialized	$v \in \mathcal{M}$	User u should consider user v as initialized.
$\mathsf{out}_{\mathsf{sim},u}!$	new	sid,grp,[sid',grp']	User u has initialized a new session.
$\mathsf{in}_{\mathsf{sim},u}?$	finish	sid,grp,[$key_{u,sim}$]	Complete session for user u. If present and allowed, assign $key_{u,sim}$ to user u.
$\mathsf{corOut}_{\mathsf{sim},u}!$	state	*state*	State of corrupted party.
$\mathsf{out}_{\mathsf{sim},u}!$	*arbitrary*	*arbitrary*	Corrupted party u sent a message.
$\mathsf{in}_{\mathsf{sim},u}?$	*arbitrary*	*arbitrary*	Send message to (corrupted) party u.

Figure B.1 Trusted host and its message types. Parts related to adaptive adversaries are in gray. Dashed lines indicate who schedules a connection.

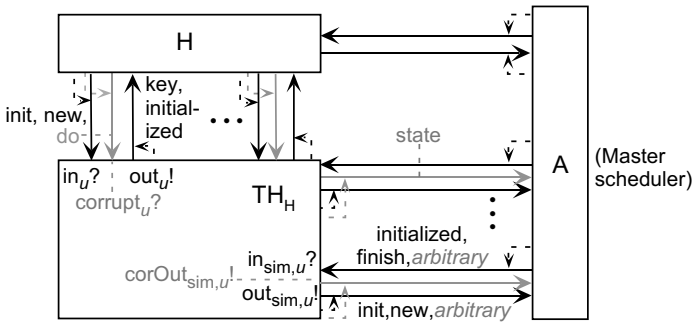


Table B.2 Variables of $\mathsf{TH}_{\mathcal{H}}$

Name	Domain	Meaning	Init.
$(state_{u,v})_{u,v\in\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by user u	undef
$(ses_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{init}, \mathsf{finished}\}$	State of sessions as seen by user u	undef
$(key_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}\}$	Session keys still in negotiation	undef
$(prev_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$(sid' \in \mathcal{SID}, grp' \subseteq \mathcal{M})$	Dependency graph of sessions	$(0, \{\})$
$(p?.cntr)_{p\in\{\mathsf{in}_u,\mathsf{corrupt}_u,\mathsf{in}_{\mathsf{sim},u} \mid u\in\mathcal{H}\}}$	$\mathbb{N}$	Activation counters	0

transition $\mathsf{in}_u?$ (new, $sid, grp, [sid', grp']$)
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  ignore if: $(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_{u,v} \neq \mathsf{init}) \vee$
    $(ses_{u,sid,grp} \neq \mathsf{undef}) \vee$
    $(\mathsf{present}(sid', grp') \wedge (u \in grp') \wedge (ses_{u,sid',grp'} \neq \mathsf{finished}))$;
  $ses_{u,sid,grp} \leftarrow \mathsf{init}$;
  if $\mathsf{present}(sid', grp')$ **then**
    $prev_{u,sid,grp} \leftarrow (sid', grp')$;
  end if;
  output: $\mathsf{out}_{\mathsf{sim},u}!$ (new, $sid, grp, [sid', grp']$), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{sim},u}?$ (finish, $sid, grp, [key_{u,sim}]$)
  enabled if: $(state_{u,u} \neq \mathsf{corrupted}) \wedge (\mathsf{in}_{\mathsf{sim},u}?.cntr < tb)$;
  ignore if: $(ses_{u,sid,grp} \neq \mathsf{init})$;
  if $\mathsf{present}(key_{u,sim}) \wedge$
    $((\exists v \in grp : state_{v,v} = \mathsf{corrupted} \vee v \in \mathcal{A}) \vee$
    $(\exists v_0, v_1 \in grp : (ses_{v_0,sid,grp} \neq \mathsf{undef}) \wedge (ses_{v_1,sid,grp} \neq \mathsf{undef}) \wedge$
    $(prev_{v_0,sid,grp} \neq prev_{v_1,sid,grp})))$ **then**
    # Corrupted or inconsistent session so ...
    $key_{u,sid,grp} \leftarrow key_{u,sim}$;*# ... use session key provided by adversary*
  else if $(\forall\, v \in grp : ses_{v,sid,grp} \neq \mathsf{finished})$ **then**
    # First to finish (ideal) session
    $key \xleftarrow{\mathcal{R}} \{0,1\}^k$; *# Generate new (random) session key ...*
    for all $v \in grp$ **do**
      $key_{v,sid,grp} \leftarrow key$; *# ... and assign it to all parties*
    end for;
  end if;
  output: $\mathsf{out}_u!$ (key, $sid, grp, key_{u,sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);*# Give key to user ...*
  $key_{u,sid,grp} \leftarrow \mathsf{undef}$; *# ... and delete it locally to enable forward secrecy*
  $ses_{u,sid,grp} \leftarrow \mathsf{finished}$;
end transition

transition $\mathsf{corrupt}_u?$ (do)
  enabled if: $(ct = \mathsf{adaptive} \wedge state_{u,u} \neq \mathsf{corrupted})$;
  $state_{u,u} \leftarrow \mathsf{corrupted}$;
  output: $\mathsf{corOut}_{\mathsf{sim},u}!$ (state, $\mathsf{encode\_state}(u)$), $\mathsf{corOut}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
end transition

function : $\mathsf{encode\_state}(u)$
  return:$(\{(u, v, state_{u,v}) \mid v \in \mathcal{M}\}, \{(sid, grp, ses_{u,sid,grp}, key_{u,sid,grp},$
    $prev_{u,sid,grp}) \mid sid \in \mathcal{SID} \wedge grp \subseteq \mathcal{M} \wedge ses_{u,sid,grp} \neq \mathsf{undef}\})$;

end function

transition $\mathsf{in}_u?$ (any_msg)

enabled if: $(state_{u,u} = \mathsf{corrupted})$; # *Transparent mode*

output: $\mathsf{out}_{\mathsf{sim},u}!$ (any_msg), $\mathsf{out}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);

end transition

transition $\mathsf{in}_{\mathsf{sim},u}?$ (any_msg)

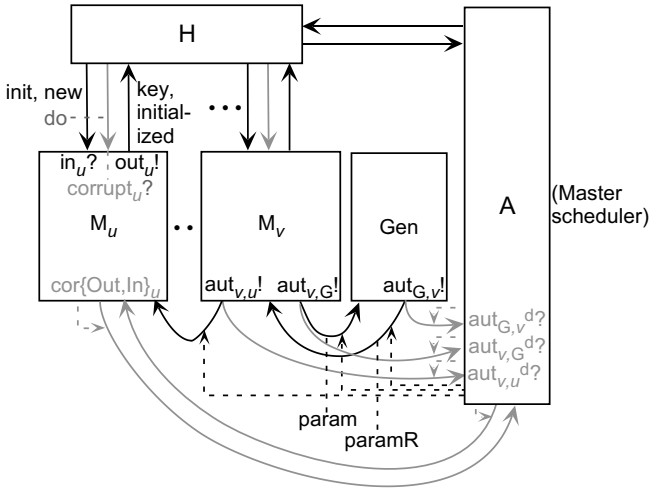
enabled if: $(state_{u,u} = \mathsf{corrupted})$; # *Transparent mode*

output: $\mathsf{out}_u!$ (any_msg), $\mathsf{out}_u{}^{\triangleleft}!$ (1);

end transition

□

Figure B.2 Sketch of the real system. Derived parts are shown in gray. Scheduling is shown only for newly introduced ports.



