Provably Secure Group Key Exchange

MARK MANULIS

Dissertation
for the Degree of Doktor-Ingenieur (Dr. Ing.)

Department for
Electrical Engineering and Information Technology
Ruhr University Bochum (Germany)

Network and Data Security (NDS) Group
Supervisor: Prof. Dr. rer. nat. Jörg Schwenk
External Referee: Prof. Dr. David Pointcheval

Bochum, 2007

Author contact information:
mark.manulis@gmail.com

This thesis was submitted to the *Department for Electrical Engineering and Information Technology* of *Ruhr University Bochum* on February 7, 2007 and defended on June 26, 2007.

Examination committee:

Prof. Dr.-Ing. *Heinz G. Göckler* (Ruhr University Bochum - committee chair)
Prof. Dr. rer. nat. *Jörg Schwenk* (Ruhr University Bochum - supervisor)
Prof. Dr. *David Pointcheval* (École Normale Supérieure Paris - external referee)
Prof. Dr.-Ing. *Ahmad-Reza Sadeghi* (Ruhr University Bochum)
Prof. Dr.-Ing. *Thomas Herrmann* (Ruhr University Bochum)

"The secret to strong security: less reliance on secrets."

Whitfield Diffie
(born 1944, pioneer of the public-key cryptography and
co-inventor of the fundamental key exchange protocol)

To my wife Katja.
To my parents Elena and Igor.
For their unconditional love and support.

Abstract

The rapid and promising development of applications and communication systems designed for groups of participants like groupware, computer supported collaborative work systems, or digital conference systems implies exigence of mechanisms providing adequate security properties. These mechanisms can be designed based on the foundations of cryptography. Group key exchange protocols are multi-party cryptographic protocols those participants compute a shared secret key that can then be used in conjunction with other cryptographic constructions like encryption schemes and message authentication codes for the purpose of privacy, confidentiality and authentication. Security confidence of modern cryptographic constructions can be increased via adequate security proofs. The paradigm of *provable security* gains in importance for all kinds of cryptographic constructions, including group key exchange protocols those security issues represent the scope of this dissertation. We give an analytical overview of the state-of-the-art research in this area and identify strengths and weaknesses of many previous approaches. We suggest a new approach in form of a security model those stronger definitions provide background for more confident security analyzes and proofs. Additionally, we present a number of generic solutions (compilers) that can be applied to independently designed group key exchange protocols in order to enhance security thereof with respect to various goals considered by our security model. Finally, we present a concrete group key exchange protocol that provably satisfies the apparently strongest currently available formally specified security requirements.

Abstract (in German)

Die schnelle und vielversprechende Entwicklung der für Gruppen von Teilnehmern konzipierten Anwendungen und Kommunikationssysteme, wie z.B. Groupware, Systeme für die computergestützte Gruppenarbeit oder digitale Konferenzsysteme, schafft die Notwendigkeit von Mechanismen zur Gewährleistung ausreichender Sicherheitseigenschaften. Der Entwurf dieser Mechanismen basiert größtenteils auf den Grundlagen der Kryptografie. Gruppen-Schlüssel-Austauschprotokolle (engl. group key exchange protocols) sind kryptografische Mehrparteien-Protokolle, die es den Teilnehmern ermöglichen, sich auf einen gemeinsamen geheimen Schlüssel zu einigen, der in Verbindung mit weiteren kryptografischen Verfahren, wie z.B. Verschlüsselungs- und Nachrichtenauthentisierungsverfahren, für die Geheimhaltung, Vertraulichkeit und Authentisierung eingesetzt werden kann. Das Vertrauen in die Sicherheit der modernen kryptografischen Verfahren kann heutzutage nur mittels eines ausreichenden Sicherheitsbeweises erzielt werden. Das Paradigma der *beweisbaren Sicherheit* gewinnt immer mehr an Bedeutung für alle kryptografischen Verfahren, einschließlich der Gruppen-Schlüssel-Austauschprotokollen, deren Sicherheitsaspekte den Hauptbestandteil dieser Dissertation darstellen. Wir geben einen analytischen Überblick über den aktuellen Stand der Forschung in diesem Bereich und heben Stärken und Schwächen vieler bekannter Ansätze hervor. Ausgehend von den durchgeführten Analysen und gefundenen Mängeln der vorhandenen Ansätze und Verfahren schlagen wir einen neuen Ansatz vor, in Form eines Sicherheitsmodells für Gruppen-Schlüssel-Austauschprotokolle, das weitaus stärkere Sicherheitsanforderungen umfasst und die Basis für tiefere Sicherheitsanalysen und Beweise bereit stellt. Zusätzlich, stellen wir eine Reihe von allgemeinen Lösungen zur Verbesserung der Sicherheit (im Rahmen unseres Modells) von unabhängig konzipierten Gruppen-Schlüssel-Austauschprotokollen vor. Abschließend beschreiben wir ein neues Gruppen-Schlüssel-Austauschprotokoll, dessen beweisbare Sicherheit den derzeit stärksten formalen Anforderungen entspricht.

Acknowledgements

This dissertation is the result of three years of research in the area of provable security and group key exchange protocols during my work as researcher at Horst Görtz Institute for IT Security (HGI) at Ruhr University Bochum (RUB). Though writing it was sometimes a lonely and isolated process, full of sleepless nights, yet it could never be completed without professional and emotional support of many wonderful people I want to thank.

First of all I want to express my deepest gratitude to my supervisor and the head of network and data security (NDS) group at RUB, *Prof. Jörg Schwenk*. His guidance, encouragement, and vision were of great help for accomplishing this serious undertaking. I am also thankful to Jörg for giving me freedom of choosing own research directions, reading my articles, commenting on my views and helping me enrich my ideas.

I am grateful to my defence committee, especially to *Prof. David Pointcheval*, the head of crypto team at École Normale Supérieure (ENS) in Paris, for taking the time to review my dissertation and travel to Bochum for my defence as well as for giving me valuable suggestions and comments.

Further, I would like to thank *Prof. Ahmad-Reza Sadeghi*, the head of system security group at RUB, for his encouragement, motivation, and practical advice, as well as for the joint work on various cryptographic protocols resulting in several published papers. In particular, I admire Ahmad-Reza for his high efficiency in work and extraordinary commitment to research.

Special thanks go to *Emmanuel Bresson* from Direction Centrale de la Sécurité des Systèmes d'Information (DCSSI Crypto Lab) in Paris for numerous discussions and intensive joint work on security models and protocols for group key exchange as well as for the time spent with me during my research visit at CELAR / Université de Rennes in June 2006.

Additionally, I would like to thank *Christian Cachin* from IBM Research Lab in Zürich, and once again *Emmanuel Bresson*, *Prof. David Pointcheval*, and *Prof. Jörg Schwenk* for their active support and participation in the organization of the 1st International Workshop on Group-Oriented Cryptographic Protocols (GOCP) in July 2007.

During my participation in the ECRYPT NoE project (PROVILAB group) I had a pleasure to meet many wonderful people whom I want to thank for the interesting discussions and valuable comments on my work and simply for the nice time spent during the project meetings, especially *Olivier Pereira* from Université Catholique de Louvain-la-Neuve (UCL) in Belgium, *Prof. Jesper Buus Nielsen* from Århus Universitet in Denmark, *Prof. Giuseppe Persiano* and *Prof. Ivan Visconti* from Università degli Studi di Salerno (UNISA) in Italy, *Prof. Berry Schoenmakers* from Technische Universiteit Eindhoven (TU/e) in the Netherlands, and once again our PROVILAB leader *Christian Cachin*.

My participation in the UbiSec&Sens project gave me opportunity to learn many high quality researchers in the area of wireless communication security. Thanks to many fruitful discussions with *Dirk Westhoff* and *Alban Hessler* from NEC Labs in Germany, *Claude Castelluccia* from INRIA in France, and *Prof. Levente Buttyán* from Budapest University of Technology and Economics in Hungary I could get quick on board and provide own contribution to the project.

Further, I am thankful to my colleagues and friends at NDS for their support and creative working atmosphere. Special thanks to the colleagues from the "first hour": *André Adelsbach* (meanwhile at Telindus, Luxembourg) for commenting my first scientific papers, *Ulrich Greveler* (meanwhile professor at Fachhochschule Münster) for interesting discussions, cheerful disposition as well as unforgettable barbecue parties and whiskey-tasting evenings, *Michael Psarros* for various practical security tips and delicious self-cooked Greek food, *Sebastian Gajek* for the discussions on future research directions during our "everyday after-dinner" coffee breaks as well as occasional cocktail parties, *Lijun Liao* for having chosen me to advise his Master thesis before joining NDS, which resulted in several publications, *Petra Winkel* for keeping a number of administrative tasks off my shoulders, *Klaus Skudlarski* and *Jürgen Weide* for providing various technical support and supply on hardware and software. Not to forget about *Sven Schäge*, *Kristina Altmann* and *Tibor Jager* who joined our group a short time ago, and *Alexander Amelin* who spent a year at NDS as a visiting researcher.

Also my thanks go to the colleagues from other research groups at HGI. Especially, to *Prof. Christoph Paar*, head of the communication security group at RUB, for his general interest in my work, *Andre Weimerskirch* (meanwhile at escrypt GmbH) and *Axel Poschmann* for the smooth joint work on the UbiSec&Sens project and the nice time spent in Grenoble and Budapest during the project meetings, *Hans Löhr* and *Alberto Escalante* for fruitful discussions and joint work on unsplittable multi-coupon schemes, and *Jan Pelze* (meanwhile at escrypt GmbH) for the joint professional courses on cryptography for the specialists from the industry.

I wish all researchers at HGI all the success in their hard work and achievement of further excellent scientific results.

Finally, I am grateful to *Prof. Dietmar Wätjen* from Technical University Braunschweig and *Prof. Stefan Fischer* from University of Lübeck for their interesting lectures on cryptography and IT security during my graduate studies and supervision of my Master thesis.

Last but not least I would like to thank my wife, my parents, and other family members for their unconditional love and support.

Mark

Contents

List of Figures

List of Tables

List of Symbols and Notations

$\mathbb{N}$, $\mathbb{R}$	a set of natural numbers (≥ 0), a set of real numbers
$\mathbb{Z}_p$, $\mathbb{Z}_p^*$	a set of integers modulo $p \in \mathbb{N}$, i.e. $\{0, \ldots, p-1\}$, a multiplicative group of $\mathbb{Z}_p$
$\mathbb{G}$	cyclic (multiplicative) group with generator g
$\in$, $\in_R$	choice, random choice
$[1, n]$	interval between 1 and n
$\equiv$, $\Longleftrightarrow$	equivalence relationship
$\stackrel{?}{=}$	equality check or verification
$:=$, $\leftarrow$	value assignment
$x \leftarrow f(y)$	x is the output of $f()$ on input y
$\mid$	divisor of, e.g. $q \mid p-1$ for q divides $p-1$, or concatenation of (binary) strings, e.g. $x \mid y$ (context-dependent)
$\lvert x \rvert$	size or length of x or absolute value of x (context-dependent)
$\oplus$	XOR operator
$\mod$	modulo operator
$\Pr[\mathtt{X}]$	probability of event $\mathtt{X}$
$\Pr[\mathtt{A}_1; \ldots; \mathtt{A}_n : \mathtt{X}]$	probability of event $\mathtt{X}$ after the sequential occurrence of $\mathtt{A}_1, \ldots, \mathtt{A}_n$ (used mostly in definitions of security goals)
$\mathsf{Game}_F^{\mathsf{prf}-b}(\kappa)$	adversarial game specified by bit $b \in \{0,1\}$ between a PPT algorithm $\mathcal{A}$ in the attack against the pseudo-randomness of a function ensemble F with the security parameter κ

$\mathsf{Game}^{\mathsf{(a)ke}-b}_{\alpha,\beta,\mathtt{P}}(\kappa)$	adversarial game specified by bit $b \in \{0,1\}$ between a PPT algorithm $\mathcal{A}$ in the attack against the (A)KE-security of a group key exchange protocol $\mathtt{P}$ with the security parameter κ and the secrecy type α; (α,β) denotes the adversarial setting against the (A)KE-security of $\mathtt{P}$
$\mathsf{Game}^{\mathsf{ma}}_{\mathtt{P}}(\kappa)$	adversarial game between a PPT algorithm $\mathcal{A}$ in the attack against the MA-security of a group key exchange protocol $\mathtt{P}$ with the security parameter κ
$\mathsf{Game}^{\mathsf{con}-t}_{\mathtt{P}}(\kappa)$	adversarial game between a PPT algorithm $\mathcal{A}$ in the attack against the t-contributiveness of a group key exchange protocol $\mathtt{P}$ with the security parameter κ
$\mathsf{Adv}^{\mathtt{req}}_{\mathtt{cc}}(\kappa)$	advantage probability of a PPT adversarial algorithm $\mathcal{A}$ in the attack against the security requirement $\mathtt{req}$ of a cryptographic construction $\mathtt{cc}$ with the security parameter κ
$\mathsf{Succ}^{\mathtt{req}}_{\mathtt{cc}}(\kappa)$	success probability of a PPT adversarial algorithm $\mathcal{A}$ in the attack against the security requirement $\mathtt{req}$ of a cryptographic construction $\mathtt{cc}$ with the security parameter κ
$U, \mathcal{U}, \mathcal{G}$	user (protocol participant, group member), set of all possible protocol participants (users, potential group members), set of actual protocol participants (group members), i.e. $\mathcal{G} \subseteq \mathcal{U}$
Π^s_U, $\Pi^{s_i}_i$	a general notation for the s-th instance (oracle) of user U, a notation for the s_i-th instance of one particular $U_i \in \mathcal{U}$
$\mathtt{sid}^s_U$, $\mathtt{pid}^s_U$, k^s_U	session id, partner id, secret key of Π^s_U
LL_i, (sk_i, pk_i)	long-lived (long-term) key of U_i, private/public key pair of U_i
T_n, $\mathcal{T}_n$	binary tree with n leaf nodes, set of all binary trees with n leaf nodes, i.e, $T_n \in \mathcal{T}_n$
d_{T_n}	depth of T_n
$\langle l, v \rangle$	label of a node in T_n where l ($0 \leq l \leq d_{T_n}$) denotes node's level and v ($0 \leq v \leq 2^l - 1$) its position within this level, $\langle 0, 0 \rangle$ denotes the root node

Overview and Organization

Part I provides general information taking the reader beyond the actual scope of the dissertation while assuming some basic knowledge of security and cryptography. In particular it provides introduction to group applications and their security issues, describes variants of group key establishment techniques, outlines the notion of "provable security" in the context of cryptographic constructions, specifies objectives of the dissertation, and summarizes its research contributions. Part I consists of the following chapters:

- Chapter 1 provides a number of examples of widely deployed group applications and describes general aspects of their security. Additionally, it focuses on the notion of group membership and gives insight into the management of the group formation.

- Chapter 2 classifies techniques of group key establishment and specifies main differences between the two major approaches – group key transport/distribution and group key exchange/agreement – from the perspective of security requirements and trust relationship whereby the description of security requirements is kept general. Finally, this chapter describes the general idea behind generic security solutions – security-enhancing compilers – and motivates their importance for group key establishment protocols.

- Chapter 3 gives a brief introduction on the issue of *provable security* and specifies how it is currently understood in cryptography. Its general description focuses on the three main classes – information-theoretic, computational, and symbolic security – and is carried out in the context of group key establishment protocols. The high-level description considers different aspects of security proofs and applied techniques including well-known auxiliary methods of the Random Oracle Model and Ideal Cipher Model.

- Chapter 4 lists main objectives of the dissertation using the terminology introduced in the previous chapters and summarizes main research contributions and obtained results. It also provides more detailed overview of single chapters of the second part than the one given in the following.

Part II is the main part of the dissertation and focuses on the issues of provable security in the context of cryptographic group key exchange protocols. In particular it provides detailed description of the state-of-the-art research in this area and contributes in different ways ranging from the theoretical aspects concerning modeling of group key exchange protocols and their security requirements up to practical issues including new protocols and generic constructions with the adequate proofs of security. Part II consists of the following chapters:

- Chapter 5 provides theoretical background on cryptography while being limited to the issues used throughout the dissertation.

Chapter 6 describes the state-of-the-art of the theoretical research on security requirements and security models for group key exchange protocols and draws conclusions that provide background for further theoretical research concerning security models for group key exchange protocols.

- Chapter 7 describes the state-of-the-art of the practical research considering currently known group key exchange protocols and draws conclusions that provide background for the further practical research on new group key exchange protocols and generic security-enhancing solutions.

- Chapter 8 provides the main theoretical contribution of this dissertation in form of an advanced security model for group key exchange protocols that provides extended security requirements considering a stronger adversarial setting compared to all currently available security models.

- Chapter 9 provides our first practical research contribution in form of several generic techniques – compilers – that can be used to enhance security of independently designed group key exchange protocols. For each compiler (in total seven) we provide confidence by proving its security in our theoretical model under standard cryptographic assumptions.

- Chapter 10 describes our second practical research contribution in form of a new group key exchange protocol `TDH1` which provides extended security properties compared to all currently known protocols. We describe two versions of the protocol for two different kinds of groups (static and dynamic). Additionally, we provide a formal treatment of the cryptographic assumption `TDDH` used in the security proof of `TDH1` and show that it can be considered as a standard cryptographic assumption.

Part I

Group Applications and Key Establishment
– Introduction on Security Issues and Dissertation Focus –

Chapter 1

Group Applications and Security

This chapter provides introduction to group applications and related security issues. Its purpose is to show that security in group applications is a large topic of research where cryptography and its techniques play a very important role.

1.1 Group Applications

The invention of integrated circuits and microprocessors in the second half of the last century and subsequent digital revolution marked a significant progress in information processing and gave every individual additional intelligent auxiliary tools in form of digital devices that are able to analyze information and prepare it for private and business use.

In combination with networking, and especially, after the establishment of the Internet as mass communication media, digital technology opened new extensive possibilities for private and industrial information retrieval and exchange, pushed innovative business models and simplified work processes within enterprizes. Recent developments in wireless technology and portable devices provide additional flexibility in handling of information and interaction with the environment. It is obvious that “connectivity” means nowadays much more than simply bringing two or more individuals or systems together.

It is reasonable to expect that in near future thanks to further hardware and software developments as well as new communication standards people and systems will remain in contact with each other regardless of time and location. This makes a very convenient atmosphere for the deployment and progress of multi-party applications and all kinds of joint information processing.

Group applications represent a special case of multi-party applications where users (group members) have some unifying relationship concerning their rights, responsibilities and application goals. In the following we list some well-known forms of group applications without claiming completeness. Interesting information on group applications can be also obtained in the literature, e.g. proceedings of GROUP, CSCW, and ECSCW conferences, and via relevant websites.

1.1.1 Digital Conferences

Digital conferences like audio/video/web conferences [187] are popular, especially in companies, since they allow a number of participants at different locations to arrange meetings and collective discussions at low cost compared to those with physical presence of participants. These real-time applications are useful in many communication scenarios, for example in business scenarios digital conferences can be used by enterprizes with many branches spread over different geographic locations for strategic discussions and reports, or in battlefield scenarios they can be used for efficient coordination of the operation development.

1.1.2 Text-Based Group Communication

The most popular example of applications for the text-based group communication are surely *chat systems*. These real-time applications allow a set of users to write messages in a public or closed space. Many chat systems provide options for admission control and moderation. New chat participants are usually able to observe past communication.

Another prominent example of the text-based group communication are *discussion forums*. The main difference to the chat systems is that the communication is asynchronous and participants can leave and return later without missing parts of the communicated information.

Newsgroups and *mailing lists* are further favorite applications for information distribution. The main difference to the previous examples is that subscription of participants is required prior to the information receival. Mailing lists are often used in business scenarios to address groups of employees of a company or inform customers about new developments.

1.1.3 File and Data Sharing

File and data sharing systems allow sharing and exchange of digital information among a group of users. The most prominent example is a distributed approach in form of peer-to-peer file sharing systems [10] where each user typically stores data intended for sharing in his own device and provides other users with access rights to this data.

There exist also centralized approaches like OpenNAP [1] where information is stored at some central logical or physical location, usually on a (distributed) database server, which can be accessed by legitimate users who may add, alter or simply retrieve the stored information.

Note that *revision control* and *concurrent versions systems* [149, 179] which are frequently used in companies and organizations to keep consistency of the information that may be modified concurrently by several users also belong to the class of file and data sharing applications.

1.1.4 Replication and Synchronization Systems

Sometimes it is desirable to store the same data at different physical locations, e.g. distributed server systems, in order to achieve higher accessability and reliability compared to the centralized data storage with a single-point-of-failure. For example, different “mirror” web servers

are often used to ensure access to the information in case that some of them are temporarily not accessible or to decrease access time by balancing the number of requests to each server. Obviously, it is important to ensure consistency of the distributed contents. *Replication* and *synchronization systems* [178] provide automatic mechanisms that allow to keep consistency of the distributed content upon occurring changes.

1.1.5 CSCW Systems and Groupware

Computer supported collaborative work (CSCW) systems and *groupware* applications [100, 101] allow a group of participants, each using own auxiliary digital device, to process some joint task. Often CSCW systems and groupware applications are used to optimize workflow management within a company. These applications consist usually of many different components each designed for some special purpose. Some of these components enable group communication via digital conferences [153] or text-based communication as described above.

However, there is a number of components which distinguish CSCW systems and groupware applications from the others, e.g. *multi-user editors* that can be used for the collective editing of documents. To this category belong also applications like shared white-boards and shared screens working based on the WYSIWIS (What-You-See-Is-What-I-See) principle [172].

Furthermore, *computer supported cooperative learning (CSCL)* systems [183] and *group decision support (GDS)* systems [155] also belong to this category.

1.2 Membership Dynamics

Some group applications assume that there exists a group $\mathcal{G}$ encompassing a set of members $U_1, \ldots, U_n$. However, this group may have its own dynamics. The number of members may remain constant or change over the whole execution period of the application. Obviously, it depends on concrete scenarios whether such changes have to be considered or not.

For example, if the executive board of an enterprise organizes a digital conference with the managers of all its branches then it is likely that all group members are known in advance and the group remains constant during the whole conference period. On the other hand if a chat system or a discussion forum is set up in some public space then it is likely that the number of participants will change and that these changes are rather spontaneous. A groupware application can be used for a project development within a company with some fixed number of employees assigned to the project or used within some project that welcomes new participants, e.g., open source projects. In the latter case it is desirable to have mechanisms that handle dynamic group changes.

Most of the group applications are designed and implemented in a way that allows them to handle changes of the group formation. The following terminology is frequently used in the literature to distinguish between groups with respect to the behavior of their members.

Static groups are groups where the initial group size remains constant over the life-time of the group. In a group application this means that the number of users does not change until the application is finished. Usually, in static groups participants are known in advance. On the other hand, *dynamic groups* allow members to join or leave the group. Sometimes, dynamic groups are built spontaneously so that application users may not be known in advance.

It is worth being mentioned that group membership dynamics may have consequences on the efficiency of the application and also on its security. It is a wide-spread opinion that higher

efficiency can be usually achieved at the expense of lower security and vice versa, so that in practice one tries to find a reasonable trade-off between the both issues.

1.3 Security Issues in Group Applications

Beside usability, reliability, and cost/performance ratio *security* is a major factor for a broader acceptance of an application and its successful deployment once put on the market. Security becomes especially important in case that the application is used in critical scenarios concerning military, politics, economics, etc.

In group applications, security is increasingly important since these applications involve many users and each of them can be considered as a possible point of the attack (or even the attacker himself). It is a challenging task to guarantee security if users are equipped with diverse digital devices, located in different networks with inhomogeneous and sometimes insufficient protection mechanisms, or have different experience with the application which may lead to additional security risks.

Surely, each group application may have its own security requirements concerning access control, communication, information storage and processing. However, most of them suit in the following three common classes of the information security requirements - *confidentiality*, *authentication*, and *integrity* [135, Chapter 1].

One of the main techniques to achieve these requirements is *cryptography*. However, cryptography alone is not sufficient. Especially, in multi-user and group applications assumptions on the *trust* relationship between participants is also of major importance. For each security requirement it is important to specify the underlying trust assumption, and in most cases security can only be guaranteed if all users during the application execution do not deviate from the stated trust relationship.

1.3.1 Confidentiality

Confidentiality is a requirement on "keeping information secret from all but those who are authorized to see it" [135, Chapter 1]. Confidentiality is often used as a synonym for *privacy* and *secrecy*, and is related to information hiding.

In many group applications confidentiality plays an important role. Especially, in scenarios where a group application is used for the purpose of the exchange or storage of sensitive information which requires protection.

Cryptography provides appropriate methods in form of encryption mechanisms to achieve confidentiality of the exchanged and stored information. There are two major types of encryption mechanisms - those based on *symmetric cryptography* and those based on *asymmetric* or also known as *public-key cryptography*.

In *symmetric encryption schemes* [135, Chapters 6,7] all authorized users share some secret value (*secret key*) k which serves as input to a symmetric encryption scheme for the purpose of information encryption and decryption. Hence, in group applications symmetric encryption schemes can be used if all authorized group members know the shared secret key k prior to the execution of the application.

In *asymmetric encryption schemes* [135, Chapter 8] each user U_i has an individual key pair (sk_i, pk_i) consisting of a public key pk_i and a mathematically related private key sk_i. If a user U_i is supposed to decrypt a message then this message should have been previously encrypted using the public key pk_i. For the decryption U_i applies own private key sk_i. Thus, in order to

exchange information securely within a group this information has to be encrypted independently with the public key of each user. Obviously, this is less practical than using symmetric encryption schemes which require information to be encrypted only once. Another approach is to let group members share the same private/public group key pair $(sk_{\mathcal{G}}, pk_{\mathcal{G}})$. However, this implicitly assumes that users share a secret value (in form of the private group key $sk_{\mathcal{G}}$), and can, therefore, derive the shared key k required for symmetric encryption schemes. Note also that symmetric encryption is more efficient than public-key encryption.

Thus, symmetric cryptography is more suitable to achieve confidentiality and privacy in group applications, and in this case it is essential that group members learn the required secret key k.

1.3.2 Authentication

Authentication has its roots in Greek and means "real" or "genuine". In information security, authentication is specific to some certain objective. Cryptography distinguishes between identification or entity authentication on the one hand, and data origin authentication or message authentication on the other hand [135, Chapter 1]. Beside this group applications may have additional authentication goals which we also mention in the following.

Identification and Entity Authentication

In case of *identification* or *entity authentication* a user U_j is assured of the identity of another user U_i through the obtained information and that U_i was actively involved in the issue of this information. An additional requirement called *non-repudiation* enables U_j to prove this fact to some third party.

Identification and entity authentication with non-repudiation can only be achieved using techniques of public-key cryptography, in particular via digital signature schemes. A *digital signature scheme* [135, Chapter 11] consists of a key generation algorithm that outputs a private/public key pair (sk, pk), a signature generation algorithm and an associate verification algorithm. The signature generation algorithm takes as input binary data m which should be signed and a private key sk known only to the signer. The verification algorithm takes as input the binary data m, a candidate signature σ and the public key pk of the potential signer and checks whether the verification is successful or not. In group applications each user U_i may have own private/public key pair (sk_i, pk_i) and thus digitally sign information intended for other group members in case that entity authentication is required by the application.

Data Origin Authentication, Message Authentication, Group Authentication

In case of *data origin authentication* or *message authentication* a user U_j is only assured of the identity of another user U_i without being able to prove that U_i issued this information. Thus, message authentication does not necessarily provide non-repudiation.

Message authentication can be achieved using techniques of symmetric cryptography, especially so-called *message authentication codes* (MACs) [135, Chapter 9] are often used for this purpose. A MAC is a symmetric cryptographic algorithm which takes as input a binary data m of an arbitrary length which should be authenticated and a secret key k, and outputs a binary string μ (*MAC value*). The main security requirement on a MAC is that given zero or more data/MAC value pairs (m, μ) it should be computationally infeasible to compute another pair for some different data $m' \neq m$. Obviously, this requirement also implies that the secret key k

may not be recovered from one or more data/MAC value pairs. Data origin can be easily verified by recomputing a MAC value and comparing it to the original one. Note that in order to recompute the MAC value it is essential to know the secret key k.

In two-user applications it is intuitively clear that if both users U_1 and U_2 share a secret key k and one of them obtains information together with a valid MAC value but is not the issuer of this information then the second user must be the issuer (as long as any third party does not obtain the shared key). Thus, in two-user applications MACs can also be used to identify the information issuer (without provability of this fact). Obviously, it is of great importance that other parties do not obtain the secret key; otherwise, they can produce MAC values accepted by both users. Thus, users have to trust each other not to reveal the shared key. Interesting is that if the key is revealed then it must have been done by at least one of the both users (as long as the MAC scheme is secure). Obviously, if one user, say U_1, remains honest then it must be another user U_2 who has misused the trust. This may cause honest user U_1 to cancel further cooperation with the dishonest U_2.

In group applications, however, we have a completely different situation. In case that the secret key is shared between a group of users $\mathcal{G} := \{U_1, \ldots, U_n\}$, $n > 2$, it is no more possible to identify the issuer since any user $U_i \in \mathcal{G}$ is a potential issuer. Hence, in group applications the notions of data origin authentication and message authentication reduce to the so-called requirement of *group authentication* [58] that guarantees only that the information is originated or altered by a member of the group who knows the corresponding secret key. Thus, in case that the application requires identification within the group then members have to use digital signatures. Also, it is not possible to identify dishonest users if they misuse the trust and reveal shared keys. Thus, in group applications the described trust relationship is more important than in two-party applications.

Note also that group authentication is also implied in case that a symmetric encryption scheme is used by group members for the purpose of confidentiality, since only group members in possession of the shared key are able to encrypt and decrypt information.

1.3.3 Integrity

Integrity has its roots in Latin and means “wholeness” or “completeness”. Information integrity is a security condition that the information remains identical during any operation, such as transfer, storage, or processing [135, Chapter 1].

Information integrity in group applications may have certain constraints depending on the required trust relationship. For example, in multi-user editors all authorized users are allowed to alter shared documents. In the text-based communication scenarios it is, however, desirable that exchanged messages are transferred to the users without being modified. In the following we overview some cryptographic methods that can be used to achieve various kinds of integrity in group application scenarios.

Collision-resistant hash functions [135, Chapter 9] can be used to achieve information and data integrity in its plain form (without additional confidentiality or authentication). These functions map binary strings of arbitrary length to binary strings of fixed length such that it is computationally infeasible to find two or more different input strings that result in the same output string (*hash value*). In order to provide information and data integrity it is necessary to protect the hash value χ of the original information. Any attempt to alter this information would be noticed by comparing the current hash value with χ. Hash functions are usually publicly known and there is no secret information needed to evaluate them (this in contrast to MACs which

are sometimes called *keyed hash functions*). Therefore, hash functions can be applied for the purpose of the integrity control only if a user knows the original hash value of the information.

This method can be used in group applications like file sharing or shared data storage in case that only authorized users are allowed to alter the stored information. In order to achieve integrity for any authorized modification of the information the updated hash value χ' has to be stored in addition to the modified data. Moreover, the application must ensure that integrity is provided for that hash value too, and that only authorized users are allowed to alter it. Thus, in group applications information integrity based on hash functions can only be achieved if applications provide additional mechanisms for the authorized access control to the data and associated hash values or if users trust each other not to alter information added by other users.

There is a number of cryptographic methods to achieve information integrity in conjunction with the previously described requirements of confidentiality, authentication, or both of them. For example, encryption schemes provide integrity for the encrypted information, however, only in case that a user is able to verify that the decrypted information is a correct one, i.e., expected by the user or satisfies some certain pattern. This follows from the security requirements on encryption schemes stating that any modification of the ciphertext would necessarily result in a different plaintext. Also, the kind of the encryption scheme used and the trust relationship between group members has consequences on the integrity of the original information. If a symmetric encryption scheme is used then any user U_i in possession of the secret key k is able to alter data by re-encryption such that other users who decrypt this data later are not able to notice its alteration. Thus, integrity of the original information in case that a symmetric encryption scheme is used can be achieved only if users trust each other not to modify information issued by other users. The same requirement holds for the asymmetric encryption schemes in case that a single private/public key pair $(sk_{\mathcal{G}}, pk_{\mathcal{G}})$ is used for the whole group. Only using individual private/public key pairs (sk_i, pk_i) prevents any modification of the original information by any other user $U_{j\neq i}$. Also, digital signatures (usually in combination with a hash function) and message authentication codes imply integrity of the information. The only difference is that if MACs are used then there are the same constraints as for the symmetric encryption schemes with respect to the alteration of the original information by other group members and the required trust relationship between group members.

1.4 Group Membership

In the previous sections we focused on group applications and their general security requirements and used termini like “group membership” or “authorized user”. Obviously, it is important to specify how a user U_i becomes member of a group $\mathcal{G}$ and how group members can distinguish whether a user belongs to the group or not?

1.4.1 Group Admission and Membership Control

The task of group membership assignment is handled by so-called *group admission and membership control* mechanisms [58, 115].

A group admission and membership control mechanism is a process or protocol whereby a user obtains or looses group membership and is able to prove its possession to other group members or third parties depending on the application scenario. The expression *membership revocation* is often used to express the loss of the group membership. Note that group membership may be revoked for a variety of reasons depending on concrete application scenarios.

1.4.2 Group Authority

If a group is not open for the public use there are always some conditions which have to be satisfied by an object or entity to obtain group membership. In dynamic groups a member may also be excluded from the group or simply leave it.

The first main question in this context is who decides which conditions have to be satisfied to obtain or loose group membership and who affirms or revokes it? This power is given to the so-called *group authority* (GA) [115] which can be either *centralized* or *distributed*.

A centralized GA assumes the existence of one party that is solely in charge for the admission to the group and revocation of the group membership. This party is often called *group controller* or *group owner* [58, 115]. It can be either a special member of the group, or even some third trusted party (TTP). The group controller alone specifies the required membership conditions, approves the group membership and revokes it.

A distributed GA consists of several parties, usually a subset of group members or even the whole group. Thus, group members have to negotiate conditions for the approval or revocation of the group membership, and are also in charge for the execution of the corresponding procedures. Negotiation of membership conditions can be done for example by voting.

Obviously, the choice of the suitable form of the group authority depends on the trust relationship between the group members, and is usually application- and situation-specific. It is reasonable to assume that if group applications are deployed in companies or organizations with a classical hierarchical structure then centralized GAs are likely to be used. In contrast, if a group application is used in a different environment where such trust relationship is undesirable then distributed GAs may be a better choice.

1.4.3 Group Admission Policy

The next important question, especially in case of dynamic groups, is how do prospective group members obtain the list of conditions that they have to satisfy in order to get the group membership approved? Note that in static groups the information about all prospective group members is usually known in advance. Admission (and revocation) conditions can be specified in the so-called *group admission policies* [115] that contain the description of the whole admission process including the communication process with the GA. In turn, these policies may be part of a global *group security policy* also called *group charter* [115] that provides additional information about the group and GAs. A prospective group member must obtain these policies prior to his group membership request.

1.4.4 Group Membership Certificates and Admission Process

As already noted GAs are in charge for the approval and revocation of the group membership. If GA approves the group membership of an object or entity then it, usually, issues a *group membership certificate* which provides access to the application, or can be used to prove membership possession to other parties (group members or third parties). Some admission protocols allow the approved group members to compute certificates on their own upon receiving some required information. The notion of the group membership certificate is rather abstract and opens different ways for concrete realizations based on different admission processes. We describe some of them in the following.

A popular way to approve group membership is to provide users with passwords that allow access to the application and can be also used for the purpose of authentication. Passwords can

be either individual or shared. Individual passwords are realistic in groups with the centralized admission control, e.g., a server acting as GA. Shared passwords can be used for the purpose of group authentication but are practical in static groups because they have to be changed whenever any group membership is revoked. If shared passwords are used in groups with a distributed GA then members of the GA must trust each other not to introduce new members to the group, unless explicitly allowed by the admission policy.

Similarly, admission process and membership certificates can be realized based on shared secret keys. If GA approves the group membership of a prospective group member then this member learns some secret key known to all other members. As already noted in Section 1.3.2 this shared secret key can then be used for the purpose of group authentication, and so for the proof of group membership. It is important to change this key on every membership revocation. This admission process works only if all GA members trust each other not to reveal the shared key to prospective members unless these satisfy the conditions stated in the group admission policy.

Another way to manage the admission process and create membership certificates is to use digital signatures with public-key infrastructure (PKI) whereby GA acts as a certification authority (CA) and digitally signs member's credentials and further information such as validity period of the certificate. Obviously, all group members are then able to verify the group membership using the corresponding public key of the GA. In case that a membership is revoked classical mechanisms of certificate revocation via certificate revocation lists (CRLs) or Online Certificate Status Protocol (OCSP) may be used. Obviously, this scenario suits well for groups with centralized GAs.

If the above approach is applied directly in groups with distributed GAs then each GA member has to sign each membership certificate resulting in linear costs for certificate generation and verification. As noted in [115] it is also possible to use *accountable subgroup multi-signatures* [143] that allow members of the GA (or a subset thereof) to create a digital signature testifying that these members have actively participated in the signing process. In order to verify the validity of such membership certificates one must verify certificates of all GA members that have signed. This is, however, practical only in static and small groups.

Other suggestions for the admission process with distributed GAs make use of the *threshold cryptography* [78] which allows to share a power of a cryptographic system. In particular, modern *threshold signature schemes* as described in [115, 156, 157] may be used by GA members to decide about the admission of prospective group members. The idea behind this technique is that each GA member provides a partial threshold signature to a prospective group member. For the approval of group membership this prospective group member must receive enough partial signatures in order to derive a complete signature based on the techniques of threshold cryptography. This complete signature represents the membership certificate of the group member. The main drawback of these approaches is inefficient revocation of the group membership. Also, group members cannot use obtained membership certificates to prove own membership to other members.

Finally, we mention that research on group admission mechanisms, especially in groups with distributed GAs, is currently in the beginning and that many of the available mechanisms have drawbacks, especially with the revocation of the group membership. Still, an increasing interest in this area can be expected along with the increasing interest in group applications.

Chapter 2

Group Key Establishment

As noted in the previous chapter many security requirements that are relevant for group applications, e.g. confidentiality, group authentication and integrity, can be achieved based on various techniques of symmetric cryptography. In order to apply these techniques in a group setting it is essential that group members obtain a common secret group key k. This chapter focuses on the process of the group key establishment.

2.1 Classification

The establishment of group keys is fundamental for a variety of security mechanisms in group applications. For example, group keys can be utilized by symmetric encryption schemes for the purpose of confidentiality which is one of the most frequent security requirements in group applications; also message authentication codes require group keys for the purpose of group authentication and integrity. Thus, it is important to have mechanisms that provide group members with shared secret keys. We classify possible mechanisms based on the following definitions from [135, Chapter 12] which we adopted to a group setting.

Definition 2.1 (Group Key Establishment). *Group key establishment is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.*

This general definition can further be shaped in two different classes: *group key transport/distribution* and *group key exchange/agreement*.

Definition 2.2 (Group Key Transport/Distribution). *A group key transport/distribution protocol or mechanism is a group key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).*

The main characteristic of group key transport protocols is that the group key k is chosen by a single party and then securely transferred to all group members. This definition leaves open whether a party which chooses the group key must be a group member. It is also imaginable to have some trusted third party (TTP) that chooses group keys on behalf of the group. Also the requirement on secure transfer of group keys forebodes the existence of secret communication channels between the party that chooses group keys and other group members.

Definition 2.3 (Group Key Exchange/Agreement). *A group key exchange/agreement protocol or mechanism is a group key establishment technique in which a shared secret is derived by two or more parties as a function of the information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value.*

Obviously, in group key exchange protocols all group members have to interact in order to compute the group key. The main difference to group key transport techniques is that no party is allowed to choose the group key on behalf of the whole group. Also, group key exchange protocols do not require the existence of secure channels between participants since no secure transfer takes place.

Note that regardless of which group key establishment technique is used by an application the resulting group key must remain secret from unauthorized parties in order to guarantee the expected requirements from the utilized cryptographic mechanisms, like encryption schemes or message authentication codes.

Both group key establishment techniques can be analyzed in context of either static or dynamic groups. Of course it is always possible to establish the group key for the modified group by re-starting the protocol. However, this may be inefficient if groups are large or the protocol is computationally expensive. Therefore, many group key establishment protocols designed for dynamic groups provide more efficient operations for addition and exclusion of group members.

2.2 Group Key Transport/Distribution

In group key transport/distribution protocols the party which chooses group keys on behalf of the group is given enormous power and may, therefore, influence the security of the protocol. Whether a group application allows this kind of trust relationship depends surely on its goals and the environment in which it is executed. However, it seems evident that group key transport protocols are primarily used in group applications with centralized control over the group admission process. In these scenarios the party acting as group authority (GA) may also be in charge for the choice of the group key and its distribution to other members.

Obviously, the most challenging task in group key transport protocols is its protection during the protocol execution.

The following general mechanism is usually applied in group key transport protocols, e.g. [50, 103]. After the party which is responsible for the choice of the group key chooses the key it encrypts it via an appropriate encryption scheme and distributes it to all other group members. Both, symmetric and public-key encryption schemes can be used for this purpose. In case that the applied scheme is symmetric the existence of shared secret keys between this party and each group member is indispensable. This means, that group members have to exchange secret keys with that party pairwise before it proceeds with the group key distribution. Another solution is to apply public-key encryption schemes which do not require any pre-shared secrets between group members and the central party, e.g. in [133]. However, public-key encryption is usually less efficient than symmetric encryption. Therefore, if group keys are distributed frequently, e.g. due to frequent group membership changes, then symmetric cryptography performs better.

The core of many group key distribution protocols builds a mechanism called *key hierarchy* [181, 186]. It arranges group members at the leaves of a logical tree and assigns some secret value to each node of the tree. The secret value at the root of the tree represents the group key or a secret material which can be used to derive the group key via some additional transformations. The goal of the distribution process in key hierarchies is to provide each group member with

the information which it can use to compute all secret values in its path up to the root, including the group key. Key hierarchies are popular in dynamic group key distribution protocols for multicast and broadcast encryption, e.g. [47, 141, 163, 180, 181, 186], since they provide various mechanisms based on modification of the logical tree structure to enhance protocol efficiency upon dynamic group changes.

2.3 Group Key Exchange/Agreement

The only trust assumption in group key exchange protocols is that members trust each other not to reveal any information which can be used to derive the group key to any third party which is not a valid member of the group. Especially, group members do not trust each other during the computation of the group key which should be composed of individual contributions of all group members. Thus, in contrast to group key transport protocols the design of group key exchange protocols is more challenging due to the distributed computation process of the group key.

Many group key exchange protocols (see Chapter 7) can be seen as modifications of the two-party key exchange protocol proposed by Diffie and Hellman in their seminal paper [79]. This protocol allows two parties upon exchange of information over a public channel to compute the shared key using specific discrete mathematical constructions which prevent eavesdroppers from learning the established key.

2.4 Session Keys

Usually, group keys returned by group key establishment protocols are not used directly in the application. Instead, additional transformations are applied in order to derive further keys, so-called *session keys*, which are used by different security mechanisms within the application. For example, if an application requires confidentiality and group authentication then one session key is derived for the encryption scheme and another session key is derived for the message authentication code. Transformations which are used to derive session keys are usually one-way, i.e., given the output of the transformation it is computationally infeasible to obtain the input. Thus, even if a session key is leaked it is still hard to compute the original group key. The use of different session keys provides additional security in terms of independence of applications and deployed security mechanisms since leakage of one session key does not imply leakage of other session keys. Thus for example, if a third party learns the session key used for group authentication then it can authenticate itself as a group member but is not able to decrypt encrypted group messages. Furthermore, session keys are *ephemeral*, i.e., they are valid for a short period of time. For example, digital conferences should use a different session key for each communication session. The use of ephemeral keys decreases the chance of cryptanalytic attacks. Also in dynamic groups any change of group membership should result in new session keys. In case that the group is static but application execution lasts long, e.g., if groupware is used in some long-term project, it is reasonable to refresh session keys periodically.

2.5 Security Requirements on Group Key Establishment Protocols

As already noted, the core security requirement on group key establishment process is to keep the established group key out of reach of unauthorized parties [50, 171]. Whether the group

key remains secret even if all group members take measurements to protect it from third parties depends on a number of facts. In case that the established group key is used in a high-level application (and this is the mostly common case) its secrecy must be protected by that application. On the other hand, each group key establishment protocol must withstand attacks aiming to compromise the key during the execution of the protocol.

However, group key secrecy is not the only security requirement. Another one is the indistinguishability of group keys from random values [119]. This requirement is stronger than group key secrecy as long as the adversary is not able to verify the key by obtaining some information where this key might have been used, e.g., a corresponding cipher text or a MAC value. There are also some further security requirements based on the knowledge of the adversary. For example security against known-key attacks [48, 189] requires that a group key used in a session remains indistinguishable from a random value even in case that group keys of other sessions are known to the adversary. This requirement is important especially in dynamic groups where members may be excluded for a variety of reasons. This is because group members after their exclusion must be treated as potential adversaries against the application which continues with participation of remaining group members.

Further, in cryptographic protocols it is common to distinguish between passive and active adversaries. Passive adversaries are limited to the attacks based on the eavesdropping of the communication channels. Active adversaries are assumed to have more power since they can modify and inject own protocol messages. In group key establishment protocols which are supposed to provide security against active adversaries it is essential that participants distinguish messages originated by valid participants from those originated by the adversary; otherwise, an unauthorized user may participate in the protocol and learn the established group key. Obviously, this requirement is related to authentication of group members during the protocol execution. Also possible replay attacks caused by injection of messages from previous protocol executions or sessions have to be considered. These scenarios are covered by the class of impersonation attacks [49, 50] against group key establishment protocols.

As noted in the previous chapter authentication of group members can be achieved using techniques of symmetric and asymmetric cryptography. In both cases each group member uses some secret information (in form of a password/shared key or private key related to some certified public key) such that other group members are able to verify this use. Usually, this secret information is independent of one particular session and remains valid over a longer time period. Therefore, it may become a potential subject of attacks. So, it is important to consider security of a group key establishment protocol in case of the leakage of the authentication information of its participants. Obviously, the knowledge of this information allows the adversary to impersonate participants in subsequent protocol sessions. Therefore, it is reasonable to focus on the security of preceding protocol sessions. These attack scenarios are covered by the requirement called (perfect) forward secrecy [80, 102].

Another potential threat comes from the attacks against the correctness of a group key establishment protocol. In particular, the adversary may try to prevent protocol participants from computing equal session keys. This may be considered as a kind of a denial-of-service attack since it may prevent group members from using the application. For example, if the established group key is used for the encryption of group messages then members holding different session keys are not able to decrypt messages of each other. Surely, a successful attack of this form may be discovered during the execution of the application, however, it is reasonable to take protective measures during the protocol without relying on the application. A class of requirements including key confirmation [135] and mutual authentication [23] deals with this sort of attacks.

It is worth being noticed that these attacks may also be carried out by valid group members who participate in the protocol but those behavior deviates from its specification. Such protocol participants are also called *malicious participants* or *malicious insiders* [111].

A different class of insider attacks concerns the computation process of the resulting group key. Especially, in group key exchange protocols it is important that protocol participants equally contribute to the resulting value of the group key without being able to influence it. This class of security requirements includes *contributiveness* and *unpredictability of group keys* [13, 173], and is of major importance for the trust relationship between the participants of a group key exchange protocol. Note that in group key transport/distribution protocols such contributiveness and unpredictability requirements have no substantiation because of the trust assumption on the party which is allowed to choose group keys on behalf of other group members. One possible threat covered by these attacks is that participants can be influenced to compute a session key which has already been used in some previous session or another application. This may cause various interference problems between different sessions or applications in which equal session keys are used.

In addition to the attacks of passive/active adversaries and malicious insiders there is a class of attacks aimed to reveal internal information used by participants during the protocol execution. These attacks can be carried out for example via malware. Such adversarial attacks are usually called *strong corruption* attacks [42, 167, 173]. The revealed internal information consists usually of ephemeral auxiliary secrets used by participants to compute the resulting session group key. Especially in dynamic group key establishment protocols, where participants have to save some auxiliary information in order to updated the group key on occurring group changes more efficiently, it is challenging to provide security with respect to strong corruptions.

2.6 Generic Security-Enhancing Solutions - Compilers

In the light of a large number of different security requirements it is convenient to have generic solutions that can be used to enhance security of any group key establishment protocol regardless of its original construction. Such security-enhancing solutions, called *compilers* [112], can also be used to design group key establishment protocols in a modular way. It is simpler to design a protocol which satisfies only a subset of security requirements, and then apply a compiler to enable additional security properties, which are not provided by the original construction. Such modular construction of group key establishment protocols has advantages in situations where a “black-box” implementation of a group key establishment protocol has to be adopted for some particular group application that requires a higher degree of security than the one currently available.

Chapter 3

Provable Security in Group Key Establishment

Security proofs are indispensable for any cryptographic protocol and provide strong arguments in its favor. This chapter focuses on the notion of *provable security* and general techniques to achieve it in the context of group key establishment protocols.

3.1 The Notion of Provable Security

The notion of *provable security* is used in the modern cryptography to specify that a cryptographic construction provides the required security goals if there exists a corresponding proof of security in some mathematically indisputable way. In order to construct such proofs one usually requires a formal setting (*security model*) that takes into account involved participants, their trust relationship, cryptographic parameters of the protocol, and communication environment, and specifies the adversarial environment including definitions of some concrete security goals that must be satisfied by the actual construction.

Security of earlier group key establishment protocols has been analyzed heuristically based on informal definitions. Such intuitive analyses may provide a first feeling for the security of a protocol but are insufficient to say that a protocol is provably secure. Indeed, many of group key establishment protocols analyzed in this way were later shown to be insecure, e.g. [144, 146].

Provable security in cryptography comes currently in three different flavors: information-theoretic (or unconditional), computational, and symbolic.

A cryptographic construction which is *information-theoretic* secure resists attacks of an adversary which is not limited in computational power and resources. Therefore, this kind of security is sometimes called unconditional.

The notion of *computational security* considers the adversary which is limited in its computational power and resources. Computational security of a cryptographic construction relies usually on some unproven (number-theoretic) assumptions which are widely believed to be true.

Finally, the *symbolic security* approach originated from formal methods and languages analyzes security of a cryptographic construction based on a specified process calculus which is applied to a formal description of the construction given in the syntax of the calculus. Security proofs carried out using this approach can be automated using appropriately implemented analyzers.

In the following we give a brief overview of these approaches in the context of group key establishment.

3.2 Information-Theoretic Security in Group Key Establishment

Information-theoretic security is considered as the strongest degree of security of a cryptographic construction. Information-theoretic security strongly relies on the probabilistic behavior of mechanisms used by a cryptographic system and statistics. The theoretical background of this security type has its roots in Shannon's research [162] on *secrecy systems*. Since that it became the inherent part of the cryptographic research and has been applied to different constructions including message authentication [93, 169, 182] and also key establishment [6, 77, 128, 129, 130, 188]. The latter is especially interesting in the context of this work.

Information-theoretic secure key establishment is possible if the adversary can be limited in its knowledge compared to the knowledge of participants. Thus, participants must share at least some partially secret information initially. Such limitation of the adversarial knowledge can be achieved by exploiting noise in communication channels [6, 77, 128, 188], limitations of adversarial memory capacity (without limiting its computational resources) [54], quantum entanglement [88] or Heisenberg uncertainty principle in quantum mechanics [25].

A significant drawback of information-theoretic secure key establishment protocols is their limited practical deployment. For example, the protocol described in [130] (whose design is exemplary for many information-theoretic secure key establishment protocols) assumes that a satellite (which can be considered as a trusted third party) sends out randomly generated binary signals which are received by both participants (and possibly by the adversary) over independent binary-symmetric channels with own error probabilities, i.e., noisy channels where any sent bit is received with its value being preserved with some probability which is independent of the initial value. General assumptions are that the error rate of the potential adversary is lower than that of the both protocol participants that are connected by a public but authentic channel. In the first phase of the protocol, denoted *advantage distillation*, both participants exploit the strength of the noiseless authentic communication channel to exchange information about the received bits in order to identify the correctly received ones using parity checks. At the end of the advantage distillation phase each participant holds a long binary string whereby strings of both participants may be distinct (contain a small number of errors). Also, the adversary may have partial information concerning these strings. The goal is, however, to achieve that both participants hold identical secret strings. Therefore, both users continue with the *information reconciliation* phase whose goal is to achieve that both strings are equal. The applied technique is related to error-correction mechanisms. The interaction between both participants in this phase may leak some information to the adversary. Therefore, in order to ensure secrecy of the established shared binary string participants execute *privacy amplification* phase whereby the entire partially secret equal binary strings are reduced. Privacy amplification phase makes use of "random-like" functions like universal hash functions [26]. As already mentioned this construction is exemplary for many information-theoretic secure key establishment protocols. Note that in some protocols, e.g., [54, 77, 188], there is no need in a trusted party so that one of the participants distributes the random bit sequence instead. Anyway, in terms of our classification all information-theoretic secure key establishment protocols are based on the one-way distribution of the random information sequences, implying that the party which distributes these sequences can also control their randomness. Thus, these protocols do not provide exchange/agreement in the sense of Definition 2.3.

All of the currently known information-theoretic secure key establishment protocols have been designed for two participants. Their security has been analyzed with respect to the secrecy of the established key in the presence of passive eavesdroppers like in [77, 128, 188] and also of active adversaries like in [129, 184], however, without considering extended security goals mentioned in the previous chapter. Furthermore, we are not aware of any information-theoretic secure key establishment protocol for a group setting. Still, information-theoretic security of key establishment protocols remains an interesting topic of research. For more details in this research area we refer to [106, 132, 151, 152, 185].

3.3 Computational Security in Group Key Establishment

Computational security limits the adversary in computational power and resources and is, therefore, weaker than information-theoretic security. However, it allows design of more practical constructions.

The core of the computational security is that all involved parties (including the adversary) are modeled by *probabilistic polynomial-time* algorithms whose outputs follow some probability distributions. In computational security each security goal is defined from the perspective of the corresponding adversarial attack specified by interaction (*game*) between the adversary algorithm and the environment, also called challenger or simulator, which is supposed to represent the collection of honest parties. In such games the adversary is considered to be successful if a certain event Win occurs with a probability which is non-negligibly greater than some "target probability". Thus, security goals provide definitions of what it means to "break" the construction whereas the security proof examines the probability of this to happen. From this point of view it is clear that computational security considers only attacks that have been previously specified (as part of the security model). Therefore, for a higher security degree of the construction it is essential that the underlying security model is strong enough to encompass all important security goals.

Provable security based on the computational approach relies on unproven assumptions and, therefore, differs from the notion of provable security in mathematics where results of proven theorems are taken "as is". These unproven assumptions deal mostly with number-theoretic cryptographic problems that are widely believed to be intractable in case that mathematical parameters used to specify these problems are chosen carefully. Intractability in this case means that it is computationally infeasible to solve the problem even with the best known solving algorithm, that is the required computation power and resources clearly exceed those (polynomially bounded) of the adversary algorithm. We refer to [135, Chapter 3] for an overview of the most frequently used cryptographic assumptions and problems.

Security proofs in computational (game-based) security models are mostly of *reductionist* nature [18] and carried out by contradiction, i.e., the success probability of an adversary in breaking a security requirement of the construction is usually reduced to the probability of breaking one or more cryptographic problems which are believed to be intractable. The idea behind the reduction is to show that the hardness of one problem P_1 like breaking some intractable cryptographic problem implies the hardness of another problem P_2 like breaking some construction specific security goal. For this purpose one usually shows that given a solving algorithm for P_2 it is possible to solve P_1 with a polynomially bounded additional complexity (polynomial reduction costs).

3.3.1 Reductionist Security Proofs

Complex cryptographic protocols like those for group key establishment usually consist of different building blocks (cryptographic primitives). Security of such protocols may rely, therefore, on more than just one cryptographic assumption. In this case each considered assumption should have own reduction and all reductions must be related to each other in order to estimate the total probability of the attack. Obviously, such security proofs may be complex and error-prone. The technique called *"sequence of games"* [168] can usually be applied in order to reduce complexity of the reductionist security proofs for complex constructions.

In the "sequence of games" approach one constructs a sequence of games (interactions between the adversary and the environment) $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_n$ starting with the original game as defined in the underlying security model. As already mentioned, the original game usually specifies a certain event Win that must occur in case of a successful attack. Thus, in the security proof for each game of the sequence one specifies events Win_i, $i = 0, \ldots, n$ whereby Win_0 is the original event Win and events Win_i, $i = 1, \ldots, n$ are usually related to Win. Then, one tries to show that the probability of Win_i is negligibly close to that of Win_{i+1}. Games are usually constructed in such way that the probability of Win_n (the event in the last game of the sequence) equals to the "target probability" from the security goal definition in the underlying security model. In order to simplify the relation of probabilities of subsequent events each game $\mathbf{G}_i$, $i = 1, \ldots, n$ is usually constructed by small changes to $\mathbf{G}_{i-1}$.

Since [42] this technique became standard in the security proofs of group key establishment protocols, e.g. [5, 84, 87, 114]. Recently several attempts to automate security proofs within the computational approach have been proposed, e.g. [31, 75]. These approaches are currently at the early stage of development and focus on automated security proofs for basic cryptographic primitives like encryption schemes and digital signatures rather than on complex cryptographic protocols.

There is a large number of the proposed group key establishment protocols that have been proven computationally secure. However, not all protocols provide the same degree of security. As already noted the security proof of a cryptographic construction based on the computational (game-based) approach considers only attacks that have been specified in the underlying security model. The development of computational security models for group key establishment protocols was initiated in [46] and further developed and refined in [42, 43, 87, 111, 112], and the number of security goals increased from model to model. Therefore, group key establishment protocols proven secure in the earlier models may loose on security in the later models. On the other hand, there is a number of group key establishment protocols whose security proofs, although based on the one of the proposed models, do not consider all security goals defined by that model but rather a subset of them. This is because it is a challenging task to design a protocol which satisfies all security requirements stated in a model, especially if that model is strong.

3.3.2 Proofs Based on Simulatability/Indistinguishability

The technique of *simulatability-* or *indistinguishability-based proofs* originated in [97] and extended in [56, 148, 167] can be considered as a special type of the computational security approach where security of a cryptographic construction is derived from the indistinguishability between the ideal execution of the construction and the real one. The ideal execution takes place in the presence of some trusted party which receives inputs from all participants, computes appropriate outputs and returns them to the participants. In this way, the ideal execution specifies

which security requirements have to be provided by the construction, i.e., the security model of the construction. In the real execution this party is replaced by the real communication environment between the participants who perform operations specified by the construction. In order to show that a cryptographic construction satisfies the stated security requirements one has to show that any damage caused by the adversary in the real execution can be simulated in the ideal execution and that this simulation is indistinguishable by any observer.

There is a number of earlier proposals (e.g. [20, 62, 167]) concerning security models with simulatability-based proofs for key establishment protocols. However, all of these models have been defined for two participants and cannot be simply scaled to a group setting.

The latest development of the indistinguishability-based computational approach is the *Universal Composability* (UC) framework [57] that specifies an additional party called *environment* which takes place of the observer representing "whatever is external to the current protocol execution". The security model is given by an *ideal functionality* which acts as a trusted party mentioned above with the difference that it processes additional inputs occurring during the execution and possibly depending on previously generated values, and maintains some local state information. Security proof in the UC framework is performed by the simulation of the real-world adversary such that the probability of the environment in being able to distinguish between the interaction with the real-world adversary and the simulator (which represents the ideal-world adversary) is negligible. The UC framework has its strength in the composition of cryptographic protocols as described by the *universal composition theorem* [57] which can be used to prove that the composition of secure sub-protocols realizing some sub-tasks results in a secure protocol realizing the task which can be seen as a combination of all sub-tasks.

There is a number of works dealing with UC-security of key establishment protocols between two participants (e.g. [60, 63, 64, 105]). The only currently published security model for UC-secure group key establishment protocols is described in [111].

3.3.3 Non-Standard Assumptions in Computational Security Proofs

A higher confidence in a security proof of a cryptographic construction based on the computational approach can be achieved if the proof is carried out in the *standard model* which is "close to" the practical execution environment. However, it is not always possible to prove security of a protocol in the standard model; mostly, for the technical reasons. Therefore, some security proofs can only be performed under auxiliary non-standard assumptions such as the Random Oracle Model [23] and the Ideal Cipher Model [162].

Random Oracle Model

The *Random Oracle Model* (ROM) [23] is a non-standard methodology frequently applied in the computational security proofs. It considers a cryptographic construction as an ideal system in which all parties (including the adversary) can evaluate an ideal random function (called *random oracle*) via adequate input queries. Security of the construction is then proven (mostly using reductions) in this ideal setting. The main assumption in this approach is that the random oracle used in the security proof can be replaced with some "good" cryptographic hash function which can be directly evaluated by all parties (including the adversary). In this way one achieves the implementation of the ideal system in the "real-world" scenario where such random oracles do not exist.

There is a number of group key establishment protocols, e.g. [41, 46], whose security has been analyzed using the methodology of ROM. However, in the cryptographic community the

ROM methodology is currently under a controversy discussion motivated by a number of papers describing cryptographic constructions whose security is provable in ROM but fails in practice, e.g. [19, 59, 142]. On the other hand, all of these constructions are artificial and will hardly be ever used in practice. In the opinion of some researchers, the practical irrelevance of the proposed constructions aiming to undermine the validity of the ROM methodology for practical cryptography achieves the opposed result [120].

In our opinion, designers of cryptographic constructions including group key establishment protocols should try to achieve provable security under weaker assumptions in the standard model rather than in ROM; at least until this controversy discussion is finished in favor of ROM.

Ideal Cipher Model

The *Ideal Cipher Model* (ICM) [162] is another non-standard methodology used in cryptographic proofs. It assumes the existence of an ideal symmetric block cipher which should be queried by all participants (including the adversary). Again having proved security of a cryptographic construction under this assumption one instantiates the ideal cipher using some practical block cipher. Note that ICM is assumed to be stronger than ROM since it is unknown whether an ideal block cipher can be derived from a random oracle. For further discussion on the relationship between ICM and ROM we refer to [81].

There are some group key establishment protocols (e.g. [5]) where symmetric encryption (with shared small entropy passwords) is used for the purpose of authentication. Security proofs of such protocols require the non-standard assumptions of ICM.

Similar to the ROM methodology there are some cryptographic constructions (e.g. [27]) which remain secure in ICM and become insecure after the instantiation of the ideal block cipher.

3.4 Symbolic Security in Group Key Establishment

The *symbolic security* approach originated in [82] and further developed in [2, 52, 89, 113, 134] is characterized by an abstract view on the cryptographic scheme and the adversary which is limited by constraints of used cryptographic primitives which are treated as ideal "black-boxes" without considering details of their implementation. This approach makes stronger assumptions on the adversary compared to the approach of computational security.

In the symbolic security approach cryptographic primitives are represented as operators of a term algebra. The approach makes use of a process calculus with own syntax and operational rules. A cryptographic construction, whose security has to be analyzed, is usually described in the syntax of the given calculus and its security goals are represented by some appropriate formal expressions. Security proofs are based on deductions, i.e., using a set of calculus rules one tries to deduce the formal expression defined for a particular security goal from the initial description of the protocol in the semantic of the calculus.

However, such symbolic analysis may not guarantee security of the construction if "black-box" primitives are replaced by real implemented schemes. Thus, symbolic analysis alone provides a questionable degree of security. The actual strength of this approach is given by the possibility to conduct automated security proofs (deductions) via so-called model checkers and protocol analyzers [11, 30, 126, 170].

Latest works in this area of provable security originated in [3] try to combine strengths of the symbolic approach, in particular its automatization, with the methods of computational se-

curity. Especially, simulatability-based approaches with composition feature (e.g., [57, 148]) seem most suitable for this combination although some recent results [31] aim to automatize reductionist proofs based on the "sequence of games" technique. When merging symbolic security analysis with the simulatability-based approach one considers symbolic terms describing "black-box" cryptographic primitives as ideal functionalities and then using the composition properties shows that the automated security proof is cryptographically sound. Several examples of this technique including a practical implementation of abstract models (*ideal functionalities*) of some cryptographic primitives in the so-called *composable cryptographic library* are [15, 57, 148].

A more direct use of the symbolic security analysis for the computationally sound security proofs has been shown in [61] on the example of the protocols for mutual authentication and key establishment where a public-key encryption scheme is the only used cryptographic primitive.

The described developments in merging symbolic security and computational security focus currently on basic cryptographic primitives and simple protocols. However, we may expect that continuous progress in this area will also consider complex cryptographic constructions including protocols for group key establishment.

Chapter 4

Dissertation Focus and Summary of Research Contributions

Having provided general information on different aspects of security in group applications, group key establishment protocols, and the issue of "provable security" we use this chapter to specify the scope of the dissertation and prepare the reader for the contents of its main part.

4.1 Dissertation Focus and Objectives

This dissertation focuses on provable security of group key exchange protocols while considering mainly its computational variant as introduced in Section 3.3. Design of a secure group key exchange protocol is more challenging compared to that of a group key distribution/transport protocol because of the missing trust between its participants during the protocol execution. Recall from Section 2.3 that in group key exchange protocols the only trust assumption is that group members do not reveal the established group key to third parties. To the contrary in group key distribution/transport protocols there is an additional assumption that trusted parties are allowed to choose group keys on behalf of other group members and this assumption reduces the number of security requirements.

This dissertation has several objectives described in the following. First, we analyze the current state of research on provable security of group key exchange protocols. For this purpose we check whether currently available computational security models for group key exchange protocols subsume all important security requirements, in particular those described informally in the earlier literature. Second, we analyze whether currently known group key exchange protocols provide sufficient degree of security. Especially, we focus on strong corruption attacks and attacks of malicious participants (both mentioned in Section 2.5) since these are the most challenging classes of attacks. Third, we intend to use the obtained results in order to specify a stronger security model that considers additional requirements missing in the previous models. Fourth, we intend to propose practical constructions providing a higher degree of security and prove their security in order to show soundness and feasibility of our new security definitions.

4.2 Summary of Research Contributions

In this section we briefly summarize main results obtained in the dissertation. The description is done along the chapters of the second part.

4.2.1 Two Surveys

Our first survey in Chapter 6 describes evolution of security requirements and models concerning group key exchange protocols. Beside the overview of informal definitions we consider all previously proposed formal security models for group key exchange protocols (and also some models for two/three-party protocols). We describe their constructions and provide analysis based on the informal security requirements described in the earlier literature. In our analysis we were able to identify some problems in the technical construction of the first published security model for group key exchange protocols in [46]. Further we were able to show that none of the currently existing security models for group key exchange protocols provides *sufficient* security definitions concerning strong corruption attacks and attacks of malicious protocol participants, especially concerning contributiveness and unpredictability of computed group keys, i.e., requirements that state the main difference between group key exchange and group key distribution protocols.

Our second survey in Chapter 7 is dedicated to the existing static and dynamic group key exchange protocols. For each mentioned protocol we briefly describe its degree of security either based on the previously published results or using our own analysis. In general, static protocols provide a higher degree of security than dynamic protocols. The reason behind this is that in dynamic group key exchange protocols participants must, usually, save additional auxiliary information in order to handle dynamic group changes in subsequent protocol sessions more efficiently whereas in static protocols the auxiliary information is chosen fresh for each new protocol execution. Therefore, strong corruption attacks against dynamic protocols become much more critical. In particular, there exists currently no dynamic group key exchange protocol with an adequate security proof (under standard cryptographic assumptions) that satisfies the requirement of forward secrecy in case of the strong corruptions. Furthermore, current security definitions concerning the issues of malicious participants, especially with respect to the requirements of key control, contributiveness and unpredictability of computed group keys, need further improvements.

4.2.2 A Stronger Computational Security Model for GKE Protocols

Having identified various weaknesses in previously proposed security models for group key exchange protocols we present in Chapter 8 a variation of security models in [32, 42, 111, 112]. Our proposed model is stronger since it provides security definitions concerning powerful adversaries with respect to strong corruptions and attacks of malicious participants. In particular, we provide new definitions of MA-security and n-contributiveness. We also extend the notion of AKE-security from [42, 46, 112] with an additional adversarial setting (*backward secrecy*) that considers damages occurred in past sessions of group key exchange protocols. Our definition of backward secrecy is symmetrically opposed to the well-known definition of forward secrecy. We also consider different flavors of backward and forward secrecy with respect to strong corruptions.

4.2.3 Seven Security-Enhancing GKE Protocol Compilers

As already mentioned in Section 2.6 it is desirable to have security-enhancing compilers that can be used in conjunction with any group key exchange protocol regardless of its natural construction. In Chapter 9 we overview existing compilers and describe their drawbacks. Additionally, we provide seven security-enhancing compilers: one for each of the three security goals (AKE-,

MA-security and n-contributiveness) specified in our security model, and additional four compilers for all combinations thereof. We prove security of each proposed compiler based on the *sequence of games* technique using standard cryptographic assumptions. These proofs also intend to show soundness and feasibility of our formal security definitions.

4.2.4 A Constant-Round GKE Protocol

In Chapter 10 we propose a novel group key exchange protocol TDH1. It appears in two versions: static and dynamic. We prove security of both versions in the standard model (without relying on non-standard assumptions of ROM or ICM) with respect to the definitions of our new security model. The static version of TDH1 achieves a higher degree of security since the internal auxiliary information used by protocol participants to compute the resulting group key is fresh in each new protocol execution. The dynamic version of TDH1 consists of operations that handle addition and deletion of protocol participants. It can be considered as the first ever proposed dynamic group key exchange protocol that is provably secure under standard cryptographic assumptions with respect to the requirement of forward secrecy in the presence of strong corruptions. Security of static TDH1 relies on the existence of one-way permutations and the so-called Tree Decisional Diffie-Hellman (TDDH) assumption, whose equivalence to the classical Decisional Diffie-Hellman assumption is also shown. Security of dynamic TDH1 relies additionally on the so-called Square-Exponent Decisional Diffie-Hellman (SEDDH) assumption.

Part II

Provably Secure Group Key Exchange
– Foundations and Solutions –

Chapter 5

Background on Cryptography

In this chapter we describe various cryptographic issues which are used in our constructions throughout this work. For the general introduction on cryptography we refer to the well-known book by Menezes, van Oorschot, and Vanstone [135] whereas for the theoretical aspects of cryptography we refer to the books by Goldreich [94, 95], and for its practical aspects to the books by Ferguson and Schneier [90, 158].

5.1 Negligible Functions

Definition 5.1 (Negligible Function [94]). *A function $\nu : \mathbb{N} \to \mathbb{R}$ is* negligible *if for every positive polynomial p there exists an $N \in \mathbb{N}$ such that for all $n > N$,*

$$\nu(n) < \frac{1}{p(n)}.$$

In other words any negligible function decreases faster than the reciprocal of any polynomial. Note that the phrase "there exists an $N \in \mathbb{N}$ such that for all $n > N$" is, usually, replaced by the shorter phrase "for all sufficiently large n".

Another important issue concerning negligible functions is that for every negligible function ν and any polynomial p, the function $\nu' := \nu(n)p(n)$ is also negligible.

Negligible functions are typically used to quantify security of cryptographic constructions.

5.2 One-Way Permutations

Definition 5.2 (One-Way Permutation). *A function $\pi : \{0,1\}^\kappa \to \{0,1\}^\kappa$, $\kappa \in \mathbb{N}$ is called a* one-way permutation *if the following three conditions hold:*

- *there exists an efficient algorithm that on input x outputs $\pi(x)$;*
- *π is a permutation;*

- *for every PPT algorithm $\mathcal{A}$, the following success probability is negligible:*

$$\mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa) := \Pr\left[\begin{array}{l} x \in_R \{0,1\}^{\kappa}; \\ \quad y := \pi(x); \quad : \pi(x') = y \\ x' \leftarrow \mathcal{A}(1^{\kappa}, y) \end{array}\right].$$

Remark 5.3. By a PPT algorithm we mean a *probabilistic polynomial-time* algorithm - a randomized algorithm (modeled as a deterministic Turing machine with an additional random tape) whose execution requires polynomial number of steps in the length of the input. We assume that the reader is familiar with this terminology.

In other words it is computationally infeasible to invert any one-way permutation. One-way permutations belong to the core cryptographic primitives. The assumption of the existence of one-way permutations is considered as a standard cryptographic assumption. A large number of number-theoretic problems like integer factorization or computation of discrete logarithms (see Section 5.4.1) that are assumed to be intractable are conjectured to provide functions satisfying the required one-way property. These conjectures are based on the absence of efficient (polynomial-time) solving algorithms.

5.3 Pseudo-Random Functions

Informally, a pseudo-random function (PRF) is specified by a (short) random key k, and can be easily computed given this key. However, if k remains secret, the input-output behavior of PRF is, for any PPT algorithm, indistinguishable from that of a truly random function with the same domain and range. This dissertation uses (efficiently computable) generalized pseudo-random functions, defined in the following (see also [94, Definition 3.6.4]).

Definition 5.4 ((Efficiently Computable) Pseudo-Random Function Ensemble F). *An ensemble of finite functions $F := \left\{\left\{f_k : \{0,1\}^{p(\kappa)} \rightarrow \{0,1\}^{p(\kappa)}\right\}_{k\in\{0,1\}^{\kappa}}\right\}_{\kappa\in\mathbb{N}}$ where $p : \mathbb{N} \rightarrow \mathbb{N}$ is upper-bounded by a polynomial, is called an* (efficiently computable) pseudo-random function ensemble *if the following two conditions hold:*

1. Efficient computation: *There exists a polynomial-time algorithm that on input k and $x \in \{0,1\}^{p(\kappa)}$ returns $f_k(x)$.*
2. Pseudo-Randomness: *Choose uniformly $k \in_R \{0,1\}^*$ and a function f in the set of all functions with domain and range $\{0,1\}^{p(\kappa)}$. Consider a PPT adversary $\mathcal{A}$ asking a polynomially bounded (in κ) number of queries of the form $Tag(x)$ and participating in one of the following two games:*
   - $\mathsf{Game}^{\mathsf{prf}-1}_F(\kappa)$ *where a query $Tag(x)$ is answered with $f_k(x)$,*
   - $\mathsf{Game}^{\mathsf{prf}-0}_F(\kappa)$ *where a query $Tag(x)$ is answered with $f(x)$.*

   At the end of the execution $\mathcal{A}$ outputs a bit b trying to guess which game was played. The output of $\mathcal{A}$ is also the output of the game. The advantage function of $\mathcal{A}$ in winning the game is defined as

$$\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) := \left|2\Pr[\mathsf{Game}^{\mathsf{prf}-b}_F(\kappa) = b] - 1\right|.$$

   We say that F is pseudo-random if the advantage $\mathsf{Adv}^{\mathsf{prf}}_F(\kappa)$ is negligible for all sufficiently large κ.

By an (efficiently computable) pseudo-random function *we mean a function $f_k \in F$ for some random $k \in_R \{0,1\}^*$.*

In other words in the above definition the goal of the adversary $\mathcal{A}$ is to distinguish whether replies on its Tag queries are generated by a pseudo-random function or by a truly random function of the same range. Note that in the above definition the pseudo-randomness of the ensemble is defined using a *black-box setting* where the adversary may indirectly (via Tag queries) obtain the value of the function chosen in the corresponding games for any arguments of its choice, but does not get any information (e.g., keys) which would allow it to evaluate the pseudo-random function itself.

Additionally, we require the following notion of collision-resistance of pseudo-random function ensembles. This definition is essentially the one used by Katz and Shin [111]. The same property has previously been defined in [91] and denoted there as *fixed-value-key-binding* property of a pseudo-random function ensemble.

Definition 5.5 (Collision-Resistance of F). *Let F be a pseudo-random function ensemble. We say that F is* collision-resistant *if there is an efficient procedure* `Sample` *such that for all PPT adversaries $\mathcal{A}$ the following success probability is negligible in κ:*

$$\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) := \Pr\left[\begin{array}{c} x := \texttt{Sample}(1^\kappa); \\ (k,k') \leftarrow \mathcal{A}(1^\kappa, x) \end{array} : \begin{array}{c} k,k' \in \{0,1\}^\kappa \wedge \\ k \neq k' \wedge \\ f_k(x) = f_{k'}(x) \end{array}\right].$$

Theorem 5.6 ([111]). *If one-way permutations exist then there exist collision-resistant pseudo-random functions.*

Proof. We show that a collision-resistant pseudo-random function ensemble $F := \left\{\left\{f_k : \{0,1\}^{p(\kappa)} \to \{0,1\}^{p(\kappa)}\right\}_{k\in\{0,1\}^\kappa}\right\}_{\kappa\in\mathbb{N}}$ can be derived via the tree-based construction proposed by Goldreich, Goldwasser, and Micali [96] using a length-doubling *pseudo-random generator* [94, Definition 3.3.1] $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ based on a one-way permutation $\pi : \{0,1\}^\kappa \to \{0,1\}^\kappa$ and its *hard-core predicate* [94, Definition 2.5.1] $h : \{0,1\}^\kappa \to \{0,1\}$. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be defined as follows:

$$G(k) := \pi^\kappa(k)|h(\pi^{\kappa-1}(k))|\ldots|h(\pi(k))|h(k),$$

where $\pi^i(k)$, $i \in [1,\kappa]$ denotes the i-fold execution of π and $|$ denotes the concatenation of binary strings. Let $G_0(k)$ denote the first κ bits of $G(k)$, and let $G_1(k)$ denote the last κ bits of $G(k)$. For a binary input $x = x_1|\ldots|x_{p(\kappa)}$, $x_j \in \{0,1\}$, $1 \leq j \leq p(\kappa)$ let

$$f_k(x) := G_{x_{p(\kappa)}}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots).$$

For the proof that $F := \left\{\left\{f_k : \{0,1\}^{p(\kappa)} \to \{0,1\}^{p(\kappa)}\right\}_{k\in\{0,1\}^\kappa}\right\}_{\kappa\in\mathbb{N}}$ is efficiently computable pseudo-random we refer to [94, Theorem 3.6.6]. Further, note that $f_k(0^{p(\kappa)}) = \pi^{p(\kappa)\cdot\kappa}(k)$. Thus, the pseudo-random function $g(k) := f_k(0^{p(\kappa)})$ is a permutation, and, therefore, collision-resistant. □

Remark 5.7. As noted in [94] there are some significant differences between using PRFs and the Random Oracle Model (ROM) [23]. In ROM, a random oracle that can be queried by the adversary is not keyed. Still, the adversary is forced to query it with chosen arguments instead of being able to compute the result by itself. Later, in the implementation the random oracle is instantiated by a public function (usually a cryptographic hash function) that can be evaluated by the adversary directly. To the contrary, when using PRFs, the oracle contains either a

pseudo-random function or a random function. The pseudo-random function is keyed and the key is supposed to be kept secret from the adversary. This requirement is also preserved during the implementation. Hence, in any case (theoretical or practical) the adversary is not able to evaluate the pseudo-random function by itself as long as the key is kept secret. Thus, with PRFs there is no difference between theoretical specification of the function and its practical instantiation. This is one of the reasons why security proofs based on pseudo-random functions instead of random oracles can be carried out in the standard model. Another reason is that existence of pseudo-random functions follows from the existence of one-way permutations, which is a standard cryptographic assumption.

5.4 Number-Theoretic Assumptions

5.4.1 Discrete Logarithm Assumption

In the following we recall the well-known Discrete Logarithm (DL) assumption. In all definitions we assume that all algorithms (including adversarial) implicitly know the available public information.

Definition 5.8 (Discrete Logarithm Assumption). *Let $\mathbb{G}$ be a cyclic group of order q generated by $g \in \mathbb{G}$, and let $y \in \mathbb{G}$. A* discrete logarithm of y to the base g *is the unique integer $x \in \mathbb{Z}_q$ with $y = g^x$. A* DL solver for $\mathbb{G}$ *is a PPT algorithm $\mathcal{A}$ whose success probability*

$$\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa) := \Pr[x \leftarrow \mathcal{A}(g, y) : y = g^x]$$

is non negligible. The DL *problem is* intractable *if there exists no* DL *solver for $\mathbb{G}$. The* DL *assumption states that this is the case for all sufficiently large κ in appropriate groups.*

There are many candidates for the appropriate choice of $\mathbb{G}$. For example, $\mathbb{G}$ can be a multiplicative group $\mathbb{Z}_p^*$ with p prime, or a cyclic subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$ of prime order q with $p = \alpha q + 1$, $\alpha \in \mathbb{N}$, or defined as a commutative group of points $E(\mathbb{F}_q)$ of an elliptic curve E over a finite (prime or binary) field $\mathbb{F}_q$. In the latter case one talks about *elliptic-curve discrete logarithms* (ECDL). For a detailed description of the possible choices for $\mathbb{G}$ and for the complexity analysis of the currently known solving algorithms for the DL problem we refer to [135].

5.4.2 Diffie-Hellman Assumptions

In the following we recall the well-known Diffie-Hellman assumptions. Let $\mathbb{G}$ be a group where the DL problem is believed to be intractable, g and q as in Definition 5.8. For $x_1, x_2 \in_R \mathbb{Z}_q$ the Diffie-Hellman distribution is defined as

$$\mathtt{DH} = (g, g^{x_1}, g^{x_2}).$$

Definition 5.9 (Computational Diffie-Hellman Assumption). *A* CDH solver for $\mathbb{G}$ *is a PPT algorithm $\mathcal{A}$ whose success probability*

$$\mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa) := \Pr[z \leftarrow \mathcal{A}(\mathtt{DH}) : z = g^{x_1 x_2}]$$

is non negligible. The CDH *problem is* intractable *if there exists no* CDH *solver for $\mathbb{G}$. The* CDH *assumption states that this is the case for all sufficiently large κ.*

The following theorem shows that if the DL problem is solvable then so is the CDH problem.