Scheme 5.2 (Real System for Group Key Establishment $Sys^{\mathsf{gke,ika1}}_{n,tb,ct}$)

An overview of the real system with its machines, their connectivity and exchanged messages is given in Figure B.2. The message types and parameters are described in the Tables B.1 and B.3. The variables of the machines are described in the Table B.4. The transitions of the machines are defined as follows, machine by machine:

Machine Gen

transition $\mathsf{aut}_{u,\mathsf{G}}?$ (param)
  enabled if: $(\mathsf{aut}_{u,\mathsf{G}}?.cntr < tb)$;
  if $(state = \mathsf{undef})$ **then**
    $(G, g, \mathsf{h}) \leftarrow \mathsf{genG}(1^k)$;
    $state \leftarrow \mathsf{init}$;
  end if
  output: $\mathsf{aut}_{\mathsf{G},u}!$ (paramR, G, g, h);
  output: $\mathsf{aut}^{\mathsf{d}}_{\mathsf{G},u}!$ (paramR, G, g, h);
end transition

Machine M_u

transition $\mathsf{in}_u?$ (init) # *Trigger initialization*
  enabled if: $(state_u = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  $state_u \leftarrow \mathsf{wait}$;

Table B.3 The message types and parameters handled by Gen and M_u. (See Table B.1 for remaining messages, i.e., the "upper" interface (specified ports) of M_u.)

Port	Type	Parameters	Meaning
$\mathsf{aut}_{u,\mathsf{G}}?$	param	—	Get system parameters.
$\mathsf{aut}_{\mathsf{G},u}!$	paramR	$(G, g, \mathsf{h}) \in \mathsf{genG}(1^k)$	Reply to above.

Port	Type	Parameters	Meaning
$\mathsf{aut}_{v,u}?$	initialized	—	Notification that M_v^* is initialized.
$\mathsf{aut}_{v,u}?$	up	$sid,\ grp,\ (m_i \in G)_{0 \le i \le \mathsf{idx}(grp,v)}$	Upflow.
$\mathsf{aut}_{v,u}?$	down	$sid,\ grp,\ (m_i \in G)_{0 \le i < \lvert grp \rvert}$	Downflow (broadcast).
$\mathsf{aut}_{v,u}?$	confirm	$sid,\ grp$	Confirmation.

For all messages on ports $\mathsf{aut}_{v,u}!$ there is an additional identical message on $\mathsf{aut}^{\mathsf{d}}_{v,u}!$, i.e., the copy to the eavesdropping A. However, to prevent clutter these messages are omitted from this and similar later tables.

output: $\mathsf{aut}_{u,\mathsf{G}}!$ (param);
output: $\mathsf{aut}^{\mathsf{d}}_{u,\mathsf{G}}!$ (param);
end transition

transition $\mathsf{aut}_{\mathsf{G},u}?$ (paramR, G', g', h') # *Get system parameters*
enabled if: $(state_u = \mathsf{wait})$;
$state_u \leftarrow \mathsf{init}$;
$(G, g, \mathsf{h}) \leftarrow (G', g', \mathsf{h}')$;
output: $\mathsf{out}_u!$ (initialized, u); $\mathsf{out}_u{}^{\triangleleft}!$ (1);
for all $v \in \mathcal{M} \setminus \{u\}$ **do**
output: $\mathsf{aut}_{u,v}!$ (initialized);
output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (initialized);
end for
end transition

transition $\mathsf{aut}_{v,u}?$ (initialized) # *Notification for other machines*
enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
$state_v \leftarrow \mathsf{init}$;
output: $\mathsf{out}_u!$ (initialized, v), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
end transition

Table B.4 Variables in Gen and M_u

Name	Domain	Meaning	Init.
$state$	$\{\mathsf{undef}, \mathsf{init}\}$	Initialized?.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(\mathsf{aut}_{v,\mathsf{G}}?.cntr)_{v \in \mathcal{H}}$	$\mathbb{N}$	Activation counters	0

Name	Domain	Meaning	Init.
$(state_v)_{v \in \mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by M_u^*.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(ses_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{\mathsf{undef}, \mathsf{upflow}, \mathsf{downflow}, \mathsf{confirm}, \mathsf{finished}\}$	State of a (potential) session.	undef
$(\mathcal{C}_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{\mathcal{I} \mid \mathcal{I} \subseteq \mathcal{M}\}$	Records received session confirmations	$\emptyset$
$(key_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}\}$	Group key of a session.	undef
$(x_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\mathbb{Z}_{\lvert G \rvert} \cup \{\mathsf{undef}\}$	Individual secret key of a session.	undef
$(\mathsf{aut}_{v,u}?.cntr)_{v \in \{\mathsf{G}\} \cup \mathcal{H} \setminus \{u\}}$	$\mathbb{N}$	Activation counters	0

transition $\mathsf{in}_u?$ (new, sid, grp) # *Start new session*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  ignore if:
    $(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_v \neq \mathsf{init}) \vee (ses_{sid,grp} \neq \mathsf{undef})$;
  $x_{sid,grp} \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}$;
  $ses_{sid,grp} \leftarrow \mathsf{upflow}$;
  if $(u = grp[1])$ **then** # *u is the first member*
    $m'_1 \leftarrow g$;
    $m'_2 \leftarrow g^{x_{sid,grp}}$
    output: $\mathsf{aut}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
    output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
    $ses_{sid,grp} \leftarrow \mathsf{downflow}$;
  end if
end transition

transition $\mathsf{aut}_{v,u}?$ (up, sid, grp, msg) # *Upflow message arrives*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  ignore if: $(ses_{sid,grp} \neq \mathsf{upflow}) \vee (v \neq grp[\mathsf{idx}(grp, u) - 1]) \vee$
    (msg is not $(m_1, \ldots, m_{\mathsf{idx}(grp,u)})$ with $m_i \in G$ having maximal order);
  $i \leftarrow \mathsf{idx}(grp, u)$; # *u's position in the group*
  $m'_1 \leftarrow m_i$;
  for $1 \leq j \leq \mathsf{min}(i, |grp| - 1)$ **do**
    $m'_{j+1} \leftarrow m_j^{x_{sid,grp}}$
  end for
  if $(i < |grp|)$ **then**
    output: $\mathsf{aut}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
    output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
    $ses_{sid,grp} \leftarrow \mathsf{downflow}$;
  else # $i = |grp|$, *i.e., u is the last member*
    $key_{sid,grp} \leftarrow \mathsf{h}((m_{|grp|})^{x_{sid,grp}})$;
    if $(ct = \mathsf{static})$ **then** # *For the static case we are done*
      $ses_{sid,grp} \leftarrow \mathsf{finished}$;
      output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
    else # *For the adaptive case wait first for the confirmation flows*
      $ses_{sid,grp} \leftarrow \mathsf{confirm}$;
      $\mathcal{C}_{sid,grp} \leftarrow \{u\}$;
      $x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
    end if
    for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" to the group members*
      output: $\mathsf{aut}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
      output: $\mathsf{aut}^{\mathsf{d}}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
    end for
  end if
end transition

transition $\mathsf{aut}_{v,u}?$ (down, sid, grp, msg) # *Downflow message arrives*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  ignore if: $(ses_{sid,grp} \neq \mathsf{downflow}) \vee (v \neq grp[|grp|]) \vee$
    (msg is not $(m_1, \ldots, m_{|grp|})$ with $m_i \in G$ having maximal order);
  $i \leftarrow \mathsf{idx}(grp, u)$; # *u's position in the group*
  $key_{sid,grp} \leftarrow \mathsf{h}((m_{|grp|+1-i})^{x_{sid,grp}})$;
  if ($ct = \mathsf{static}$) **then** # *For the static case we are done*
    $ses_{sid,grp} = \mathsf{finished}$;
    output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
  else # *For the adaptive case, start confirmation*
    $ses_{sid,grp} \leftarrow \mathsf{confirm}$;
    $\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{u, v\}$;
    $x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
    for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" confirmation to group members*
      output: $\mathsf{aut}_{u,v}!$ (confirm, sid, grp);
      output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (confirm, sid, grp);
    end for
    if ($\mathcal{C}_{sid,grp} = grp$) **then** # *We got* down *after all* confirm *...*
      $ses_{sid,grp} = \mathsf{finished}$; # *... so we are done: Give key to user ...*
      output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
      $key_{sid,grp} \leftarrow \mathsf{undef}$; # *... and delete it locally*
    end if
  end if
end transition

transition $\mathsf{aut}_{v,u}?$ (confirm, sid, grp) # *Confirmation message arrives*
  enabled if: $(ct = \mathsf{adaptive}) \wedge (state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  ignore if: $(v \notin grp \setminus \mathcal{C}_{sid,grp}) \vee (ses_{sid,grp} \notin \{\mathsf{downflow}, \mathsf{confirm}\})$;
  $\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{v\}$;
  if $(\mathcal{C}_{sid,grp} = grp) \wedge (ses_{sid,grp} = \mathsf{confirm})$ **then** # *All* confirm *received ...*
    $ses_{sid,grp} \leftarrow \mathsf{finished}$; # *... so we are done: Give key to user ...*
    output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
    $key_{sid,grp} \leftarrow \mathsf{undef}$; # *... and delete it locally*
  end if
end transition

transition $\mathsf{corrupt}_u?$ (do) # *We get corrupted*
  enabled if: $(ct = \mathsf{adaptive} \wedge (state_u \neq \mathsf{corrupted}) \wedge state_u \neq \mathsf{corrupted})$
  $state_u \leftarrow \mathsf{corrupted}$;
  output: $\mathsf{corOut}_u!$ (state, encode_state()), $\mathsf{corOut}_u{}^{\triangleleft}!$ (1);
end transition