Theorem 5.10 (CDH $\leq$ DL). *The* CDH *problem is polynomial-time reducible to the* DL *problem and*

$$\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa) \leq \mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa).$$

Proof. Assuming that there exists a DL solver for $\mathbb{G}$ denoted $\mathcal{A}'$ we construct a CDH solver for $\mathbb{G}$ denoted $\mathcal{A}$ as follows. On input the DH distribution (g, g^{x_1}, g^{x_2}) the CDH solver $\mathcal{A}$ calls the DL solver $\mathcal{A}'$ on input (g, g^{x_1}) and obtains x_1 with probability $\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa)$. Then, $\mathcal{A}$ returns $z := (g^{x_2})^{x_1}$. Obviously, $\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa) \leq \mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa)$. □

Whether the DL problem is reducible to the CDH problem remains one of the open problems in cryptography. There are some scientific results, e.g. [131], showing computational equivalence of the CDH and the DL problems in certain groups. In particular, it remains an open question if there could also be groups where the CDH problem is easy but the DL problem remains hard.

In order to define the decisional version of CDH we use the DH distribution to derive further two distributions with $r \in_R \mathbb{Z}_q$:

$$\mathtt{DDH}^{\star} := (\mathtt{DH}, g^{x_1 x_2}) \text{ and } \mathtt{DDH}^{\$} := (\mathtt{DH}, g^{r}).$$

Definition 5.11 (Decisional Diffie-Hellman Assumption). *A* DDH distinguisher for $\mathbb{G}$ *is a PPT algorithm $\mathcal{A}$ whose advantage probability defined as*

$$\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) := |\Pr[\mathcal{A}(\mathtt{DDH}^{\star}) = 1] - \Pr[\mathcal{A}(\mathtt{DDH}^{\$}) = 1]|$$

is non negligible. The DDH *problem is* intractable *if there exists no* DDH *distinguisher for $\mathbb{G}$. The* DDH *assumption states that this is the case for all sufficiently large κ.*

The following theorem shows that if the CDH problem is solvable then so is the DDH problem.

Theorem 5.12 (DDH $\leq$ CDH). *The* DDH *problem is polynomial-time reducible to the* CDH *problem and*

$$\mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa) \leq \mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + \frac{1}{q}.$$

Proof. Assuming that there exists a CDH solver for $\mathbb{G}$ denoted $\mathcal{A}$ we construct a DDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}'$ as follows. On input the distribution (g, g^{x_1}, g^{x_2}, z) the DDH distinguisher $\mathcal{A}'$ calls the CDH solver $\mathcal{A}$ on input the DH distribution (g, g^{x_1}, g^{x_2}) and obtains $g^{x_1 x_2}$ with probability $\mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa)$. Then, $\mathcal{A}'$ checks whether $z \stackrel{?}{=} g^{x_1 x_2}$ and returns 1 if the equation holds.

In case that $\mathcal{A}'$ receives a $\mathtt{DDH}^{\star}$ distribution it outputs 1 with the probability $\mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa)$. Thus, $\Pr[\mathcal{A}(\mathtt{DDH}^{\star}) = 1] = \mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa)$.

In case that $\mathcal{A}'$ receives a $\mathtt{DDH}^{\$}$ distribution the value z is uniformly distributed in $\mathbb{G}$. Hence, $\mathcal{A}'$ outputs 1 with the probability of a random choice over the size of $\mathbb{G}$, i.e, $\Pr[\mathcal{A}(\mathtt{DDH}^{\$}) = 1] = \frac{1}{q}$.

This implies the desired inequality $\mathsf{Succ}^{\mathtt{CDH}}_{\mathbb{G}}(\kappa) \leq \mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + \frac{1}{q}$. □

Corollary 5.13 (DDH $\leq$ DL). *The* DDH *problem is polynomial-time reducible to the* DL *problem and*

$$\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa) \leq \mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + \frac{1}{q}.$$

Proof. The proof follows directly from Theorems 5.10 and 5.15. □

In other words, if the DL problem is easy to solve then the DDH problem is also easy to solve, and if the DDH problem is intractable then the DL problem is also intractable. The relationship between the DDH, CDH and DL problems is a hot topic of research. In particular, there are so-called *Gap* Diffie-Hellman groups [37] where DDH is easy but CDH and DL remain hard.

In the following we focus on a variant of the DDH assumption considered in [154, 185], called *Square-Exponent Decisional Diffie-Hellman* (SEDDH) assumption. Considering $x, r \in_R \mathbb{Z}_q$, we define two distributions:

$$\mathtt{SEDDH}^{\star} := (g, g^x, g^{x^2}) \text{ and } \mathtt{SEDDH}^{\$} := (g, g^x, g^r).$$

Definition 5.14 (Square-Exponent Decisional Diffie-Hellman Assumption). *A* SEDDH distinguisher for $\mathbb{G}$ *is a PPT algorithm* $\mathcal{A}$ *whose advantage probability defined as*

$$\mathsf{Adv}^{\mathtt{SEDDH}}_{\mathbb{G}}(\kappa) := |\Pr[\mathcal{A}(\mathtt{SEDDH}^{\star}) = 1] - \Pr[\mathcal{A}(\mathtt{SEDDH}^{\$}) = 1]|$$

is non negligible. The SEDDH *problem is* intractable *if there exists no* SEDDH *distinguisher for* $\mathbb{G}$*. The* SEDDH *assumption states that this is the case for all sufficiently large* κ*.*

Wolf [185] showed that the intractability of the SEDDH problem depends on the actual choice of $\mathbb{G}$. More precisely, the probability that any SEDDH distinguisher $\mathcal{A}$ can correctly distinguish between $\mathtt{SEDDH}^{\star}$ and $\mathtt{SEDDH}^{\$}$ is directly proportional to the inverse of the smallest prime factor of q. Hence, the SEDDH assumption is believed to hold for groups where q is free of small prime factors [154].

Additionally, Wolf [185] showed that the SEDDH problem is reducible to the DDH problem but that the converse does not hold (see also [154]). The following theorem shows that if the DDH problem is solvable then so is the SEDDH problem.

Theorem 5.15 ($\mathtt{SEDDH} \leq \mathtt{DDH}$**).** *The* SEDDH *problem is polynomial-time reducible to the* DDH *problem and*

$$\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) \leq \mathsf{Adv}^{\mathtt{SEDDH}}_{\mathbb{G}}(\kappa).$$

Proof. Assuming that there exists a DDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}'$ we construct a SEDDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}$ as follows. On input a distribution (g, g^x, z) (either $\mathtt{SEDDH}^{\star}$ or $\mathtt{SEDDH}^{\$}$) the SEDDH distinguisher $\mathcal{A}$ chooses $s, t \in_R \mathbb{Z}_q$, calls $\mathcal{A}'$ on input $(g, (g^x)^s, (g^x)^t, z^{st})$ and returns its output. Note that if $\mathcal{A}$ has received $\mathtt{SEDDH}^{\star}$ then $(g, (g^x)^s, (g^x)^t, z^{st})$ corresponds to the distribution $\mathtt{DDH}^{\star}$. On the other hand, if $\mathcal{A}$ has received $\mathtt{SEDDH}^{\$}$ then $(g, (g^x)^s, (g^x)^t, z^{st})$ corresponds to the distribution $\mathtt{DDH}^{\$}$.

Thus, $\Pr[\mathcal{A}(\mathtt{SEDDH}^{\star}) = 1] = \Pr[\mathcal{A}'(\mathtt{DDH}^{\star}) = 1]$ and $\Pr[\mathcal{A}(\mathtt{SEDDH}^{\$}) = 1] = \Pr[\mathcal{A}'(\mathtt{DDH}^{\$}) = 1]$. This implies the desired inequality $\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) \leq \mathsf{Adv}^{\mathtt{SEDDH}}_{\mathbb{G}}(\kappa)$. □

5.5 Digital Signatures

As already noted in the introduction part digital signatures are useful cryptographic primitives for the purpose of identification and entity authentication. In the following we provide the formal treatment of this notion.

Definition 5.16 (Digital Signature Scheme). *A signature scheme* $\Sigma := (\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *consists of the following algorithms:*

`Gen`*: A probabilistic algorithm that on input a security parameter* 1^κ *outputs a secret key* sk *and a public key* pk*.*
`Sign`*: A probabilistic algorithm that on input a secret key* sk *and a message* $m \in \{0,1\}^*$ *outputs a signature* σ*.*
`Verify`*: A deterministic algorithm that on input a public key* pk*, a message* $m \in \{0,1\}^*$ *and a candidate signature* σ *outputs 1 or 0, indicating whether* σ *is valid or not.*

For our constructions we require digital signatures to be existentially unforgeable under chosen message attacks [99].

Definition 5.17 (EUF-CMA Security). *A digital signature scheme* $\Sigma := (\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *from Definition 5.16 is said to be* existentially unforgeable under chosen message attacks *(EUF-CMA) if for any PPT algorithm (forger)* $\mathcal{F}$ *the following success probability is negligible in* κ*:*

$$\mathsf{Succ}_{\Sigma}^{\texttt{euf-cma}}(\kappa) := \Pr\left[\begin{array}{c}(sk,pk) := \texttt{Gen}(1^\kappa);\\ (m,\sigma) \leftarrow \mathcal{F}^{Sign(sk,\cdot)}(pk);\end{array} : \begin{array}{c}\texttt{Verify}(pk,m,\sigma) = 1 \wedge\\ \neg Sign(sk,m)\end{array}\right].$$

Remark 5.18. There exists a stronger security requirement called *strong EUF-CMA* [9]. It allows $\mathcal{F}$ to query m to the signing oracle, and the forgery is successful if $\mathcal{F}$ returns (m,σ) such that σ was never returned in response to any query $\texttt{Sign}(sk,m)$. However, in our constructions we do not need this stronger property.

5.6 Techniques used in Security Proofs

5.6.1 The "Sequence of Games" Technique

As already mentioned in the introduction the technique called *"sequence of games"* described by Shoup [168] can be used to reduce complexity of reductionist security proofs.

Recall that the "sequence of games" technique subsumes the construction of a sequence of games $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_n$ starting with the original game between the adversarial algorithm $\mathcal{A}$ and its environment (simulator) where event Win symbolizes a successful attack. In the security proof for each game of the sequence one specifies events Win_i, $i = 0, \ldots, n$ whereby Win_0 is the original event Win and events Win_i, $i = 1, \ldots, n$ are usually related to Win. Then, one tries to show that $\Pr[\mathsf{Win}_i]$ is negligibly close to $\Pr[\mathsf{Win}_{i+1}]$. Games are usually constructed in such way that $\Pr[\mathsf{Win}_n]$ equals to some specified "target probability".

In order to simplify the estimation of $|\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}]|$ for any $i \in [0, n-1]$ each game $\mathbf{G}_{i+1}$ is usually constructed from small changes to $\mathbf{G}_i$. Shoup distinguishes between the following three transition types for the construction of successive games, i.e., transitions based on indistinguishability, transitions based on "failure events", and transitions based on "bridging steps". In the following we briefly describe these transitions and extend the framework by an additional type which we call *transitions based on "condition events"*.

Transitions based on indistinguishability

Such transitions are made based on the assumption that there are two computationally indistinguishable distributions D_0 and D_1, i.e., for any PPT algorithm (distinguisher) $\mathcal{D}$ its advantage probability in distinguishing between D_0 and D_1, denoted $\mathsf{Adv}_{\mathcal{D},D}(\kappa) := |\Pr[\mathcal{D}(D_0) =$

$1] - \Pr[\mathcal{D}(D_1) = 1]|$, is assumed to be negligible. One constructs games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ such that if their difference is detected by the adversary $\mathcal{A}$ then it is possible to construct a distinguisher $\mathcal{D}$ that on input D_0 outputs 1 with probability $\Pr[\mathsf{Win}_i]$ and on input D_1 outputs 1 with probability $\Pr[\mathsf{Win}_{i+1}]$. The indistinguishability assumption between D_0 and D_1 implies that $|\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}]|$ is negligible.

Transitions based on "failure events"

In this case one constructs $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ in a way that both games proceed identically unless some specified "failure event" F occurs. If both games are defined over the same probability space then both events $\mathsf{Win}_i \wedge \neg\mathsf{F}$ and $\mathsf{Win}_{i+1} \wedge \neg\mathsf{F}$ are equivalent.

The following lemma is essential to estimate the difference between probabilities $\Pr[\mathsf{Win}_i]$ and $\Pr[\mathsf{Win}_{i+1}]$.

Lemma 5.19 (Difference Lemma [168]). *Let* A, B, F *be events defined in some probability distribution, and suppose that* $\mathsf{A} \wedge \neg\mathsf{F} \iff \mathsf{B} \wedge \neg\mathsf{F}$. *Then*

$$|\Pr[\mathsf{A}] - \Pr[\mathsf{B}]| \leq \Pr[\mathsf{F}].$$

Proof. We compute

$$\begin{aligned} |\Pr[\mathsf{A}] - \Pr[\mathsf{B}]| &= |\Pr[\mathsf{A} \wedge \mathsf{F}] + \Pr[\mathsf{A} \wedge \neg\mathsf{F}] - \Pr[\mathsf{B} \wedge \mathsf{F}] - \Pr[\mathsf{B} \wedge \neg\mathsf{F}]| \\ &= |\Pr[\mathsf{A} \wedge \mathsf{F}] - \Pr[\mathsf{B} \wedge \mathsf{F}]| \\ &\leq \Pr[\mathsf{F}]. \end{aligned}$$

Note that the second equality follows from the assumption that $\mathsf{A} \wedge \neg\mathsf{F} \iff \mathsf{B} \wedge \neg\mathsf{F}$ which implies $\Pr[\mathsf{A} \wedge \neg\mathsf{F}] = \Pr[\mathsf{B} \wedge \neg\mathsf{F}]$. □

Following this lemma, in order to show that $|\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}]|$ is negligible it is sufficient to show the negligibility of $\Pr[\mathsf{F}]$.

Transitions based on "bridging steps"

These are transitions where $\Pr[\mathsf{Win}_i] = \Pr[\mathsf{Win}_{i+1}]$. Such transitions can be build by restating the way of computation of certain entities while keeping their resulting values equivalent. Transitions based on "bridging steps" are usually used to prepare transitions of the other two types described above.

Transitions based on "condition events"

In order to perform some of our proofs using the "sequence of games" technique we require an additional transition type for the construction of successive games. The background is given by Lemma 5.20, which we call Difference Lemma II.

Lemma 5.20 (Difference Lemma II). *Let* A, B, C *be events defined in some probability distribution, and suppose that* $\Pr[\mathsf{B}] = \Pr[\mathsf{A}|\mathsf{C}]$. *Then*

$$\Pr[\mathsf{A}] - \Pr[\mathsf{B}] \leq \Pr[\neg\mathsf{C}].$$

Proof. We compute

$$\begin{aligned}\Pr[\mathsf{A}] &= \Pr[\mathsf{A}|\mathsf{C}]\Pr[\mathsf{C}] + \Pr[\mathsf{A}|\neg\mathsf{C}]\Pr[\neg\mathsf{C}] \\ &= \Pr[\mathsf{B}]\Pr[\mathsf{C}] + \Pr[\mathsf{A}|\neg\mathsf{C}]\Pr[\neg\mathsf{C}] \\ &\leq \Pr[\mathsf{B}] + \Pr[\neg\mathsf{C}].\end{aligned}$$

□

In order to construct a new game $\mathbf{G}_{i+1}$ from the previous game $\mathbf{G}_i$ via a transition based on a "condition event" one proceeds as follows. One defines an appropriate "condition event" C and sets Win_{i+1} as the event that Win_i occurs given C. Then according to Lemma 5.20

$$\Pr[\mathsf{Win}_i] - \Pr[\mathsf{Win}_{i+1}] \leq \Pr[\neg\mathsf{C}].$$

Therefore, in order to estimate the probability distance between $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ it is sufficient to compute the probability of $\neg\mathsf{C}$. Note that in this form $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ proceed identical from the perspective of the adversary, and we are only interested in the probability of $\neg\mathsf{C}$. Therefore, it is not necessary for the simulator to detect whether this "condition event" occurs or not. This is an important difference to games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ built via transitions based on "failure events" where both games proceed identically unless the specified "failure event" has occurred so that in some cases the simulator must be able to detect this event in order to change the process of $\mathbf{G}_{i+1}$. Thus, transitions based on "condition events" additionally simplify the proof and can be applied in cases where transitions based on "failure events" are not directly applicable because of the problems with the detection of "failure events" by the simulator.

Another important observation of Lemma 5.20 is that by conditioning the success of the adversary with the event C we do not restrict the adversarial strategy. Note that the inequality

$$\Pr[\mathsf{Win}_i] = \Pr[\mathsf{Win}_{i+1}]\Pr[\mathsf{C}] + \Pr[\mathsf{Win}_i|\neg\mathsf{C}]\Pr[\neg\mathsf{C}] \leq \Pr[\mathsf{Win}_{i+1}] + \Pr[\neg\mathsf{C}]$$

considers both events, Win_{i+1} and $\mathsf{Win}_i|\neg\mathsf{C}$, thus focusing on one adversarial strategy represented by $\mathsf{Win}_{i+1} = \mathsf{Win}_i|C$ in the subsequent sequence game $\mathbf{G}_{i+1}$ does not rule out all other strategies represented by $\mathsf{Win}_i|\neg\mathsf{C}$ because the total probability for Win_i is still upper-bounded by $\Pr[\mathsf{Win}_{i+1}] + \Pr[\neg\mathsf{C}]$.

Remark 5.21. We apply transitions based on "condition events" in the proofs of Theorems 9.11, 9.16, 9.26, 9.34, 10.8, and 10.13.

5.6.2 The "Hybrid Technique"

The *"hybrid technique"* [94] (a.k.a. *"hybrid argument"*) is a special type of reductionist proofs where computational indistinguishability of *complex* distributions is proved by construction of computational indistinguishable *hybrid* distributions. Thus, the efficient distinguishability of the hybrid distributions is reduced to the efficient distinguishability of the complex distributions. Let D and D' be two complex distributions. In order to show their computational indistinguishability one defines a sequence of hybrid distributions $D_1, \ldots, D_n$ of a polynomially bounded length such that $D_1 = D$, $D_n = D'$, and shows that *any* two neighboring distributions D_i and D_{i+1}, $i \in [1, n-1]$ are computationally indistinguishable. The computational indistinguishability of D and D' follows from the polynomially bounded length of the sequence.

Remark 5.22. In some sense games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ constructed within the "sequence of games" technique via transitions based on indistinguishability can be considered as hybrid distributions.

Chapter 6

Analytical Survey on Security Requirements and Models for Group Key Exchange Protocols

In this chapter we describe security issues that are relevant for group key exchange protocols. We start in Section 6.1 with the security notions that have been informally described in the literature and widely used in security analyses of earlier protocols. Most of these definitions were originally stated for two-party protocols and then adapted to a group setting. These definitions can be considered as foundational for the later appeared formal security models whose development, strengths, and weaknesses we describe and analyze in Section 6.2. We summarize main results of our analysis in Section 6.3.

6.1 Survey on Informal Security Definitions

Security properties of cryptographic schemes are usually defined based on certain assumptions about the adversary whose goal is to break these properties. In case of cryptographic protocols it is common to distinguish between passive and active adversaries. A *passive adversary*, usually, only eavesdrops the communication channel without being able to modify or inject messages. An *active adversary* is more powerful since it is assumed to have a complete control over the communication channel resulting in its ability to alter sent messages or inject own messages during the execution of the protocol. In particular, an active adversary is able to mount so-called *man-in-the-middle attacks*. Additionally, security of a cryptographic protocol may depend on the behavior of its participants. Obviously, it is more challenging for a protocol to guarantee its security properties in case where legitimate participants are *malicious*, or dishonest, and do not act according to the protocol specification.

In the following we consider blocks of related security notions which we describe following the chronology of their appearance in the literature. We also specify which type of the adversary is reasonable to be assumed for each notion.

6.1.1 Semantic Security and Known-Key Attacks

The notion of *key privacy*, also called *key confidentiality* or *key secrecy* [80], was surfaced by Diffie and Hellman [79], and described later in the context of group key establishment [50, 171]. According to the definition of Burmester and Desmedt [50] a group key establishment protocol guarantees privacy if it is computationally infeasible for a passive adversary to compute the group key k. Obviously, similar definition should hold against an active adversary who is not a legitimate protocol participant. A stronger definition of key privacy requires the indistinguishability of the computed group key from a random number. Thus, an adversary given either a real group key or a random string sampled from the same space should not be able to distinguish which value it has been given. This is in spirit of the *semantic security* requirement proposed by Goldwasser and Micali [98] in the context of digital encryption schemes. Note, however, that this requirement can only hold under the assumption that the adversary does not obtain any additional information that would allow it to verify the given value, e.g., if the adversary obtains a cipher text computed using the real group key then it can easily distinguish between the real group key and some random value by comparing the corresponding cipher text.

The notion of *known-key security* [48, 189], strengthens the above requirements by assuming a stronger adversary who knows the group keys of past sessions. For example, members excluded from the group should not be able to compute or distinguish the updated group keys. The related notion of *key freshness* [135] requires that the protocol guarantees that the key is new, that is participants compute group keys which have not been used in the past. Steiner *et al.* [175] introduced the notion of *key independence* in the context of dynamic group key exchange protocols meaning that previously used group keys must not be discovered by joined group members and that former group members must not be able to compute the group keys used in the future. Obviously, this definition considers that the adversary was a legitimate protocol participant or may become one in the future.

Kim *et al.* [116, 117] summarized the above requirements as follows: *weak backward secrecy* guarantees that previously used group keys must not be discovered by new group members; *weak forward secrecy* guarantees that new keys must remain out of reach of former group members; *computational group key secrecy* guarantees that it is computationally infeasible for a passive adversary to discover any group key; *forward secrecy* guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys; *backward secrecy* guarantees that a passive adversary who knows a contiguous set of group keys cannot discover any preceding group keys; *key independence* guarantees that a passive adversary who knows any proper subset of group keys cannot discover any other group key. Note that the last four requirements do not make any assumptions about the group membership of the adversary. In their subsequent work, Kim *et al.* [119] introduced the *decisional group key secrecy* whereby a passive adversary must not be able to distinguish the group key from a random number. Although [116, 117] use passive adversaries in their definitions, it is mentioned that the same security requirements should also hold in the presence of active adversaries.

Remark 6.1. Unfortunately, Kim *et al.*'s definitions concerning (weak) forward secrecy are in conflict with the commonly used meaning of the term forward secrecy (see Section 6.1.4 for further details).

6.1.2 Impersonation Attacks

Security against *impersonation attacks* in the context of group key establishment was addressed by Burmester and Desmedt [50] and defined as a property of the protocol where an impersonator

together with active or passive adversaries should be prevented from the computation of the group key. By an impersonator [50] denotes an adversary whose goal is to replace a legitimate participant in the execution of the protocol (thus impersonator is not considered to be a malicious participant but rather some external party). Further, [48, 49] extend the notion of known-key attacks by requiring that an active adversary who knows past session keys must not be able to impersonate one of the protocol participants.

The notion of *entity authentication* [22], introduced by Bellare and Rogaway in the context of two-party authentication protocols, specifies a process whereby one party is assured of the identity of the second party involved in the protocol, and of the actual protocol participation of the latter. This requirement is equivalent to the requirement on resistance against impersonation attacks in the context of group key exchange protocols. The related notion called *(implicit) key authentication* [135] requires that each legitimate protocol participant is assured that no other party except for other legitimate participants learns the established group key. According to this definition a group key exchange protocol is *authenticated* if it provides (implicit) key authentication. Ateniese *et al.* [13] proposed a requirement on *group integrity* meaning that each protocol participant must be assured of every other party's participation in the protocol. Obviously, this notion is similar to the requirement of the entity authentication applied to a group setting.

All of these impersonation/authentication requirements consider an adversary that represents some external party and not a legitimate protocol participant. Therefore, these requirements are similar to those of the previous section assuming that the adversary is active, i.e., that the indistinguishability of the real group keys and the random numbers sampled from the same space remains preserved with respect to the attacks of an active adversary which is allowed to modify and inject messages during the protocol execution.

Another related requirement called *unknown key-share resilience* surfaced in [80] means that an active adversary must not be able to make one protocol participant believe that the key is shared with one party when it is in fact shared with another party. Note that in this attack the adversary may be a malicious participant and does not need necessarily to learn the established group key [38].

Finally, we mention *key-compromise impersonation resilience* [29]. This security property prevents the adversary who obtains a long-term key of a user from being able to impersonate other users to that one. Note that long-term (a.k.a. long-lived) keys are usually either private keys used for signature generation or digital decryption, or shared secret (small entropy) passwords that remain unchanged for a long period of time. In both cases long-lived keys are used primarily for the purpose of authentication rather than for the actual computation of the group key. Obviously, this attack concerns only protocols whose goal is to establish a session key which is then used for the purpose of authentication. Therefore, it is arguable (e.g. [111]) whether this requirement is general for all group key exchange protocols. Note that if an adversary obtains long-lived keys of participants then it can usually act on behalf of these participants in subsequent protocol executions.

6.1.3 Key Confirmation and Mutual Authentication

The requirement called *key confirmation* [135] means that each protocol participant must be assured that every other protocol participant actually has possession of the computed group key. According to [135] key confirmation in conjunction with (implicit) key authentication results in *explicit key authentication*, i.e., each identified protocol participant is known to actually possess the established group key. The same goal states the requirement of *mutual authentication*

introduced in [23] when considered for group key exchange protocols. As noted in [13] key confirmation makes a group key exchange protocol a more robust and a more autonomous operation. According to [38] key confirmation mechanisms can be used to provide resistance against unknown key-share attacks mentioned in the previous section. We stress that the requirements on key confirmation and mutual authentication should also be considered from the perspective of the attacks by malicious protocol participants who try to prevent honest participants from computing identical group keys.

6.1.4 Perfect Forward Secrecy

The notion of *(perfect) forward secrecy* (sometimes called *break-backward protection* [135]) was surfaced by Günter [102] and rephrased by Diffie *et al.* [80] as a property of an authenticated key agreement protocol requiring that the disclosure of long-term keying material does not compromise the secrecy of the established keys from earlier protocol sessions. The idea behind this notion is to maintain the protection of the secure traffic in the future. Note that the compromised long-term keys make future protocol sessions nonetheless susceptible to impersonation attacks. A weaker form of (perfect) forward secrecy is *partial forward secrecy* [38] which considers the case where one (or more but not all) principals' long-term keys become compromised.

6.1.5 Key Control and Contributiveness

The issue of *key control* described by Mitchel *et al.* [138] in the context of two-party group key exchange protocols considers malicious protocol participants who wish to influence the computation of the group key.

Ateniese *et al.* [13] introduced a notion of *contributory group key agreement* meaning such protocols where each party equally contributes to the established group key and guarantees its freshness (see also [173]). This notion also subsumes the requirement of *unpredictability* of the computed group keys. Note that these requirements clearly exclude group key distribution protocols (see Section 2.2) where some trusted party is responsible for the generation of the group keys. Ateniese *et al.* defined additionally *complete group key authentication* as a property of a group key exchange protocol whereby all parties compute the same group key only if each of the parties have contributed to its computation. This notion can be seen as a combination of contributiveness and mutual (or explicit key) authentication. Ateniese *et al.* [13] proposed further a more stronger notion of *verifiable contributory group key agreement* meaning protocols where each participant is assured of every other participant's contribution to the group key. We stress that these requirements should also hold in the presence of malicious protocol participants.

Another related requirement that is stated from the perspective of an adversary that is not a malicious participant is *key integrity* [108] which requires that the established group key has not been modified by the adversary, or equivalently only has inputs from legitimate protocol participants. Ateniese *et al.* [13] extended the definition of key integrity by requiring that the established group key is a function of only the individual contributions of legitimate protocol participants such that extraneous contribution(s) to the group key must not be tolerated even if it does not afford the adversary with any additional knowledge. Obviously, this can be achieved if the protocol is contributory and provides mutual authentication.

6.2 Analytical Survey on Formal Security Models

As already noted in the introduction provable security of cryptographic protocols can be achieved using an appropriate security model that considers protocol participants, their trust relationship, communication environment, and further relevant aspects, and contains definitions of required security goals.

Unfortunately, there exist no common *goodness* criteria for the evaluation of such security models. In our opinion a security model should be *abstract* meaning that it should not depend on any implementation-specific definitions or assumptions. Further, a model should be *self-contained*, i.e., there should be no parameters whose specification is not defined within the model or depends on certain assumptions beyond it. This property allows design of autonomous protocols. A model should be *precise*, i.e., it should disallow any ambiguous interpretations for its definitions and requirements. A security model should be *modular*, i.e., allow security proofs for protocols that provide only a subset of specified security goals. This property allows design of protocols with respect to their practical deployment in applications that do not require the full range of security; on the other hand it allows construction of generic solutions. Another advantage of modular security models is a possible integration of additional security definitions which may become essential in the future.

In the following we provide an analytical survey of security models proposed for group key exchange protocols (note that these models can or have already been used for security proofs of the protocols described in Chapter 7). In addition to the description we specify which of the most important informal security requirements have been considered by the definitions of a model, thereby focusing on semantic security or indistinguishability of the computed group keys from random numbers considering known-key attacks and active adversaries, key confirmation and mutual authentication with respect to malicious protocol participants, (perfect) forward secrecy, and the issues related to key control and contributiveness. Additionally, we judge each model with respect to the specified goodness criteria. Some of the models described in this section were proposed in the context of two- or three-party key exchange protocols. However, they provide some interesting definitions and constructions that became foundational for a variety of the later appeared security models designed for the group setting.

6.2.1 Models by Bellare and Rogaway (BR, BR$^+$)

BR

Bellare and Rogaway [22] proposed the first computational security model for authentication and security goals of two-party key exchange protocols which we refer to as the BR model. This model allows reductionist proofs of security. Each protocol participant is assumed to have an identity and a long-lived key. The adversary can initiate different sessions between the same participants. It has an infinite collection of *oracles* $\Pi_{i,j}^{s}$ such that each oracle represents participant i trying to authenticate participant j in session s. The adversary communicates with the oracles via queries which contain sender and receiver identities, the session id, and the actual message. Hence, the adversary is considered to be active. However, the model assumes a *benign* adversary which faithfully forwards all message flows between the oracles, and is, therefore, not allowed to modify the messages. It can invoke any oracle to start the protocol execution. A variable $\kappa_{i,j}^{s}$ keeps track of the conversation between i and j in session s. The security goal of mutual authentication is defined based on the notion of *matching conversations* between the

participants. Roughly, this means that all messages sent by one participant have been subsequently delivered to another participant without modification, and vice versa. According to the BR model a protocol provides mutual authentication if for any polynomial time adversary the oracles $\Pi^s_{i,j}$ and $\Pi^s_{j,i}$ have matching conversations and if one of the oracles, say $\Pi^s_{i,j}$, accepts (does not fail) then there is always another oracle $\Pi^s_{j,i}$ with the engaged matching conversation. Note that from this definition mutual authentication also results in key confirmation since the exchanged message flows, and thus computed keys, must be equal. The authors also show the uniqueness of matching partners. Additionally, the adversary is allowed to ask *Reveal* queries to obtain session keys computed by $\Pi^s_{i,j}$. In order to model known-key attacks the BR model specifies the notion of *fresh* oracles, i.e., an oracle $\Pi^s_{i,j}$ is fresh if it has accepted (computed the session key) and no *Reveal* query was asked to $\Pi^s_{i,j}$ or to its matching partner $\Pi^s_{j,i}$. Further, the adversary is allowed to ask exactly one *Test* query to any oracle which is fresh. In response to this query it receives either the real session key computed by the oracle or a random number of the same range and has to decide which value it has received. The BR model calls an authenticated key exchange protocol secure if in the presence of the benign adversary both oracles, $\Pi^s_{i,j}$ and $\Pi^s_{j,i}$, accept with the equal session keys which are randomly distributed over the key space, and the success probability of the adversary to decide correctly in response to the value obtained from its *Test* query is non-negligibly greater that $1/2$.

The main weakness of the BR model is that it disallows the adversary to modify messages, or to corrupt participants obtaining their long-lived keys. Hence, the model does not consider the notion of (perfect) forward secrecy. Further, the model does not deal with attacks of malicious participants. Thus, definition of mutual authentication is defined only for honest protocol participants, and no definitions concerning issues related to key control and contributiveness are available in the BR model.

BR$^+$

In their subsequent work, Bellare and Rogaway [24] extended the BR model to deal with key distribution scenarios in a three-party setting which involves two participants wishing to establish a shared key and a key distribution center (key server). Nonetheless, this model, denoted BR$^+$, is of particular interest for us since it provides some interesting definitions which are also relevant for the models dealing with key exchange protocols. The actions of the adversary are specified by a number of queries which it may ask to the *instances* of parties participating in the protocol. Each party may have a multiple number of instances and so participate in different sessions of the protocol. Using a *SendPlayer* or a *SendS* query the adversary can send messages to one of the participants or to the key distribution center, respectively, that reply according to the protocol specification or do not reply if the received message is unexpected. So the adversary is active. With a *Reveal* query to a specified instance the adversary may obtain the final key computed by that instance. Additionally, the adversary is allowed to ask *Corrupt* queries which return the complete internal state of the instance to the adversary together with the long-lived key of the party and allows the adversary to replace this long-lived key with any value of its choice. Note that this kind of corruptions became later known as *strong corruptions*. After an adversary asks a *Corrupt* query for some party all instances of this party use the changed value for the long-lived key in all subsequent protocol executions. Although the adversary is allowed to corrupt parties and reveal their session keys the security of the protocol may also depend on instances of other parties who participate in the same session. To consider all protocol participants the BR$^+$ model specifies an abstractly defined *partner function* which roughly speaking

means that two instances are partnered if they participate in the same session and compute the shared key. Further, *at the end* of its execution the adversary asks a *Test* query to an instance which holds a *fresh session key*, i.e., a key computed during the session such that no *Reveal* or *Corrupt* queries have been previously asked to the instance or any of its partners. The adversary receives either the session key computed by that instance or a random number of the same range, and similar to the BR model must decide which value it has received. This security definition subsumes the informal requirement on indistinguishability of computed group keys from random numbers while also considering active adversaries.

The BR^+ model has some weaknesses described in the following. Although the adversary is allowed to reveal long-lived keys of participants through *Corrupt* queries it is allowed to ask its *Test* query only at the end of its execution. Obviously, the adversary may not use the knowledge of corrupted long-term keys to make its guess because of the freshness requirement. Therefore, the model does not capture the requirement of (perfect) forward secrecy. Second, the BR^+ model does not deal with the attacks concerning key confirmation or mutual authentication, neither with respect to honest participants nor to malicious. This observation has also been mentioned in [72, 73]. Third, the BR^+ model does not consider attacks related to the issue of key control. This, however, is reasonable since the model has been proposed for key distribution protocols for which such requirements are not relevant because of the trust assumption concerning the key server. The partnering function in the BR^+ model is not concretely specified. This contradicts to our goodness criteria for the security models.

6.2.2 Model by Bellare, Canetti, and Krawczyk (BCK)

Bellare, Canetti, and Krawczyk [20] proposed a computational security model for authentication and key exchange, which we denote as the BCK model. This model allows security proofs based on the simulatability approach. The BCK model supports modular constructions and deals with *message-driven* protocols, i.e., after being invoked by a party the protocol waits for an activation which may either be caused by the arrival of a certain message or by an external request (which may come from other processes executed by the party). The authors essentially define two different adversarial models: *authenticated-links model* (AM) and *unauthenticated-links model* (UM). The AM model considers a passive adversary who has a full control over the communication channel but is assumed to deliver messages faithfully without modifying any of them, however, is allowed to change their delivery order. Further the adversary is allowed to activate any party using external requests, but not own protocol messages. The UM model assumes that the adversary is active, i.e., can activate parties with arbitrary incoming messages. Further, the BCK model specifies the notion of *emulation of protocols* in UM using a so-called authenticator which is considered to be a compiler translating the execution of a protocol in AM into UM while preserving its security requirements. Similar to the BR^+ model the BCK model considers several executions of the protocols, and calls each execution a *session*. In order to distinguish different sessions the model uses *session ids* which should be unique for the sender and the receiver (recall that the model was defined for two parties). The adversary in the AM and UM models is allowed to *corrupt sessions* such that it learns the internal state associated with a particular session identified via unique session IDs for which the model does not provide any concrete construction. Although the BCK model was proposed in the context of two-party key exchange protocols, the authors tried to provide definitions which also hold in a multi-party setting. They defined the notion of the *ideal key exchange* and the *ideal adversary*. The ideal adversary is allowed to invoke any party to establish the session key with any other

party such that the adversary learns the transcript of the exchanged protocol messages and the session id value, but not the established key. Further, if the ideal adversary corrupts a session using the corresponding session id then it obtains the established key for this session, and if the adversary corrupts a party then it obtains all keys (including long-lived key) known to this party. Corrupted parties may continue participating in the protocol. However, in this case the model allows the adversary to choose the established keys. Therefore, no security definitions related to the requirement on (perfect) forward secrecy are considered. For the same reason, the BCK model does not provide any security definitions for the case in which honest participants interacting with the adversary represented as a (subset of) malicious participant(s) try to contribute to the resulting group key. Hence, the BCK model does not consider attacks concerning key control and unpredictability. Also, the BCK model does not capture possible attacks of malicious participants against key confirmation and mutual authentication. Note that the corrupt query reveals internal state information together with the long-lived key of a party. This, however, disallows consideration of scenarios where long-lived keys are revealed without revealing the internal state information (note that long-lived keys may have a different protection mechanism). In order to define security goals the BCK model specifies the notion of a *global output* of the protocol execution in the presence of the adversary. It consists of cumulative concatenations of the outputs (sent messages) of all parties and their random inputs, together with the output of the adversary which is a function of its random input and all information seen by the adversary throughout its execution. A key exchange protocol is called secure in the BCK model if the global output of the ideal protocol execution is indistinguishable from the global output of the protocol execution in either the AM or UM model. This is the typical approach for security models that allow security proofs based on simulatability/indistinguishability as described in Section 3.3.

Interesting about the BCK model is that its modular construction allows to prove protocol security in the AM model and then apply the described authenticator to obtain a protocol which is secure in the UM model.

6.2.3 Model by Bellare, Pointcheval and Rogaway (BPR)

The following model proposed by Bellare, Pointcheval, and Rogaway in [21], which we denote BPR, is based on the previously described BR^+ model, and considers two-party key exchange protocols. The BPR model is described w.r.t. the communication between a client and a server. Similar to the BR^+ model each participant may have different instances, called *oracles*. In addition to the queries *Reveal* and *Test* which return the computed key of the instance respectively the computed session key or a random number of the same range (in contrast to the BR^+ model the adversary in the BPR model may ask the *Test* query at any time during its execution and not only at the end), the BPR model specifies *Execute* and *Send* queries. *Execute* queries can be used by the adversary to invoke an honest execution of the protocol and obtain a transcript of exchanged messages. The *Send* query allows the adversary to send messages to the instances, i.e., behave actively. Send queries can also be used to achieve honest execution (as *Execute* queries) simply by invoking the protocol execution at instances of adversary's choice and then forwarding messages between these instances without any modification. However, *Execute* query allows on the one hand a better handling of dictionary attacks in case where long-lived keys are shared passwords, because the adversary can be granted access to plenty of honest executions, and on the other hand it allows to treat passive adversaries separately though the BPR model does not take use of this second advantage. Additionally, the BPR model allows the adversary to ask *Oracle* queries in order to deal with non-standard models, like the Random Oracle Model (ROM)

[23] or the Ideal Cipher Model (ICM) [27, 162]. In case when the protocol is designed to achieve security in the standard model the *Oracle* query can be omitted. The BPR model specifies two forms of a *Corrupt* query (unlike the previously described models): a strong form (called *strong corruption model*) means that the adversary obtains the long-lived key of the party and its internal state (excluding the session key), and a weak form (called *weak corruption model*) means that the adversary obtains only the long-lived key of the party.

The model assumes the existence of unique *session ids* and specifies *partner ids*. The partner id of an instance is a public value and consists of the identities of all parties with which the oracle believes it has just exchanged the session key with. According to the BPR model *two oracles are partnered* if they compute (accept with) equal session keys, equal session ids, have each other's identity as part of the computed partner ids, and there is no other oracle who accepts with the same partner id. The BPR model defines two flavors of session key freshness: a session key is *fresh* if there have been no *Reveal* queries to the oracle or any of its partners, and no *Corrupt* queries at all; a session key is *fs-fresh* (*fs* for forward secrecy) if there have been no *Reveal* queries to the oracle or any of its partners, and if there have been no *Corrupt* queries prior to the *Test* query such that further *Send* queries have been asked to the tested oracle. The latter means that the adversary can corrupt participants before the test session but is then not allowed to send any messages to the oracle whose key (or a random value instead) it later receives in response to its *Test* query. Based on these two flavors the model provides two definitions of security against known-key attacks: (1) security without forward-secrecy meaning that the adversary asks a *Test* query to an oracle which holds a fresh session key, and (2) security with forward-secrecy meaning that the adversary asks a *Test* query to an oracle which holds a fs-fresh session key. Similar to the BR and BR^+ models the goal of the adversary is to distinguish whether it obtains a session key or a random number. Obviously, this security definition subsumes the informal requirements related to the indistinguishability of group keys from random numbers with respect to active adversaries. Furthermore, security proofs in which the *Test* query is asked to an oracle holds a fs-fresh session key consider attacks related to (perfect) forward secrecy.

Additionally, the BPR model gives a definition of *server-to-client* and *client-to-server authentication* which are violated in case where a server respectively a client accepts with a session key but does not have any partner. *Mutual authentication* is, therefore, violated if either a server-to-client or client-to-server authentication is violated (recall that the BPR model was proposed for the two-party key exchange protocols).

In the BPR model partnering is defined using session ids. However, the authors do not provide further details within the model concerning the construction of the unique session ids. In the proposed protocol, however, they are constructed as a concatenation of all flows exchanged between both participants. Note that this is an appropriate method in case of two parties, however, cannot be generally applied to the multi-party case where participants do not generally need to send each message to every other participant. Similar to the BR^+ model the BPR model does not consider attacks related to the issues of key control and contributiveness.

6.2.4 Model by Canetti and Krawczyk (CK)

Canetti and Krawczyk [62] proposed a formal model, which we refer to as CK, based on the methodology of the BR and BCK models. Similar to the BCK model the CK model deals with message-driven protocols that involve only two parties. Different executions of a protocol are called sessions which are identified by unique session ids. The CK model describes the notion of *matching sessions* (related to the matching conversations in the BR model), and treats participants of matching sessions as *partners*.

As the BCK model the CK model specifies an unauthenticated-links (UM) and an authenticated-links (AM) adversarial models. In the UM model the adversary passes messages from one participant to another, but has control over their scheduling (including initiation of the protocol), and is allowed to ask *Reveal* queries to obtain the computed session keys and *Corrupt* queries to obtain all the internal memory of the party including its long-lived key and specific session-internal information (such as internal state of incomplete sessions and session-keys of already completed sessions). From the moment a party is corrupted it is fully controlled by the adversary. This models attacks against (perfect) forward secrecy. Additionally, in the CK model the adversary is allowed to reveal internal state of a party for an incomplete session without necessarily corrupting that party (we call this *RevealState* queries). In the CK model a session becomes *locally exposed* if any of these three queries (*Reveal*, *RevealState*, or *Corrupt*) have been asked to a party during that session, and the session becomes *exposed* if it or any of its matching sessions are locally exposed. Further, the CK model specifies *session expiration* which can be performed by a party causing the erasure of that session key and any session-specific information from the party's memory, and used in the model for the definition of security without (perfect) forward secrecy. In the AM model the adversary has the same capabilities as in the UM model, but is required to pass messages between the parties truly, i.e., without modifying them (this is comparable to a passive adversary who only eavesdrops the communication). The UM and AM models are linked together over the emulation paradigm based on authenticators as in the BCK model.

The CK model introduces the notion of *session-key security* as a security goal for the key exchange protocols. The definition in the UM model allows the adversary to ask a *Test* query (with similar response as in the BR, BR^+, and BPR models) to a party during the session which is completed, unexpired and unexposed at that time. Having asked this *Test* query the adversary is allowed to continue with regular actions according to the UM model but is not allowed to expose the test-session. At the end of its execution the adversary has to output a guess concerning the response of the *Test* query. A protocol is called *session-key secure* if for any UM-adversary holds that if any two parties complete matching sessions then they compute the same session key, and that the probability of the adversary's correct guess is no more than $1/2$ plus a negligible fraction. This definition also captures informal requirements on indistinguishability notions considering known-key attacks and active adversaries. Additionally, the CK model provides a weaker definition of security for protocols in which no (perfect) forward secrecy is available or desirable. For this purpose the model disallows session expirations. A protocol is called *session-key secure without (perfect) forward secrecy* if it is session-key secure and the UM-adversary is not allowed to corrupt any partner from the test-session, i.e., the security of the session key can be no more guaranteed if any partner who computes this key gets corrupted. The authors mention that similar definitions are applicable for the AM model.

Compared to the BCK and BR models, the CK model provides a stronger adversarial setting since it allows *RevealState* queries. Interesting is that session-key security in the CK model implies the known-key security of the protocol in the BR^+ and BPR models. For the proof of this fact and further analysis of the relations between the security definitions in the BR, BR^+, BPR and CK models we refer to the work of Choo, Boyd and Hitchcock [73]. Note that one drawback of the CK model in the context of key exchange is that it leaves the construction of the session ids open, and this might have an impact on the security of the protocols designed based on this model. Also the CK model does not deal with the issues of key confirmation and mutual authentication as well as key control and contributiveness.

6.2.5 Model by Shoup

Shoup [167] proposed a modular formal model for secure key exchange between two parties that can be seen as an extension of the BCK model. Shoup's model considers protocol composition, i.e., it contains security definitions for the case where the key exchange protocol should be used within another high-level application protocol. The model allows security proofs based on simulatability approach. It consists of two settings: the *ideal-world* model and the *real-world* model (this is somewhat similar to the ideal key exchange process in the BCK model). For each setting there exists a different adversary. For both the real-world and the ideal-world adversaries a transcript is generated which consists of all occurred events and messages. The security of the protocol for every real-world adversary means that there exists a corresponding ideal-world adversary, such that the generated transcripts are computationally indistinguishable. Shoup's model specifies three corruption settings: *static*, *adaptive*, and *strong adaptive* corruptions.

In case of *static corruptions* the ideal model allows the ideal adversary to initialize the party and its instance (oracle) using the identity of its partner, abort a session between two instances, invoke a session between two instances using a so-called *connection assignment* which specifies how the session key is computed, i.e., either honestly, or using the key of another connection, or chosen by the adversary (compromised). The latter is available only if this instance is not partnered with any other party's instance. As noted by Shoup, connection assignments are used instead of session ids to identify the connection between two parties. Further, the ideal-world adversary is allowed to call the *application* query which returns a partial information about the session key computed as a function within the application protocol. Additionally, the ideal-world adversary is allowed to ask an *implementation* query which simply adds the attached comment to the transcript. At the end of the adversary's execution a transcript which contains all actions (and outputs) taken by the adversary is generated. In the real-world model there is an additional trusted party which can be queried by the adversary in order to register the identities of the parties. However, the model requires that if a party is registered by the adversary then its identity should not be in the set of identities of the parties for which the adversary used its initialization query. So the adversary is able to obtain a long-lived key for a party which will not be initialized, and, therefore, will not participate actively in the honest protocol execution (this is the reason for the notion of static corruptions). In this case the model does not consider attacks concerning the interaction between honest and malicious participants, that are issues of key confirmation and mutual authentication, (perfect) forward secrecy, and key control. Instead of session invocations the real-world adversary is allowed to *deliver* messages (via a corresponding query) to the instances who process them and return a protocol-specific output together with a status information which specifies whether the instance is waiting for further messages (*continue*), has already computed the session key (*accept*), or is finished without being able to compute the key (*reject*). At the end of the execution of the real-world adversary a transcript with all its actions is generated. For the setting of static corruptions Shoup specifies three security goals: *termination* (each instance either accepts or rejects after a polynomially bounded number of sessions), *liveness* (whenever the real-world adversary faithfully delivers all protocol messages both instances accept with the same session key), and *simulatability* (for every real-world adversary there exists a ideal-world adversary such that the transcripts of their actions are computationally indistinguishable).

In case of *adaptive corruptions* the model extends the requirements from the setting of static corruptions. The real-world adversary is additionally allowed to corrupt parties and reveal their long-lived keys before their initialization, and then register them by the trusted party. This enables the adversary to participate in the protocol on behalf of the corrupted user. The same

operation is available to the ideal-world adversary, however, the adversary is allowed to choose a compromised connection assignment in its session invocation query to a party's instance either if this party is not partnered with any other party, or the party is partnered with another corrupted party, or the party itself is already corrupted. These changes capture the notions of key confirmation and mutual authentication, and (perfect) forward secrecy.

The setting of *strong adaptive corruptions* extends the setting of adaptive corruptions and considers a more powerful real-world adversary who obtains not only the long-lived keys, but also the internal state information if this has not been previously erased by the high-level application protocol. The same operation is given to the ideal-world adversary. This kind of corruptions is comparable to the strong corruption model of the BPR model.

Additionally, Shoup's model specifies similar security definitions for key exchange protocols between anonymous parties, i.e., parties whose identities are not registered with a trusted party. In these protocols the established key is used to derive a password which can be used for the purpose of authentication. Further, Shoup compares his model to the BR$^+$ and BCK models. This results in the equivalence of some security notions, like security against static and adaptive corruptions in the Shoup's and the BR$^+$ models.

As already noted Shoup's model is strongly focused on the composition of key exchange protocols with high-level application protocols. Therefore, it is unavoidable that some definitions of the model rely on the assumptions about the application protocol, e.g., computation and erasure of session keys. Further, security definitions in Shoup's model do not consider attacks related to key control and contributiveness.

6.2.6 Model by Bresson, Chevassut, Pointcheval, and Quisquater (BCPQ)

The formal model proposed by Bresson, Chevassut, Pointcheval and Quisquater [46], which we refer to as the BCPQ model, is truly the first computational security model which has been designed for group key exchange protocols. The model allows reductionist security proofs and extends the methodology used in the BR, BR$^+$, and BPR models to a group setting. Similar to the mentioned models each protocol participant $U_i \in ID$[1], $i = 1, \ldots, n$ is modeled by an unlimited number of instances called *oracles* and denoted $\Pi_i^{s_i}$ (s_i-th instance of U_i) that can be involved in different concurrent protocol executions. Each user U_i is assumed to have a long-lived key LL_i (either symmetric or asymmetric). As in the BPR model the BCPQ model uses session ids to define the notion of partnering used in the definition of security goals. Unlike the BPR model which assumes the existence of unique session ids the BCPQ model describes their concrete construction. A session id of an oracle $\Pi_i^{s_i}$ is defined as $\mathrm{SID}(\Pi_i^{s_i}) := \{\mathrm{SID}_{ij} \mid U_j \in ID\}$ where SID_{ij} is the concatenation of all flows that $\Pi_i^{s_i}$ exchanges with another oracle $\Pi_j^{s_j}$. According to the BCPQ model two oracles $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are called *directly partnered*, denoted $\Pi_i^{s_i} \leftrightarrow \Pi_j^{s_j}$, if both oracles *accept* (compute the session key) and if $\mathrm{SID}(\Pi_i^{s_i}) \cap \mathrm{SID}(\Pi_j^{s_j}) \neq \emptyset$. Further, oracles $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are *partnered* if there exists a graph $G_{\mathrm{SIDS}} := (V, E)$ with $V := \{\Pi_l^{s_l} | \ U_l \in ID, \ l = 1, \ldots, n\}$ and $E := \{(\Pi_l^{s_l}, \Pi_{l'}^{s_{l'}}) | \ \Pi_l^{s_l} \leftrightarrow \Pi_{l'}^{s_{l'}}\}$ such that there exists a sequence of oracles $(\Pi_{l_1}^{s_{l_1}}, \Pi_{l_2}^{s_{l_2}}, \ldots, \Pi_{l_k}^{s_{l_k}})$ with $l_k > 1$, $\Pi_i^{s_i} = \Pi_{l_1}^{s_{l_1}}$, $\Pi_j^{s_j} = \Pi_{l_k}^{s_{l_k}}$, and $\Pi_{l-1}^{s_{l-1}} \leftrightarrow \Pi_l^{s_l}$ for all $l = l_2, \ldots, l_k$. This kind of partnering is denoted $\Pi_i^{s_i} \leftrightsquigarrow \Pi_j^{s_j}$. The BCPQ model uses graph G_{SIDS} to construct (in polynomial time $|V|$) the graph of partnering $G_{\mathrm{PIDS}} := (V', E')$ with $V' = V$ and $E' = \{(\Pi_l^{s_l}, \Pi_{l'}^{s_{l'}}) \mid \Pi_l^{s_l} \leftrightsquigarrow \Pi_{l'}^{s_{l'}}\}$, and defines the *partner id* for an oracle $\Pi_i^{s_i}$ as $\mathrm{PIDS}(\Pi_i^{s_i}) = \{\Pi_l^{s_l} \mid \Pi_i^{s_i} \leftrightsquigarrow \Pi_l^{s_l} \ \forall \, l \in \{1, n\} \setminus \{i\}\}$.

[1] ID is a set of n participants involved in the *current* protocol execution and is part of a larger set that contains all possible participants.

The adversary $\mathcal{A}$ in the BCPQ model is allowed to send messages to the oracles (and invoke the protocol execution) via $Send$ queries, reveal the session key computed by the oracles via $Reveal$ queries, obtain long-lived keys of the users via $Corrupt$ queries (note that the oracle's internal state information is not revealed, that is similar to the weak-corruption notion in the BPR model), and ask a $Test$ query to obtain either a session key or a random number. Using this adversarial setting the BCPQ model specifies two security goals for a group key exchange protocol: *authenticated key exchange (AKE) security* and *mutual authentication (MA) security*, both based on the notion of partnering.

For the AKE-security the model requires that during its execution $\mathcal{A}$ which is given access to the above mentioned queries asks a single $Test$ query to an oracle which is fresh. An oracle Π_i^s is *fresh* if: (1) it has accepted, (2) neither Π_i^s nor any of its partners have been asked for a $Corrupt$ query before Π_i^s accepts, and (3) neither Π_i^s nor any of its partners have been asked for a $Reveal$ query. A group key exchange protocol is said to be AKE-secure if the probability that $\mathcal{A}$ correctly guesses which value it has received in response to its $Test$ query, i.e., the session key or a random number, is negligibly greater than that of a random guess. This definition of AKE-security subsumes the informal security goals related to the indistinguishability of group keys from random numbers with respect to known group keys of other sessions in the presence of active adversaries, as well as the requirement of (perfect) forward secrecy.

The definition of MA-security in the BCPQ model is intended to capture the intuitive notion that it should be hard for a computationally bounded adversary $\mathcal{A}$ to impersonate any participant U_i through its oracle $\Pi_i^{s_i}$. For this purpose the authors require that the probability that during the execution of $\mathcal{A}$ which is given access to the above queries (thereby $Test$ query is irrelevant) there exists at least one oracle $\Pi_i^{s_i}$ which accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| \neq n-1$ is negligible. In other words, if each participating oracle $\Pi_i^{s_i}$ has accepted with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$ then no impersonation attacks could have occurred, thus the informal notion of mutual authentication meaning that each participating oracle is assured of every other oracle's participation in the protocol is satisfied. In the following paragraph we show that this is not generally the case, i.e., that there exists protocols where $\mathcal{A}$ impersonates U_i through some $\Pi_i^{s_i}$ but nevertheless all participating oracles accept and remain partnered, i.e., $|\mathrm{PIDS}(\Pi_j^{s_j})| = n-1$ for every participating $\Pi_j^{s_j}$. Further, the authors claim

> In the definition of partnering, we do not require that the session key computed by partnered oracles be the same since it can easily be proven that the probability that **partnered** oracles come up with different session keys is negligible. [46, Footnote 3]

> We are not concerned with partnered oracles coming up with different session keys, since our definition of partnering implies the oracles have exchanged *exactly* the same flows. [46, Section 7.4]

If these claims hold then the above definition of MA-security captures further informal security goals related to key confirmation and mutual authentication (but only for honest protocol participants). In the following paragraph we explain that these claims do not hold for just any GKE protocol either. We show that an impersonation attack may likely result in different group keys accepted by different partnered oracles. In fact we are able to show that the definition of MA-security in the BCPQ model is not general enough to be used just for any GKE protocol, i.e., if in a GKE protocol every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$ then it does not necessarily mean that this protocol provides mutual authentication and key confirmation.

Additionally, we stress that the BCPQ model does not consider attacks aiming to reveal the internal state information (strong corruptions) and attacks of malicious participants aiming to control the resulting key value.

Problems with the definition of MA-Security in the BCPQ model

We provide examples for the following two problems: (1) there exists GKE protocols where an active adversary $\mathcal{A}$ can impersonate one of the participants through its oracle but nevertheless every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$ (the definition of MA-security in the BCPQ remains satisfied even though impersonation attacks have occurred); (2) there exists GKE protocols where each participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$ but there are at least two partnered oracles that have computed different keys (the definition of MA-security in the BCPQ remains satisfied even though some of the oracles complete with different group keys). Note that these problems become visible only in the group setting with at least three protocol participants. Therefore, it does not concern the original definition of mutual authentication given by Bellare and Rogaway [22] based on matching conversations.

Before we give examples using a concrete GKE protocol we provide an abstract description. Figure 6.1 shows the abstract messages denoted m_i (index i specifies the order in which messages have been sent) that have been exchanged between the oracles (at least three participants are required) during the honest execution of any GKE protocol from [41, 42, 43, 46]. A concrete equivalent message of each abstract message m_i can be found in the corresponding *up-* or *downflow* stage of any of these GKE protocols. By m_i at the beginning of the arrow we mean the original message sent by the oracle, and by m_i at the end of the arrow we mean the corresponding message received by another oracle. If both messages are equal then the original message was not modified during the transmission.

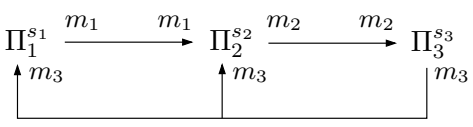

Fig. 6.1. Example: Honest Execution of Protocols in [41, 42, 43, 46]

$\mathrm{SID}(\Pi_i^{s_i})$	SID_{i1}	SID_{i2}	SID_{i3}
$\mathrm{SID}(\Pi_1^{s_1})$	$\emptyset$	m_1	m_3
$\mathrm{SID}(\Pi_2^{s_2})$	m_1	$\emptyset$	(m_2, m_3)
$\mathrm{SID}(\Pi_3^{s_3})$	m_3	(m_2, m_3)	$\emptyset$

Fig. 6.2. Example: $\mathrm{SID}(\Pi_i^{s_i})$ in the Honest Protocol Execution

Obviously, Figure 6.1 shows a correct execution of the protocol since there are no modified messages. Figure 6.2 specifies the session ids of the oracles $\Pi_1^{s_1}, \ldots, \Pi_3^{s_3}$ during this honest protocol execution using the construction from the BCPQ model. We assume that the protocol is correct, thus it is clear that each participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$. To show the first problem we consider the case where $\mathcal{A}$ impersonates U_1 and modifies message m_1 to $\tilde{m}_1$ (Figure 6.3) such that $\mathrm{SID}_{21} = \tilde{m}_1$ (Figure 6.4). Our goal is to show that nevertheless

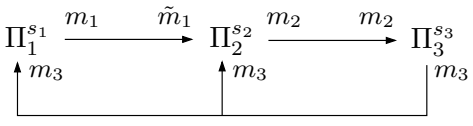

Fig. 6.3. Example: Protocol Execution with Impersonation Attack

$\mathrm{SID}(\Pi_i^{s_i})$	SID_{i1}	SID_{i2}	SID_{i3}
$\mathrm{SID}(\Pi_1^{s_1})$	$\emptyset$	m_1	m_3
$\mathrm{SID}(\Pi_2^{s_2})$	$\tilde{m}_1$	$\emptyset$	(m_2, m_3)
$\mathrm{SID}(\Pi_3^{s_3})$	m_3	(m_2, m_3)	$\emptyset$

Fig. 6.4. Example: $\mathrm{SID}(\Pi_i^{s_i})$ in the Attacked Protocol Execution

every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$. We cannot generally assume that

all oracles accept after this modification but we may assume that there exists protocols where this is the case (our example later is such a protocol where the oracles nevertheless accept). With this assumption the first part of our goal, i.e., the acceptance of every participating $\Pi_i^{s_i}$, is satisfied. In order to show that $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ holds for every $\Pi_i^{s_i}$ we need to show that $\Pi_i^{s_i} \leftrightsquigarrow \Pi_j^{s_j}$ (or $\Pi_i^{s_i} \leftrightarrow \Pi_j^{s_j}$) still holds for any two participating $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$. For this purpose we need to look more precisely on the session ids of the oracles. Note that $\mathrm{SID}_{12} = m_1$. Though $\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_2^{s_2}) = \{m_1, m_3\} \cap \{\tilde{m}_1, m_2|m_3\} = \emptyset$ and thus $\Pi_1^{s_1} \not\leftrightarrow \Pi_2^{s_2}$, there still exists a sequence of oracles $\Pi_1^{s_1}$, $\Pi_3^{s_3}$, $\Pi_2^{s_2}$ such that

$$\begin{aligned}\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_3^{s_3}) &= \\ &= \{m_1, m_3\} \cap \{m_3, m_2|m_3\} \\ &= m_3\end{aligned}$$

$$\begin{aligned}\mathrm{SID}(\Pi_3^{s_3}) \cap \mathrm{SID}(\Pi_2^{s_2}) &= \\ &= \{m_3, m_2|m_3\} \cap \{\tilde{m}_1, m_2|m_3\} \\ &= m_2|m_3\end{aligned}$$

so that $\Pi_1^{s_1} \leftrightsquigarrow \Pi_2^{s_2}$. Note also that these equations imply the direct partnering $\Pi_1^{s_1} \leftrightarrow \Pi_3^{s_3}$ and $\Pi_2^{s_2} \leftrightarrow \Pi_3^{s_3}$. Hence, we have shown that every $\Pi_i^{s_i}$, $i \in \{1, 2, 3\}$ has $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$. Thus all oracles are still partnered though the impersonation attack occurred whereby $\Pi_2^{s_2}$ received a different message than the one originally sent by $\Pi_1^{s_1}$. This may result in different group keys computed by $\Pi_1^{s_1}$ and $\Pi_2^{s_2}$.

In order to illustrate the described attack on a concrete example we consider the GKE protocol described in the same paper as the BCPQ model [46] but without the additional confirmation round which the authors described independently of the protocol. We stress that the additional confirmation round belongs to a concrete protocol design but not to a general security model; otherwise the model cannot be applied to the protocols that do not have this round. Recall, our goal is to show that despite of the acceptance of each participating oracle $\Pi_i^{s_i}$ with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ mutual authentication and key confirmation are not necessarily provided. The protocol proceeds as described in Figure 6.5 (for simplicity we consider three participants). $[m]_{U_i}$ denotes a digital signature on m computed by the corresponding $\Pi_i^{s_i}$ (the signature is attached to m), and $V(m) \stackrel{?}{=} 1$ its verification; g is a generator of a cyclic group of prime order p. Upon computing $K = g^{x_1 x_2 x_3}$ each oracle derives the resulting group key $k := \mathcal{H}(ID, FL_3, K)$ with a cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^l$ where l is the security parameter. In order to apply the above attack we consider that $\Pi_1^{s_1}$ chooses $x_1 \in \mathbb{Z}_p^*$ but $\mathcal{A}$ drops the original message $[Fl_1]_{U_1}$ and replays a corresponding message from some previous protocol execution (note $\mathcal{A}$ can invoke several subsequent protocol executions with the same ID via its $Send$ query). The replayed message is likely to be $[\widetilde{Fl_1}]_{U_1}$ with $\widetilde{Fl_1} := (ID, \widetilde{X_1})$ where $\widetilde{X_1} := \{g, g^{\widetilde{x_1}}\}$ for some $\widetilde{x_1} \neq x_1$ (since each x_i is chosen at random for every new session). Obviously, $\Pi_2^{s_2}$ can still verify the replayed message, i.e., $V(\widetilde{Fl_1}) = 1$ holds. It is easy to see that $X_2 = \{g^{\widetilde{x_1}}, g^{x_2}, g^{\widetilde{x_1} x_2}\}$ and $X_3 := \{g^{\widetilde{x_1} x_2}, g^{\widetilde{x_1} x_3}, g^{x_2 x_3}\}$ so that $\Pi_1^{s_1}$ computes $K = g^{x_1 x_2 x_3}$ whereas $\Pi_2^{s_2}$ and $\Pi_3^{s_3}$ compute another value, i.e., $K = g^{\widetilde{x_1} x_2 x_3}$. This also implies that the derived group keys are different. Note also that all oracles accept since all signature verifications remain correct. Beside that we have (similar to the abstract problem description above)

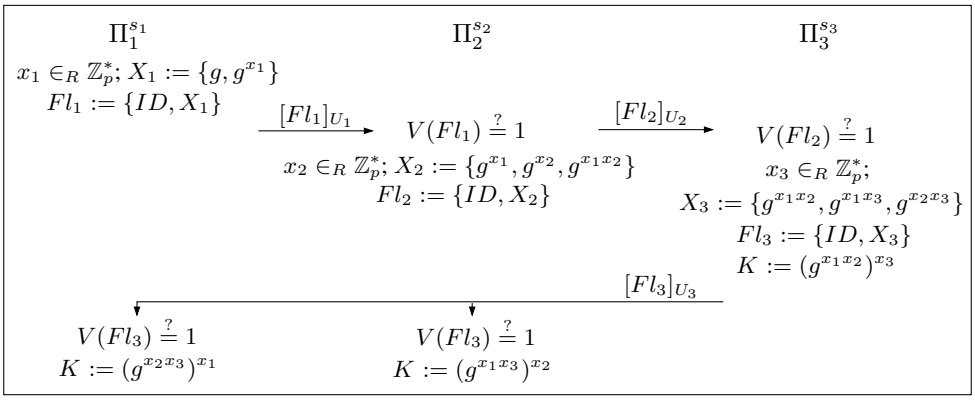

Fig. 6.5. Example: Execution of the Protocol in [46] with Three Participants

$$\begin{aligned}\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_3^{s_3}) &= \\ &= \{[Fl_1]_{U_1}, [Fl_3]_{U_3}\} \cap \{[Fl_3]_{U_3}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} \\ &= [Fl_3]_{U_3}\end{aligned}$$

$$\begin{aligned}\mathrm{SID}(\Pi_3^{s_3}) \cap \mathrm{SID}(\Pi_2^{s_2}) &= \\ &= \{[Fl_3]_{U_3}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} \cap \{\widetilde{[Fl_1]}_{U_1}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} \\ &= [Fl_2]_{U_2}|[Fl_3]_{U_3}\end{aligned}$$

so that $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ for every $\Pi_i^{s_i}$, $i \in \{1, 2, 3\}$. Thus, we could show that although all oracles accept with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ the protocol does not provide mutual authentication and key confirmation. This contradicts to the idea behind the definition of MA-security in the BCPQ model. Thus, it is not always true that if every $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n - 1$ then mutual authentication and key confirmation are provided. Note that this is true if the protocol from [46] is executed with the additional confirmation round, but there may exist other protocols (including our example) for which this statement is not true (i.e., if MA-security is achieved by some other techniques). This shows that the definition of MA-security given in the BCPQ model is not generally applicable just for any GKE protocol.

Furthermore, we stress that a more general definition of MA-security should also consider possible attacks of malicious protocol participants those goal is to influence honest participants to come up with different group keys. As described in [72, 73], not considering malicious participants is the reason why the BCPQ model and some of its later appeared variants do not capture unknown key-share attacks in their definitions. Note also that the construction of session ids based on concatenation of exchanged messages has one significant drawback - it becomes available only after the protocol is executed. However, some protocols use uniqueness of session ids as protection against replay and protocol interference attacks. In this case it is desirable to have a unique session id prior to the protocol execution.

6.2.7 Models by Bresson, Chevassut, and Pointcheval (BCP, BCP$^+$)

BCP

In their subsequent work, Bresson, Chevassut, and Pointcheval [41] extend the BCPQ model to deal with dynamic group key exchange protocols where group membership may change during the protocol execution. We denote this extended model BCP. According to it a dynamic GKE

protocol consists of an initialization algorithm executed for each participant, a setup protocol between all founding group members for the initialization of the group and computation of initial session key, a join protocol executed between current group members and a set of joining members, and a remove protocol executed between the remaining group members after the exclusion of a subset of members from the group. The protocols for setup, join, and remove are called *operations*. The BCP model also specifies three additional queries, $Setup$, $Join$, and $Remove$, enabling an adversary to invoke the corresponding operation between the protocol participants. The BCP model defines AKE-security and MA-security as in the BCPQ model, i.e., based on the adversary's guess on the response of its $Test$ query to a fresh oracle and based on the partnering condition, respectively.

The BCP model has the same drawbacks as the BCPQ model concerning the security of the protocol in case of the attacks that aim to reveal the internal state information of the oracles, attacks by malicious participants, and the identified problems with the definition of MA-security.

BCP$^+$

Bresson, Chevassut, and Pointcheval [42] revised their BCP model to cover attacks against the protocol based on the revealed internal state information of the oracles (strong corruptions). We denote this revised model as BCP$^+$. The model assumes that the security-relevant internal state information is maintained within a secure coprocessor, and that the long-lived keys of participants are stored within a smart card. Therefore, the model specifies additional queries which an adversary is allowed to ask, i.e., a $Send_c$ query which allows the adversary to communicate directly with the coprocessor, a $Corrupt_c$ query which reveals the private memory of the device together with all messages which have been exchanged between the coprocessor and the smart card, a $Send_s$ query which allows the adversary to communicate directly with the smart card, and a $Corrupt_s$ query which reveals the oracle's long-lived key.

Further, the BCP$^+$ model defines two flavors of forward secrecy: *weak forward secrecy* (wfs) and *strong forward secrecy* (fs). For the case of weak forward secrecy the BCP$^+$ model defines a *weak corruption model* in which the adversary is allowed to ask $Send$, $Setup$, $Join$, $Remove$, $Reveal$, and $Test$ queries (all of which are answered as in the BCP model) as well as $Send_c$, $Send_s$, and $Corrupt_s$ queries. According to the weak corruption model an oracle is called *wfs-fresh* if no $Corrupt_s$ query has been asked by the adversary since the beginning of its execution, and in the execution of the current operation the oracle has accepted (holds the session key) and neither this oracle nor any of its partners (although the model does not specify the definition of partnering it seems to be the same as in the BCP model) have been asked for a $Reveal$ query. The adversary must ask its $Test$ query to an oracle which is wfs-fresh.

Consequently, for the case of strong forward secrecy the BCP$^+$ model defines a *strong corruption model* in which the adversary may additionally ask $Corrupt_c$ queries to obtain the internal state of the coprocessor and all messages which have been exchanged between the coprocessor and the smart card, and $Reveal$ queries reveal not only the session key but also all messages which have been exchanged between the oracle and its secure coprocessor. An oracle is called *fs-fresh* if neither $Corrupt_c$ nor $Corrupt_s$ queries have been asked by the adversary since the beginning of its execution, and in the execution of the current operation the oracle has accepted and neither this oracle nor any of its partners have been asked for a $Reveal$ query. In the strong corruption model the adversary must ask its $Test$ query to an oracle which is fs-fresh.

The BCP$^+$ model defines AKE-security of a GKE protocol using the adversary's guess for the response to its $Test$ query as in the BCP model but can be of two types with respect to the

chosen corruption model. Hence, the BCP$^+$ model considers definitions of semantic security with respect to known-key attacks and active adversaries as well as (perfect) forward secrecy. The BCP$^+$ model does not explicitly specify MA-security, however, the authors mention that its definition can be taken from their BCP model. Although the model deals with the attacks concerning internal states of participants it still does not consider attacks of malicious participants, neither for MA-security nor for the issues of key control and contributiveness. It is also arguable whether the assumptions about the existence of smart cards and secure coprocessors for the protocol execution are of major importance and should be considered within an abstract model. The ability of the adversary to reveal the internal state information and the long-lived keys of participants can also be modeled by corresponding queries without these assumptions. This would also simplify the model.

6.2.8 Modifications of the BCPQ, BCP, and BCP$^+$ Models

In this section we describe some existing modifications of the models in Sections 6.2.6 and 6.2.7.

Modification by Bresson, Chevassut, and Pointcheval

In [43] the authors slightly modified the BCPQ model to be used with group key exchange protocols where authentication is achieved by the means of shared passwords. In addition to $Send$, $Reveal$, $Corrupt$, and $Test$ queries the adversary is allowed to ask an $Execute$ which models an honest protocol execution between the participants specified in the query, or in other words the protocol is executed in the presence of a passive adversary. This allows to provide tighter security proofs with respect to dictionary attacks because the number of $Send$ queries which an adversary is allowed to ask and that it actually uses to try own passwords is independent of the number of $Execute$ queries for which the adversary obtains the transcript of an honest execution where participants use the shared password.

Modification by Katz and Yung (KY)

Katz and Yung [112] revised the BCPQ model from the perspective of static group key exchange protocols in which all messages are sent over a broadcast channel, i.e., received by all participants. Similar as the modification in [43] the authors consider an additional $Execute$ query. However, the main difference to the BCPQ model is a different construction of session ids and partner ids. The session id of an oracle Π_i^s is simply the concatenation of all message flows that were sent or received by Π_i^s. Since each protocol message is received by every protocol participant it is clear that at the end of an honest execution all participants compute the same session id. The partner id of an oracle Π_i^s consists of the identities of participants with whom the oracle intends to establish the session key including U_i (note that in the BCPQ model U_i is not part of $\mathrm{PIDS}(\Pi_i^s)$). Note that according to this definition partner ids are known in advance whereas in the BCPQ model they become known at the end of the protocol execution. Therefore, the equality of partner ids alone is not sufficient to decide whether oracles have actually participated in the same protocol session or not. For this reason Katz and Yung define two oracles as being *partnered* if they have equal partner ids and equal session ids. Also, Katz and Yung slightly modified the definition of the freshness of an oracle, i.e., an oracle Π_i^s is *fresh* if the adversary did not ask a $Send$ query to Π_i^s or any of its partners after having corrupted U_i or any of its partners, and neither Π_i^s nor any of its partners have been asked for a $Reveal$ query. Hence,

this modification allows the adversary to corrupt a party but then the adversary is not allowed to participate in the protocol on behalf of the corrupted party. This does not give the adversary any advantage compared to the definition of freshness in the BCPQ model. For the definition of security of a protocol Katz and Yung consider a modular approach. They call a protocol: (a) a *secure group key exchange protocol* if a passive adversary which is not allowed to ask any $Send$ queries (note that this is the reason for the additional $Execute$ query) is successful in its guess concerning the $Test$ query, and (b) a *secure authenticated group key exchange protocol* if the same holds for an active adversary. However, their modifications do not explicitly consider mutual authentication and key confirmation and also do not deal with the attacks of malicious participants concerning the issues of key control and contributiveness. Note also that the proposed construction of session ids can be used only in the group key exchange protocols where each message is sent over a broadcast channel, i.e., received by each other party. Thus, the model is not abstract enough.

Modification by Kim, Lee, and Lee

Kim, Lee, and Lee [114] proposed a modification of the BCP model considering modifications done by Katz and Yung for the BCPQ model. In their model partner id of an oracle Π_i^s corresponds to the set of group members excluding the identity U_i, so that the partner ids are already known prior to the execution of the protocol. The proposed model specifies unique session ids for each oracle, however, they do not describe how these session ids are constructed. Obviously, they assume the same construction as in the modification by Katz and Yung. Two oracles are defined to be partnered if: (1) their session ids are equal, (2) each oracle's partner id consists of identities of all other group members' oracles, and (3) if the oracles compute equal session keys. Obviously, the latter requirement on the equality of computed session keys is somehow redundant, because if oracles compute equal session ids which are built by the concatenation of exchanged messages then they also compute equal session keys. Note also that this modification has similar limitations as the modification by Katz and Yung concerning protocols in which messages are not exchanged over a broadcast channel, and it also does not deal with attacks related to mutual authentication and key control.

Modification by Dutta, Barua, and Sarkar

Dutta, Barua, and Sarkar [87] proposed another variant of the BCPQ model. Similar modifications have been later applied by Dutta and Barua [83, 84, 86] to the BCP model for dynamic protocols and to the model in [43] for the password-based authenticated protocols. The authors use the same construction of partner ids as Katz and Yung, however, they proposed a different construction of session ids. Instead of using the concatenation of exchanged messages the authors set the session id of an oracle Π_i^s to be a set of pairs $\{(U_1, s_1), \ldots, (U_n, s_n)\}$ where each pair (U_j, s_j), $j \in \{1, \ldots, n\}$ corresponds to the instance oracle $\Pi_j^{s_j}$ of the protocol participant U_j, and say that two oracles are partnered if they have equal partner ids and equal session ids.

In order to keep session ids unique the authors require the uniqueness of oracles for each new session. In [83] the authors suggest to use a counter value as an additional parameter which should be increased for every new oracle of the user. Though this construction makes unique session ids available prior to the protocol execution it has the following weakness if used in the actual protocol implementation: the counter value must be saved after each execution of the protocol and, furthermore, it must be protected from manipulation; otherwise, an adversary may reset it to some previous value and cause impacts on the security of the protocol. A more

practical approach seems to be using random values (nonces) for each new initialization of the oracle. However, in this case one has to consider possible collisions between nonces used in different sessions. Note that the if session ids are not unique then an adversary may mount attacks based on interference of different sessions, e.g., replay attacks.

6.2.9 Models by Katz and Shin (KS, UC-KS)

Katz and Shin [111] proposed two different security models for GKE protocols: a computational model (referred to as the KS model), and a model in the framework of Universal Composability (UC) [57] (referred to as the UC-KS model). These models provide the first formal treatment of security of GKE protocols in the presence of malicious participants.

KS

The KS model is an extension of the BCPQ model and is also based on the modification of the latter by Katz and Yung [112]. However, Katz and Shin assume that unique session ids are already provided to the protocol by some high-level application protocol whereas the BCPQ model and the mentioned extension provide their own construction of the session ids. Partner ids of the oracles and the partnering relation are specified in the same way as proposed by Katz and Yung. The KS model allows the adversary to ask $Execute$, $Send$, $Reveal$, $Corrupt$ and $Test$ queries capturing the informal security definitions of semantic security with respect to known-key attacks and active adversaries as well as (perfect) forward secrecy. Similar to the BCP$^+$ model the KS model specifies two types of oracle freshness with respect to two different corruption models. In the *weak corruption model* an oracle Π_i^s is fresh if no $Reveal$ query has been asked to Π_i^s or to any of its partners, and no $Corrupt$ query has been asked to U_i or to any other participants in the session before the oracles have computed the session key and terminated. In the *strong corruption model* an oracle Π_i^s is fresh if no $Reveal$ query has been asked to Π_i^s or to any of its partners, and no $Corrupt$ query has been asked to a party whose identity belongs to the partner id of Π_i^s before Π_i^s has computed the session key and terminated. Thus, in the strong corruption model the adversary is allowed to corrupt partners of Π_i^s. Further, in the strong corruption model the $Corrupt$ query returns not only the long-lived key of a party but also the internal state of any active oracle which belongs to this party. Based on these two corruption models the KS model defines *security of authenticated group key exchange* protocols based on the adversary's guess with respect to its $Test$ query (similar to the AKE-security from the BCPQ model and its previously described variants).

Additionally, the KS model considers insider attacks executed by misbehaving, malicious participants. It defines a security goal called *agreement* such that an adversary *violates agreement* if there exist two oracles, Π_i^s and $\Pi_j^{s'}$, which are partnered and neither U_i nor U_j are corrupted but Π_i^s and $\Pi_j^{s'}$ have accepted with different session keys. Intuitively, this considers key confirmation in case that all other participants are malicious (corrupted).

Further, the KS model says that $\mathcal{A}$ *impersonates* U_j *to* (accepted) Π_i^s if U_j is uncorrupted and belongs to the partner id of Π_i^s but in fact there exists no oracle $\Pi_j^{s'}$ which is partnered with Π_i^s. In other words, the oracle Π_i^s computes the session key and U_i believes that U_j does so, but in fact an adversary has participated in the protocol on behalf of U_j. Note that there are no assumptions about the corruption of other protocol participants. This is a subject of the following two different definitions of the protocol security against impersonation attacks. A protocol is secure against *outsider impersonation attacks* if there exists a party U_j and an oracle Π_i^s such that for any adversary $\mathcal{A}$ the probability that $\mathcal{A}$ impersonates U_j to Π_i^s and no parties which belong to

the partner id of Π_i^s are corrupted before Π_i^s accepts is negligible. Thus, the adversary is not allowed to corrupt protocol participants during the execution of the protocol. Hence, an active adversary may try to inject messages on behalf of U_j or manipulate messages sent by valid protocol participants. Further, a protocol is secure against *insider impersonation attacks* if there exists a party U_j and an oracle Π_i^s such that for any adversary $\mathcal{A}$ the probability that $\mathcal{A}$ impersonates U_j to Π_i^s and neither U_j nor U_i are corrupted before Π_i^s accepts is negligible. Obviously, this (stronger) definition requires the existence of at least two uncorrupted protocol participants and allows the adversary to corrupt other participants. Intuitively, this requirement considers mutual authentication and unknown key-share resilience in the presence of malicious participants. Katz and Shin say that an authenticated group key exchange protocol is *secure against insider attacks* if it guarantees agreement and is secure against insider impersonation attacks. Note that the KS model does not describe any relationship between their formal definitions and other well-known informal definitions.