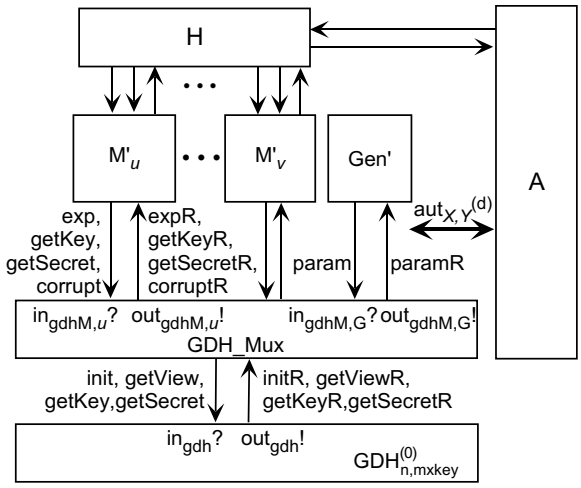
function : encode_state()
return:$((G, g, \mathsf{h}),\ \{(v, state_v) \mid v \in \mathcal{M}\},\ \{(sid, grp, ses_{sid,grp}, \mathcal{C}_{sid,grp},$
$x_{sid,grp}, key_{sid,grp}) \mid sid \in \mathcal{SID} \wedge grp \subseteq \mathcal{M} \wedge ses_{sid,grp} \neq \mathsf{undef}\});$
end function

transition $port?\ (any\_msg)$ # *Transparent mode*
enabled if: $(state_u = \mathsf{corrupted}) \wedge (port \in \{\mathsf{in}_u?\} \cup \{\mathsf{aut}_{v,u}? \mid v \in \mathcal{M} \cup \{\mathsf{G}\}\});$
output: $\mathsf{corOut}_u!\ (port, any\_msg)$, $\mathsf{corOut}_u{}^{\triangleleft}!\ (1);$
end transition

transition $\mathsf{corIn}_u?\ (port, any\_msg)$ # *Transparent mode*
enabled if: $(state_u = \mathsf{corrupted})$
ignore if: $(port \notin \{\mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!\} \cup \{\mathsf{aut}_{u,v}! | v \in \mathcal{M} \cup \{\mathsf{G}\}\});$
output: $port\ (any\_msg);$
end transition

□

Figure B.3 Semi-real system. (Clocking of new components GDH_Mux and $\mathsf{GDH}^{(0)}_{n,mxkey}$ is RPC-style.)



Scheme 5.4 (Semi-real system $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$)

An overview of the semi-real system with its machines, their connectivity and the exchanged messages is given in Figure B.3. The message types and parameters are described in the Tables B.1, B.3 and B.5. The variables of the machines are described in the Tables B.6 and B.7. The transitions of the machines are defined as follows, machine by machine:

Machine Gen'

transition $\mathsf{aut}_{u,\mathsf{G}}?$ (param)
  enabled if: $(\mathsf{aut}_{u,\mathsf{G}}?.cntr < tb)$;
  if $(state = \mathsf{undef})$ **then**
    output: $\mathsf{in}_{\mathsf{gdhM,G}}!$ (param), $\mathsf{in}_{\mathsf{gdhM,G}}{}^{\triangleleft}!$ (1);
    input: $\mathsf{out}_{\mathsf{gdhM,G}}?$ (paramR, G', g', h');
    $(G, g, \mathsf{h}) \leftarrow (G', g', \mathsf{h}')$;
    $state \leftarrow \mathsf{init}$;
  end if
  output: $\mathsf{aut}_{\mathsf{G},u}!$ (paramR, G, g, h);
  output: $\mathsf{aut}^{\mathsf{d}}_{\mathsf{G},u}!$ (paramR, G, g, h);
end transition

Table B.5 The message types and parameters handled by GDH_Mux and $\mathsf{GDH}^{(b)}_{n,mxkey}$. (See Table B.1 (specified ports) and Table B.3 for the remaining message types and parameters handled in the semi-real system.)

Port	Type	Parameters	Meaning
$\mathsf{in}_{\mathsf{gdhM,G}}?$	param	—	Get system parameters
$\mathsf{out}_{\mathsf{gdhM,G}}!$	paramR	G, g, h	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	corrupt	—	Corruption
$\mathsf{out}_{\mathsf{gdhM},u}!$	corruptR	—	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	exp	sid, grp, γ	Exponentiate γ with secret for u in this session. Limited to the computation of partial keys!
$\mathsf{out}_{\mathsf{gdhM},u}!$	expR	γ^{x_u}	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	getKey	sid, grp, γ	Get derived key matching final partial key γ
$\mathsf{out}_{\mathsf{gdhM},u}!$	getKeyR	K	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	getSecret	sid, grp	Get secret of this session (to hand it over during corruption)
$\mathsf{out}_{\mathsf{gdhM},u}!$	getSecretR	x_u	Reply to above

Port	Type	Parameters	Meaning
$\mathsf{in}_{\mathsf{gdh}}?$	init	—	Get system parameters
$\mathsf{out}_{\mathsf{gdh}}!$	initR	G, g, h	Reply to above
$\mathsf{in}_{\mathsf{gdh}}?$	getView	n'	Get GDH partial keys of a new session
$\mathsf{out}_{\mathsf{gdh}}!$	getViewR	$i, \{(\beta, g^{\prod_{\beta_j=1} x_{i,j}}) \mid \beta \in I_{n_i} \setminus \{1^{n_i}\}\}$	Reply to above, i is the session reference identifier
$\mathsf{in}_{\mathsf{gdh}}?$	getKey	i	Get key of session i
$\mathsf{out}_{\mathsf{gdh}}!$	getKeyR	z_i	Reply to above
$\mathsf{in}_{\mathsf{gdh}}?$	getSecret	i	Get secret exponents of session i
$\mathsf{out}_{\mathsf{gdh}}!$	getSecretR	$(x_{i,1}, \ldots, x_{i,n_i})$	Reply to above

Table B.6 Variables in Gen' and M'_u

Name	Domain	Meaning	Init.
$state$	$\{\mathsf{undef}, \mathsf{init}\}$	Initialized?.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(\mathsf{aut}_{v,\mathsf{G}}?.cntr)_{v \in \mathcal{H}}$	$\mathbb{N}$	Activation counters	0

Name	Domain	Meaning	Init.
$(state_v)_{v \in \mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by M'_u.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(ses_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{\mathsf{undef}, \mathsf{upflow}, \mathsf{downflow}, \mathsf{confirm}, \mathsf{finished}\}$	State of a (potential) session.	undef
$(\mathcal{C}_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{\mathcal{I} \mid \mathcal{I} \subseteq \mathcal{M}\}$	Records received session confirmations	$\emptyset$
$(key_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\{0,1\}^k \cup G \cup \{\mathsf{undef}\}$	Group key of a session.	undef
$(x_{sid,grp})_{sid \in \mathcal{SID}, grp \subseteq \mathcal{M}}$	$\mathbb{Z}_{\lvert G \rvert} \cup \{\mathsf{undef}, \mathsf{exists}\}$	Individual secret key of a session.	undef
$(\mathsf{aut}_{v,u}?.cntr)_{v \in \{\mathsf{G}\} \cup \mathcal{H} \setminus \{u\}}$	$\mathbb{N}$	Activation counters	0