UC-KS

The UC-KS model has a different concept (which is common for all UC-based models) that describes what an ideal GKE protocol execution is. Security proofs carried out in this model are based on the simulatability/indistingushability approach (as mentioned in Section 3.3.2). This is different to the security proofs carried out in computational security models (like other mentioned variants of the BCPQ model) that use the reductionist approach (as described in Section 3.3.1). Note that in addition to both models Katz and Shin proposed a compiler to turn any GKE protocol which is secure in the BCPQ model into a protocol which is secure in their UC-based model, and provided simulatability/indistingushability-based security proofs for this case. However, Katz and Shin did not provide reductionist proofs to show that this compiler satisfies security against insider attacks defined in their computational model (this is important to be mentioned even though UC-security is considered to be stronger). This leaves open whether definitions of agreement and security against insider impersonation attacks in their computational model are practical enough for the construction of reductionist security proofs.[2] Another, more significant drawback of the KS model is that its definitions are still not strong enough since they do not consider the issues related to key control and contributiveness. As we show in Section 9.5 Katz and Shin's construction proven secure in the UC-KS model still allows a malicious participant to bias the resulting value of the session group key.

6.2.10 Model by Bohli, Vasco, and Steinwandt (BVS)

Bohli, Vasco, and Steinwandt [32] proposed in their unpublished work an extension (which we refer to as the BVS model) of the BCPQ model and its modification by Katz and Yung towards security goals in the presence of malicious participants. Their definitions of session ids, partner ids, the notion of partnering, the adversarial queries, oracle freshness, and security of authenticated group key exchange are identical to those in [112]. Therefore, the BVS model captures indistinguishability of group keys from random numbers with respect to known-key

[2] In fact in our model in Section 8.2.9 we provide an alternative definition (which we call MA-security to keep consistency with all previous models) that can be used to replace definitions of agreement and security against insider impersonation attacks of the KS model. One advantage is that for the same requirements in our model we need only one definition (and consequently one reductionist proof) whereas in the KS model two definitions (and consequently two reductionist proofs) are needed. Furthermore, we prove in Section 8.3 that our definition really unifies the informal notions of key confirmation, mutual authentication and unknown key-share resilience in the presence of malicious participants while the KS model does not show it explicitly.

attacks and active adversaries as well as (perfect) forward secrecy. Additionally, the BVS model defines a security goal called *session integrity* which is provided if all oracles of uncorrupted participants that have accepted with equal session ids hold identical session keys and partner ids which encompass the identities of all honest parties having accepted with the same session id. The second security goal defined by the BVS model is *strong entity authentication* to an oracle Π_i^s meaning that Π_i^s accepts and for all uncorrupted U_j which belong to the partner id of Π_i^s there exists an oracle $\Pi_j^{s'}$ which holds the same session id as Π_i^s and U_i belongs to the partner id of $\Pi_j^{s'}$. Intuitively, compared to the KS model the definition of session integrity is related to the definition of agreement, and the notion of *strong entity authentication* is similar to the security against impersonation attacks. Therefore, a combination of session integrity and strong entity authentication subsumes informal definitions of mutual authentication and key confirmation in the presence of malicious participants.

The BVS model also deals with the issues of key control and contributiveness. The adversary is given access to $Execute$, $Send$, $Reveal$, and up to $t-1$ $Corrupt$ queries and has to output a tuple (i, s, χ_κ, a) where i and s correspond to an unused oracle Π_i^s (oracle is unused if it has not been initialized earlier) of an uncorrupted participant U_i, χ_κ is a boolean-valued algorithm with $\kappa := \{k \in \mathcal{K} |\ \chi_\kappa(k) = true\}$ such that $\mathcal{K}$ is a key space and $|\kappa|$ is polynomial in the security parameter, and a is some state information. Then on input a the adversary tries to make Π_i^s accept a session key $k \in \kappa$. In this second stage the adversary is allowed to ask $Execute$, $Send$, $Reveal$, and $Corrupt$ queries, but is not allowed to corrupt U_i, and the total number of $Corrupt$ queries in both stages should remain $\leq t-1$. The BVS model defines a group key establishment protocol as being *t-contributory* if the adversary succeeds with only negligible probability. In case that a protocol is n-contributory where n is a number of protocol participants then the BSV model calls it a *key agreement*. It is clear that the above definition enforces each participant to provide own contribution to the computation of the session key. Unfortunately the authors do not show feasibility of their contributiveness definition since their proofs are heuristic.

The BVS model has some drawbacks discussed in the following. First, the adversary is required to commit to a certain oracle Π_i^s which remains uncorrupted and whose computation of the session key it tries to influence. Hence, the adversary is not adaptive in the sense that it can freely choose Π_i^s during the second stage of the attack. Second, the BCPQ model and consequently the BVS model disallows the adversary to reveal internal states of the oracles (strong corruptions). Therefore, the model considers neither (perfect) forward secrecy with respect to strong corruptions nor its definition of contributiveness does capture attacks aiming to influence the computation of the session key by Π_i^s using the knowledge of its internal state information but without corrupting U_i. Third, it is not clear how to specify the algorithm χ_κ, i.e., according to which criteria one can distinguish whether the key computed by Π_i^s is influenced by the adversary or is real in the sense of the protocol.

6.3 Summary and Discussion

In the following we summarize results of our analysis of currently known security models for group key exchange protocols. We focus on the BCPQ, BCP, BCP$^+$, KY, KS/UC-KS, and BVS models while leaving out security models specified only for two or three protocol participants. We also do not consider variations of the above models in [43, 83, 84, 86, 87, 114] since these are minor modifications, mostly of technical nature, and without significant consequences for the actual security definitions. Note that almost all considered models (BCPQ, BCP, BCP$^+$,

KY, KS, and BVS) have been designed for reductionist security proofs whereas UC-KS has been designed for simulatability/indistinguishability-based proofs. Still, the security requirements stated in the UC-KS model correspond to those stated in the KS model. Table 6.3 provides a comparison of these models. Columns two to five specify blocks of the most important

Table 6.1. Analysis of Security Models for Group Key Exchange Protocols

Model		IND	MA	FS	CON	Strong Corr.	S/D
BCPQ	[46]	+	H	+	-	-	S
BCP	[41]	+	H	+	-	-	D
BCP$^+$	[43]	+	-	+	-	+	D
KY	[112]	+	-	+	-	-	S
KS/UC-KS	[111]	+	M	+	-	+	S
BVS	[32]	+	M	+	+	-	S

informally defined security requirements from Section 6.1 whereby

- IND is the requirement on the indistinguishability of the group key computed in one session from a random value with respect to active adversaries and known group keys of other sessions,
- MA is the requirement on mutual authentication between all participants of the group key exchange protocol that also subsumes the requirement on key confirmation and unknown key-share resilience (M points out that definitions of the model consider malicious participants; H points out that definitions of the model consider only honest participants;),
- FS is the requirement on (perfect) forward secrecy, in particular that IND still holds if the adversary is able to reveal the long-lived keys of all participants in later sessions,
- CON is the requirement on contributiveness of the group key exchange protocol, in particular that each participant equally contributes to the resulting group key and guarantees its freshness (this also subsumes the notion of key control and unpredictability).

Additionally, Table 6.3 specifies whether the considered model provides definitions with respect to strong corruptions, i.e., the adversary should be allowed to reveal internal (private) information of protocol participants used in the protocol execution. This may have consequences on the definitions of considering FS and CON. The last column provides the type of the protocol dynamics considered by the model (S for static, D for dynamic).

INFORMAL REQUIREMENTS Obviously, all considered security models provide definitions that consider the requirements on indistinguishability of computed group keys (IND) and forward secrecy (FS). The requirement on mutual authentication and key confirmation has been found to be not general enough in the definitions of the BCPQ and BCP models. Furthermore, these definitions do not consider attacks of malicious participants. The BCP$^+$ and KY models do not contain any definitions concerning MA-security. Only the KS/UC-KS and BVS models provide sufficient definitions concerning mutual authentication, key confirmation and unknown key-share resilience (MA) concerning attacks of malicious participants. Note that BVS is the only

model that considers issues concerning key control and contributiveness (CON).

STRONG CORRUPTIONS Only the BCP$^+$ and KS/UC-KS models consider strong corruptions in their security definitions, however, only for the security requirement concerning forward secrecy (FS). The BVS model as an extension of the KY model does not deal with strong corruptions so that its definitions concerning key control and contributiveness are weak. We stress that the consideration of a more powerful adversary against key control and contributiveness which is given access to strong corruptions is important since this kind of the adversary is also considered for other requirements, e.g., FS.

GROUP DYNAMICS Only the BCP model and its stronger variant BCP$^+$ provide definitions concerning dynamic group key exchange protocols. All other models focus on static GKE protocols. Note that dynamic protocols provide additional operations for the efficient update of the group key upon occurring changes of the group formation. Due to the risk that efficiency is achieved at the expense of weaker security it is important to consider these operations in the stated security requirements.

MAIN RESULT Obviously, the models BCP$^+$, KS/UC-KS and BVS are the strongest currently available security models for GKE protocols. However, all of them have different drawbacks as shown throughout this chapter. In particular, none of the currently existing security models for group key exchange protocols provides sufficient security definitions that unify all important informal security requirements from the earlier literature and consider malicious participants, strong corruptions and dynamic operations at the same time. This fact emphasizes the need of an advanced security model that does not have the identified limitations.

Chapter 7

Security-Focused Survey on Group Key Exchange Protocols

In this chapter we give a survey of currently known key exchange protocols while focusing on the protocols that have been designed for more than three participants. For an overview of two- and three-party key exchange protocols we refer to [38, 85]. Most of the protocols described in this chapter can be seen as extensions of the well-known Diffie-Hellman key exchange protocol between two parties proposed by Diffie and Hellman in their foundational work [79].

We distinguish between protocols with heuristic security arguments in Section 7.2 and protocols in Section 7.3 that have been proven secure in one of the security models described in the previous chapter.

We stress that our survey focuses on the security aspects of the protocols. It does not aim to provide any efficiency comparison. The reader interested in this kind of surveys we refer to [8, 150].

7.1 Preliminaries

7.1.1 Two-Party Key Exchange Protocol by Diffie and Hellman

The protocol proposed by Diffie and Hellman in [79] is the earliest key exchange protocol that allows two participants, U_1 and U_2, compute a secret key k over a public communication

channel. Mathematical operations of the protocol are performed in a multiplicative group $\mathbb{G}$ where the DL problem is believed to be intractable (see Section 5.4.1 for further details). Let g be a generator of $\mathbb{G}$. Figure 7.1 describes the generalized version of the protocol.

- Each $U_i, i \in \{1,2\}$ chooses a random $x_i \in_R \mathbb{Z}_q$ and sends $z_i := g^{x_i}$ to U_{3-i}.
- Each $U_i, i \in \{1,2\}$ computes $k_i := (z_{3-i})^{x_i}$.

Fig. 7.1. Two-Party Key Exchange Protocol by Diffie and Hellman [79]

Obviously, the resulting shared key has the form $k = g^{x_1 x_2}$. The semantic security of k against passive adversaries relies on the DDH assumption. The original Diffie-Hellman protocol does not provide protection against impersonation attacks. A large number of variations has been proposed after the invention of the protocol to improve its security degree, the most recent are [121, 122]. Mostly all group key exchange protocols considered in our survey can be seen as more or less complex extensions related to this original Diffie-Hellman key exchange protocol.

7.1.2 Three-Party Key Exchange Protocol by Joux

Joux [109] proposed the following efficient key exchange protocol designed for three participants. The protocol uses a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ where $\mathbb{G}_1$ is an additive group of prime order q and $\mathbb{G}_2$ a multiplicative group of the same order, e.g., $\mathbb{G}_1$ is a subgroup of the group of points on an elliptic curve E over a finite field, $\mathbb{G}_2$ a subgroup of a multiplicative group over a related finite field, and $\hat{e}$ is an appropriate pairing on E (we refer to [28] for more details on pairings in elliptic curves). Also an element (point) $P \in \mathbb{G}_1$ with $\hat{e}(P,P) \neq 1_{\mathbb{G}_2}$ should be publicly known. The protocol between U_0, U_1, and U_2 proceeds as follows.

- Each U_i chooses $x_i \in_R \mathbb{Z}_q^*$ and broadcasts $y_i := x_i P$ to all other users.
- Each U_i computes $k_i := \hat{e}(y_{(i+1) \bmod 3}, y_{(i+2) \bmod 3})^{x_i}$.

Fig. 7.2. Three-Party Key Exchange Protocol by Joux [109]

Obviously, at the end of the protocol each user computes the group key $k = \hat{e}(P,P)^{x_0 x_1 x_2}$. The protocol requires only one communication round. Although not explicitly shown in [109], the semantic security of the protocol against passive adversaries is based on the Bilinear Diffie-Hellman (BDH) assumption [36]. Joux' protocol does not provide any form of authentication. Several attempts have been done to add authentication to the Joux' protocol, e.g., certification-based [7, 104, 166] and identity-based [68, 139, 140, 164, 191] protocol some of which could be broken in [67, 164, 165, 177].

7.1.3 A Comment on Relationships between the Protocols

Regardless of the separation into group key exchange protocols with heuristic security arguments and protocols with security proofs in the available security models some of the protocols included in our survey have certain similarities which we describe in the following.

The protocols in Sections 7.3.1, 7.3.2, 7.3.3, and 7.3.4 can be considered as modifications of the static group key exchange protocol proposed by Burmester and Desmedt [50] which we

describe in Section 7.2.1. These protocols are characterized by the constant number of communication rounds and are, therefore, scalable for large groups. Some of these protocols derive the group key from bases whose discrete logarithms are outputs of an additive cyclic function.

The protocols in Sections 7.2.3, 7.2.4, and 7.3.6 can be considered as modifications of the static group key exchange protocol proposed by Ingemarsson, Tang, and Wong [107] which we describe in Section 7.2.2. Most of these protocols derive the group key from bases whose discrete logarithms are outputs of a symmetric multiplicative function. In particular, from the value of the form $g^{x_1 \cdots x_n}$ where g is a generator of a cyclic group $\mathbb{G}$ where the DL problem is believed to be intractable, and every x_i, $i \in [1, n]$ is a private exponent of participant U_i. This form can be seen as a "natural" extension of the two-party Diffie-Hellman key exchange protocol described above.

The protocols in Sections 7.2.6, 7.2.7, derive the group key from a value obtained by an iterative application of the two-party Diffie-Hellman protocol. The earliest protocol of this class was proposed by by Steer, Strawczynski, Diffie, and Wiener [171] which we describe in Section 7.2.5. Most of the protocols of this class arrange participants into a logical binary tree structure which is either linear or balanced. In general, each user is logically assigned to a leaf node of a binary tree T. We use *labels* $\langle l, v \rangle$ to uniquely identify a node of a tree where $l \in \{0, d_T\}$ is a corresponding level of T, d_T the depth of T, and $v \in \mathbb{N}$ the nodes' position within the level. Note that in linear binary trees the depth d_T is linear in the number of participants whereas in the balanced binary trees d_T is logarithmic. Figure 7.3 shows an example of both tree types for $n = 4$. Every node of the tree contains a pair $(x_{\langle l,v \rangle}, y_{\langle l,v \rangle})$ where $x_{\langle l,v \rangle}$

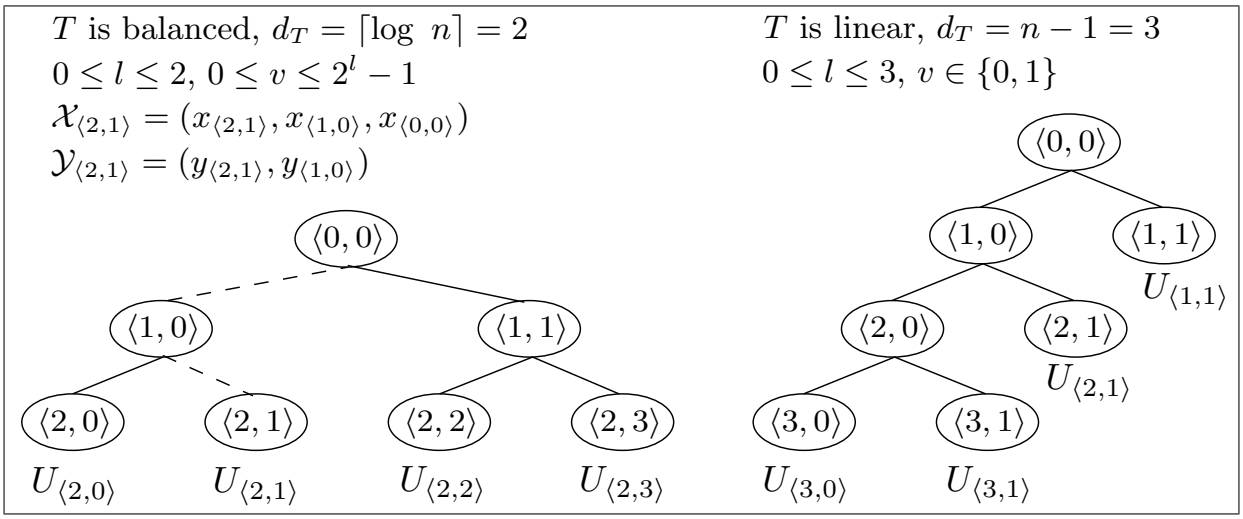

Fig. 7.3. Example: Balanced and Linear Trees for $n = 4$

is considered to be a secret key, and $y_{\langle l,v \rangle}$ a public value derived from $x_{\langle l,v \rangle}$. The root of the tree, denoted $\langle 0, 0 \rangle$ contains only the secret value $x_{\langle 0,0 \rangle}$, which is usually used in the protocols to derive the resulting group key k. By a *secret key path of a node* $\langle l, v \rangle$ we denote the list $\mathcal{X}_{\langle l,v \rangle} := (x_{\langle l,v \rangle}, x_{\langle l-1,\lfloor v/2 \rfloor \rangle}, \ldots, x_{\langle 1,\lfloor v/2^{l-1} \rfloor \rangle}, x_{\langle 0,0 \rangle})$, and by a *public key path of a node* $\langle l, v \rangle$ the corresponding list $\mathcal{Y}_{\langle l,v \rangle} := (y_{\langle l,v \rangle}, y_{\langle l-1,\lfloor v/2 \rfloor \rangle}, \ldots, y_{\langle 1,\lfloor v/2^{l-1} \rfloor \rangle})$. The protocols differ in a way of how this group key is computed. Furthermore, some of the described protocols update the logical tree structure and the the secret value $x_{\langle 0,0 \rangle}$ upon occurring dynamic changes of the group formation.

The protocols in Sections 7.2.8, 7.2.9, and 7.3.7 arrange participants into a ternary tree like the one in Figure 7.4. These protocols are extensions of the Joux' three-party key exchange protocol from Section 7.1.2.

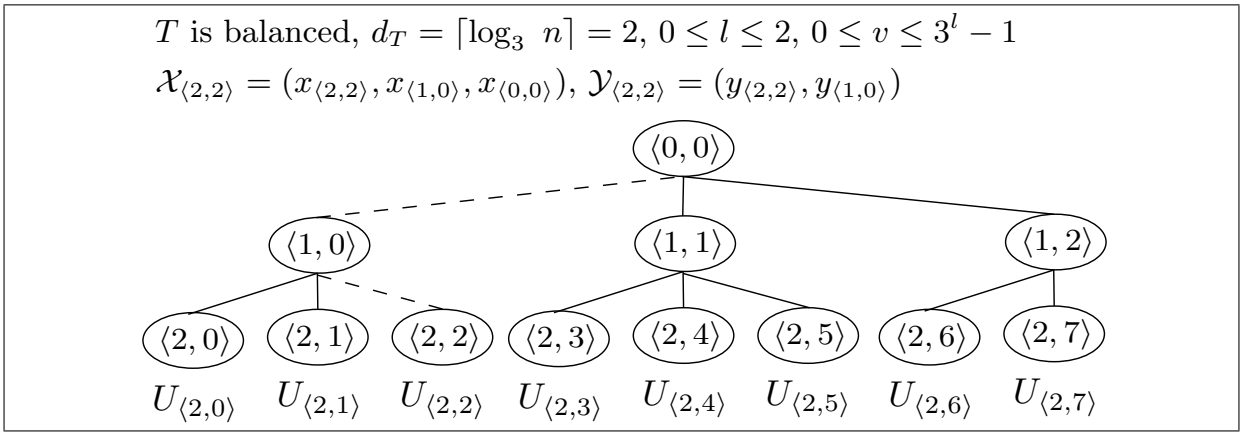

Fig. 7.4. Example: Balanced Ternary Tree for $n = 8$

7.2 Group Key Exchange Protocols with Heuristic Security Arguments

7.2.1 Protocol by Burmester and Desmedt

Burmester and Desmedt [50, 51] describe several protocols that allow a set of n users (group members) $U_1, \ldots, U_n$ to compute a secret group key k. The proposed protocols differ with respect to the underlying network topology. Although the majority of their protocols belongs to the class of group key distribution there are two protocols that can be considered as group key exchange protocols between users that are connected either over a broadcast network or other a bi-directional cyclic network. In the following we give a brief description of the static protocol designed for a broadcast network (denoted in [50] as Protocol 3).

All group members are logically ordered into a cycle, i.e., the indices are taken modulo n so that user U_0 is U_n and user U_{n+1} is U_1. All mathematical operations are performed in a cyclic group $\mathbb{G} \subseteq \mathbb{Z}_p$ with prime p generated by $g \in \mathbb{Z}_p$ of order q. It is assumed that the description of $\mathbb{G}$ is implicitly known to all users. The protocol proceeds as follows.

- Each U_i chooses a random $r_i \in_R \mathbb{Z}_q$ and broadcasts $z_i := g^{r_i} \pmod p$.
- Each U_i broadcasts $X_i := (z_{i+1}/z_{i-1})^{r_i} \pmod p$.
- Each U_i computes $k_i := (z_{i-1})^{nr_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i+n-2} \pmod p$ for $i = 1, \ldots, n$.

Fig. 7.5. Protocol by Burmester and Desmedt [50]

Note that after the protocol is completed every user holds the same group key $k = k_i = g^{r_1 r_2 + r_2 r_3 + \ldots + r_n r_1} \pmod p$. The protocol designed for bi-directional cyclic network (denoted in [50] as Protocol 4) proceeds similar except that corresponding messages are sent between any two directly connected participants. This increases the communication and computation overhead compared to the above protocol.

The heuristic security proof given in the pre-proceedings version considers only key secrecy requirement with respect to passive eavesdroppers under the CDH assumption. The insecurity against active adversaries follows from the absence of authentication. The authors also mention a variant of an authenticated protocol where each user U_i authenticates corresponding z_i to the subsequent user U_{i+1} using a zero-knowledge proof technique [66]. However this technique does not provide security against impersonation attacks due to the missing identification.

Variants by Choi *et al.* and Manulis

Choi *et al.* [71] proposed a variant of the unauthenticated Burmester-Desmedt protocol based on the technique of bilinear pairings [28, 36]. The heuristic security analysis considers only indistinguishability of group keys from random numbers with respect to passive adversaries under the so-called Decisional Bilinear Diffie-Hellman (DBDH) assumption [28, 35, 36, 92, 109, 136, 137] in the Random Oracle Model. Choi *et al.* have also constructed a protocol that provides identity-based authentication. Its security is argued in the Random Oracle Model under the so-called Decisional Hash Bilinear Diffie-Hellman (DHBDH) assumption [16] which is a non-standard cryptographic assumption strictly stronger than the DBDH assumption. Later, Zhang and Chen [190] showed an attack against the authentication property of the identity-based version of Choi *et al.*'s protocol where two malicious participants impersonate an honest participant using his authentication transcript from some previous protocol execution.

Manulis [127] described an elliptic curve equivalent of the original Burmester-Desmedt protocol in the context of mobile ad-hoc communication. The deployment of the elliptic curve cryptography results in a better trade-off between computation and communication costs due to the smaller sizes of the operands. The informally argued semantic security relies on the elliptic-curve version of the DDH assumption [28] if a non-supersingular and a non-trace-2 elliptic curve is used as required in [110].

7.2.2 Protocol by Ingemarsson, Tang, and Wong

In [107] Ingemarsson, Tang, and Wong proposed a family of group key exchange protocols from which we describe the mostly known one in Figure 7.6. It is assumed that all participants $U_1, \ldots, U_n$ implicitly know the description of the multiplicative group $\mathbb{G}$ of prime order q with the corresponding generator g. Participants are logically ordered into a cycle (similar to the Burmester-Desmedt protocol), i.e., the indices are taken modulo n so that member U_0 is U_n and member U_{n+1} is U_1. At the end of the protocol every U_i computes the group key $k :=$

- In round 0, each U_i chooses a random $x_i \in \mathbb{Z}_q^*$, computes g^{x_i} and forwards it to U_{i+1}.
- In round t, $t = 1, \ldots, n-2$ each U_i computes $g^{\prod\{x_j \mid j \in [i-t,i]\}}$ (using x_i as an exponent for $g^{\prod\{x_j \mid j \in [i-t,i-1]\}}$ received in the previous round) and forwards it to U_{i+1}.

Fig. 7.6. Protocol by Ingemarsson, Tang, and Wong [107]

$g^{x_1 \cdots x_n}$. The informal security proof considers semantic security against passive adversaries under the DDH assumption. The absence of the authentication implies the insecurity against active adversaries.

7.2.3 Protocols by Steiner, Tsudik, and Waidner

Steiner *et al.* [174] proposed a generic group key exchange protocol and three realizations, called GDH.1, GDH.2, and GDH.3, respectively. The generic construction considers a cyclic group $\mathbb{G}$ of prime order q generated by g. Through a distributed computation of the subsets of $\{g^{\prod X} | X \subset \{x_1, \ldots, x_n\}\}$ every member U_i computes $g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n}$. This allows every U_i to derive the resulting secret group key k_i with an additional exponentiation of the mentioned value with the private exponent x_i. The first proposed realization, i.e., the protocol GDH.1 consists of two stages: upflow and downflow, described in Figure 7.7.

- *Upflow:* In round i, $i = 1, \ldots, n-1$ the user U_i chooses a random $x_i \in \mathbb{Z}_q^*$ and forwards $\{g^{\prod \{x_t | t \in [1,j]\}} | j = 1, \ldots, i\}$ to U_{i+1}.
- *Downflow:* In round $n-1+i$, $i = 1, \ldots, n-1$ the user U_{n-i+1} forwards $\{g^{\prod \{x_t | t \in [1,n] \wedge t \notin [j,n-i]\}} | j = 1, \ldots, n-i\}$ to U_{n-i}. Upon receiving $g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n}$ each U_i computes the group key $k_i := (g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n})^{x_i}$.

Fig. 7.7. Protocol GDH.1 [174]

The second protocol GDH.2 consists of the upflow stage and an additional broadcast round, and proceeds as described in Figure 7.8.

- *Upflow:* In round i, $i = 1, \ldots, n-1$ the user U_i chooses a random $x_i \in \mathbb{Z}_q^*$ and forwards $\{g^{\prod \{x_t | t \in [1,i] \wedge t \neq j\}} | j = 1, \ldots, i\}$ and $g^{x_1 \ldots x_i}$ to U_{i+1}.
- *Broadcast:* In round n the user U_n chooses a random $x_n \in \mathbb{Z}_q^*$ and broadcasts a set $\{g^{\prod \{x_t | t \in [1,n] \wedge t \neq i\}} | i = 1, \ldots, n-1\}$. Upon receiving $g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n}$ each U_i computes the group key $k_i := (g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n})^{x_i}$.

Fig. 7.8. Protocol GDH.2 [174]

The third protocol GDH.3 consists of the upflow stage, two broadcast rounds and one response round and proceeds as described in Figure 7.9.

- *Upflow:* In round i, $i = 1, \ldots, n-2$ the user U_i chooses a random $x_i \in \mathbb{Z}_q^*$ and forwards $g^{\prod (x_t | t \in [1,i])}$ to U_{i+1}.
- *Broadcast:* In round $n-1$ the user U_{n-1} chooses a random $x_{n-1} \in \mathbb{Z}_q^*$ and broadcasts $g^{\prod (x_t | t \in [1,n-1])}$ to all other users.
- *Response:* In round n the user U_i factors out x_i and sends $g^{\prod (x_t | t \in [1,n-1] \wedge t \neq i)}$ to U_n.
- *Broadcast:* In round $n+1$ the user U_n chooses a random $x_n \in \mathbb{Z}_q^*$ and broadcasts a set $\{g^{\prod (x_t | t \in [1,n] \wedge t \neq i)} | i = 1, \ldots, n-1\}$ to all other users. Upon receiving $g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n}$ each U_i computes the secret group key $k_i := (g^{x_1 \ldots x_{i-1} x_{i+1} \ldots x_n})^{x_i}$.

Fig. 7.9. Protocol GDH.3 [174]

The heuristic security analysis shows that the generic construction is semantically secure against passive adversaries under the DDH assumption. The insecurity against active adversaries comes from the absence of authentication.

For the protocols GDH.2 and GDH.3 Steiner *et al.* proposed two additional extensions that handle join and leave events. In order to proceed with these events in GDH.2 user U_n has to save the contents of the received message during the upflow stage whereas in GDH.3 U_n saves the contents of the first broadcast and the response messages. The authors argue that their dynamic extensions preserve semantic security against passive adversaries.

In their subsequent work, Steiner *et al.* [175] presented a dynamic group key agreement protocol suite called CLIQUES. It consists of several protocols, that allow the initial key agreement between the founding group members, and auxiliary handling of possible dynamic events (join, leave, group fusion, and subgroup exclusion). In order to proceed with auxiliary protocols for dynamic events each user has to maintain an internal state information. The initial key agreement (IKA) protocol is given by the GDH.2 protocol from [174]. For the addition of a group member [175] suggests two different protocols that differ in the choice of a controller, i.e., the member who sends the broadcast message enabling other members to update the group key. Further, CLIQUES offers two efficient protocols for the simultaneous addition of multiple members (mass addition), and suggests several forms to process the group fusion event. It can be handled as a special case of the mass join or by the construction of a new super-group via the IKA protocol. Another proposed approach to merge two different groups G_1 and G_2 each having corresponding group keys k_1 and k_2 is to exchange the values g^{k_1} and g^{k_2} and compute the new group key as $k := g^{k_1 k_2}$. Obviously, this approach does not guarantee semantic security with respect to known key attacks. In case of mass exclusion the set broadcasted by the controller in the last protocol stage does not contain values that would allow excluded members to compute the updated key. In case where a group has to be partitioned into several independent smaller groups each smaller group performs the mass exclusion protocol for all other members.

Security of the CLIQUES protocols has been analyzed based on a snapshot of a current group formation. The protocols do not implicitly provide authentication, and the authors assume that authentic communication channels are used. Therefore, the notion of perfect forward secrecy is not treated, and the adversary is considered to be passive and is represented by a set of all future and former group members with respect to a given snapshot. Thus, the adversary is in possession of all private exponents of these members. Steiner *et al.* address only the issue of key independence and show that the probability of the adversary to distinguish a current group key from a random number is negligible under the DDH assumption.

Later, Steiner *et al.* [176] extended the CLIQUES suite by another initial key agreement IKA.2 that corresponds to the GDH.3 protocol from [174] and a protocol that allows to refresh the group key where one group member generates a fresh private exponent and repeats the last broadcast round of the original IKA.1 or IKA.2 protocol using the updated values. The authors subsequently repeat their heuristic proof from [175] to show that the extended CLIQUES suite is semantically secure against passive adversaries under the DDH assumption. The authors also claim that the protocol is contributory. This holds only if the adversary is not allowed to reveal private exponents of honest participants, that is only in the weak corruption model.

A Variant by Manulis

Manulis [127] describes an elliptic curve variant of the initial key agreement protocol of CLIQUES and its dynamic extensions achieving a better trade-off between computation and communication costs, and analyzes the deployment of the protocol suite in mobile ad-hoc group communication scenarios. The key secrecy of this modification has been argued intuitively based on the elliptic-curve version of the CDH assumption [28].

7.2.4 Protocols by Ateniese, Steiner, and Tsudik

Ateniese *et al.* [13] proposed two authenticated group key agreement protocols, A-GDH.2 and SA-GDH.2, based on the modifications of the GDH.2 protocol from [174]. In the proposed

protocols every user U_i holds a corresponding long-term key pair (sk_i, g^{sk_i}). The protocol A-GDH.2 proceeds during its first stage similar to GDH.2 but in the last stage U_n broadcasts a set $\{g^{K_{in}\prod\{x_t|t\in[1,n]\wedge t\neq i\}}|i = 1,\ldots,n-1\}$ where $K_{in} := F(g^{sk_i sk_n})$ with $F()$ either a reduction modulo q or a cryptographic hash function with domain $\{0,1\}^*$ and image $\mathbb{Z}_q^*$ where q is the order of $\mathbb{G}$. At the end of the protocol every user U_i computes the secret group key $k := (g^{K_{in}\prod\{x_t|t\in[1,n]\wedge t\neq i\}})^{K_{in}^{-1}x_i}$. In this form the authentication is performed indirectly via the controller U_n. Ateniese *et al.* provide a heuristic security analysis of A-GDH.2. They argue that the protocol is resistant against known-key attacks, and provides implicit authentication and perfect forward secrecy in the presence of passive adversaries. As for the active adversaries the authors point out that some attacks against the semantic security of the protocol are possible due to the missing key confirmation property, i.e., it is possible for the active adversary to share a group key with a subset of group members. Also, the implicit authentication in A-GDH.2 is given in a weak form, since there is no direct authentication between the members, but the controller which is assumed to be trusted authenticates himself to all other members. The authors point out that the protocol is susceptible to the attacks by dishonest participants wishing to alter the group formation during the protocol execution by excluding (or skipping) some of its participants. Thus, the protocol does not provide key confirmation in case that some of its participants are dishonest.

In order to prevent some of the described attacks Ateniese *et al.* proposed a modified protocol version, called SA-GDH.2, with the intention to achieve the informally defined notion of a complete group key authentication, i.e., any two members compute the same group key only if every member has contributed to its computation. The protocol proceeds as described in Figure 7.10.

- *Upflow:* In round i, $i = 1,\ldots,n-1$ the user U_i receives a set of n intermediate values $\{V_t|t = 1,\ldots,n\}$ with

$$V_t = \begin{cases} g^{\frac{x_1\cdots x_{i-1}}{x_t}\cdot K_{t1}\cdots K_{t(i-1)}} & \text{if } t \leq i-1 \\ g^{x_1\cdots x_{i-1}\cdot K_{t1}\cdots K_{t(i-1)}} & \text{if } t > i-1, \end{cases}$$

  updates each value as follows

$$V_t' = \begin{cases} V_t^{K_{it}x_i} = g^{\frac{x_1\cdots x_i}{x_t}\cdot K_{t1}\cdots K_{ti)}} & \text{if } t < i \\ V_t^{K_{it}x_i} = g^{x_1\cdots x_i\cdot K_{t1}\cdots K_{t(i)}} & \text{if } t > i \\ V_t & \text{if } t = i, \end{cases}$$

  and forwards the updated set $\{V_t'|t = 1,\ldots,n\}$ to U_{i+1}. Note that U_1 starts his computation with an empty set and defines $V_1' := g$.
- *Broadcast:* In round n the user U_n chooses a random $x_n \in \mathbb{Z}_q^*$, updates received $\{V_t|t = 1,\ldots,n\}$ as described above, and broadcasts $\{V_t'|t = 1,\ldots,n\}$ to all other users. Upon receiving the message each U_i selects the appropriate V_i' and computes the group key as $k := (V_i')^{x_i\cdot K_{1i}^{-1}\cdots K_{ni}^{-1}} = g^{x_1\cdots x_n}$.

Fig. 7.10. Protocol SA-GDH.2 [13]

Ateniese *et al.* informally argue that SA-GDH.2 provides complete group key authentication and is resistant against known-key attacks in the presence of active adversaries. Further, the authors claim that both protocols can be easily extended to provide key confirmation by including a value $g^{F(k_n)}$ where k_n is the group key computed by U_n into the last message broadcasted by U_n such that each U_i who receives this message is able to verify whether $g^{k_i} \stackrel{?}{=} g^{k_n}$ holds.

In their subsequent work Ateniese *et al.* [14] used the ideas from [13] to add authentication to the protocols for the initial key agreement and handling of dynamic events of the CLIQUES suite from [175]. However, later Pereira and Quisquater [144, 146] discovered some attacks

against implicit key authentication, perfect forward secrecy, and resistance against known-key attacks of A-GDH.2 and its dynamic extensions, as well as attacks against complete group key authentication of SA-GDH.2 protocol in the presence of an active adversary. Note that these attacks do not concern the security of the original CLIQUES protocols in [175] that remain semantically secure against passive adversaries (in case of weak corruptions).

7.2.5 Protocol by Steer, Strawczynski, Diffie, and Wiener

The protocol proposed by Steer *et al.* [171] to secure audio teleconference systems is the earliest protocol that computes the group key using the structure of a linear tree. Although, the authors do not mention the tree structure explicitly, the mathematical structure of the computed group key is similar to the one obtained from a linear tree. All operations are performed in a cyclic group $\mathbb{G}$ of prime order p generated by g. It is assumed that all users have public-key certificates generated by a trusted party that can be used to sign messages. The protocol proceeds as described in Figure 7.11. Note that X_n has the algebraic form $g^{x_n g^{x_{n-1} g^{\cdots g^{x_3 g^{x_1 x_2}}}}}$. The authors also

- In round 1 each U_i chooses a random $x_i \in_R \mathbb{Z}_p$, and broadcasts $y_i := g^{x_i}$. Upon receiving these values all users get indices according to the ordered list of their identities, i.e., $U_1, \ldots, U_n$, and U_1 computes $X_{i+1} := y_{i+1}^{X_i}$ for all $i = 1, \ldots, n-1$ starting with $X_1 = x_1$.
- In round i, $i = 2, \ldots, n-1$ user U_i, $i = 2, \ldots, n-1$ receives Y_{i-1} (U_2 uses $Y_1 := y_1$), computes $X_i := Y_{i-1}^{x_i}$, $Y_i := g^{X_i}$, broadcasts Y_i to all other users, and computes $X_{j+1} := y_{j+1}^{X_j}$ for all $j = i, \ldots, n-1$.
- In round n all users learn X_n and use it to derive the group key k. Each U_i broadcasts own certificate. Upon receiving all certificates U_i verifies each of them.
- In round $n+1$ each U_i signs a hash value of $(y_1, \ldots, y_n)$ and broadcasts it to other users. Upon receiving all messages each U_i verifies the signature and the hash value.

Fig. 7.11. Protocol by Steer, Strawczynski, Diffie, and Wiener [171]

describe an efficient addition mechanism for new members by considering a new member as U_{n+1} and appending his input y_{n+1} to the accumulated chain calculation as in the second stage of the protocol. However, this mechanism is not semantically secure against known-key attacks. The authors claim that the protocol provides key secrecy and security against impersonation attacks. However, they do not give any security analysis with respect to either a passive or an active adversary.

7.2.6 Protocol by Becker and Wille

Becker and Wille [17] proposed two static group key exchange protocols, called Octopus and Hypercube. Although, the protocols do not assign users to the leaf nodes of a tree, the algebraic structure of the computed group key is similar to the one that can be obtained from a balanced binary tree. The main building block of the Octopus protocol is a four-party key agreement described in Figure 7.12. In the first round U_0 and U_1 in parallel with U_2 and U_3 compute $x_{0,1} := g^{x_0 x_1}$ respectively $x_{2,3} := g^{x_2 x_3}$, respectively. In the second round U_0 and U_2 in parallel with U_1 and U_3 compute the resulting shared key $y_{0,1,2,3} := g^{x_{0,1} x_{2,3}} = g^{g^{x_0 x_1} g^{x_2 x_3}}$. In the Octopus protocol all users are ordered into four subgroups of almost equal sizes in which one user, denoted U_i with $i \in \{0, \ldots, 3\}$, takes the role of a controller (similar to the role of the sponsor in [117]). The protocol consists of the three stages described in Figure 7.13. For the case where $n = 2^d$, $d \in \mathbb{N}$ Becker and Wille proposed a Hypercube protocol where all users are arranged into a d-dimensional hypercube, i.e., a graph in the form of a cube with 2^d vertices, each of

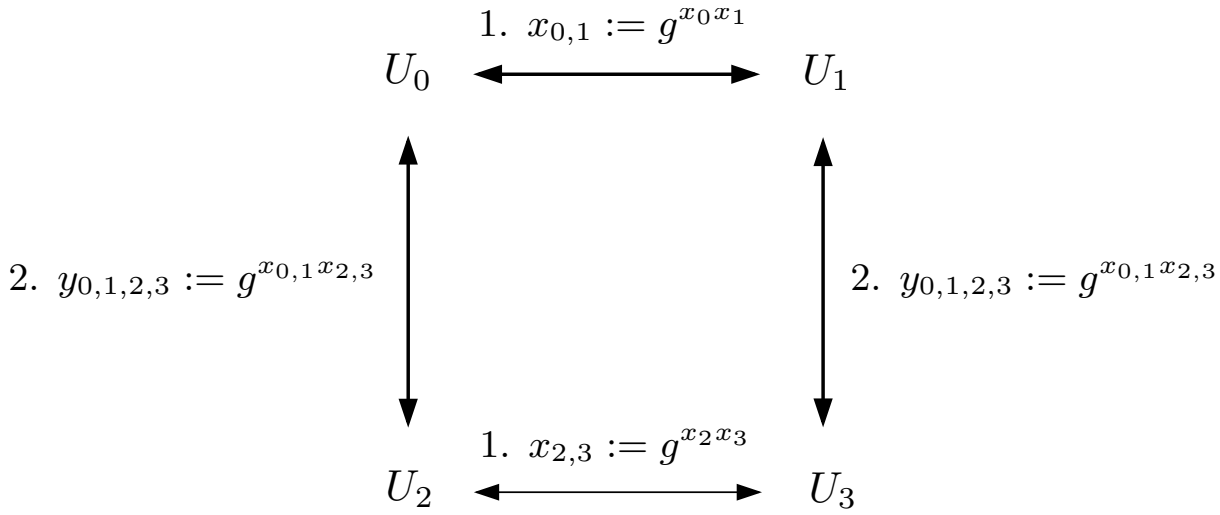

Fig. 7.12. Example: Main Building Block of Protocols Octopus and Hypercube[17]

- *Stage 1:* Each subgroup controller $U_i, i \in \{0, \ldots, 3\}$, computes an individual Diffie-Hellman key with each of the subgroup members. By $x_{i,j}$ we denote the secret key shared between U_i and the j-th member of the i-th subgroup.
- *Stage 2:* Each controller U_i computes $\hat{x}_i := \prod_j x_{i,j}$ and uses it in the protocol from Figure 7.12 to compute the group key $k := g^{\hat{x}_1 \hat{x}_2 g^{\hat{x}_3 \hat{x}_4}}$.
- *Stage 3:* Each controller U_i computes $(g^{\hat{x}_{(i+2) \bmod 4}})^{\hat{x}_i / x_{i,j}}$ (using the received value $g^{\hat{x}_{(i+2) \bmod 4}}$ from Stage 2) and sends this together with another received value $g^{g^{\hat{x}_{(i+1) \bmod 4} \hat{x}_{(i+3) \bmod 4}}}$ to the j-th member of the i-th subgroup who uses $x_{i,j}$ to compute $g^{\hat{x}_{(i+2) \bmod 4} \hat{x}_i}$ and k.

Fig. 7.13. Protocols Octopus and Hypercube [17]

them connected to d other vertices. It is assumed that for each user U_i there is a label i of d bits with $i \in \mathbb{Z}_n$. The protocol proceeds in d communication rounds such that in the j-th round each user U_i performs a Diffie-Hellman key exchange with the user $U_{i \oplus 2^{j-1}}$ using the key computed in the previous round $j-1$, i.e., in the j-th round users along the j-th dimension of the hypercube compute the shared Diffie-Hellman key. After a total number of d rounds and dn unicast messages all users agree on a group key k. The protocol remains efficient if n equals to a power of two. For the opposite case [17] suggests a mixed solution, called 2^d-Octopus, which is based on a combination of the Hypercube and Octopus protocols. Becker and Wille prove the semantic security of their protocols against passive adversaries under the DDH assumption using a heuristic method similar to [174]. Both protocols as described in [17] do not provide any form of authentication and are, therefore, insecure against impersonation attacks.

A Variant by Asokan and Ginzboorg

Asokan and Ginzboorg [12] adopted the password-based authentication to the protocol of Becker and Wille for the scenarios of small ad-hoc groups. They also described a solution for a user to find a partner for the Diffie-Hellman key exchange if the round partner of this user is faulty (this can be seen as a step towards denial of service attacks). Although, the authors described the desirable security properties for their protocol, i.e., group key secrecy, perfect forward secrecy, contributiveness, as well as the tolerance against disruption attempts in the mobile ad-hoc setting, they do not give any security analysis of these issues.

7.2.7 Protocols by Kim, Perrig, and Tsudik

Perrig [147] designed a static group key exchange protocol that outputs group keys with the same algebraic structure as in the Hypercube protocol using the logical structure of a balanced binary tree T from Figure 7.3. In the following description by $T_{\langle l,v\rangle}$ we denote a subtree of T rooted at node $\langle l, v\rangle$, and for any non-leaf node $x_{\langle l,v\rangle} := g^{x_{\langle l+1,2v\rangle} x_{\langle l+1,2v+1\rangle}}$ holds. The protocol proceeds as described in Figure 7.14.

- In round 1 each $U_{\langle l,v\rangle}$, $0 \leq v \leq 2^l - 1$ randomly chooses $x_{\langle l,v\rangle} \in_R \mathbb{Z}_p$ and broadcasts $y_{\langle l,v\rangle} := g^{x_{\langle l,v\rangle}}$ to every other user.
- In round i, $i = 2, \ldots, d_T + 1$ each $U_{\langle l,v\rangle}$, $l > d_T + 1 - i$, computes $x_{\langle d_T+1-i,\lfloor v/2^{i-1}\rfloor\rangle}$. If $i \neq d_T + 1$ then for each subtree $T_{\langle d_T+1-i,v\rangle}$, $0 \leq v \leq 2^{d_T+1-i} - 1$, a user assigned to one of the leaf nodes of $T_{\langle d_T+1-i,v\rangle}$ broadcasts $y_{\langle d_T+1-i,\lfloor v/2^{i-1}\rfloor\rangle} := g^{x_{\langle d_T+1-i,\lfloor v/2^{i-1}\rfloor\rangle}}$ to every other user.

Fig. 7.14. Protocol by Perrig [147]

Obviously, at the end of the protocol each user computes $x_{\langle 0,0\rangle}$. This value is then used to derive the group key as $k := H(x_{\langle 0,0\rangle})$.

Kim *et al.* [116, 119] extended Perrig's protocol to the TGDH (for Tree-based Group Diffie-Hellman) protocol suite that handles various dynamic group changes. For this purpose each user $U_{\langle l,v\rangle}$ has to store the structure of T including the public keys of all nodes and the own secret key path $\mathcal{X}_{\langle l,v\rangle}$. TGDH changes the initial tree structure and updates the group key with respect to the changes in the group formation. The authors also propose a policy which keeps the updated tree mostly balanced. Informal security analysis of the protocol in [119] provides arguments for the issues of key independence and group key secrecy. It focuses on the semantic security against passive adversaries under the DDH assumption. TGDH does not provide implicit authentication or key confirmation. Also, perfect forward secrecy is not considered because of the absence of any long-term keys.

Kim, Perrig, and Tsudik [117, 118] proposed a dynamic extension of the Steer *et al.*'s protocol, which they have called the STR protocol (Figure 7.15). They applied a linear tree structure for the computation of the group key, and have extremely increased the communication efficiency. We use the label-based notation from Figure 7.3 to describe their protocol. It is assumed that each user $U_{\langle l,v\rangle}$ is assigned to a leaf node of a linear binary tree T, i.e., v is either 0 or 1. All computations are performed in a special multiplicative cyclic group $\mathbb{G} = \langle g\rangle$ of prime order q where the group operation can be used to derive a bijection from $\mathbb{Z}_q$ to $\mathbb{Z}_q$.

- In round 1 each $U_{\langle l,v\rangle}$, $1 \leq l \leq n-1$, $v \in \{0,1\}$, randomly chooses $x_{\langle l,v\rangle} \in_R \mathbb{Z}_q^*$, and broadcasts $y_{\langle l,v\rangle} := g^{x_{\langle l,v\rangle}}$ to every other user.
- In round 2 users $U_{\langle n-1,0\rangle}$ and $U_{\langle n-1,1\rangle}$ compute $\mathcal{X}_{\langle n-2,0\rangle} := \{x_{\langle l-1,0\rangle} := y_{\langle l,1\rangle}^{x_{\langle l,0\rangle}} \,|\forall\, n-1 \geq l \geq 1\}$ ($U_{\langle n-1,1\rangle}$ starts the computation with $y_{\langle n-1,0\rangle}$). Then, $U_{\langle n-1,0\rangle}$ computes and broadcasts $\mathcal{Y}_{\langle n-2,0\rangle} := \{y_{\langle l,0\rangle} := g^{x_{\langle l,0\rangle}} \,|\, \forall\, x_{\langle l,0\rangle} \in \mathcal{X}\}$. Then, each $U_{\langle l,1\rangle}$, $1 \leq l \leq n-2$ computes $\mathcal{X}_{\langle l-1,0\rangle} := \{x_{\langle l-1,0\rangle} := y_{\langle l,0\rangle}^{x_{\langle l,1\rangle}}\} \cup \{x_{\langle j-1,0\rangle} := y_{\langle j,1\rangle}^{x_{\langle j,0\rangle}} \,|\forall\, l-1 \geq j \geq 1\}$.

Fig. 7.15. Protocol STR [118]

Note that each $U_{\langle l,v\rangle}$ learns $x_{\langle 0,0\rangle} = g^{x_{\langle 1,1\rangle} g^{x_{\langle 2,1\rangle} g^{\cdots g^{x_{\langle n-2,1\rangle} g^{x_{\langle n-1,1\rangle} x_{\langle n-1,0\rangle}}}}}}$ after the execution of the protocol. Kim *et al.* suggested to derive the group key using a cryptographic hash function, i.e., $k := H(x_{\langle 0,0\rangle})$. The authors also propose efficient operations to deal with dynamic group

changes. In order to handle these events the protocol requires from each user to save the whole structure of T including all public keys $y_{\langle l,v\rangle}$, $1 \leq l \leq n-1$, $v \in \{0,1\}$, and the secret key path of the user's leaf node. STR updates the tree and the group key with respect to the changes of the group formation. In [118], Kim *et al.* intuitively argue that the protocol provides key independence with respect to the known-key attacks under the DDH assumption. The protocol does not provide implicit authentication, however, the authors assume that all communication channels are authentic. The security analysis does not consider forward secrecy in its actual sense as described in Section 6.1.4 due to the absence of long-term keys.

Variants by Liao, Manulis, and Schwenk

For completeness we mention that Schwenk *et al.* [161] proposed a protocol suite for multimedia communications which is quite similar to that of TGDH but developed independently and patented [160].

Manulis [127] briefly described elliptic curve variants of the TGDH and STR protocol suites in the context of mobile ad-hoc communication which allows to achieve a better trade-off between computation and communication costs of the protocols. The heuristic analysis of the proposed protocols shows that their semantic security against passive adversaries relies on the elliptic-curve version of the DDH assumption [28].

Recently, Liao and Manulis [124, 125] proposed a tree-based framework for group key agreement in ad-hoc networks, called TFAN. The framework consists of a protocol which can be seen as a combination of the optimized elliptic curve variants of TGDH and STR protocols from [117, 118]. The heuristic security proof considers semantic security of the protocol based on the elliptic-curve version of the DDH assumption.

7.2.8 Protocol by Lee, Kim, Kim, and Ryu

In [123], Lee *et al.* extend the TGDH protocol suite [116, 119] using Joux' protocol. All members are assigned to the leaf nodes of a ternary tree T. Obviously, every node of T may be a leaf node, a parent node of two, or a parent node of three child nodes. In case that a non-leaf node $\langle l,v\rangle$ is a parent of three child nodes $\langle l+1,3v+i\rangle$, $i = 0,1,2$, its secret key is computed as $x_{\langle l,v\rangle} := H_1(\hat{e}(P,P)^{x_{\langle l+1,3v\rangle}x_{\langle l+1,3v+1\rangle}x_{\langle l+1,3v+2\rangle}})$ using the computations of the Joux' protocol with a cryptographic hash function $H_1 : \mathbb{G}_2 \rightarrow \mathbb{Z}_q^*$. In case that $\langle l,v\rangle$ is a parent of two child nodes $\langle l+1,3v+i\rangle$, $i = 0,1$ its secret key is computed as $x_{\langle l,v\rangle} := H_2(x_{\langle l+1,3v\rangle}y_{\langle l+1,3v+1\rangle}) = H_2(x_{\langle l+1,3v+1\rangle}y_{\langle l+1,3v\rangle}) = H_2(x_{\langle l+1,3v\rangle}x_{\langle l+1,3v+1\rangle}P)$ using the computations of an elliptic curve equivalent of the two-party Diffie-Hellman protocol with a cryptographic hash function $H_2 : \mathbb{G}_1 \rightarrow \mathbb{Z}_q^*$. The protocol proceeds as described in Figure 7.16. At the end of the protocol each user computes the group key $k := x_{\langle 0,0\rangle}$. Dynamic operations

- In round 1 each $U_{\langle l,v\rangle}$, $0 \leq v \leq 3^l - 1$ randomly chooses $x_{\langle l,v\rangle} \in_R \mathbb{Z}_q^*$ and broadcasts $y_{\langle l,v\rangle} := x_{\langle l,v\rangle}P$ to every other user.
- In round i, $i = 2,\ldots,d_T+1$ each $U_{\langle l,v\rangle}$, $l > d_T+1-i$, computes $x_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle}$. If $i \neq d_T+1$ then for each subtree $T_{\langle d_T+1-i,v\rangle}$, $0 \leq v \leq 3^{d_T+1-i}-1$, a user assigned to one of the leaf nodes of $T_{\langle d_T+1-i,v\rangle}$ broadcasts $y_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle} := x_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle}P$ to every other user.

Fig. 7.16. Protocol by Lee, Kim, Kim, and Ryu [123]

are handled similarly to those of TGDH by updating the group key and the tree structure which

is kept mostly balanced. Each user $U_{\langle l,v\rangle}$ is supposed to save own secret key path $\mathcal{X}_{\langle l,v\rangle}$ and the structure of T including all public keys to process dynamic events. The protocol does not provide authentication. Lee *et al.* specify a special assumption which they call a Decisional Ternary Tree Group Bilinear Diffie-Hellman (DTGBDH) assumption (which is polynomial-time reducible to the decisional version of the BDH assumption) and apply a heuristic security proof (similar to the one from [119]) to show that under this assumption a passive adversary is not able to distinguish $\hat{e}(P,P)^{x_{\langle 1,0\rangle}x_{\langle 1,1\rangle}x_{\langle 1,2\rangle}}$ from a random number. However, the proof does not consider hash functions H_1 and H_2 used to derive the secret keys of the tree nodes. Beside that, the protocol provides neither implicit key authentication nor key confirmation nor security against key control attacks.

7.2.9 Protocols by Barua, Dutta, and Sarkar

The protocol proposed by Barua *et al.* [16] uses a similar approach as Lee *et al.* to extend the Joux' protocol to the group setting. Barua *et al.* described a top down recursive procedure which constructs a balanced ternary tree down to the (d_T-1)*th*-level. All nodes at level d_T-1 have either one, two, or three child nodes, and all nodes at level $l < d_T-1$ are either leaf nodes or have three child nodes. In general, the protocol proceeds as in Figure 7.16 with two main differences. First, [16] replaces the elliptic curve equivalent of the two-party Diffie-Hellman key exchange protocol used in [123] for the parent nodes with two child nodes by the computations according to the Joux' protocol whereby one of the two users, say $U_{\langle d_T,0\rangle}$, simulates the third user by choosing two secret keys $x_{\langle d_T,0\rangle}$, and $x'_{\langle d_T,0\rangle}$, so that both real users can compute $\hat{e}(P,P)^{x_{\langle d_T,0\rangle}x'_{\langle d_T,0\rangle}x_{\langle d_T,1\rangle}}$. Similar to [123] the protocol uses hash functions to map the secrets of non-leaf nodes to $\mathbb{Z}_q^*$. The second difference is that unlike [123] the sponsor does not broadcast the node's public key, but sends it directly to the users in the subtree(s) rooted at sibling node(s). In addition to the unauthenticated protocol [16] describes an authenticated version which is based on the protocol by Zhang *et al.* [191] which in turn adapts ID-based authentication to the original Joux' protocol. Barua *et al.* also described two operations to handle joins and leaves of users. The heuristic security analysis of the protocol uses the Decisional Hash Bilinear Diffie-Hellman (DHBDH) assumption which is related to the non-standard Hash Decisional Diffie-Hellman (HDDH) assumption described in [4] which in turn is strictly stronger than DDH. The analysis considers only semantic security against passive adversaries and implicit key authentication. In [145], Pereira and Quisquater described a successful replay attack against the authenticated version of [16].

7.3 Provably Secure Group Key Exchange Protocols

7.3.1 Protocol by Katz and Yung

Katz and Yung [112] proposed a static group key exchange protocol (as a modification of an earlier heuristically analyzed protocol by Burmester and Desmedt [50, 51]). All mathematical operations are performed in the cyclic group $\mathbb{G}=\langle g\rangle$ of prime order q such that $\mathbb{G}$ is a subgroup of a cyclic group of prime order $p=\beta q+1$, $\beta\in\mathbb{N}$ where the DDH assumption holds. Figure 7.17 describes an unauthenticated version of the protocol. Note that after the protocol is completed every user holds the same group key $k=k_i=g^{r_1r_2+r_2r_3+\ldots+r_nr_1} \pmod p$. Additionally, Katz and Yung proposed an authentication compiler based on digital signatures. This compiler requires one additional communication round for participants to agree on a list of

- In round 1 each U_i chooses a random $r_i \in_R \mathbb{Z}_q$ and broadcasts $z_i := g^{r_i}$.
- In round 2 each U_i broadcasts $X_i := (z_{i+1}/z_{i-1})^{r_i}$. Then each U_i computes $k_i := (z_{i-1})^{nr_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i+n-2}$ for $i = 1, \ldots, n$.

Fig. 7.17. Protocol by Katz and Yung [112]

nonces that is then used to signature generation of all outgoing messages and verification of all incoming messages (more on Katz and Yung's authentication compiler in Section 9.3). For the security analysis Katz and Yung apply their KY model (Section 6.2.8) and show that the protocol is AKE-secure based on the DDH assumption. Due to the limitations of the KY model described in Section 6.2.8 the provided security proof does not consider strong corruptions. Also the protocol does not provide mutual authentication and key confirmation. Bohli *et al.* [32] describe an attack against key confirmation of the authenticated version of the protocol in the presence of an adversary $\mathcal{A}$ represented by malicious participants. The attack is successful for all $n > 3$, and proceeds as follows: $\mathcal{A}$ corrupts users U_1 and U_3 and continues with the protocol execution according to its specification up to round 3 (corresponds to round 2 in the unauthenticated version). Then $\mathcal{A}$ swaps X_1 and X_3, i.e., it broadcasts $X_1 := (z_4/z_2)^{r_3}$ and $X_3 := (z_2/z_4)^{r_1}$ (instead of original $X_1 := (z_2/z_4)^{r_1}$ and $X_3 := (z_4/z_2)^{r_3}$). Due to the absence of the key confirmation uncorrupted users U_2 and U_4 compute different group keys $k_2 \neq k_4$. This can be seen by computing the quotient $\frac{k_2}{k_4} = X^n \cdot \left(\frac{z_2}{z_4}\right)^{nr_3} \neq 1$. For example, if $n = 4$ then $k_2 = z_1^{4r_2} \cdot X_2^3 \cdot X_1^2 \cdot X_4 = \frac{z_2^{3r_1} \cdot z_3^{3r_2}}{z_4^{r_1} \cdot z_3^{r_4}}$ and $k_4 = z_3^{4r_3} \cdot X_4^3 \cdot X_3^2 \cdot X_2 = \frac{z_4^{3r_3} \cdot z_1^{3r_4}}{z_2^{r_3} \cdot z_1^{r_2}}$. Note that $z_i^{r_j} = z_j^{r_i}$ for any $i \neq j$ with $1 \leq i, j \leq n$. Therefore, $\mathcal{A}$ who acts on behalf of U_1 and U_3 is able to compute $k_2 := \frac{z_2^{3r_1} \cdot z_2^{3r_3}}{z_4^{r_1} \cdot z_4^{r_3}}$ and $k_4 := \frac{z_4^{3r_3} \cdot z_4^{3r_1}}{z_2^{r_3} \cdot z_2^{r_1}}$. Bohli *et al.* notice that verification $\prod_i X_i \stackrel{?}{=} 1$ by every user U_i prior to the computation of the group key k_i helps to prevent this attack; however, only in the presence of at most one malicious participant.

Additionally, we present an attack whereby a malicious participant U_i being in the strong corruption model is able to control the resulting value of the group key. The attack proceeds as follows: U_i chooses some $\tilde{r} \in \mathbb{Z}_q$ before the execution is started and his goal is to influence other users to accept with the key $\tilde{k} := g^{\tilde{r}}$. During the execution of the protocol U_i waits for all contributions $z_{j \neq i}$ (the common assumption is that communication channel is asymmetric) and reveals internal states of all other participants that include their private exponents $r_{j \neq i}$. Then, U_i uses

$$r_i := \frac{\tilde{r} - (r_1 r_2 + \ldots + r_{i-2} r_{i-1} + r_{i+1} r_{i+2} + \ldots + r_n r_1)}{r_{i-1} + r_{i+1}}$$

to compute own contribution z_i. It is easy to check that $k := g^{r_1 r_2 + r_2 r_3 + \ldots + r_n r_1} = g^{\tilde{r}} = \tilde{k}$ holds.

7.3.2 Protocol by Abdalla, Bresson, Chevassut, and Pointcheval

Abdalla *et al.* [5] described a variant of the KY protocol where authentication is achieved by the means of the password-based encryption using a secure symmetric encryption scheme $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ modeled as an ideal cipher, and three functions H_1, H_2, and $Auth$ modeled as random oracles. Each participant U_i holds a secret password pw which is common for the whole group. The protocol proceeds as described in Figure 7.18. The authors prove the AKE-security of their protocol using the BCP$^+$ model (Section 6.2.8) under the DDH assumption with additional non-standard assumptions of ROM and ICM. Abdalla *et al.* also proved that their protocol resists dictionary attacks unless the adversary is able to test several passwords in one session.

- In round 1 each U_i chooses a random nonce N_i and broadcasts (U_i, N_i).
- In round 2 each U_i computes a session identifier $S := U_1|N_1|\ldots|U_n|N_n$ and its symmetric key $pw_i := H_1(S, i, pw)$. Each U_i chooses a secret exponent $r_i \in_R \mathbb{Z}_q$ and broadcasts $z_i^* := \texttt{Enc}(pw_i, z_i)$ where $z_i := g^{r_i}$.
- In round 3 each U_i decrypts $z_{i-1} := \texttt{Dec}(pw_{i-1}, z_{i-1}^*)$ and $z_{i+1} := \texttt{Dec}(pw_{i+1}, z_{i+1}^*)$, and broadcasts $X_i := (z_{i+1}/z_{i-1})^{r_i}$. Then each U_i computes the temporary key $k_i := (z_{i-1})^{nr_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i+n-2}$ for $i = 1, \ldots, n$, and broadcasts a confirmation token $Auth_i := Auth(S, \{z_j^*, X_j\}_j, k_i, i)$. Then, each user receives and verifies all confirmation tokens and (if all verifications are successful) accepts with the session group key $K_i := H_2(S, \{z_j^*, X_j, Auth_j\}_j, k_i)$.

Fig. 7.18. Protocol by Abdalla, Bresson, Chevassut, and Pointcheval [5]

They prove it by showing that the advantage of the adversary to break the AKE-security of the protocol grows linearly with the number of messages that have been built by the adversary. Note that password-based protocols cannot achieve security against malicious participants because of the adversary can use the shared password pw to authenticate messages on behalf of other participants. The attack of Bohli *et al.* against Katz and Yung's protocol works obviously in this protocol too. Assume that $n = 4$. By swapping X_1 and X_3 the adversary $\mathcal{A}$ achieves that U_2 and U_4 compute different temporary keys $k_2 \neq k_4$. Then $\mathcal{A}$ sends to U_4 resp. U_2 a forged confirmation token $Auth_2 := Auth(S, \{z_j^*, X_j\}_j, k_4, 2)$ resp. $Auth_4 := Auth(S, \{z_j^*, X_j\}_j, k_2, 4)$ where X_1 and X_3 are swapped. Both users U_2 and U_4 verify corresponding confirmation tokens successfully and believe that $k_2 = k_4$. But in fact their temporary session keys are different so that their session group keys K_2 and K_4 are different too.

7.3.3 Protocol by Kim, Lee, and Lee

Kim *et al.* [114] proposed a dynamic extension of Katz and Yung's protocol. Although, some of the computation steps in [114] have certain similarity with the protocol in [112] the mathematical structure of the computed group key is completely different. In Figure 7.19 we describe the setup operation of the protocol. Again, all group members $U_1, \ldots, U_n$ are arranged into a circle. All computations are performed in a cyclic multiplicative group $\mathbb{G}$ of prime order p generated by g. The protocol uses a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^l$, and a secure digital signature scheme $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ with each U_i having a corresponding signature key pair (sk_i, pk_i). In order to handle dynamic events each U_i has to store k, $h_i^L := H(z_{i-1}^{r_i}|k|0)$,

- In round 1 each U_i randomly chooses $N_i \in_R \{0,1\}^l$ and $r_i \in_R \mathbb{Z}_p^*$ and computes $z_i := g^{r_i}$. Additionally, U_n computes $H(N_n|0)$. Then, each U_i generates $\sigma_i^1 := \texttt{Sign}(sk_i, z_i|ID|0)$ (resp. U_n generates $\sigma_n^1 := \texttt{Sign}(sk_n, H(N_n|0)|z_n|ID|0))$ where $ID := \{U_1, \ldots, U_n\}$ is a set of identities, and broadcasts σ_i^1 together with z_i (resp. $H(N_n|0)|z_n$).
- In round 2 each U_i verifies the received signatures and halts if this process fails. Otherwise, U_i computes $t_i^L := H(z_{i-1}^{r_i}|ID|0)$, $t_i^R := H(z_{i+1}^{r_i}|ID|0)$, and $T_i := t_i^L \oplus t_i^R$. Additionally, U_n generates $\hat{T} := N_n \oplus t_n^R$. Each U_i generates $\sigma_i^2 := \texttt{Sign}(sk_i, N_i|T_i|ID|0)$ (resp. U_n generates $\sigma_n^2 := \texttt{Sign}(sk_n, \hat{T}|T_n|ID|0))$ and broadcasts the signature σ_i^2 together with $N_i|T_i$ (resp. $\hat{T}|T_n$). Then, each U_i verifies the received signatures and halts if this process fails. Otherwise, each U_i computes $\tilde{t}_{i+1}^R := T_{i+1} \oplus t_i^R$, $\tilde{t}_{i+2}^R := T_{i+2} \oplus \tilde{t}_{i+1}^R$, …, $\tilde{t}_{i+n-1}^R := T_{i+n-1} \oplus \tilde{t}_{i+n-2}^R$, and checks that $t_i^L \stackrel{?}{=} \tilde{t}_{i+n-1}^R$. Then, each U_i decrypts $\tilde{N}_n := \hat{T} \oplus \tilde{t}_n^R$, checks whether $H(\tilde{N}_n|0) \stackrel{?}{=} H(N_n|0)$, and computes the session group key $k_i := H(N_1|\ldots|N_n|0)$.

Fig. 7.19. Protocol by Kim, Lee, and Lee [114] (Setup Operation)

$h_i^R := H(z_{i+1}^{r_i}|k|0)$, and $X := H(N_n|k|0)$. These values are used to update the circle structure and the group key upon occurring dynamic group changes in a way which is more efficient compared to the new execution of the setup operation. For the detailed description of the dynamic operations we refer to [114].

The authors prove the AKE-security of their protocol against active adversaries using the BCP and KY models with minor modifications (Section 6.2.8) under the CDH assumption. However, their proof requires, additionally, the non-standard assumptions of ROM. Though the provided proof does not consider strong corruptions the authors claim that their protocol guarantees strong-forward secrecy with respect to the BCP$^+$ model [42]. Their argumentation is that the stored hash values h_i^L, h_i^R, and X upon being revealed can be used to compute group keys of the subsequent sessions but not of the previous sessions. However, the authors do not provide a formal proof of this claim. Also the authors claim that their protocol resists attacks aimed to control the value of the resulting group key without explaining what do they understand under the key control and without providing any formal proofs for their claims (also because no formal definitions of key control were available at that time). Bohli *et al.* [32] present some impersonation attacks on this protocol for the case that session identifiers are generated via concatenation of the exchanged message flows as in the BCP and KY models. As a result of their replay attack two honest participants compute identical session group keys without being partnered. For the successful attack $\mathcal{A}$ must replay σ_1^1 together with z_1 addressed to U_3. The reason for this attack is that U_1 is not directly neighbored with U_3 so that the substitution does not affect the computation of the session group key, i.e., $k_1 = k_3$. This attack is of technical nature since it allows $\mathcal{A}$ to ask a $Reveal$ query to U_3 and obtain k_3 while asking $Test$ query to U_1 ($\mathcal{A}$ is allowed to proceed like that because U_1 and U_3 are not partnered). Since $k_3 = k_1$ the adversary makes a correct guess in response to its $Test$ query with non-negligible probability and breaks, therefore, the AKE-security of the protocol. In order to prevent this technical attack Bohli *et al.* suggest that in the beginning of the second round each user U_i computes a session identifier $\mathtt{sid}_i := H(ID|N_1|\ldots|H(N_n))$ and use it for signature generation and verification in the continuation of the protocol execution.

7.3.4 Protocols by Barua and Dutta

Dutta and Barua [83] described an extension of the Katz and Yung's protocol towards dynamic groups. The modified unauthenticated setup protocol proceeds as described in Figure 7.20. The

- In round 1 each U_i randomly chooses $r_i \in_R \mathbb{Z}_p^*$ and sends $z_i := g^{r_i}$ to U_{i-1} and U_{i+1} (note that $U_0 = U_n$ and $U_{n+1} = U_1$).
- In round 2 each U_i computes $t_i^L := z_{i-1}^{r_i}$, $t_i^R := z_{i+1}^{r_i}$, and broadcasts $X_i := t_i^L / t_i^R$ (note that $t_i^R = t_{i+1}^L$ for $1 \leq i \leq n-1$, $t_n^R = t_1^L$, and $t_{i+n-1}^R = t_i^L$). Then, each U_i computes $\tilde{t}_{i+1}^R := X_{i+1} t_i^R$, $\tilde{t}_{i+2}^R := X_{i+2} \tilde{t}_{i+1}^R$, ..., $\tilde{t}_{i+n-1}^R := X_{i+n-1} \tilde{t}_{i+n-2}^R$, and checks that $t_i^L \stackrel{?}{=} \tilde{t}_{i+n-1}^R$. Then, each U_i computes the session group key $k_i := \tilde{t}_1^R \cdots \tilde{t}_n^R$, the seed $x := H(k_i)$, and saves t_i^L, t_i^R.

Fig. 7.20. Protocol by Dutta and Barua [83] (Unauthenticated Setup Operation)

authors prove the AKE-security of their protocol against a passive adversary using a minor modification of the KY model (Section 6.2.8) under the DDH assumption.

Additionally, Dutta and Barua described an authenticated version of their protocol where digital signatures are used to sign every protocol message assuming that every member U_i has a signature key pair (sk_i, pk_i). This authentication approach is similar to the one described by Katz and Yung [112] for the only difference that it does not uses nonces. The authors prove that the authenticated version is AKE-secure under the DDH assumption using the same model as for the unauthenticated version. They also prove that the protocol provides the weak form of forward secrecy. However, it seems that there are several inaccuracies in their proof. First, the

proof does not consider corrupt queries which reveal long-term keys (in this case sk_i). However, the forward secrecy requirement assumes that the adversary is allowed to reveal these keys. Second, in contrast to the Katz and Yung's technique the protocol by Dutta and Barua does not use nonces as part of the signed messages. Note that nonces (or any other fresh randomness) are essential to resist replay attacks (as considered in [112]). Therefore, it is not clear whether the proposed protocol remains secure if the adversary replays a message from some previous session. Indeed, no such attacks are covered by the proof. Moreover, the simulation described in the proof fails if the adversary replays a message as part of its *Send* query, because these queries are answered from the predefined transcripts obtained through execute queries, and, therefore, any unpredictable *Send* query would not be answered.

Additionally, Dutta and Barua described dynamic operations allowing new members to join to the group and current members to leave it. In order to handle these operations efficiently (without restarting the initial protocol) participants use the saved values: $x := H(k_i)$, t_i^L, and t_i^R. Note that these values are considered as part of internal states of protocol participants. For the proof of AKE-security of the dynamic protocol version the authors apply a variant of the BCP model (Section 6.2.7). They also claim that the dynamic protocol version achieves forward secrecy. However, their proof has the same weaknesses as the proof of the static authenticated version. Additionally, recall that the BCP model does not consider strong corruptions. Therefore, it is not clear whether the dynamic protocol by Dutta and Barua still provides AKE-security if strong corruptions are considered. Beside these weaknesses the protocols by Dutta and Barua are also susceptible to the attack against key control in a similar way as described for the protocol by Katz and Yung in Section 7.3.1.

It is worth being noticed that Dutta and Barua [86] described a variant of Kim *et al.*'s protocol where they used a password pw shared between all users $U_1, \ldots, U_n$ together with three secure symmetric encryption schemes $(\mathtt{Gen}_i, \mathtt{Enc}_i, \mathtt{Dec}_i)$, $i = 1, 2, 3$, for the purpose of authentication instead of the originally used digital signatures. The protocol proceeds as described in Figure 7.21. To prove the AKE-security of their protocol, Dutta and Barua applied a variant of the

- In round 1 each U_i randomly chooses $N_i \in_R \{0,1\}^l$ and $r_i \in_R \mathbb{Z}_p^*$ and sends $z_i^* := \mathtt{Enc}_1(pw, z_i)$ where $z_i := g^{r_i}$ to U_{i-1} and U_{i+1}.
- In round 2 each U_i extracts z_{i-1} and z_{i+1}, and computes $t_i^L := H(z_{i-1}^{r_i}|ID|0)$ and $t_i^R := H(z_{i+1}^{r_i}|ID|0)$. For $i = 1, \ldots, n-1$ each U_i computes $T_i := t_i^L \oplus t_i^R$ while U_n computes $T_n := N_n \oplus t_n^R$. For $i = 1, \ldots, n-1$ each U_i broadcasts $\mathtt{Enc}_2(pw, N_i|T_i)$ while U_n broadcasts $\mathtt{Enc}_3(pw, T_n)$. Then, each U_i recovers N_n (as in Kim *et al.*'s protocol) and computes the session group key $k_i := H(N_1|\ldots|N_n)$.

Fig. 7.21. Protocol by Dutta and Barua with Password-Based Authentication [86]

model proposed in [43] (see also Section 6.2.8) together with the non-standard assumptions of the Random Oracle Model. The authors also claimed that the proposed protocol is secure against off-line dictionary attacks. However, [5] described an efficient substitution attack on this protocol which allows to mount a successful dictionary attack revealing the shared password pw.

7.3.5 Protocols by Bresson and Catalano

Bresson and Catalano [40] proposed a family of static constant-round authenticated group key exchange protocols based on the following generalized protocol where each user U_i has a pair of private/public keys (sk_i, pk_i) generated by a key generation algorithm of a public key encryption scheme. These protocols derive the group key from the interpolation of secretly shared

polynomials. All computations are performed modulo a sufficiently large prime number p. The generalized protocol proceeds as specified in Figure 7.22. The authors prove the AKE-security

- In round 1 each U_i chooses random values $s_i, b_{i,1}, \ldots, b_{i,n-1} \in_R \mathbb{Z}_p$, defines $f_i(z) := s_i + b_{i,1}z + \ldots + b_{i,n-1}z^{n-1} \mod p$ and sends $f_i(j)$ to user U_j.
- In round 2 each U_i computes $f(i) := \sum_{j=1}^{n} f_j(i) \mod p$, encrypts $f(i)$ with the public keys of all users, denoted $ENC_j(f(i))$ and sends the corresponding value to U_j.
- In round 3 each U_i decrypts all received values from the previous round, and interpolates them in $\mathbb{Z}_p$ retrieving the secret $f(0) := s'_i = s_1 + \ldots + s_n \mod p$. Then, each U_i computes $k'_i := F_{s'_i}(U_i)$ where F is a pseudo-random function and broadcasts it to other users. Upon receiving these messages from all other users each U_i checks whether $F_{s'_i}(U_j) = k'_j$ holds for all the received values, and if so computes the group key $k := F_{s'_i}(ID)$, where $ID = \{U_1, \ldots, U_n\}$ is the set of identities.

Fig. 7.22. Protocol by Bresson and Catalano [40] (Generalized Version)

of their protocol under standard assumptions, i.e., the existence of one-way functions (Section 5.2), following the requirements of the BCP model. The protocol also achieves key confirmation. Furthermore, the authors provide two realizations based on the El-Gamal and RSA encryption schemes where the function F is instantiated by a cryptographic hash function. For the El-Gamal-based realization the authors show resilience against key control whereby considering its weaker version with honest participants who are assumed to have a biased source of randomness so that a curious adversary tries to gain extra information and break the AKE-security of the protocol. The authors stress that the case where participants are malicious and try to bias the resulting group key deliberately is not considered. Note that even this weaker form of key control is achieved in an inefficient way by requiring that each participant chooses in the first step an additional random value r_i, which is then encrypted with the El-Gamal public keys of all other participants and broadcasted. Note also that all protocols proposed in [40] are static.

7.3.6 Protocols by Bresson, Chevassut, Pointcheval, and Quisquater

Bresson *et al.* [46] proposed a static group key exchange protocol (as an extension of the earlier heuristically analyzed protocols by Steiner *et al.* [174]). Each member U_i is in possession of a long-lived key-pair (sk_i, pk_i) for the digital signature scheme $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$. All mathematic operations are performed in a finite cyclic group $\mathbb{G}$ of the prime order q, $|q| = \kappa$ generated by g. The protocol proceeds as described in Figure 7.23 whereby $ID := \{U_1, \ldots, U_n\}$ is a set of identities of protocol participants and $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ is a cryptographic hash function.

- *Upflow stage:* In round $i = 1, \ldots, n-1$ the user U_i chooses a random $x_i \in_R \mathbb{Z}_q^*$, computes $X_i := \{g^{\prod\{x_t | t \in [1,i] \wedge t \neq j\}} | j = 1, \ldots, i\}$ and $Z_i := g^{x_1 \cdots x_i}$, generates $\sigma_i := \texttt{Sign}(sk_i, ID|X_i|Z_i)$ and forwards $X_i|Z_i$ and σ_i to U_{i+1}. Upon receiving the corresponding message U_{i+1} verifies the attached signature and halts if this verification is not successful.
- *Downflow stage:* In round n the user U_n chooses a random $x_n \in_R \mathbb{Z}_q^*$ computes $X_n := \{g^{\prod\{x_t | t \in [1,n] \wedge t \neq i\}} | i = 1, \ldots, n-1\}$, $Z_n := Z_{n-1}^{x_n}$, generates $\sigma_n := \texttt{Sign}(sk_n, ID|X_n)$ and broadcasts X_n and σ_n. After the verification of σ_n each U_i, $i \in [1, n-1]$ extracts $Z'_i := g^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_n} \in X_n$ and computes the temporary key $k_i := {Z'}_i^{x_i}$ while U_n computes $k_n := Z_n$. Finally, each U_i accepts with the session group key $K_i := H(ID|X_n|k_i)$

Fig. 7.23. Protocol by Bresson, Chevassut, Pointcheval, and Quisquater [46]

To prove the AKE-security of their protocol Bresson *et al.* specify a Group Computational Diffie-Hellman assumption (GCDH) previously surfaced in [34, 174], which is polynomial-time reducible to the standard cryptographic assumptions CDH and DDH [44]. The proof itself is performed in the BCPQ model with the additional non-standard assumptions of ROM. Due to the

limitations of the BCPQ model the proof does not consider strong corruptions. Also the issue of key control is not considered. Interesting is that an adversary $\mathcal{A}$ represented by a malicious participant U_j (being in the strong corruption model) can control the value of the temporary key k_i computed by some uncorrupted user U_i. For this purpose $\mathcal{A}$ simply chooses $\tilde{x} \in \mathbb{Z}_q^*$ prior to the execution of the protocol and computes own exponent after having revealed exponents of other participants as $x_j := \frac{\tilde{x}}{\prod_{i \neq j} x_i}$. Also set ID may be known prior to the protocol execution. Obviously, whether $\mathcal{A}$ is able to control the resulting value K_i depends either on the collision-resistance property of H, or on the ability of $\mathcal{A}$ to find appropriate values in X_n such that the input of H corresponds to some value chosen by $\mathcal{A}$ prior to the protocol execution. The protocol in Figure 7.23 does not provide key confirmation and mutual authentication.