Table B.7 Variables in GDH_Mux and $\mathsf{GDH}^{(b)}_{n,mxkey}$

Variables	Domain	Meaning	Init.
$(i_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\mathbb{N}$	Index used for this session with $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(corr_u)_{u\in\mathcal{M}}$	$\{\mathsf{true}, \mathsf{false}\}$	Corrupted machine?	true iff $u \in \mathcal{M}\backslash\mathcal{H}$
$(ses_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	{undef, finished, corrupted}	Session status related to u	undef
$(key_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup$ {undef}	Session key from $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(view_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	View of a session	undef
$(secrets_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	Secrets of a session	undef
$(\mathsf{in}_{\mathsf{gdhM},u}?.cntr)_{u\in\in\mathcal{M}\cup\{\mathsf{G}\}}$ $\mathsf{out}_{\mathsf{gdh}}?.cntr$	$\mathbb{N}$	Activation counters	0

Name	Domain	Meaning	Init.
(G, g, h)	Range of $\mathsf{genG}(1^k)$	System parameters	
i	$\mathbb{N}$	Session counter	0
$(c_i)_{i\in\mathbb{N}}$	{undef, init, finished, corrupted }	Session status	undef
$(n_i)_{i\in\mathbb{N}}$	$\mathbb{N}$	Number of session participants	
$(x_{i,j})_{i,j\in\mathbb{N}}$	$\mathbb{Z}_{\lvert G\rvert}$	Secret exponents	
$(z_i)_{i\in\mathbb{N}}$	G	Session keys	
$\mathsf{in}_{\mathsf{gdh}}?.cntr$	$\mathbb{N}$	Activation counter	0

Machine M'_u

transition $\mathsf{in}_u?$ (init) # *Trigger initialization*
  enabled if: $(state_u = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  $state_u \leftarrow \mathsf{wait}$;
  output: $\mathsf{aut}_{u,\mathsf{G}}!$ (param);
  output: $\mathsf{aut}^{\mathsf{d}}_{u,\mathsf{G}}!$ (param);
end transition

transition $\mathsf{aut}_{\mathsf{G},u}?$ (paramR, G', g', h') # *Get system parameters*
  enabled if: $(state_u = \mathsf{wait})$;
  $state_u \leftarrow \mathsf{init}$;
  $(G, g, \mathsf{h}) \leftarrow (G', g', \mathsf{h}')$;
  output: $\mathsf{out}_u!$ (initialized, u); $\mathsf{out}_u{}^{\triangleleft}!$ (1);
  for all $v \in \mathcal{M} \setminus \{u\}$ **do**
    output: $\mathsf{aut}_{u,v}!$ (initialized);
    output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (initialized);
  end for
end transition

transition $\mathsf{aut}_{v,u}?$ (initialized) # *Notification for other machines*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  $state_v \leftarrow \mathsf{init}$;
  output: $\mathsf{out}_u!$ (initialized, v), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_u?$ (new, sid, grp) # *Start new session*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{in}_u?.cntr < tb)$;
  ignore if:
    $(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_v \neq \mathsf{init}) \vee (ses_{sid,grp} \neq \mathsf{undef})$;
  $x_{sid,grp} \leftarrow \mathsf{exists}$;# *Just remember that exponent did not get erased yet*
  $ses_{sid,grp} \leftarrow \mathsf{upflow}$;
  if $(u = grp[1])$ **then** # *u is the first member*
    $m'_1 \leftarrow g$;
    output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (exp, sid, grp, g), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
    input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (expR, $g^{x_{sid,grp}}$);
    $m'_2 \leftarrow g^{x_{sid,grp}}$;
    output: $\mathsf{aut}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
    output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
    $ses_{sid,grp} \leftarrow \mathsf{downflow}$;
  end if
end transition

transition $\mathsf{aut}_{v,u}?$ (up, sid, grp, msg) # *Upflow message arrives*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  ignore if: $(ses_{sid,grp} \neq \mathsf{upflow}) \vee (v \neq grp[\mathsf{idx}(grp,u)-1]) \vee$
    (msg is not $(m_1, \ldots, m_{\mathsf{idx}(grp,u)})$ with $m_i \in G$ having maximal order);
  $i \leftarrow \mathsf{idx}(grp,u)$; # *u's position in the group*
  $m'_1 \leftarrow m_i$;
  for $1 \leq j \leq \mathsf{min}(i, |grp|-1)$ **do**
    output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (exp, sid, grp, m_j), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
    input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (expR, $m_j^{x_{sid,grp}}$);
    $m'_{j+1} \leftarrow m_j^{x_{sid,grp}}$;
  end for
  if $(i < |grp|)$ **then**
    output: $\mathsf{aut}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
    output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
    $ses_{sid,grp} \leftarrow \mathsf{downflow}$;
  else # $i = |grp|$, *i.e., u is the last member*
    $key_{sid,grp} \leftarrow m_{|grp|}$; # *Just remember the pre-key*
    if $(ct = \mathsf{static})$ **then** # *For the static case we are done*
      $ses_{sid,grp} \leftarrow \mathsf{finished}$;
      output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
      input: $\mathsf{out}_{\mathsf{gdhM},u}?$ ($\mathsf{getKeyR}$, key);
      $key_{sid,grp} \leftarrow key$;
      output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
    else # *For the adaptive case wait first for the confirmation flows*
      $ses_{sid,grp} \leftarrow \mathsf{confirm}$;
      $\mathcal{C}_{sid,grp} \leftarrow \{u\}$;
      $x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
    end if
    for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" to the group members*
      output: $\mathsf{aut}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
      output: $\mathsf{aut}^{\mathsf{d}}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
    end for
  end if
end transition

transition $\mathsf{aut}_{v,u}?$ (down, sid, grp, msg) # *Downflow message arrives*
  enabled if: $(state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
  ignore if: $(ses_{sid,grp} \neq \mathsf{downflow}) \vee (v \neq grp[|grp|]) \vee$
    (msg is not $(m_1, \ldots, m_{|grp|})$ with $m_i \in G$ having maximal order);
  $i \leftarrow \mathsf{idx}(grp,u)$; # *u's position in the group*
  $key_{sid,grp} \leftarrow m_{|grp|+1-i}$; # *Just remember the pre-key*
  if $(ct = \mathsf{static})$ **then** # *For the static case we are done*
    $ses_{sid,grp} = \mathsf{finished}$;

output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
else # *For the adaptive case, start confirmation*
$ses_{sid,grp} \leftarrow$ confirm;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{u, v\}$;
$x_{sid,grp} =$ undef; # *Erase secret exponent*
for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" confirmation to group members*
output: $\mathsf{aut}_{u,v}!$ (confirm, sid, grp);
output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (confirm, sid, grp);
end for
if ($\mathcal{C}_{sid,grp} = grp$) **then** # *We got* down *after all* confirm *...*
$ses_{sid,grp} =$ finished; # *...so we are done: Give key to user ...*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow$ undef; # *...and delete it locally*
end if
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (confirm, sid, grp) # *Confirmation message arrives*
enabled if: $(ct =$ adaptive$) \wedge (state_u \neq$ corrupted$) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(v \notin grp \setminus \mathcal{C}_{sid,grp}) \vee (ses_{sid,grp} \notin \{$downflow, confirm$\})$;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{v\}$;
if ($\mathcal{C}_{sid,grp} = grp$)$\wedge$($ses_{sid,grp} =$ confirm) **then** # *All* confirm *received ...*
$ses_{sid,grp} \leftarrow$ finished; # *...so we are done: Give key to user ...*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{out}_u!$ (key, $sid, grp, key_{sid,grp}$), $\mathsf{out}_u{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow$ undef; # *...and delete it locally*
end if
end transition

transition $\mathsf{corrupt}_u?$ (do) # *We get corrupted*
enabled if: $(ct =$ adaptive $\wedge (state_u \neq$ corrupted$) \wedge state_u \neq$ corrupted$)$
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (corrupt), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (corruptR);
$state_u \leftarrow$ corrupted;
output: $\mathsf{corOut}_u!$ (state, encode_state()), $\mathsf{corOut}_u{}^{\triangleleft}!$ (1);
end transition

function : encode_state()

for all $(key_{sid,grp} :: key_{sid,grp} \neq \mathsf{undef})$ **do** # *Perform delayed key computation*

output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);

input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);