Additionally, Bresson *et al.* describe a mechanism to achieve MA-security (key confirmation and mutual authentication). It consists of one additional communication round where each user U_i after having computed K_i as described above computes and broadcasts $H(K_i, U_i)$. Every other member U_j receives this message and checks whether $H(K_j, U_i) \stackrel{?}{=} H(K_i, U_i)$ holds (note, this implies $K_i \stackrel{?}{=} K_j$). If this verification holds for all participants then each user U_i computes the actual group key as $K_i^{'} := H(K_i, 0)$. The authors prove that in the Random Oracle Model this additional round adds mutual authentication and key confirmation with respect to the definition of MA-security in the BCPQ model while preserving AKE-security. Recall that this definition is flawed. Hence, the provided security proof is no more reliable. It is easy to show that the above approach does not provide key confirmation in the presence of malicious participants. This is because the hash value $H(K_i, U_i)$ does not provide sender identification.

Bresson, Chevassut, and Pointcheval [41] extended the protocol from [46] to handle additions (join protocol) and exclusions (remove protocol) of group members. For this purpose every user has to save the last broadcasted set X_n which is then updated with freshly chosen private exponent(s). The remove protocol consists of a single downflow stage where the highest-indexed remaining user U_n deletes from X_n all values addressed to the excluded group members, raises all remaining values to the power of the freshly generated exponent x_n' and broadcasts the updated set X_n'. The join protocol requires an upflow stage where private exponents of joined members are collected in a way described in the setup protocol starting with the position of the highest-indexed member U_n. The downflow stage is similar to that of the setup protocol. For the detailed description of the dynamic operations we refer to [41]. The authors proof the AKE-security of this dynamic protocol in their BCP model (see also Section 6.2.7) using the non-standard assumptions of ROM under the GCDH assumption. Their proof is similar to the proof in [46] and does not consider strong corruptions. Indeed, it is possible to show that if an adversary $\mathcal{A}$ obtains private exponents of participants in one protocol sessions then it is able to compute group keys from the previous sessions using public values in X_n. Considerations concerning key control issues are similar to that of the static protocol.

In their subsequent work Bresson, Chevassut, and Pointcheval [42] revised the protocols from [41] and proposed a variant that is secure under standard assumptions, i.e., without considering assumptions of ROM. Instead of digital signatures as in [41] the authentication in the protocols from [42] is carried out by a message authentication code (MAC) function. Each user U_i is in possession of an El-Gamal-like long-lived key (s_i, g^{s_i}) where g is a generator of some group $\mathbb{G}$ where the DDH assumption holds. The MAC-key K_{ij} used for authentication between users U_i and U_j is derived as $F_1(g^{s_i s_j})$ where F_1 is a universal hash function $H_r()$ ([95, Section 6.4.3]) which takes as input beside $g^{s_i s_j}$ an additional random string $r = r_{ij}$ which U_i and U_j receive during the registration of their identities from the Certification Authority which is part

of the Public-Key Infrastructure (PKI). Note that K_{ij} does not expose. The group key is derived as $K := F_2(ID, X_n, k)$ where F_2 is a universal hash function $H_r()$ where the required random string $r = r_k$ is chosen by the user U_n who sends the final broadcast message. The authors prove the AKE-security of their protocol using their BCP$^+$ model (see also Section 6.2.7) under the Group Decisional Diffie-Hellman GDDH and the Multi Decisional Diffie-Hellman (MDDH) assumptions which are polynomial-time reducible to the DDH assumption. It is worth being noticed that the protocol in [42] does not explicitly provide MA-security. Indeed, if the mechanism from [41, 46] is applied then the proof requires non-standard assumptions of the Random Oracle Model. As for the requirement of key control we notice that the adversary $\mathcal{A}$ (being in the strong corruption model) represented by a malicious participant U_j can control the value of k (the computation is similar to that of the protocol in [46]) and may learn ID prior to the execution of the protocol. In addition to that if U_n is malicious then it can choose the random string $r = r_k$ used in the universal hash function $H_r()$ non-uniformly implying that resulting hash values are not uniformly distributed. Hence, the probability that $\mathcal{A}$ controls the value of the key is given by the probability that for two different input values of F_2, say α and β, the adversary finds r_α and r_β such that $H_{r_\alpha}(\alpha) = H_{r_\beta}(\beta)$. Obviously, this is some non-standard requirement of collision-resistance of universal hash functions.

In their other work [43], Bresson *et al.* proposed a static variant of [46] with the password-based authentication. The authors show the AKE-security of their protocol in the BCPQ model [46] using the non-standard assumptions of ROM under the Trigon Group Computational Diffie-Hellman (TGCDH) assumption (which is a special form of the GCDH assumption and thus reducible to CDH and DDH). Their proof also shows that the protocol is resistant against dictionary attacks. For this purpose in addition to private exponents x_i each U_i has a second private exponent ν_i such that the resulting group key is derived as $K := H(ID, X_n, k)$ where H is a cryptographic hash function and $k = g^{\prod_i (x_i \nu_i)}$. However, the protocol does not deal with dynamic group changes, and its proof does not consider the requirement on forward secrecy. The authors also mention that MA-security can be achieved using the hash function based mechanism from [46]. However, it does not guarantee security in the presence of malicious participants.

Recently, Bresson *et. al.* [45] proposed another static password-based group key exchange protocol, called GOKE, for IEEE802.11's ad-hoc mode. This protocol proceeds as described in Figure 7.24 whereby $VP(x_i)$ (validity proof) is a non-interactive zero-knowledge proof [159] for the knowledge of x_i in the exponent of the elements of X_i, pw_i is a secret password shared between U_n and U_i, f is a symmetric encryption function, and H_1, H_2, H_3 are cryptographic hash functions.

- *Upflow stage:* In round $i = 1, \ldots, n-1$ the user U_i chooses random $x_i \in_R \mathbb{Z}_q^*$, $r_i \in_R \{0,1\}^\kappa$ computes $X_i := \{g^{\prod \{x_t | t \in [1,i] \wedge t \neq j\}} | j = 1, \ldots, i\}$ and $Z_i := g^{x_1 \cdots x_i}$, generates $VP(x_i)$ and forwards $r_i | X_i | Z_i$ and $VP(x_i)$ to U_{i+1}. Upon receiving the corresponding message U_{i+1} verifies the attached validity proof and halts if this verification is not successful.
- *Downflow stage:* In round n the user U_n chooses a random $x_n \in_R \mathbb{Z}_q^*$, computes for all $i = 1, \ldots, n$, the temporary key $k := g^{\prod_i x_i}$, $k_i := K^{1/x_i}$ using elements from X_{n-1}, $k_i' := k_i^{\alpha_i}$ where $\alpha_i \in_R \mathbb{Z}_q^*$, and broadcasts $k_i^* := k_i' \cdot f(pw_i)$. Upon receiving this value each U_i "unmasks" k_i', computes $k_i'' := k_i'^{x_i}$, an authenticator $\texttt{Auth}_i := H_1(r_1 | \ldots | r_n | i | k_i'')$ and sends $\texttt{Auth}_i$ to U_n.
- In round $n+1$ user U_n checks $\texttt{Auth}_i \stackrel{?}{=} H_1(r_1 | \ldots | r_n | i | k^{\alpha_i})$ for each received $\texttt{Auth}_i$. If all received authenticators are valid then U_n broadcasts $\texttt{Auth}_i' := H_2(r_1 | \ldots | r_n | i | k^{\alpha_i} | k_i)$ and k_i.
- In round $n+2$ each U_i checks $\texttt{Auth}_i' \stackrel{?}{=} H_2(r_1 | \ldots | r_n | i | k_i'' | k_i)$. If this verification holds then U_i accepts with the session group key $K_i := H_3(r_1 | \ldots | r_n | k_i^{x_i})$.

Fig. 7.24. Protocol by Bresson, Chevassut, and Pointcheval with Password-Based Authentication [45]

The authors prove the AKE-security of the protocol using a formal setting from [43] together with the non-standard assumptions of the Random Oracle Model (for the applied hash functions) under the TGCDH assumption. Note that in this protocol U_n acts as a sole authenticator and checks whether all participants computed the same key. There is no direct authentication and key confirmation between any to participants U_i and U_j with $1 \leq i, j < n$. Thus, if the authenticator U_n is malicious then he can mount a successful attack against the mutual authentication and key conformation properties by sending $\texttt{Auth}'_i := H_2(r_1| \ldots |r_n|i|k''_i|\tilde{k}_i)$ with some fake $\tilde{k}_i$.

7.3.7 Protocols by Dutta, Barua, and Sarkar

Dutta, Barua, and Sarkar [87] extended their heuristically analyzed unauthenticated protocol from [16] (Section 7.2.9) by the authentication mechanism based on the non-interactive multi-signature scheme by Boldyreva [33] and on the pairing-based signature scheme by Boneh, Lynn, and Shacham [37]. Recall that the protocol assigns users to the leaf nodes of a balanced ternary key tree T and applies iterations of Joux' protocol to compute the resulting key at the root of T which is then used to derive the session group key. The tree is constructed in such a way that all nodes at level $d_T - 1$ are parent nodes of either one, two, or three child nodes, and all nodes at levels $l < d_T - 1$ are either leaf nodes or parents of exactly three child nodes. In general the protocol proceeds as described in Figure 7.25 with the following computation rules for the secret values $x_{\langle d_T-1,v\rangle}$ where $H : \mathbb{G}_2 \rightarrow \mathbb{Z}_q^*$ is a cryptographic hash function:

- if $\langle d_T - 1, v\rangle$ is a parent of one leaf node then $x_{\langle d_T-1,v\rangle}$ is exactly the secret value chosen by the user assigned to that leaf node,
- else if $\langle d_T - 1, v\rangle$ is a parent of two leaf nodes $\langle d_T, 3v + i\rangle$, $i = 0, 1$ then $x_{\langle d_T-1,v\rangle} = H(\hat{e}(P, P)^{x_{\langle d_T,3v\rangle} x_{\langle d_T,3v+1\rangle} x'})$ where $x_{\langle d_T,3v\rangle}$ and x' are secret values chosen by the user assigned to $\langle d_T, 3v\rangle$, and $x_{\langle d_T,3v+1\rangle}$ is chosen by the user assigned to $\langle d_T, 3v + 1\rangle$,
- else if $\langle d_T - 1, v\rangle$ is a parent of three leaf nodes $\langle d_T, 3v + i\rangle$, $i = 0, 1, 2$ then $x_{\langle d_T-1,v\rangle} = H(\hat{e}(P, P)^{x_{\langle d_T,3v\rangle} x_{\langle d_T,3v+1\rangle} x_{\langle d_T,3v+2\rangle}})$ where each $x_{\langle d_T,3v+i\rangle}$ is chosen by the user assigned to $\langle d_T, 3v + i\rangle$

Note that each node $\langle l, v\rangle$ with $l < d_T - 1$ is either a leaf node or a parent of exactly three nodes. Therefore, $x_{\langle l,v\rangle}$ is either chosen by the user assigned to $\langle l, v\rangle$ or computed as $H(\hat{e}(P, P)^{x_{\langle l+1,3v\rangle} x_{\langle l+1,3v+1\rangle} x_{\langle l+1,3v+2\rangle}})$ via Joux' technique.

- In round 1 each $U_{\langle l,v\rangle}$, $0 \leq v \leq 3^l - 1$ randomly chooses $x_{\langle l,v\rangle} \in_R \mathbb{Z}_q^*$, computes and sends $y_{\langle l,v\rangle} := x_{\langle l,v\rangle} P$ together with the corresponding digital signature σ to every user in the subtree(s) rooted at the sibling node(s) of $\langle l, v\rangle$. Every user verifies received signatures before he proceeds with the protocol.
- In round i, $i = 2, \ldots, d_T + 1$ each $U_{\langle l,v\rangle}$, $l > d_T + 1 - i$, computes $x_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle}$. If $i \neq d_T + 1$ then for each subtree $T_{\langle d_T+1-i,v\rangle}$, $0 \leq v \leq 3^{d_T+1-i} - 1$, a user (sponsor) assigned to one of the leaf nodes of $T_{\langle d_T+1-i,v\rangle}$ computes and sends $y_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle} := x_{\langle d_T+1-i,\lfloor v/3^{i-1}\rfloor\rangle} P$ together with the corresponding non-interactive multi-signature σ to every user in the subtree(s) rooted at the sibling node(s) of $\langle d_T + 1 - i, v\rangle$. Every user verifies received signatures before he proceeds with the protocol.

Fig. 7.25. Protocol by Dutta, Barua, and Sarkar [87]

The secret value $x_{\langle 0,0\rangle}$ at the root of the tree is then used as the session group key.

Dutta *et al.* prove the AKE-security of their protocol against active adversaries under the non-standard cryptographic assumption DHBDH in the modified version of the KY security model (Section 6.2.8). Unlike the authentication procedure in Katz and Yung's protocol the protocol by Dutta *et al.* does not use nonces as part of signed messages. Note that nonces are useful to

resist replay attacks. Therefore, it is not clear whether the proposed protocol remains secure in case where an active adversary replays previous messages. The simulation described in the proof fails if the adversary replays a message as part of his send query, because send queries are answered from the predefined transcripts obtained through execute queries, and, therefore, any unpredictable send query (such as a replayed message) cannot be answered. Also, the security of some parts of Dutta *et al.*'s modifications to the original BCPQ model are arguable as discussed in Section 6.2.8. Beside this it is possible to show that the protocol is susceptible to the attack against key control (in the strong corruption model). The idea behind the attack is that the adversary represented by a malicious participant may know the tree structure prior to the execution of the protocol and own position within it. The adversary adaptively computes all secret values in its path up to $x_{\langle 0,0\rangle}$ prior to the execution of the protocol. Then, during the protocol execution it influences honest participants that are assigned to the leaf nodes of the subtrees rooted at nodes that are siblings of the nodes in the adversarial path to compute each $x_{\langle l,v\rangle}$ as chosen by the adversary. In the strong corruption model this attack is simple. For example, consider that two honest participants are assigned to the leaf nodes $\langle d_T, 3v+1\rangle$ and $\langle d_T, 3v+2\rangle$ and the malicious participant is assigned to $\langle d_T, 3v\rangle$, and chooses $x_{\langle d_T-1,v\rangle}$ prior to the protocol execution as the output of $H(\hat{e}(P,P)^{\tilde{x}})$ for some chosen $\tilde{x}$. Then during the protocol execution it reveals $x_{\langle d_T,3v+1\rangle}$ and $x_{\langle d_T,3v+2\rangle}$ as part of the internal information of honest participants and computes $x_{\langle d_T,3v\rangle} := \frac{\tilde{x}}{x_{\langle d_T,3v+1\rangle} x_{\langle d_T,3v+2\rangle}}$. Further, this kind of the attack can be performed for all $x_{\langle l,v\rangle}$ in the adversarial path including the session group key $x_{\langle 0,0\rangle}$.

In [84], Dutta and Barua extended the above protocol by additional operations that handle dynamic group changes, i.e., addition and deletion of group members. Both events are handled using a sponsor and result in the updated logical tree T' and the updated secret value at the root of T' whereby some secret values $x_{\langle l,v\rangle}$ remain unchanged. The authentication is achieved using digital signatures and multi-signatures as in [87]. The authors prove the AKE-security of their dynamic protocol against active adversaries under the DHBDH assumption using a mix of the BCP$^+$ and KY models with own technical modifications (Section 6.2.8). The proof considers only weak corruptions. Obviously, no forward secrecy in case of strong corruptions is provided because the knowledge of any unchanged $x_{\langle l,v\rangle}$ can be used to compute the previous value of $x_{\langle 0,0\rangle}$.

7.4 Summary and Discussion

In Table 7.4 we summarize results of our security-focused survey of group key exchange protocols while considering only provably secure protocols from Section 7.3 since security models used in their proofs provide a solid background for a fair comparison of their security degrees. We consider only protocols that have not been found to be flawed despite of their security proof. For each considered protocol we specify the applied security model together with some possibly used non-standard models like Random Oracle Model (ROM) or Ideal Cipher Model (ICM). Additionally, we specify the underlying cryptographic assumption, and point out whether the proof considers strong (S) or weak (W) corruptions. In the last columns we give the protocol type (S for static; D for dynamic).

STRONG VS. WEAK CORRUPTIONS Observe, only few security proofs of the described protocols consider a powerful adversary which is given access to strong corruptions. From the analysis of security models in the previous chapter we know that only the BCP$^+$ and KS/UC-KS

Table 7.1. Analysis of Provably Secure Group Key Exchange Protocols

Protocol		Model(s)	Assumption(s)	Corr.	S/D
Abdalla *et al.*	[5]	BCP$^+$ + ICM, ROM	DDH	S	S
Barua and Dutta	[83]	KY	DDH	W	S
Barua and Dutta	[83]	BCP	DDH	W	D
Barua and Dutta	[84]	BCP	DHBDH	W	D
Bresson and Catalano	[40]	BCP	OW	W	S
Bresson *et al.*	[46]	BCPQ + ROM	GCDH	W	S
Bresson *et al.*	[41]	BCP + ROM	GCDH	W	D
Bresson *et al.*	[42]	BCP$^+$	GDDH, MDDH	W	D
Bresson *et al.*	[43, 45]	BCPQ + ROM	TGCDH	W	S
Dutta *et al.*	[87]	BCPQ	DHBDH	W	S
Katz and Yung	[112]	KY	DDH	W	S
Kim, Lee, and Lee	[114]	BCP,KY + ROM	CDH	W	D

models provide definitions that consider strong corruptions. However, there exists no group key exchange protocol proven secure in the KS/UC-KS models. The only protocols proven secure in the BCP$^+$ model have been proposed by Abdalla *et al.* [5] and by Bresson *et al.* [42]. The protocol proposed by Abdalla *et al.* is static. Intuitively, all static protocols provide security in the strong corruption model as long as they provide security in the weak corruption model. This is because in static protocols internal (ephemeral) information used for the computation of the group key is chosen independently at random for each new protocol execution (session). However, this is not the case in dynamic protocols. The protocol proposed by Bresson *et al.* [42] is dynamic. However, it does not provide security (in particular in case of forward secrecy) in the strong corruption model. Therefore, its proof considers only weak corruptions. For the protocol proposed by Dutta *et al.* [87] and its dynamic version in [84] we pointed out that given security proofs have mistakes concerning possible replay attacks. For the dynamic protocol proposed by Kim, Lee, and Lee in [114] security in the strong corruption model has been claimed but not formally proven (also due to the absence of adequate security models at that time).

STANDARD VS. NON-STANDARD ASSUMPTIONS Security proofs of the protocols in [5, 41, 46, 114] require non-standard assumptions of ROM or ICM. As for the cryptographic assumptions, security of the protocols in [83] and [112] is based on the standard cryptographic assumption DDH. Security of the generalized protocol by Bresson and Catalano [40] is based on the standard assumption on the existence of one-way functions (OW). Security of the protocols proposed by Bresson *et al.* in [41, 43, 45, 46] has been proven under the assumptions GCDH and TGCDH which are polynomial-time reducible to the standard cryptographic assumptions CDH and DDH [44]. The CDH assumption has also relevance for the security of the protocol proposed by Kim, Lee, and Lee [114]. The protocol proposed by Bresson, Chevassut, and Pointcheval in [42] re-

lies on the GDDH and MDDH assumptions which are polynomial-time reducible to DDH. The only non-standard cryptographic assumption is DHBDH which is used in the protocols from [84, 87].

ATTACKS OF MALICIOUS PARTICIPANTS We focus on the attacks of malicious participants against the properties of key confirmation and mutual authentication (Section 6.1.3) and key control and contributiveness (Section 6.1.5). From the analysis of security models in the previous chapter we know that only the KS/UC-KS and BVS models provide definitions that consider requirements on key confirmation and mutual authentication with respect to the malicious participants. However, none of the considered protocols has been proven secure in any of these models. Intuitively, all password-based authentication protocols including [5, 43, 45] are susceptible to such attacks of malicious participants (see Section 7.3.2 for an example of the attack against [5]) because password-based authentication does not provide identification when used in the group setting. However, such identification is important if all protocol participants must authenticate mutually. Obviously, protocols where mutual authentication is performed via digital signatures are more suitable for this purpose. Still, none of the security proofs of the protocols in [40, 41, 46, 83, 84, 87, 112, 114] that apply digital signatures considers this kind of attacks of malicious participants.

According to our analysis in the previous chapter, none of the security models used in security proofs of the GKE protocols in Table 7.4 provides formal definitions concerning key control and contributiveness in case of strong corruptions. This is the reason why none of the currently existing group key exchange protocols could be proven secure against this kind of attacks (see Section 7.3.1 for an example attack against [112] in case of strong corruptions).

MAIN RESULT Based on the above arguments we emphasize that none of the currently existing dynamic group key exchange protocols is provably secure with respect to the strong corruptions and attacks of malicious participants. Therefore, one of the goals of this dissertation (achieved in Chapter 10) is to design a dynamic GKE protocol that can be proven to satisfy these strong security requirements.

Chapter 8

A Modular Security Model for Group Key Exchange Protocols

According to the analysis in Section 6.2, currently existing security models for group key exchange (GKE) protocols still have some limitations concerning their security definitions w.r.t. a powerful adversary who is given access to the strong corruptions and who may represent a (subset of) malicious participant(s) mounting attacks against various security requirements, such as mutual authentication, key confirmation, unknown key-share resilience, key control and contributiveness.

In this chapter we propose a modular computational security model for GKE protocols which allows reductionist security proofs and considers strong corruptions for all given security definitions. In fact, we extend the ideas of Bresson *et al.* [42] and Katz and Yung [112] concerning the AKE-security with additional secrecy types, specify MA-security alternatively to the definitions of the insider security by Katz and Shin [111], and define contributiveness strictly stronger than in Bohli *et al.* [32], while also considering dynamic GKE protocols.

8.1 Execution Parameters and Definitions

8.1.1 Protocol Participants, Instance Oracles

Let $\mathcal{U}$ be a set of N users (their identities) that may participate in a GKE protocol P. Note that one execution of P may consist of several subsequent executions of the sub-protocols of P called *operations*. If P is static then it provides only one operation (setup) so that the execution of P

is finished by the end of this operation. If P is dynamic then it provides additional operations (join and leave) so that after the setup operation several subsequent dynamic operations can be executed until the execution of P is finished.

In order to handle participation of $U \in \mathcal{U}$ in distinct concurrent operation executions of P we consider U having an unlimited number of instances called *oracles*. By Π_U^s with $s \in \mathbb{N}$ we denote the s-th instance oracle of U. When we need to distinguish between the oracles of two different users U_i and U_j we use the notation $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$, respectively. Note that every instance oracle Π_U^s might be represented as a process controlled by the user U and modeled as a PPT algorithm.

For each concurrent operation execution of P we consider a non-empty set of oracles $\mathcal{G}$ of size $n \in [1, N]$, called a *group*. A new group of oracles is created for every invoked operation execution. Each oracle in $\mathcal{G}$ is called a *group member*. By $\mathcal{G}_i$ for $i = 1, \ldots, n$ we denote the index of the user related to the i-th oracle involved in this group. This i-th oracle in $\mathcal{G}$ is further denoted $\Pi(\mathcal{G}, i)$. Thus, for every $i \in \{1, \ldots, n\}$ there exists $\Pi(\mathcal{G}, i) = \Pi_{\mathcal{G}_i}^s \in \mathcal{G}$ for some $s \in \mathbb{N}$.

8.1.2 Long-Lived Keys

Every user $U \in U$ may hold a long-lived (long-term) key LL_U generated by some probabilistic algorithm $\mathtt{GenLL}(1^\kappa)$. Every LL_U is generated in advance and becomes known to all oracles Π_U^s, $s \in \mathbb{N}$ upon their initialization (see Section 8.1.5). LL_U is usually represented by a private/public key pair (sk_U, pk_U) of user U or by a symmetric key (password) pw. If we need to distinguish between different the long-lived keys of different users U_i and U_j we use the notation LL_i and LL_j, respectively.

8.1.3 Internal State Information

Every Π_U^s maintains an *internal state information* $\mathtt{state}_U^s$ which is composed of all private ephemeral information used during the protocol execution excluding the long-lived key LL_U. This information is typically used by the oracles to compute or update group keys during the execution of the protocol. For example, $\mathtt{state}_U^s$ may contain ephemeral secret exponents used by a participant of the Diffie-Hellman key exchange protocol.

There are several reasons for separating LL_U from $\mathtt{state}_U^s$. First, LL_U is not ephemeral, moreover it belongs to the user and not to the oracle. Second, in spirit of [42] long-lived keys may have different protection mechanisms, e.g., smart cards. Thus, revealing LL_U does not necessarily imply that $\mathtt{state}_U^s$ is revealed and vice versa. Third, by distinguishing between LL_U and $\mathtt{state}_U^s$ we are able to model curious adversaries who do not get full control over the users but reveal their ephemeral secrets. We stress that this second reason is essential for our definition of contributiveness (cf. Section 8.2.10).

Further, we stress that $\mathtt{state}_U^s$ contains private information. Especially, $\mathtt{state}_U^s$ does not contain any public information, such as public keys of participants or public parameters of cryptographic schemes. This is convenient for security proofs like in [21, 42, 114, 167] where a popular *(secure) erasure technique* [76] is applied, i.e., the erasure of the internal state information does not imply the erasure of the public information which is usually required to continue with the protocol execution.

8.1.4 Session Group Key, Session ID, Partner ID

Every operation execution of P constitutes a separate *session* with own group $\mathcal{G}$. In each session every participating oracle Π_U^s computes the session group key $k_U^s \in \{0,1\}^\kappa$ (notation k is used in case of generalization). We do not consider the session group key k_U^s as part of $\mathtt{state}_U^s$ in order to model AKE-security in case of strong corruptions. Therefore, in our security model we allow the adversary to obtain k_U^s without obtaining $\mathtt{state}_U^s$ and vice versa. This is also because k_U^s is, usually, used in the actual group application where its protection might be insufficient.

The session where Π_U^s participates in is identified by a unique *session id* $\mathtt{sid}_U^s$. Note that this value is known to all oracles participating in the same session. We stress that either the unique session id is provided by the environment of the protocol (e.g., a high-level application) or is computed during the actual operation execution (e.g., via nonces). By q_s we denote the total number of (concurrent) sessions (operation executions) of P.

Similarly, each oracle $\Pi_U^s \in \mathcal{G}$ has a *partner id* $\mathtt{pid}_U^s$ that contains the identities of all users holding oracles in $\mathcal{G}$ (including U), or formally

$$\mathtt{pid}_U^s := \{U_{\mathcal{G}_j} \mid \Pi(\mathcal{G}, j) \in \mathcal{G},\ \forall j = 1, \ldots, n\}.$$

We say that two oracles $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are *partnered* if

$$U_i \in \mathtt{pid}_j^{s_j}, U_j \in \mathtt{pid}_i^{s_i},\ \text{and}\ \mathtt{sid}_i^{s_i} = \mathtt{sid}_j^{s_j}.$$

Obviously, all oracles in $\mathcal{G}$ are considered as partners. This notion of *partnering* is helpful for formal definitions of some security goals concerning P.

8.1.5 Instance Oracle States

An oracle Π_U^s may be either *used* or *unused*. The oracle is considered as unused if it has never been initialized. Each unused oracle Π_U^s must be initialized with the long-lived key LL_U before it can participate in the operation execution. After the initialization the oracle is marked as used, and turns into the *stand-by* state where it waits for an invocation to execute a protocol operation.

Upon receiving such invocation the oracle Π_U^s learns its partner id $\mathtt{pid}_U^s$ (and possibly $\mathtt{sid}_U^s$) and turns into a *processing* state where it sends, receives and processes messages according to the description of the invoked operation. During the whole processing state the internal state information $\mathtt{state}_U^s$ is maintained by the oracle. The oracle Π_U^s remains in the processing state until it collects enough information to compute the session group key k_U^s.

After Π_U^s computes k_U^s it *accepts* and *terminates* the execution of the protocol operation (possibly after some additional auxiliary steps). The oracle turns then back into the stand-by state where it can be invoked for the new operation. If the operation execution fails (due to any adversarial actions) then Π_U^s turns into the stand-by state without having accepted, i.e., the session group key k_U^s is set to some undefined value. Whether Π_U^s has accepted or not can be modeled by a boolean variable which should be set true as soon as k_U^s is computed, and false after the oracle receives invocation for a new protocol operation.

Remark 8.1. We need the stand-by state in order to model dynamic group key exchange protocols where each protocol execution may consist of several operation executions. Thus, in dynamic protocols there is no state for termination. On the other hand, in static protocols there is only one operation. Therefore, in static protocols the oracle Π_U^s after having accepted does not turn back into the stand-by state anymore, but remains *terminated*, that is the stand-by state is reached only once, straight after the initialization.

8.1.6 Static GKE Protocol

The following definition considers only static GKE protocols.

Definition 8.2 (Static GKE Protocol). *A* static group key exchange protocol S-GKE *consists of the key generation algorithm* KeyGen*, and an operation* Setup *defined as follows:*

- P.KeyGen(1^κ)*: On input a security parameter* 1^κ *provides each user* U *in* $\mathcal{U}$ *with a long-lived key* LL_U*.*
- P.Setup$(\mathcal{S})$*: On input a set* $\mathcal{S}$ *of* n *unused oracles a new group* $\mathcal{G}$ *is created and set to be* $\mathcal{S}$*. A probabilistic interactive protocol is executed between* $\Pi(\mathcal{G},1)$*, . . .,* $\Pi(\mathcal{G},n)$ *such that all oracles accept with the session group key and terminate.*

Remark 8.3. Note that in the definition of P.Setup oracles must be unused and must, therefore, be initialized before they proceed with the interaction.

8.1.7 Dynamic GKE Protocol

In the following we extend Definition 8.2 towards dynamic GKE protocols. The main difference are additional dynamic operations which may be used to update the session group key without a new execution of P.Setup. In the description of the dynamic operations we assume that n is the size of the initial group $\mathcal{G}$ prior to the execution. Note that after the operation is executed the new group size may be different.

Definition 8.4 (Dynamic GKE Protocol). *A* dynamic group key exchange protocol D-GKE *consists of the key generation algorithm* KeyGen *and operations* Setup*,* Join$^+$*, and* Leave$^+$ *defined as follows:*

- P.KeyGen(1^κ)*: as in Definition 8.2.*
- P.Setup$(\mathcal{S})$*: as in Definition 8.2, except that oracles turn into the stand-by state.*
- P.Join$^+(\mathcal{G},\mathcal{J})$*: On input a group* $\mathcal{G}$ *of* n *used oracles and a set* $\mathcal{J}$ *of* n_J *(unused) oracles a new group* $\mathcal{G}$ *of* $n+n_J$ *oracles is created and set to be* $\mathcal{G}\cup\mathcal{J}$*. A probabilistic interactive protocol is executed between* $\Pi(\mathcal{G},1)$*, . . .,* $\Pi(\mathcal{G},n+n_J)$ *such that all oracles accept with the updated session group key and turn into the stand-by state.*
- P.Leave$^+(\mathcal{G},\mathcal{L})$*: On input a group of* n *used oracles and a set* $\mathcal{L}$ *of* n_L *used oracles a new group* $\mathcal{G}$ *of* $n-n_L$ *oracles is created and set to be* $\mathcal{G}\setminus\mathcal{L}$*. A probabilistic interactive protocol is executed between* $\Pi(\mathcal{G},1)$*, . . .,* $\Pi(\mathcal{G},n-n_L)$ *such that all oracles accept with the updated session group key and turn into the stand-by state.*

Remark 8.5. In the definition of P.Join$^+$ the joining set $\mathcal{J}$ may consist either of the unused or used oracles. $\mathcal{J}$ being a set of the unused oracles models the case where completely new users join to the group. Since the oracles are unused they have to be initialized before the interaction starts. $\mathcal{J}$ being a set of the used oracles models the case where users that have already participated in the previous operations join to the group (possibly after having leaved it before). In this case no initialization of the oracles is required. In both cases oracles in the initial group $\mathcal{G}$ must be used. This is because users can join to and leave from already existing groups.

Remark 8.6. Our model for S-GKE and D-GKE is independent of the underlying communication channel between the group members. It can be used to model GKE protocols where group members send messages over broadcast / multicast and unicast channels. Note that if a GKE protocol assumes only a broadcast channel then all exchanged messages are received by all oracles.

In the remainder of this part, unless otherwise specified, by P we mean a dynamic GKE protocol D-GKE. Note also that S-GKE can be considered as an operation of D-GKE.

8.2 Adversarial Model and Security Requirements

In order to analyze security of a GKE protocol we need to define an adversarial setting which specifies capabilities and possible actions of the attacker. In our model, the adversary $\mathcal{A}$ is represented by a PPT algorithm. It is assumed to have complete control over all communication in the network and may interact with group members by making queries to an unlimited number of oracles Π_U^s as described in the next section. Note that similar to other models we do not deal with denial-of-service attacks (neither by non-legitimate participants nor by malicious participants) which generally aim to prevent an oracle from accepting. Our security definitions (similar to those of other models) state requirements on the session group keys that are accepted by the oracles.

8.2.1 Queries to the Instance Oracles

The following queries model the attacks the adversary could mount through the network.

- $Setup(\mathcal{S})$: This query models $\mathcal{A}$ eavesdropping the honest execution of $\mathtt{P.Setup}$. This query is only available to $\mathcal{A}$ if the oracles in $\mathcal{S}$ are unused. $\mathtt{P.Setup}(\mathcal{S})$ is executed and $\mathcal{A}$ is given the transcript of the execution.
- $Join^{+}(\mathcal{G}, \mathcal{J})$: This query models $\mathcal{A}$ eavesdropping the honest execution of $\mathtt{P.Join}^{+}$. This query is only available to $\mathcal{A}$ if the oracles in $\mathcal{G}$ are partnered, have previously accepted, and are in the stand-by state (this is because oracles can only join to already existing groups), and the oracles in $\mathcal{J}$, only if used, are in the stand-by state. $\mathtt{P.Join}^{+}(\mathcal{G}, \mathcal{J})$ is executed and $\mathcal{A}$ is given the transcript of the execution.
- $Leave^{+}(\mathcal{G}, \mathcal{L})$: This query models $\mathcal{A}$ eavesdropping the honest execution of $\mathtt{P.Leave}^{+}$. This query is only available to $\mathcal{A}$ if the oracles in $\mathcal{G}$ are partnered, have previously accepted and are in the stand-by state (this is because oracles can only leave from the already existing groups), and if $\mathcal{L} \subset \mathcal{G}$ (this is because only current group members can leave the group). $\mathtt{P.Leave}^{+}(\mathcal{G}, \mathcal{L})$ is executed and $\mathcal{A}$ is given the transcript of the execution.
- $Send(op, \Pi_U^s, m)$: This query models $\mathcal{A}$ sending messages to the oracles. $\mathcal{A}$ receives the response which Π_U^s would have generated after having processed the message m according to the description of the protocol operation specified by a string $op \in \{'\mathtt{setup}', '\mathtt{join}', '\mathtt{leave}'\}$. The response may also be an empty string if m is incorrect or unexpected. $\mathcal{A}$ can use the following $Send$ queries to ask an oracle Π_U^s to invoke the operation execution of P with other oracles:
  - $Send('\mathtt{setup}', \Pi_U^s, \mathcal{S})$ asks Π_U^s for the first message that invokes the setup operation between Π_U^s and other oracles in $\mathcal{S}$. Note that $\Pi_U^s \in \mathcal{S}$ must hold.
  - $Send('\mathtt{join}', \Pi_U^s, \mathcal{G}, \mathcal{J})$ asks Π_U^s for the first message that invokes the join operation between Π_U^s and oracles in $\mathcal{G}$ and $\mathcal{J}$. Note that $\Pi_U^s \in \mathcal{G}$ or $\Pi_U^s \in \mathcal{J}$ must hold.
  - $Send('\mathtt{leave}', \Pi_U^s, \mathcal{G}, \mathcal{L})$ asks Π_U^s for the first message that invokes the leave operation between Π_U^s, and other oracles in $\mathcal{G}$. Note that $\Pi_U^s \in \mathcal{G}$ and $\mathcal{L} \subset \mathcal{G}$ must hold.

  The described send queries are answered according to the specification of the protocol. For any other send queries of $\mathcal{A}$ related to the execution of the invoked operation a corresponding identifier op is the first argument of the query.

- $RevealKey(\Pi_U^s)$: This query models the attacks which reveal the session group key. $\mathcal{A}$ is given the session group key k_U^s. Note that this query is answered only if Π_U^s has previously accepted, i.e., the session group key k_U^s is defined.
- $RevealState(\Pi_U^s)$: This query models the attacks resulting in the internal state of the oracle being revealed. $\mathcal{A}$ is given the internal state information $\texttt{state}_U^s$. Note that $\texttt{state}_U^s$ contains ephemeral secrets used by Π_U^s. We stress that this query does not reveal LL_U (this allows to model separate protection mechanisms for long-lived keys) and k_U^s (this allows to model AKE-security with respect to strong corruptions).
- $Corrupt(U)$: This query models the attacks resulting in the group member's long-lived key being revealed. $\mathcal{A}$ is given the long-lived key LL_U.
- $Test(\Pi_U^s)$: This query will be used to model the (A)KE-security of a GKE protocol. It can be asked by $\mathcal{A}$ at any time during $\mathcal{A}$'s execution, but only once. The query is answered only if Π_U^s has previously accepted. The oracle generates a random bit b. If $b = 1$ then $\mathcal{A}$ is given k_U^s, and if $b = 0$ then $\mathcal{A}$ is given a random string.

A *passive* adversary can eavesdrop the execution of the protocol operations via $Setup$, $Join^+$, and $Leave^+$ queries, reveal session group keys, internal states and corrupt participants via $RevealKey$, $RevealState$, and $Corrupt$ queries, respectively, and is also allowed to ask $Test$ queries. Additionally, it is given access to the $Send$ queries, however, with the restriction that it is not allowed to inject, replay, or modify messages. Thus, a passive adversary can truly forward, drop, and delay messages, or deliver them out of order. Although the queries $Setup$, $Join^+$, and $Leave^+$, can be simulated using the query $Send$ with the appropriate invocation messages, the presence of these queries still allows a separate treatment of the (weaker) passive adversaries who are restricted to the eavesdropping of the protocol execution in the sense of [112]. Note that in our model the passive adversary is stronger than in [112] and is comparable to the one from [62].

To the contrary an *active* adversary is given access to all queries and is also allowed to modify, inject, and replay messages via the $Send$ query.

The separation between passive and active adversaries is convenient for the definition of security goals in a modular way. Note that modular definitions of security goals can be used to simplify security proofs, design application-specific protocols for which not all security goals might be required, and construct so-called "compilers" that may be used to add specific security properties to a protocol.

Further, we distinguish between honest and malicious participants. We say that Π_U^s is a *malicious participant* if the adversary has previously asked the $Corrupt(U)$ query, thus the adversary can participate in $\texttt{P}$ on behalf of U. In all other cases Π_U^s is *honest*. Finally, we say that the adversary is *curious* if it asks a $RevealState(\Pi_U^s)$ query for some honest Π_U^s.

8.2.2 Correctness

Our model should be able to exclude "useless" GKE protocols. The following definition of correctness is based on the uniqueness of session ids and ensures that all oracles participating in the same session of $\texttt{P}$ compute the same group key.

Definition 8.7 (Correctness). *A GKE protocol* $\texttt{P}$ *is* correct *if for any operation execution between the oracles* $\Pi(\mathcal{G}, 1)$, ..., $\Pi(\mathcal{G}, n)$ *with the same session ids* $\texttt{sid}$ *all oracles accept with the same session group key* k.

8.2.3 Forward Secrecy

The notion of forward secrecy allows to distinguish between damages to the AKE-security of previously computed session group keys caused by different actions of the adversary in subsequent sessions. In the following we list possible attack scenarios aiming to reveal a previous session group key: (1) $\mathcal{A}$ reveals a subset of session group keys of subsequent sessions; (2) $\mathcal{A}$ corrupts the long-lived key of a group member in any subsequent session; (3) $\mathcal{A}$ reveals internal states of the oracles in some subsequent sessions; and (4) the combination of any subset of the three previous cases.