$key_{sid,grp} \leftarrow key$;

end for

for all $(x_{sid,grp} :: x_{sid,grp} = \mathsf{exists})$ **do** # *Get real exponents*

output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getSecret, sid, grp), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);

input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getSecretR, $secret$);

$x_{sid,grp} \leftarrow secret$;

end for

return:$((G, g, \mathsf{h}),\ \{(v, state_v) \mid v \in \mathcal{M}\},\ \{(sid, grp, ses_{sid,grp}, \mathcal{C}_{sid,grp}, x_{sid,grp}, key_{sid,grp}) \mid sid \in \mathcal{SID} \wedge grp \subseteq \mathcal{M} \wedge ses_{sid,grp} \neq \mathsf{undef}\})$;

end function

transition $port?$ (any_msg) # *Transparent mode*

enabled if: $(state_u = \mathsf{corrupted}) \wedge (port \in \{\mathsf{in}_u?\} \cup \{\mathsf{aut}_{v,u}? \mid v \in \mathcal{M} \cup \{\mathsf{G}\}\})$;

output: $\mathsf{corOut}_u!$ $(port, any\_msg)$, $\mathsf{corOut}_u{}^{\triangleleft}!$ (1);

end transition

transition $\mathsf{corIn}_u?$ $(port, any\_msg)$ # *Transparent mode*

enabled if: $(state_u = \mathsf{corrupted})$

ignore if: $(port \notin \{\mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!\} \cup \{\mathsf{aut}_{u,v}! | v \in \mathcal{M} \cup \{\mathsf{G}\}\})$;

output: $port$ (any_msg);

end transition

Machine GDH_Mux

transition $\mathsf{in}_{\mathsf{gdhM},\mathsf{G}}?$ (param)

output: $\mathsf{in}_{\mathsf{gdh}}!$ (init), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);

input: $\mathsf{out}_{\mathsf{gdh}}?$ (initR, G, g, h);

output: $\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}!$ (paramR, G, g, h), $\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}{}^{\triangleleft}!$ (1);

end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (exp, sid, grp, γ)
  require: $(u \in grp) \ \wedge \ ((i_{sid,grp} = \mathsf{undef})$
    $\vee \ ((\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted}) \wedge (key_{sid,grp} = \mathsf{undef}))$
    $\vee \ ((\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted}) \wedge (\exists \beta : (\beta, \gamma) \in view_{sid,grp} \wedge$
    $\mathsf{bit}(\beta, \mathsf{idx}(grp, u)) = 0 \wedge \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) \neq 1^{|grp|})));$
    # A legitimate caller and either session is completely undefined or ses-
    # sion is corrupted but key is not yet divulged or session is uncorrupted
    # and query is for one of "our" partial keys.
  if $(i_{sid,grp} = \mathsf{undef})$ **then** *# New session*
    output: $\mathsf{in}_{\mathsf{gdh}}!$ (getView, $|grp|$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
    input: $\mathsf{out}_{\mathsf{gdh}}?$ (getViewR, $i, view$);
    $i_{sid,grp} \leftarrow i$; $view_{sid,grp} \leftarrow view$
    for all $(v :: corr_v = \mathsf{true})$ **do** $ses_{v,sid,grp} \leftarrow \mathsf{corrupted}$; **end for**
  end if
  if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** *# Session uncorrupted*
    $\beta' \leftarrow \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) :: (\beta, \gamma) \in view_{sid,grp}$; *# Index of exponentiation*
    output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (expR, $\gamma' :: (\beta', \gamma') \in view_{sid,grp}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
  else *# Group contains a corrupted participant*
    if $(secrets_{sid,grp} = \mathsf{undef})$ **then** *# Secrets not yet known*
      output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
      input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, $secrets$);
      $secrets_{sid,grp} \leftarrow secrets$;
    end if
    output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (expR, $\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}}$); $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
  end if
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (getKey, sid, grp, γ)
  require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}) \ \wedge$
    $(((\exists \beta : (\beta, \gamma) \in view_{sid,grp}) \wedge (\mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) = 1^{|grp|})) \ \vee$
    $((key_{sid,grp} = \mathsf{undef}) \wedge (\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted})));$
    # A legitimate caller of an initialized but unfinished session either ask-
    # ing for a correct key or being corrupted without somebody having
    # asked for the ideal key before
  if $key_{sid,grp} \neq \mathsf{undef}$ **then** *# (Ideal) key already defined...*
    # ...so just return this key
    $ses_{u,sid,grp} \leftarrow \mathsf{finished}$;
    output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
  else *# (Ideal) key does not yet exist and ...*
    if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** *# ...uncorrupted session*
      output: $\mathsf{in}_{\mathsf{gdh}}!$ (getKey, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
      input: $\mathsf{out}_{\mathsf{gdh}}?$ (getKeyR, key);
      $key_{sid,grp} \leftarrow key$;
      $ses_{u,sid,grp} \leftarrow \mathsf{finished}$; *# Mark only uncorrupted sessions as finished!*

output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
else # *Group contains corrupted participants and (ideal) key undefined*
if ($secrets_{sid,grp} = \mathsf{undef}$) **then** # *Secrets not yet known*
output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, $secrets$);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getKeyR, $\mathsf{h}(\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}})$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end if
end if
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (corrupt)
$corr_u \leftarrow \mathsf{true}$;
for all $(sid, grp :: (u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}))$
do
$ses_{u,sid,grp} \leftarrow \mathsf{corrupted}$; # *Mark only locally unfinished sessions*
end for
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (corruptR), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (getSecret, sid, grp)
require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} = \mathsf{corrupted}) \wedge (key_{sid,grp} = \mathsf{undef})$;
A legitimate caller of a started session and we are corrupted but the
key has not been exposed
if ($secrets_{sid,grp} = \mathsf{undef}$) **then** # *Secrets not yet known*
output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, $secrets$);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getSecretR, $secrets_{sid,grp,\mathsf{idx}(grp,u)}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition

Machine $\mathsf{GDH}^{(b=0)}_{n,mxkey}$

transition $\mathsf{in}_{\mathsf{gdh}}?$ (init)
enabled if: $(i = 0)$;
$(G, g, \mathsf{h}) \xleftarrow{\mathcal{R}} \mathsf{genG}(1^k)$;
$i \leftarrow 1$;
output: $\mathsf{out}_{\mathsf{gdh}}!$ (initR, G, g, h), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdh}}?$ (getView, n')
  enabled if: $(1 \leq i \leq mxkey)$; # *Initialized & maxima not exceeded*
  ignore if: $\neg(2 \leq n' \leq n)$; # *Illegal number of participants*
  $c_i \leftarrow \mathsf{init}$;
  $n_i \leftarrow n'$;
  $(x_{i,1}, \ldots, x_{i,n_i}) \xleftarrow{\mathcal{R}} \mathbb{Z}_{|G|}^{n_i}$;
  if $b = 0$ **then** # *Depending on type of machine ...*
    $z_i \leftarrow \mathsf{h}(g^{x_{i,1} \cdots x_{i,n_i}})$; # *... set real key ...*
  else
    $z_i \xleftarrow{\mathcal{R}} \{0,1\}^k$; # *... or random key.*
  end if
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getViewR, $i, \{(\beta, g^{\prod_{\beta_j=1} x_{i,j}}) \mid \beta \in I_{n_i} \setminus \{1^{n_i}\}\}$), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
  $i \leftarrow i + 1$;
end transition

transition $\mathsf{in}_{\mathsf{gdh}}?$ (getKey, i)
  ignore if: $(c_i \neq \mathsf{init})$; # *Session not yet initialized or already terminated*
  $c_i \leftarrow \mathsf{finished}$;
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getKeyR, z_i), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition

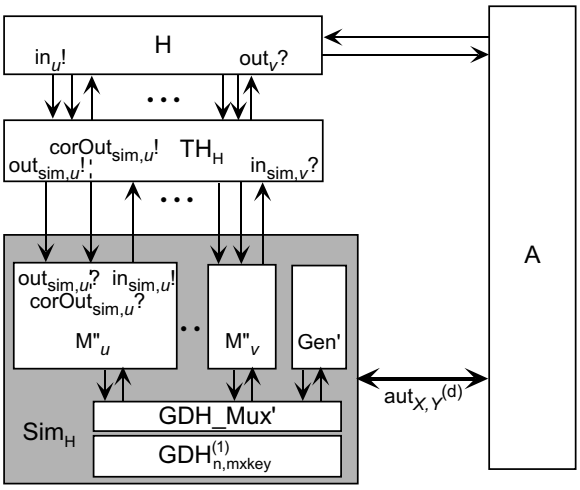
transition $\mathsf{in}_{\mathsf{gdh}}?$ (getSecret, i)
  ignore if: $(c_i \neq \mathsf{init})$; # *Session not yet initialized or already terminated*
  $c_i \leftarrow \mathsf{corrupted}$;
  output: $\mathsf{out}_{\mathsf{gdh}}!$ (getSecretR, $(x_{i,1}, \ldots, x_{i,n_i})$), $\mathsf{out}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
end transition □

Scheme 5.4' (Semi-ideal system $Sys^{\mathsf{gke,ika1,si}}_{n,tb,ct}$)

The same as $Sys^{\mathsf{gke,ika1,sr}}_{n,tb,ct}$ except with $\mathsf{GDH}^{(b=0)}_{n,mxkey}$ replaced by $\mathsf{GDH}^{(b=1)}_{n,mxkey}$.

□

Figure B.4 Simulator



Scheme 5.5 (Simulator for Scheme 5.2)

An overview of the simulator with its sub-machines and the overall connectivity is given in the grey box in Figure B.4. Gen' and $\mathsf{GDH}^{(1)}_{n,mxkey}$ are identical to their counterparts in the semi-ideal system and are not repeated here.

The message types and parameters of $\mathsf{GDH\_Mux}'$ are described in Table B.9. The message types and parameters of M''_u are described in the Tables B.8 and B.3. Note that contrary to M_u and M'_u (and contrary to the corresponding remark in the caption of Table B.3) the "upper" interface of M''_u corresponds to the messages at the adversary ports in Table B.1, not the specified ports. Furthermore, do not let you confuse by the implied renaming of "upper" interface ports since the usual $\mathsf{in}?$ ports (e.g., $\mathsf{in}_u?$) became now $\mathsf{out}?$ ports (e.g, $\mathsf{out}_{\mathsf{sim},u}?$) and vice versa. The variables of the machines M''_u and $\mathsf{GDH\_Mux}'$ are described in the Table B.10. The transitions of the machines are defined as follows, machine by machine:

Machine Gen'

See page 180 for the definition of the machine.

Machine M''_u

transition $\mathsf{out}_{\mathsf{sim},u}?$ (init) # *Trigger initialization*

enabled if: $(state_u = \mathsf{undef}) \wedge (\mathsf{in}_u?.cntr < tb)$;

$state_u \leftarrow \mathsf{wait}$;

output: $\mathsf{aut}_{u,\mathsf{G}}!$ (param);

Table B.8 The message types and parameters handled on "upper" interface of M''_u. (See Table B.3 for remaining messages.)

Port	Type	Parameters	Meaning
$\mathsf{out}_{\mathsf{sim},u}?$	init		User u is initializing.
$\mathsf{in}_{\mathsf{sim},u}!$	initialized	$v \in \mathcal{M}$	User u should consider user v as initialized.
$\mathsf{out}_{\mathsf{sim},u}?$	new	sid,grp,[sid',grp']	User u has initialized a new session.
$\mathsf{in}_{\mathsf{sim},u}!$	finish	sid,grp,[$key_{u,sim}$]	Let $\mathsf{TH}_{\mathcal{H}}$ complete the session for user u. If present and allowed, assign $key_{u,sim}$ to user u.
$\mathsf{corOut}_{\mathsf{sim},u}?$	state	*state*	$\mathsf{TH}_{\mathcal{H}}$'s state of corrupted party.
$\mathsf{out}_{\mathsf{sim},u}?$	*arbitrary*	*arbitrary*	Corrupted party u sent a message.
$\mathsf{in}_{\mathsf{sim},u}!$	*arbitrary*	*arbitrary*	Send message to (corrupted) party u.

Table B.9 The message types and parameters handled by $\mathsf{GDH\_Mux}'$

Port	Type	Parameters	Meaning
$\mathsf{in}_{\mathsf{gdhM},\mathsf{G}}?$	param	—	Get system parameters
$\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}!$	paramR	G, g, h	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	corrupt	*state*	Corruption
$\mathsf{out}_{\mathsf{gdhM},u}!$	corruptR	—	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	exp	sid, grp, γ	Exponentiate γ with secret for u in this session. Limited to the computation of partial keys!
$\mathsf{out}_{\mathsf{gdhM},u}!$	expR	γ^{x_u}	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	getKey	sid, grp, γ	Get derived key matching final partial key γ
$\mathsf{out}_{\mathsf{gdhM},u}!$	getKeyR	K	Reply to above
$\mathsf{in}_{\mathsf{gdhM},u}?$	getSecret	sid, grp	Get secret of this session (to hand it over during corruption)
$\mathsf{out}_{\mathsf{gdhM},u}!$	getSecretR	x_u	Reply to above

Table B.10 Variables in M''_u and $\mathsf{GDH\_Mux}'$

Name	Domain	Meaning	Init.
$(state_v)_{v\in\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{wait}, \mathsf{init}, \mathsf{corrupted}\}$	Long-term states as seen by M''_u.	undef
(G, g, h)	Range of $\mathsf{genG}(1^k)$	Global parameters.	—
$(ses_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{upflow}, \mathsf{downflow}, \mathsf{confirm}, \mathsf{finished}\}$	State of a (potential) session.	undef
$(\mathcal{C}_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathcal{I} \mid \mathcal{I} \subseteq \mathcal{M}\}$	Records received session confirmations	$\emptyset$
$(key_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup G \cup \{\mathsf{undef}, \mathsf{ideal}\}$	Group key of a session.	undef
$(x_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\mathbb{Z}_{\lvert G\rvert} \cup \{\mathsf{undef}, \mathsf{exists}\}$	Individual secret key of a session.	undef
$(\mathsf{aut}_{v,u}?.cntr)_{v\in\{\mathsf{G}\}\cup\mathcal{H}\setminus\{u\}}$	$\mathbb{N}$	Activation counters	0

Variables	Domain	Meaning	Init.
$(i_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\mathbb{N}$	Index used for this session with $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(corr_u)_{u\in\mathcal{M}}$	$\{\mathsf{true}, \mathsf{false}\}$	Corrupted machine?	true iff $u \in \mathcal{M}\setminus\mathcal{H}$
$(ses_{u,sid,grp})_{u\in\mathcal{M},sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{\mathsf{undef}, \mathsf{finished}, \mathsf{corrupted}\}$	Session status related to u	undef
$(key_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	$\{0,1\}^k \cup \{\mathsf{undef}, \mathsf{ideal}\}$	Session key from $\mathsf{GDH}^{(b)}_{n,mxkey}$	undef
$(view_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	View of a session	undef
$(secrets_{sid,grp})_{sid\in\mathcal{SID},grp\subseteq\mathcal{M}}$	As output by $\mathsf{GDH}^{(b)}_{n,mxkey}$	Secrets of a session	undef
$(\mathsf{in}_{\mathsf{gdhM},u}?.cntr)_{u\in\in\mathcal{M}\cup\{\mathsf{G}\}\}}$ $\mathsf{out}_{\mathsf{gdh}}?.cntr$	$\mathbb{N}$	Activation counters	0

output: $\mathsf{aut}^{\mathsf{d}}_{u,\mathsf{G}}!$ (param);
end transition

transition $\mathsf{aut}_{\mathsf{G},u}?$ (paramR, G', g', h') # *Get system parameters*
enabled if: ($state_u = \mathsf{wait}$);
$state_u \leftarrow \mathsf{init}$;
$(G, g, \mathsf{h}) \leftarrow (G', g', \mathsf{h}')$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (initialized, u); $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
for all $v \in \mathcal{M} \setminus \{u\}$ **do**
output: $\mathsf{aut}_{u,v}!$ (initialized);
output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (initialized);
end for
end transition

transition $\mathsf{aut}_{v,u}?$ (initialized) # *Notification for other machines*
enabled if: ($state_u \neq \mathsf{corrupted}$) $\wedge$ ($\mathsf{aut}_{v,u}?.cntr < tb$);
$state_v \leftarrow \mathsf{init}$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (initialized, v), $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{out}_{\mathsf{sim},u}?$ (new, sid, grp) # *Start new session*
enabled if: ($state_u \neq \mathsf{corrupted}$) $\wedge$ ($\mathsf{in}_u?.cntr < tb$);
ignore if:
$(u \notin grp) \vee (|grp| < 2) \vee (\exists v \in grp : state_v \neq \mathsf{init}) \vee (ses_{sid,grp} \neq \mathsf{undef})$;
$x_{sid,grp} \leftarrow \mathsf{exists}$;# *Just remember that exponent did not get erased yet*
$ses_{sid,grp} \leftarrow \mathsf{upflow}$;
if ($u = grp[1]$) **then** # *u is the first member*
$m'_1 \leftarrow g$;
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (exp, sid, grp, g), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (expR, $g^{x_{sid,grp}}$);
$m'_2 \leftarrow g^{x_{sid,grp}}$;
output: $\mathsf{aut}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[2]}!$ (up, sid, grp, (m'_1, m'_2));
$ses_{sid,grp} \leftarrow \mathsf{downflow}$;
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (up, sid, grp, msg) # *Upflow message arrives*
enabled if: ($state_u \neq \mathsf{corrupted}$) $\wedge$ ($\mathsf{aut}_{v,u}?.cntr < tb$);
ignore if: ($ses_{sid,grp} \neq \mathsf{upflow}$) $\vee$ ($v \neq grp[\mathsf{idx}(grp, u) - 1]$) $\vee$
(msg is not $(m_1, \ldots, m_{\mathsf{idx}(grp,u)})$ with $m_i \in G$ having maximal order);
$i \leftarrow \mathsf{idx}(grp, u)$; # *u's position in the group*
$m'_1 \leftarrow m_i$;
for $1 \leq j \leq \mathsf{min}(i, |grp| - 1)$ **do**

output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (exp, sid, grp, m_j), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (expR, $m_j^{x_{sid,grp}}$);
$m'_{j+1} \leftarrow m_j^{x_{sid,grp}}$;
end for
if $(i < |grp|)$ **then**
output: $\mathsf{aut}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
output: $\mathsf{aut}^{\mathsf{d}}_{u,grp[i+1]}!$ (up, sid, grp, $(m'_1, \ldots, m'_{i+1})$);
$ses_{sid,grp} \leftarrow$ downflow;
else # *$i = |grp|$, i.e., u is the last member*
$key_{sid,grp} \leftarrow m_{|grp|}$; # *Just remember the pre-key*
if $(ct =$ static$)$ **then** # *For the static case we are done*
$ses_{sid,grp} \leftarrow$ finished;
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (finish, $sid, grp, key_{sid,grp}$), $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
else # *For the adaptive case wait first for the confirmation flows*
$ses_{sid,grp} \leftarrow$ confirm;
$\mathcal{C}_{sid,grp} \leftarrow \{u\}$;
$x_{sid,grp} =$ undef; # *Erase secret exponent*
end if
for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" to the group members*
output: $\mathsf{aut}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
output: $\mathsf{aut}^{\mathsf{d}}_{u,v'}!$ (down, sid, grp, $(m'_1, \ldots, m'_i)$);
end for
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (down, sid, grp, msg) # *Downflow message arrives*
enabled if: $(state_u \neq$ corrupted$) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(ses_{sid,grp} \neq$ downflow$) \vee (v \neq grp[|grp|]) \vee$
$(msg$ is not $(m_1, \ldots, m_{|grp|})$ with $m_i \in G$ having maximal order);
$i \leftarrow \mathsf{idx}(grp, u)$; # *$u$'s position in the group*
$key_{sid,grp} \leftarrow m_{|grp|+1-i}$; # *Just remember the pre-key*
if $(ct =$ static$)$ **then** # *For the static case we are done*
$ses_{sid,grp} =$ finished;
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (finish, $sid, grp, key_{sid,grp}$), $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
else # *For the adaptive case, start confirmation*
$ses_{sid,grp} \leftarrow$ confirm;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{u, v\}$;

$x_{sid,grp} = \mathsf{undef}$; # *Erase secret exponent*
for all $v' \in grp \setminus \{u\}$ **do** # *"Broadcast" confirmation to group members*
output: $\mathsf{aut}_{u,v}!$ (confirm, sid, grp);
output: $\mathsf{aut}^{\mathsf{d}}_{u,v}!$ (confirm, sid, grp);
end for
if $(\mathcal{C}_{sid,grp} = grp)$ **then** # *We got* down *after all* confirm ...
$ses_{sid,grp} = \mathsf{finished}$; # *... so we are done: Give key to user ...*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (finish, $sid, grp, key_{sid,grp}$), $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow \mathsf{undef}$; # *... and delete it locally*
end if
end if
end transition

transition $\mathsf{aut}_{v,u}?$ (confirm, sid, grp) # *Confirmation message arrives*
enabled if: $(ct = \mathsf{adaptive}) \wedge (state_u \neq \mathsf{corrupted}) \wedge (\mathsf{aut}_{v,u}?.cntr < tb)$;
ignore if: $(v \notin grp \setminus \mathcal{C}_{sid,grp}) \vee (ses_{sid,grp} \notin \{\mathsf{downflow}, \mathsf{confirm}\})$;
$\mathcal{C}_{sid,grp} \leftarrow \mathcal{C}_{sid,grp} \cup \{v\}$;
if $(\mathcal{C}_{sid,grp} = grp) \wedge (ses_{sid,grp} = \mathsf{confirm})$ **then** # *All* confirm *received ...*
$ses_{sid,grp} \leftarrow \mathsf{finished}$; # *... so we are done: Give key to user ...*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;
output: $\mathsf{in}_{\mathsf{sim},u}!$ (finish, $sid, grp, key_{sid,grp}$), $\mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$ (1);
$key_{sid,grp} \leftarrow \mathsf{undef}$; # *... and delete it locally*
end if
end transition

transition $\mathsf{corOut}_{\mathsf{sim},u}?$ (state, $state_{\mathsf{TH}}$) # *We get corrupted*
enabled if: $(ct = \mathsf{adaptive} \wedge (state_u \neq \mathsf{corrupted}) \wedge state_u \neq \mathsf{corrupted})$
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (corrupt, $state_{\mathsf{TH}}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (corruptR);
$state_u \leftarrow \mathsf{corrupted}$;
output: $\mathsf{corOut}_u!$ (state, encode_state()), $\mathsf{corOut}_u{}^{\triangleleft}!$ (1);
end transition

function : encode_state()
for all $(key_{sid,grp} :: key_{sid,grp} \neq \mathsf{undef})$ **do** # *Perform delayed key computation*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getKey, sid, grp, $key_{sid,grp}$), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getKeyR, key);
$key_{sid,grp} \leftarrow key$;

end for
for all $(x_{sid,grp} :: x_{sid,grp} = \mathsf{exists})$ **do** # *Get real exponents*
output: $\mathsf{in}_{\mathsf{gdhM},u}!$ (getSecret, sid, grp), $\mathsf{in}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdhM},u}?$ (getSecretR, $secret$);
$x_{sid,grp} \leftarrow secret$;
end for
return:$((G, g, \mathsf{h}),\ \{(v, state_v) \mid v \in \mathcal{M}\},\ \{(sid, grp, ses_{sid,grp}, \mathcal{C}_{sid,grp},$
$x_{sid,grp}, key_{sid,grp}) \mid sid \in \mathcal{SID} \wedge grp \subseteq \mathcal{M} \wedge ses_{sid,grp} \neq \mathsf{undef}\})$;
end function

transition $port?$ (any_msg) # *Transparent mode*
enabled if: $(state_u = \mathsf{corrupted}) \wedge (port \in \{\mathsf{out}_{\mathsf{sim},u}?\} \cup \{\mathsf{aut}_{v,u}? \mid v \in \mathcal{M} \cup \{\mathsf{G}\}\})$;
output: $\mathsf{corOut}_u!$ $(port, any\_msg)$, $\mathsf{corOut}_u{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{corIn}_u?$ $(port, any\_msg)$ # *Transparent mode*
enabled if: $(state_u = \mathsf{corrupted})$
ignore if: $(port \notin \{\mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!\} \cup \{\mathsf{aut}_{u,v}! | v \in \mathcal{M} \cup \{\mathsf{G}\}\})$;
if $(port = \mathsf{out}_u!)$ **then**
$port \leftarrow \mathsf{in}_{\mathsf{sim},u}!$;# *Rename port to reflect name change in simulator*
else if $(port = \mathsf{out}_u{}^{\triangleleft}!)$ **then**
$port \leftarrow \mathsf{in}_{\mathsf{sim},u}{}^{\triangleleft}!$;# *Rename port to reflect name change in simulator*
end if
output: $port$ (any_msg);
end transition

Machine GDH_Mux′

transition $\mathsf{in}_{\mathsf{gdhM},\mathsf{G}}?$ (param)
output: $\mathsf{in}_{\mathsf{gdh}}!$ (init), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
input: $\mathsf{out}_{\mathsf{gdh}}?$ (initR, G, g, h);
output: $\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}!$ (paramR, G, g, h), $\mathsf{out}_{\mathsf{gdhM},\mathsf{G}}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (exp, sid, grp, γ)
require: $(u \in grp)\ \wedge\ ((i_{sid,grp} = \mathsf{undef})$
$\vee\ ((\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted}) \wedge (key_{sid,grp} = \mathsf{undef}))$
$\vee\ ((\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted}) \wedge (\exists \beta : (\beta, \gamma) \in view_{sid,grp} \wedge$
$\mathsf{bit}(\beta, \mathsf{idx}(grp, u)) = 0 \wedge \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) \neq 1^{|grp|})))$;
A legitimate caller and either session is completely undefined or ses-
sion is corrupted but key is not yet divulged or session is uncorrupted
and query is for one of "our" partial keys.

if $(i_{sid,grp} = \mathsf{undef})$ **then** # *New session*
output: $\mathsf{in_{gdh}}!$ (getView, $|grp|$), $\mathsf{in_{gdh}}^{\triangleleft}!$ (1);
input: $\mathsf{out_{gdh}}?$ (getViewR, $i, view$);
$i_{sid,grp} \leftarrow i$; $view_{sid,grp} \leftarrow view$
for all $(v :: corr_v = \mathsf{true})$ **do** $ses_{v,sid,grp} \leftarrow \mathsf{corrupted}$; **end for**
end if
if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** # *Session uncorrupted*
$\beta' \leftarrow \mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) :: (\beta, \gamma) \in view_{sid,grp}$; # *Index of exponentiation*
output: $\mathsf{out_{gdhM,u}}!$ (expR, $\gamma' :: (\beta', \gamma') \in view_{sid,grp}$), $\mathsf{out_{gdhM,u}}^{\triangleleft}!$ (1);
else # *Group contains a corrupted participant*
if $(secrets_{sid,grp} = \mathsf{undef})$ **then** # *Secrets not yet known*
output: $\mathsf{in_{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in_{gdh}}^{\triangleleft}!$ (1);
input: $\mathsf{out_{gdh}}?$ (getSecretR, $secrets$);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out_{gdhM,u}}!$ (expR, $\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}}$); $\mathsf{out_{gdhM,u}}^{\triangleleft}!$ (1);
end if
end transition

transition $\mathsf{in_{gdhM,u}}?$ (getKey, sid, grp, γ)
require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}) \wedge$
$(((\exists \beta : (\beta, \gamma) \in view_{sid,grp}) \wedge (\mathsf{setbit}(\beta, \mathsf{idx}(grp, u)) = 1^{|grp|})) \vee$
$((key_{sid,grp} = \mathsf{undef}) \wedge (\exists v \in grp : ses_{v,sid,grp} = \mathsf{corrupted})))$;
A legitimate caller of an initialized but unfinished session either ask-
ing for a correct key or being corrupted without somebody having
asked for the ideal key before
if $key_{sid,grp} \neq \mathsf{undef}$ **then** # *(Ideal) key already defined...*
...so just return this key
$ses_{u,sid,grp} \leftarrow \mathsf{finished}$;
output: $\mathsf{out_{gdhM,u}}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out_{gdhM,u}}^{\triangleleft}!$ (1);
else # *(Ideal) key does not yet exist and ...*
if $(\forall v \in grp : ses_{v,sid,grp} \neq \mathsf{corrupted})$ **then** # *...uncorrupted session*
$key_{sid,grp} \leftarrow \mathsf{ideal}$;
$ses_{u,sid,grp} \leftarrow \mathsf{finished}$; # *Mark only uncorrupted sessions as finished!*
output: $\mathsf{out_{gdhM,u}}!$ (getKeyR, $key_{sid,grp}$), $\mathsf{out_{gdhM,u}}^{\triangleleft}!$ (1);
else # *Group contains corrupted participants and (ideal) key undefined*
if $(secrets_{sid,grp} = \mathsf{undef})$ **then** # *Secrets not yet known*
output: $\mathsf{in_{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in_{gdh}}^{\triangleleft}!$ (1);
input: $\mathsf{out_{gdh}}?$ (getSecretR, $secrets$);
$secrets_{sid,grp} \leftarrow secrets$;
end if
output: $\mathsf{out_{gdhM,u}}!$ (getKeyR, $\mathsf{h}(\gamma^{secrets_{sid,grp,\mathsf{idx}(grp,u)}})$), $\mathsf{out_{gdhM,u}}^{\triangleleft}!$ (1);
end if
end if

end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (corrupt, $state_{\mathsf{TH}}$)
  for all $(sid', grp', ses_{u,sid',grp'}, key_{u,sid',grp'}, prev_{u,sid',grp'}) \in state_{\mathsf{TH}}$ **do**
    if $(key_{u,sid',grp'} \neq \mathsf{undef})$ **then**
      $key_{sid',grp'} \leftarrow key_{u,sid',grp'}$;
    end if
  end for
  $corr_u \leftarrow \mathsf{true}$;
  for all $(sid, grp :: (u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} \neq \mathsf{finished}))$ **do**
    $ses_{u,sid,grp} \leftarrow \mathsf{corrupted}$; # *Mark only locally unfinished sessions*
  end for
  output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (corruptR), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition

transition $\mathsf{in}_{\mathsf{gdhM},u}?$ (getSecret, sid, grp)
  require: $(u \in grp) \wedge (i_{sid,grp} \neq \mathsf{undef}) \wedge (ses_{u,sid,grp} = \mathsf{corrupted}) \wedge (key_{sid,grp} = \mathsf{undef})$;
  # *A legitimate caller of a started session and we are corrupted but the*
  # *key has not been exposed*
  if $(secrets_{sid,grp} = \mathsf{undef})$ **then** # *Secrets not yet known*
    output: $\mathsf{in}_{\mathsf{gdh}}!$ (getSecret, $i_{sid,grp}$), $\mathsf{in}_{\mathsf{gdh}}{}^{\triangleleft}!$ (1);
    input: $\mathsf{out}_{\mathsf{gdh}}?$ (getSecretR, $secrets$);
    $secrets_{sid,grp} \leftarrow secrets$;
  end if
  output: $\mathsf{out}_{\mathsf{gdhM},u}!$ (getSecretR, $secrets_{sid,grp,\mathsf{idx}(grp,u)}$), $\mathsf{out}_{\mathsf{gdhM},u}{}^{\triangleleft}!$ (1);
end transition

Machine $\mathsf{GDH}^{(b=1)}_{n,mxkey}$
  See page 189 for the definition of the machine.

□