We do not consider the attack scenario (1) in the definition of forward secrecy, because any GKE protocol `P` must provide security against it implicitly as part of its AKE-security. So when talking about forward secrecy we distinguish between attack scenarios (2), (3) and (4). Since long-lived keys in `P` are used primarily for authentication of messages, but internal state information is used explicitly to compute the session group key, the attack scenario (3) is more damageable than (2). In the following definition of forward secrecy we distinguish only between two types of attacks: (a) the attack scenario (2), and (b) the combination of attack scenarios (2) and (3). Note that the corruption of the long-lived key does not necessarily reveal the internal state information of the oracle, and vice versa, because it may have a different protection mechanism.

Based on these considerations we distinguish between *weak forward secrecy* (`wfs`) where the adversary is allowed to ask the queries $Setup$, $Join^{+}$, $Leave^{+}$, $Send$, $RevealKey$, and $Corrupt$, and *strong forward secrecy* (`sfs`) where the adversary is additionally allowed to ask the query $RevealState$.

8.2.4 Backward Secrecy

The notion of backward secrecy allows to distinguish between damages to the AKE-security of session group keys computed in the future caused by different actions of the adversary in previous sessions. Obviously, backward secrecy has the opposite meaning of forward secrecy. In the following we list possible attack scenarios aiming to reveal session group keys computed in future sessions: (1) $\mathcal{A}$ reveals a subset of session group keys of previous sessions; (2) $\mathcal{A}$ corrupts the long-lived key of a group member in any previous session; (3) $\mathcal{A}$ reveals internal states of the oracles used in some previous sessions; and (4) the combination of any subset of the three previous cases.

Similar to the case of forward secrecy, we do not consider the attack scenario (1) in the definition of backward secrecy, because any GKA protocol `P` protocol must provide security against it implicitly. So when talking about backward secrecy we distinguish between attack scenarios (2), (3) and (4). Opposed to the case of forward secrecy the attack scenario (2) is more damageable than (3). This is because by corrupting the long-lived key of a group member the adversary can impersonate that group member in future sessions. Therefore in the definition of backward secrecy we distinguish between the following two types of attacks: (a) the attack scenario (3), and (b) the combination of attack scenarios (3) and (2).

Based on these considerations we distinguish between *weak backward secrecy* (`wbs`) where the adversary is allowed to ask the queries $Setup$, $Join^{+}$, $Leave^{+}$, $Send$, $RevealKey$, and $RevealState$, and *strong backward secrecy* (`sbs`) where the adversary is additionally allowed to ask the query $Corrupt$.

Remark 8.8. We refer to Section 8.4 for some additional comments on strong backward secrecy.

8.2.5 Freshness

The notion of freshness of an oracle Π_U^s is required to distinguish between various definitions of AKE-security of P with respect to different flavors of forward and backward secrecy.

Definition 8.9 (α-Freshness). *Let $\alpha \in \{\emptyset, \texttt{wfs}, \texttt{wbs}, \texttt{sfs}, \texttt{sbs}\}$. The oracle $\Pi_U^s \in \mathcal{G}$ is*

- *$\emptyset$-fresh if: neither Π_U^s nor any of its partners is asked for a $RevealKey$ query after having accepted;*
- *wfs-fresh if: (1) no $U_i \in \texttt{pid}_U^s$ is asked for a $Corrupt$ query prior to a query of the form $Send(op, \Pi_j^{s_j}, m)$ such that $U_j \in \texttt{pid}_U^s$ before Π_U^s and all its partners accept, and (2) neither Π_U^s nor any of its partners is asked for a $RevealKey$ query after having accepted;*
- *sfs-fresh if: (1) no $U_i \in \texttt{pid}_U^s$ is asked for a $Corrupt$ query prior to a query of the form $Send(op, \Pi_j^{s_j}, m)$ such that $U_j \in \texttt{pid}_U^s$ before Π_U^s and all its partners accept, (2) neither Π_U^s nor any of its partners is asked for a $RevealState$ query before they accept, and (3) neither Π_U^s nor any of its partners is asked for a $RevealKey$ query after having accepted;*
- *wbs-fresh if: (1) neither Π_U^s nor any of its partners is asked for a $RevealState$ query after $\mathcal{G}$ is created, and (2) neither Π_U^s nor any of its partners is asked for a $RevealKey$ query after having accepted;*
- *sbs-fresh if: (1) no $U_i \in \texttt{pid}_U^s$ is asked for a $Corrupt$ query prior to a query of the form $Send(op, \Pi_j^{s_j}, m)$ such that $U_j \in \texttt{pid}_U^s$ after $\mathcal{G}$ is created, (2) neither Π_U^s nor any of its partners is asked for a $RevealState$ query after $\mathcal{G}$ is created, and (3) neither Π_U^s nor any of its partners is asked for a $RevealKey$ query after having accepted.*

We say that a session is α-fresh *if all participating oracles are α-fresh.*

The above definition is given from the perspective of an oracle which participates in a concrete operation execution of P. Note that in our model a new group $\mathcal{G}$ is created for every invoked operation, i.e., new session. In the following we provide some additional explanations concerning our definition of α-freshness.

Obviously, the wfs-freshness allows $Corrupt$ queries to any user in $\mathcal{U}$ after the oracles in $\mathcal{G}$ have accepted whereas the sfs-freshness allows, additionally, $RevealState$ queries to any oracle of any user in $\mathcal{U}$ after the oracles in $\mathcal{G}$ have accepted. Beside this, the wfs-freshness allows $Corrupt$ queries in previous and concurrent operations to any user in $\mathcal{U}$ who does not have an oracle in $\mathcal{G}$ whereas the sfs-freshness allows, additionally, $RevealState$ queries in previous and concurrent operations to all oracles that do not belong to $\mathcal{G}$.

On the other hand, the wbs-freshness allows $RevealState$ queries to any oracle of any user in $\mathcal{U}$ before the group $\mathcal{G}$ is created whereas the sbs-freshness allows, additionally, $Corrupt$ queries to any user in $\mathcal{U}$ before $\mathcal{G}$ is created. Beside this, the wbs-freshness allows $RevealState$ queries in concurrent and later operations to all oracles that do not belong to $\mathcal{G}$ whereas the sbs-freshness allows, additionally, $Corrupt$ queries in concurrent and later operations to any user who does not have an oracle in $\mathcal{G}$.

Remark 8.10. Note that the requirements in Definition 8.9 guarantee that if at least one oracle Π_U^s in $\mathcal{G}$ is α-fresh then all other oracles in $\mathcal{G}$ must be α-fresh too. Thus, the whole session is then α-fresh. Additionally, these requirements ensure that if a session specified by a group $\mathcal{G}$ is α-fresh then: (1) if a user holding an oracle in $\mathcal{G}$ is corrupted during the session execution then no subsequent $Send$ queries can be asked to any of the participating oracles, (2) none of the oracles in $\mathcal{G}$ can be asked to reveal its internal state information during the session execution, and (3) none of the oracles in $\mathcal{G}$ that has accepted at the end of the session execution can

be asked to reveal the computed session group key. Note that the first statement prevents an adversary from the active participation in an α-fresh session on behalf of corrupted users and restricts the adversarial behavior in this case to that of a passive adversary. This observation is important for the reductions in the security proofs of Theorems 9.3, 9.18, 9.21, and 9.31. The other two statements ensure that during the whole α-fresh session there can be no $RevealState$ and $RevealKey$ (to accepted oracles) queries which are asked to the oracles in $\mathcal{G}$.

The notion of α-fresh sessions becomes important in AKE-security proofs since it allows to distinguish between "honest" sessions, in which the computed key is kept secret, and "corrupted" sessions, in which the adversary may learn or compute its value.

8.2.6 Corruption Models

To properly manage the adversarial capabilities for each scenario of freshness, we distinguish between the following corruption models.

Definition 8.11 (Corruption Model β). *For any PPT adversary $\mathcal{A}$ a possible corruption model $\beta \in \{\texttt{wcm}, \texttt{wcm-fs}, \texttt{wcm-bs}, \texttt{scm}\}$ is specified according to the following description:*

- `wcm` (weak corruption model): *An adversary $\mathcal{A}$ is given access to the queries $Setup$, $Join^+$, $Leave^+$, $Send$, $Test$ and $RevealKey$.*
- `wcm-fs` (weak corruption model for forward secrecy): *An adversary $\mathcal{A}$ is given access to the queries $Setup$, $Join^+$, $Leave^+$, $Send$, $Test$, $RevealKey$, and $Corrupt$.*
- `wcm-bs` (weak corruption model for backward secrecy): *An adversary $\mathcal{A}$ is given access to the queries $Setup$, $Join^+$, $Leave^+$, $Send$, $Test$, $RevealKey$ and $RevealState$.*
- `scm` (strong corruption model): *An adversary $\mathcal{A}$ is given access to the queries $Setup$, $Join^+$, $Leave^+$, $Send$, $Test$, $RevealKey$, $RevealState$ and $Corrupt$.*

8.2.7 Adversarial Setting

For a concrete proof of (A)KE-security (defined in the next section) we will need to specify capabilities of the adversary depending on the intended freshness type. Combining corresponding definitions for freshness and corruption we obtain a set of possible *adversarial settings* $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{wbs}, \texttt{wcm-bs}), (\texttt{sbs}, \texttt{scm}), (\texttt{sfs}, \texttt{scm})\}$, where $\emptyset$ denotes the freshness type for GKE protocols that do not provide any form of forward or backward secrecy.

8.2.8 (A)KE-Security

Informally, the (A)KE-security of a GKE protocol `P` requires the indistinguishability of session group keys computed in operations of `P` from random numbers. The formal description is given in the following definition.

Definition 8.12 (Game $\mathsf{Game}^{\mathsf{(a)ke}-b}_{\alpha,\beta,\mathtt{P}}(\kappa)$). *Let `P` be a GKE protocol from Definition 8.4 and b a uniformly chosen bit. Consider an adversarial setting (α, β) sampled from $\{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sbs}, \texttt{scm}), (\texttt{sfs}, \texttt{scm})\}$ and an (active) adversary $\mathcal{A}$ against (A)KE-security of `P`. We define game $\mathsf{Game}^{\mathsf{(a)ke}-b}_{\alpha,\beta,\mathtt{P}}(\kappa)$ as follows:*

- *after initialization $\mathcal{A}$ interacts with instance oracles using queries;*

- *if $\mathcal{A}$ asks a $Test$ query to an oracle α-**fresh** oracle Π_U^s which has accepted, it receives either $k_1 := k_U^s$ (if $b = 1$) or $k_0 \in_R \{0,1\}^\kappa$ (if $b = 0$);*
- *$\mathcal{A}$ continues interacting with instance oracles;*
- *when $\mathcal{A}$ terminates, it outputs a bit b' trying to guess which case it was dealing with.*

If Π_U^s is still α-fresh then the output of $\mathcal{A}$ is the output of the game. The advantage function of $\mathcal{A}$ in winning the game is defined as

$$\mathsf{Adv}^{\mathsf{(a)ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) := \left|2\Pr[\mathsf{Game}^{\mathsf{(a)ke}-b}_{\alpha,\beta,\mathtt{P}}(\kappa) = b] - 1\right|$$

Based on the definition of $\mathcal{A}$ we define the (A)KE-security of a GKE protocol $\mathtt{P}$ as follows.

Definition 8.13 ((A)KE-Security).

(1) $\mathtt{P}$ *is a* KE-secure protocol with α-secrecy ($\mathtt{GKE}$-α) *if for any* passive *PPT $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa)$ is negligible. Note, if $\alpha = \emptyset$, we say that $\mathtt{P}$ is a* KE-secure protocol.

(2) $\mathtt{P}$ *is a* AKE-secure protocol with α-secrecy ($\mathtt{AGKE}$-α) *if for any* active *PPT $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{P}}(\kappa)$ is negligible. Note, if $\alpha = \emptyset$, we say that $\mathtt{P}$ is a* AKE-secure protocol.

The (A)KE-security of $\mathtt{P}$ deals with the secrecy of the computed session group key against parties who are not legitimate protocol participants during the session where the $Test$ query occurs. The following definitions deal with the attacks of malicious protocol participants and curious adversaries.

Remark 8.14. In Section 9.3 we describe a generic solution to achieve AKE-security for any KE-secure group key exchange protocol.

8.2.9 MA-Security

In the following we provide the definition of MA-security which is supposed to subsume the following informal requirements of *unknown key-share resilience* [80], *key confirmation* [135], *explicit key authentication* [135], and the original notion of *mutual authentication* [22] as described in Chapter 6.

Our definition differs from the one in [42, 46] since we also consider malicious protocol participants. The main difference is that malicious participants are able to send valid protocol messages that may prevent honest participants from computing the same session group key. Hence, malicious oracles may become partners of honest oracles. Therefore, it is not enough to define this requirement based alone on public criteria like session ids or partner ids but also to use session group keys accepted by the oracles. Note, since MA-security is supposed to subsume key confirmation and mutual authentication requirements there should be at least two uncorrupted participants during the execution of the protocol operation which is attacked by the adversary. In our definition of MA-security malicious participants are modeled by the ability of the active adversary to corrupt oracles in every operation execution of the protocol. This allows the adversary to influence the protocol execution by malicious activities and so achieve that uncorrupted participants compute different session keys. Additionally, we allow the adversary to reveal internal states of the oracles. This models the attack scenario where private information in the internal states of uncorrupted participants may be obtained by malicious participants.

Definition 8.15 (Game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{P}}(\kappa)$). *Let $\mathtt{P}$ be a GKE protocol from Definition 8.4 and $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{P}}(\kappa)$ the interaction between an active adversary $\mathcal{A}$ who is allowed to query $Send$, $Setup$, $Join^+$, $Leave^+$, $RevealKey$, $RevealState$, and $Corrupt$, and the oracle instances. We say that $\mathcal{A}$*

wins *if at some point during the interaction there exist an uncorrupted user* U_i *whose instance oracle* $\Pi_i^{s_i}$ *has accepted with* $k_i^{s_i}$ *and another user* U_j *with* $U_j \in \mathtt{pid}_i^{s_i}$ *that is uncorrupted at the time* $\Pi_i^{s_i}$ *accepts such that*

1. *there exists* ***no*** *instance oracle* $\Pi_j^{s_j}$ *with* $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$, ***or***
2. *there exists* ***an*** *instance oracle* $\Pi_j^{s_j}$ *with* $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$ *that has accepted with* $k_j^{s_j} \neq k_i^{s_i}$.

The probability of this event is denoted $\mathsf{Succ}_{\mathtt{P}}^{\mathsf{ma}}(\kappa)$.

The first condition deals with unknown key-share attacks since it allows the adversary to introduce "fake" users that do not participate in the protocol. The second condition subsumes additionally the notions of key confirmation and mutual authentication. Note that in $\mathsf{Game}_{\mathtt{P}}^{\mathsf{ma}}(\kappa)$ we do not deal with α-fresh sessions, and the adversarial $Test$ query is useless.

Definition 8.16 (MA-Security). $\mathtt{P}$ *is a* MA-secure *GKE protocol* $(\mathtt{MAGKE})$ *if for any PPT adversary* $\mathcal{A}$ *the advantage* $\mathsf{Succ}_{\mathtt{P}}^{\mathsf{ma}}(\kappa)$ *is negligible.*

Remark 8.17. In Section 9.4 we describe a generic solution to achieve MA-security for any group key exchange protocol.

8.2.10 t-Contributiveness

In the following we propose the definition of the security goal which deals with the issues of the key control, contributiveness and unpredictability of session group keys which are essential for the security against malicious protocol participants. Informally, we consider an active PPT adversary that is allowed to corrupt group members and/or reveal internal states of their oracles (be curious) during the execution of the protocol operations such that there exists at least one honest oracle who accepts the session group key chosen previously by the adversary.

Definition 8.18 (Game $\mathsf{Game}_{\mathtt{P}}^{\mathsf{con}-t}(\kappa)$**).** *Let* $\mathtt{P}$ *be a correct GKE protocol from Definition 8.4 and* $\mathcal{A}$ *be an adversary against* t*-contributiveness which is executed over two stages,* prepare *and* attack*, in the following game denoted* $\mathsf{Game}_{\mathtt{P}}^{\mathsf{con}-t}(\kappa)$ *with* $t \in \mathbb{N}$*:*

- $\mathcal{A}(\mathsf{prepare})$ *is given access to the queries* $Send$*,* $Setup$*,* $Join^+$*,* $Leave^+$*,* $RevealKey$*,* $RevealState$*, and* $Corrupt$*. At the end of the stage it outputs* $\tilde{k} \in \{0,1\}^\kappa$*, and some state information* St*. As soon as* $\mathcal{A}$ *makes its output and all previously asked queries are processed the following sets are built:* $\mathcal{G}_{\mathtt{us}}$ *consisting of all used oracles* $\Pi_i^{s_i}$ *with uncorrupted* U_i*,* $\mathcal{G}_{\mathtt{std}}$ *consisting of the oracles* $\Pi_i^{s_i}$ *in the stand-by state with uncorrupted* U_i*, and* Ψ *consisting of session ids* $\mathtt{sid}_i^{s_i}$ *for every* $\Pi_i^{s_i} \in \mathcal{G}_{\mathtt{std}}$*. Then,* $\mathcal{A}$ *is invoked for the* attack *stage.*
- $\mathcal{A}(\mathsf{attack}, St)$ *is given access to the queries* $Send$*,* $Setup$*,* $Join^+$*,* $Leave^+$*,* $RevealKey$*,* $RevealState$*, and* $Corrupt$*. At the end of the stage* $\mathcal{A}$ *outputs* (s, U)*.*

The adversary $\mathcal{A}$ *wins in* $\mathsf{Game}_{\mathtt{P}}^{\mathsf{con}-t}(\kappa)$ *if* ***all*** *of the following holds:*

1. Π_U^s *is in the stand-by state, has accepted with* $\tilde{k}$*, no* $Corrupt(U)$ *has been asked,* $\Pi_U^s \notin \mathcal{G}_{\mathtt{us}} \setminus \mathcal{G}_{\mathtt{std}}$ *and* $\mathtt{sid}_U^s \notin \Psi$*.*
2. *There are at most* $t-1$ *corrupted users* U_i *having oracles* $\Pi_i^{s_i}$ *partnered with* Π_U^s*.*

The success probability of $\mathcal{A}$ *in winning the game is defined as*

$$\mathsf{Succ}_{\mathtt{P}}^{\mathsf{con}-t}(\kappa) := \Pr[\mathcal{A} \text{ wins in } \mathsf{Game}_{\mathtt{P}}^{\mathsf{con}-t}(\kappa)]$$

The first requirement ensures that Π_U^s belongs to an uncorrupted user. The condition $\Pi_U^s \notin \mathcal{G}_{\mathtt{us}} \setminus \mathcal{G}_{\mathtt{std}}$ prevents the case where $\mathcal{A}$ while being an operation participant outputs $\tilde{k}$ for the still running operation which is then accepted by Π_U^s that participates in the same operation (this is not an attack since participants do not compute group keys synchronously). Note that $\mathcal{G}_{\mathtt{us}} \setminus \mathcal{G}_{\mathtt{std}}$ consists of all oracles that at the end of the prepare stage are in the processing state. Similarly, the condition $\mathtt{sid}_U^s \notin \Psi$ prevents that $\mathcal{A}$ while being in the attack stage outputs (s, U) such that Π_U^s has accepted with $\tilde{k}$ already in the prepare stage; otherwise as soon as Π_U^s computes some ksU in the prepare stage $\mathcal{A}$ can trivially output its value as $\tilde{k}$ and after turning into the attack stage output (s, U) without asking any further queries and invoking any further operations. Note that in every session $\mathtt{sid}_U^s$ is unique so that $\mathtt{sid}_U^s \notin \Psi$ holds if at least one new operation has been executed with the ("clone" of) Π_U^s in the attack stage. The second requirement allows $\mathcal{A}$ to corrupt at most $t-1$ participants (out of totally n) in the session where Π_U^s accepts with $\tilde{k}$.

Note also that the oracle Π_U^s that has been influenced by $\mathcal{A}$ to accept $\tilde{k}$ must be uncorrupted but $\mathcal{A}$ is allowed to reveal its internal state during the execution of the attack stage (this is because our model separates LL_U from $\mathtt{state}_U^s$). This curious behavior models adaptive attacks by malicious participants against honest participants aiming to reveal their local secrets used in the protocol operation execution in order to influence them to accept the chosen key. This is the reason why our definition is strictly stronger than the one given in the BVS model.

Remark 8.19. In case that P is a static GKE protocol set $\mathcal{G}_{\mathtt{std}}$ consists of the terminated oracles $\Pi_i^{s_i}$ with uncorrupted U_i.

The following definition allows to classify GKE protocols and compare their resistance to the attacks of $\mathcal{A}$.

Definition 8.20 (t-Contributiveness). P *is a* t-contributory GKE protocol $(t\text{-}\mathtt{CGKE})$ *if there exists no PPT adversary* $\mathcal{A}$ *such that* $\mathsf{Succ}_{\mathtt{P}}^{\mathtt{con}-t'}(\kappa)$ *is non-negligible for all* $t' \leq t$. P *is called* contributory *if it achieves* n*-contributiveness.*

In other words, $\mathcal{A}$ can mount a successful attack against a t-contributory GKE protocol only if it corrupts at least t session participants.

Remark 8.21. In Section 9.5 we describe a generic solution to achieve n-contributiveness for any group key exchange protocol.

Remark 8.22. Definition 8.18 ensures unpredictability of group keys accepted by the honest protocol participants. This definition is sufficient to prevent (interference) attacks where the same group key occurs twice due to the actions of malicious participants. Think of a GKE protocol that is invoked by two different applications: if the same session key is obtained twice (deliberately), this is surely an interesting open door for attacks. Also, if the same group key is computed (deliberately) in two different sessions of the same protocol then session interference attacks may become possible, e.g., if the group key is used for the symmetric encryption then it would be no more possible to distinguish between cipher texts computed in both sessions. However, Definition 8.18 does not deal with the unpredictability of *some bits* of the group key, i.e., it does not require the indistinguishability of the group key k_U^s accepted by the honest oracle Π_U^s from a random number in the same space in case that up to $t-1$ oracles are corrupted and the internal states of all honest oracles can be revealed (note that in Definition 8.18 $RevealState$ queries can be asked to all honest oracles). Independent of the question on reasonability of such *decisional contributiveness* it is unclear whether it can be achieved in our strong adversarial

setting. The main problem is that all ephemeral secrets used by the honest participants during the protocol execution can be revealed by the adversary. This may allow the adversary to compute own messages adaptively as a function of the obtained information biasing the probability distribution of the resulting value with non-negligible probability. Intuitively, the problem of decisional contributiveness is related to the well-known problem of the *asynchronous distributed coin tossing* without any trusted parties for which there exists a theoretical bound of at most $(n-1)/2$ corrupted participants [74]. On the other hand, in the weak corruption model (where no $RevealState$ queries are allowed) decisional contributiveness can be easily achieved, for example, using commitments as suggested in [114, 138], whereas in the strong corruption model committed secrets can be revealed as part of the internal state $\mathtt{state}_U^s$.

8.3 Unifying Relationship of MA-Security and t-Contributiveness

In this section we present some claims to illustrate that our definitions of MA-security and contributiveness unify many mostly important informally defined security requirements, i.e., key confirmation, mutual authentication, explicit key authentication, unknown key-share resilience, key control, unpredictability of computed group keys, and contributiveness mentioned in Section 6.1 since it is not obvious in some cases. Note that missing formalism in these requirements allows only argumentative proofs.

Claim 8.23. *If* P *is a* MAGKE *protocol then it provides key confirmation and mutual authentication (explicit key authentication) in the sense of the [135, Def. 12.6-12.8], i.e., every legitimate protocol participant is assured of the participation of every other participant, and all participants that have accepted hold identical session group keys.*

Proof (informal). If P does not provide key confirmation and mutual authentication then there exists at least one uncorrupted user U_i whose oracle $\Pi_i^s \in \mathcal{G}$ has accepted with a session group key $k_i^{s_i}$ and there exists at least one another uncorrupted user $U_j \in \mathtt{pid}_i^{s_i}$ whose oracle $\Pi_j^{s_j}$ has accepted with a different session group key $k_j^{s_j} \neq k_i^{s_i}$. According to Definition 8.15 this is a successful attack against the MA-security of P. This, however, contradicts to the assumption that P is a MAGKE protocol. □

Claim 8.24. *If* P *is a* MAGKE *protocol then it is resistant against unknown key-share attacks in the sense of [38, Sec. 5.1.2], i.e., the adversary $\mathcal{A}$ cannot make one protocol participant, say U_j, believe that the session group key k is shared with $\mathcal{A}$ when it is in fact shared with a different participant U_i.*

Proof (informal). With respect to our model we assume that oracles $\Pi_j^{s_j}$ and $\Pi_i^{s_i}$ participate in the protocol on behalf of U_j and U_i, respectively. If an unknown key-share attack occurs then $\Pi_j^{s_j}$ and $\Pi_i^{s_i}$ accepted with the identical session group k, but since $\Pi_j^{s_j}$ believes that the key is shared with $\mathcal{A}$ we conclude that $U_i \notin \mathtt{pid}_j^{s_j}$ must hold (otherwise after having accepted U_j would believe that the key is shared with U_i) whereas $U_j \in \mathtt{pid}_i^{s_i}$. This implies $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) \neq (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$ On the other hand, P is by assumption MAGKE. Thus, according to Definition 8.15 for any $U_j \in \mathtt{pid}_i^{s_i}$ there must exist a corresponding oracle $\Pi_j^{s_j}$ such that $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$. This is a contradiction. □

Claim 8.25. *If* P *is a* t-CGKE *protocol then the output of any operation of* P *is unpredictable by any subset of* $t-1$ *session participants.*

Proof (informal). If the output of some operation of P is predictable by a subset of $t-1$ session participants then there exists $\tilde{k}$ which was predicted by this subset and accepted by some uncorrupted user's U oracle Π_U^s which does not belong to this subset. However, this implies that there exists an adversary $\mathcal{A}$ who corrupts $t-1$ users whose oracles are partnered with Π_U^s and predicts the session group key accepted by Π_U^s. This is a contradiction to the assumption that P is a t-CGKE protocol. □

Claim 8.26. *If* P *is a* n-CGKE *protocol then* P *is contributory in the sense of [13, Def. 3.2], i.e., each participant equally contributes to the resulting session group key and guarantees its freshness.*

Proof (informal). If P is not contributory then there exists an honest oracle Π_U^s who accepts a session group key without having contributed to its computation, i.e., the session group key accepted by Π_U^s is composed of at most $n-1$ contributions. This, however, implies that there exists an adversary $\mathcal{A}$ who corrupts up to $n-1$ users and influences Π_U^s to accept a session group key built from contributions of these corrupted users. This is a contradiction to the assumption that P is a CGKE protocol. □

Claim 8.27. *If* P *is a* n-CGKE *and a* MAGKE *protocol then* P *provides complete group key authentication in the sense of [13, Def. 6.3], i.e., any two participants compute the same session group key only if all other participants have contributed to it.*

Proof (informal). Since P is a n-CGKE protocol then according to the previous claim P is contributory. Hence, no oracle owned by an honest user accepts the key without having contributed to its computation. Since P is a MAGKE protocol oracles of all honest users accept the same session group key. Hence, the oracle of every honest user has contributed to it. Therefore, there can be no pair of honest users' oracles which accept the same group key which is not contributed to by all other honest users' oracles. Thus, P provides complete group key authentication. □

The notion of verifiable contributiveness is relevant to MA-security, since this mechanism is designed for providing confirmation (and thus, verification) that the protocol actually fits the security requirements. In the case of contributory protocols, it is intuitively true that the MA-security guarantees that the contributiveness was satisfied (otherwise, some player would be able to check that his own contribution was not properly taken into account). Hence,

Claim 8.28. *If* P *is a* n-CGKE *and* MAGKE *protocol then* P *is verifiable contributory in the sense of [13, Def. 7.3], i.e., each participant is assured of every other participant's contribution to the group key.*

Proof (informal). Since P is a MAGKE protocol oracles of all honest users accept the same session group key. Since P is also a n-CGKE protocol and, therefore, contributory the accepted group key is contributed to by the oracle of each honest user. □

8.4 A Comment on Backward Secrecy

In Section 8.2.8 we considered $(\texttt{sbs}, \texttt{scm})$ as one of the possible adversarial settings for the adversary $\mathcal{A}'$ against the AKE-security of a GKE protocol P. In this section we give some comments for this setting.

We stress that the adversarial setting $(\texttt{sbs}, \texttt{scm})$ in the definition of AKE-security is mostly of theoretical interest and is provided for the purpose of completeness. In practice long-lived keys

of group members are usually used to achieve authentication for the protocol messages and not for the actual computation of the group key. Therefore, it is intuitively clear that if an adversary is able to corrupt a group member in an earlier session and obtain its long-lived key then it can impersonate that group member in a subsequent session and learn the established group key. Obviously, in order to achieve strong backward secrecy for this case (assuming that the protocol provides weak backward secrecy) the long-lived keys have to be chosen at random for each new operation execution of the protocol (that is the statistical distance $\delta_{\mathtt{LL}}$ from Definition 8.29 must be negligible).

Definition 8.29 (Statistical Distance $\delta_{\mathtt{LL}}$). *Let* $\mathtt{P.KeyGen}(1^\kappa)$ *be a probabilistic algorithm for the generation of long-lived keys for the participants of a GKA protocol* P*. The statistical distance between the outputs of* $\mathtt{P.KeyGen}(1^\kappa)$ *and a uniform distribution over* $\{0,1\}^\kappa$ *is defined as*

$$\delta_{\mathtt{LL}} := \frac{1}{2} \sum_{x \in \{0,1\}^\kappa} \left| \Pr_{LL :=\mathtt{P.KeyGen}(1^\kappa)}[LL = x] - \Pr_{LL \in_R \{0,1\}^\kappa}[LL = x] \right|$$

Theorem 8.30. *If* P *is a* AGKE-wbs *protocol and a new long-lived key* LL_U *is obtained from* $\mathtt{P.KeyGen}(1^\kappa)$ *with negligible statistical distance* $\delta_{\mathtt{LL}}$ *by every oracle* $\Pi_U^s \in \mathcal{G}$ *prior to the operation execution with other oracles in* $\mathcal{G}$ *then* P *is* AGKE-sbs*, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{sbs,scm,P}}(\kappa) \leq 2Nq_s\delta_{\mathtt{LL}} + q_s\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{wbs,wcm\text{-}bs,P}}(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Proof. We show that for any active PPT adversary $\mathcal{A}$ against AGKE-sbs, the advantage of $\mathcal{A}$ in winning the game $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sbs,scm,P}}(\kappa)$ can be upper-bounded by the advantage of an active PPT adversary against AGKE-wbs and a function in $\delta_{\mathtt{LL}}$.

We define three games $\mathbf{G}_0$, $\mathbf{G}_1$, and $\mathbf{G}_2$ with corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ with $i \in \{0,2\}$ meaning that the output bit b' of $\mathcal{A}$ in $\mathbf{G}_i$ is identical to the randomly chosen bit b in this game.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sbs,scm,P}}(\kappa)$ from Definition 8.12 where a simulator Δ answers all queries of $\mathcal{A}$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ except that the following rule is added: Δ chooses $q_{\mathsf{s}}^* \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q_{s}^*-th session then the simulation fails and bit b' is set at random. Let Q be the event that this guess is correct. Obviously, $\Pr[\mathsf{Q}] = 1/q_{\mathsf{s}}$. Then we get

$$\begin{aligned}
\Pr[\mathsf{Win}^{\mathsf{ake}}_1] &= \Pr[\mathsf{Win}^{\mathsf{ake}}_1 \wedge \mathsf{Q}] + \Pr[\mathsf{Win}^{\mathsf{ake}}_1 \wedge \neg\mathsf{Q}] \\
&= \Pr[\mathsf{Win}^{\mathsf{ake}}_1 | \mathsf{Q}] \Pr[\mathsf{Q}] + \Pr[\mathsf{Win}^{\mathsf{ake}}_1 | \neg\mathsf{Q}] \Pr[\neg\mathsf{Q}] \\
&= \Pr[\mathsf{Win}^{\mathsf{ake}}_0] \frac{1}{q_{\mathsf{s}}} + \frac{1}{2}\left(1 - \frac{1}{q_{\mathsf{s}}}\right).
\end{aligned}$$

This implies

$$\Pr[\mathsf{Win}^{\mathsf{ake}}_0] = q_{\mathsf{s}} \left(\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \frac{1}{2} \right) + \frac{1}{2}. \tag{8.1}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the only exception that in the q_{s}^*-th session the long-lived key LL_U is chosen at random for every participating oracle Π_U^s. Since there are at most N session participants we get

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_2^{\mathsf{ake}}]| \leq N\delta_{\mathrm{LL}}. \tag{8.2}$$

Since the long-lived keys used in the q_{s}^*-th session are random and independent of the long-lived keys used in other sessions we can bound the success probability of $\mathcal{A}$ in Game $\mathbf{G}_2$ by the success probability of an active adversary in $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{wbs},\mathtt{wcm\text{-}bs},\mathtt{P}}(\kappa)$, i.e.,

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = \frac{1}{2}\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{wbs},\mathtt{wcm\text{-}bs},\mathtt{P}}(\kappa) + \frac{1}{2}. \tag{8.3}$$

In the following we successively estimate the upper bound of $\Pr[\mathsf{Win}_0^{\mathsf{ake}}]$, and so the advantage of $\mathcal{A}$ in winning the game $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sbs},\mathtt{scm},\mathtt{P}}(\kappa)$ followed from its definition as $\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{sbs},\mathtt{scm},\mathtt{P}}(\kappa) := 2\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sbs},\mathtt{scm},\mathtt{P}}(\kappa) = b] - 1$. Considering the Equations 8.1 to 8.3 we obtain

$$\begin{aligned}
\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sbs},\mathtt{scm},\mathtt{P}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&= q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq Nq_{\mathsf{s}}\delta_{\mathrm{LL}} + \frac{1}{2}q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{wbs},\mathtt{wcm\text{-}bs},\mathtt{P}}(\kappa) + \frac{1}{2}
\end{aligned}$$

By transformation we obtain the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{sbs},\mathtt{scm},\mathtt{P}}(\kappa) \leq 2Nq_{\mathsf{s}}\delta_{\mathrm{LL}} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{wbs},\mathtt{wcm\text{-}bs},\mathtt{P}}(\kappa)$$

□

Note, the requirement that long-lived keys are changed before each operation execution is not only impractical but also opposed to the idea behind the long-lived key terminology. To the contrary, the weak corruption model for backward secrecy, i.e., the adversarial setting (`wbs`, `wcm-bs`), is also of practical interest since it concerns only internal state information of the oracles and is independent of the long-lived keys of their users.

Conjecture 8.31. *If* `P` *is a* **dynamic** *GKE protocol from Definition 8.4 then* $\mathsf{Adv}^{(\mathsf{a})\mathsf{ke}}_{\mathtt{wbs},\mathtt{wcm\text{-}bs},\mathtt{P}}(\kappa)$ *is non-negligible.*

This conjecture appears likely to be true since all currently known dynamic GKE protocols provide efficient update operations for the group key based on the ephemeral secret information computed in previous sessions. Intuitively, the knowledge of this information would allow the adversary computing the updated session group key and, thus, breaking the (A)KE-security of the protocol.

On the other hand, in static GKE protocols all ephemeral secrets are, usually, chosen fresh for each new protocol execution. Therefore, resulting from the above conjecture we conclude that static GKE protocols, in nature, provide a higher degree of security than dynamic GKE protocols. This is an interesting observation in favor of the wide-spread opinion that efficiency comes at the expense of security.

Chapter 9

Seven Security-Enhancing Compilers for GKE Protocols

In this chapter we describe seven different generic techniques which can be applied to enhance security of GKE protocols providing additional properties.

9.1 Compilers and their Goals

Imagine, there exists a *"black-box"* implementation of a GKE protocol which should be used by some group application. On the one hand, this GKE implementation may provide too strong security properties but not all of them may be needed for that particular application. Usually, higher security properties are achieved at additional expense of resources like communication or computation costs. Hence, if the available GKE implementation is too strong for the application then using it may result in the decreased efficiency of the application.

On the other hand, a given GKE implementation may not satisfy all security requirements required for a particular group application. Instead of designing and implementing a new GKE protocol that satisfies the stated stronger requirements it is desirable to have a generic technique which can be applied to the given "black-box" implementation in order to enhance its security.

In general we are concerned with the following question. What is a good strategy for the implementation of GKE protocols? Obviously, it depends on the relationship between the application and the protocol. If a GKE protocol is designed and implemented for one specific application and will not be re-used in any other application (which may have different security requirements) when it is better to consider all stated requirements in the implementation and optimize the protocol accordingly. However, what to do if the implementation of the GKE protocol should be flexible and easily modifiable to be applied for various applications without any significant additional effort? Obviously, a good strategy is to implement a GKE protocol in a modular way starting with the basic implementation that satisfies the most common set of security requirements and continuing with the implementation of optional modules that can be used together with the basic implementation to provide extended security requirements. The

main goal of GKE security-enhancing GKE protocol compilers is to enable secure construction of GKE protocols in a modular way.

Definition 9.1 (Security-Enhancing GKE Protocol Compiler $\mathtt{C}$). *A* security-enhancing GKE protocol compiler $\mathtt{C}$ *is a protocol which takes as input a GKE protocol* $\mathtt{P}$ *and outputs a compiled GKE protocol* $\mathtt{C}_{\mathtt{P}}$ *with security properties not provided by* $\mathtt{P}$.

9.2 Preliminaries

9.2.1 On Separation of Long-Lived Keys and Internal States

One of the main objectives for our compilers is to provide security against strong corruptions. Therefore, we need some specification on the information stored in the internal state $\mathtt{state}_U^s$ of the oracle Π_U^s that participates in the compiled protocol $\mathtt{C}_{\mathtt{P}}$.

Mostly all of the compilers described in the following use a digital signature schemes Σ for the purpose of authentication. Σ is supposed to be existentially unforgeable (see Definition 5.17). In order to generate signatures each user U requires a private/public key pair (sk'_U, pk'_U) whereby sk'_U is the long-lived key LL_U w.r.t. our security model in Chapter 8 (this is in addition to long-lived keys possibly required by underlying protocols). Note that in our security model we separate between LL_U and $\mathtt{state}_U^s$. At the same time we allow curious behavior of the adversary, i.e., we allow the adversary via a $RevealState$ query to reveal $\mathtt{state}_U^s$ without obtaining LL_U. This, implicitly means that LL_U has a different protection mechanism compared to $\mathtt{state}_U^s$. For example, LL_U can be stored in a smart card as suggested in [42], or in any other trusted device. On the other hand, the signing algorithm $\Sigma.\mathtt{Sign}(sk'_U, m)$ usually uses some freshly generated random information in addition to the private key sk'_U in order to generate the signature σ. Now, if this random information is obtained by a curios adversary via a $RevealState$ query then it can be possibly misused to reveal the private key sk'_U. In order to prevent that $RevealState$ queries can be "misused" as the $Corrupt$ queries we assume that the execution of the signing algorithm is protected by the same mechanism as the long-lived key sk'_U, e.g., the signature generation can be performed in a smart card or in some other trusted device. In fact we assume that at any time during the execution of the compiled protocol $\mathtt{C}_{\mathtt{P}}$ the internal state information $\mathtt{state}_U^s$ contains secrets which are independent of the user's long-lived key LL_U.

9.2.2 Changes in Notation

The description of our compilers is done from the perspective of one particular operation execution (session). Therefore, by $\Pi_i^s \in \mathcal{G}$ we consider the i-th oracle in $\mathcal{G}$ (thus we use the notation Π_i^s instead of $\Pi(\mathcal{G}, i)$ used in the description of our model) assuming that there exists an index $j \in [1, N]$ such that U_j owns Π_i^s. Similar, by (sk'_i, pk'_i) resp. (sk_i, pk_i) we denote the private/public key pair of U_j used in the compiler resp. in the underlying protocol. Note also, that in our compilers (and also in many group key exchange protocols) participating oracles are ordered into a sequence.

9.3 Compiler for AKE-Security

The requirement of KE-security, i.e., indistinguishability of computed group keys with respect to passive adversaries states the basic security requirement for any GKE protocol. To the contrary, the requirement of AKE-security may be optional. For example, if a network or high-level

application provides implicit authentication then it is sufficient to use KE-secure GKE protocol. Therefore, it is reasonable to specify AKE-security as an additional property and design a compiler which adds AKE-security to any KE-secure GKE protocol. A compiler which achieves this requirement was originally proposed by Katz and Yung in [112]. In the following we give a slightly modified version of their compiler.

Definition 9.2 (Compiler for AKE-Security, C-A (Modification of Katz and Yung's Compiler in [112])). *Let* P *be a GKE protocol from Definition 8.4, and* $\Sigma := (\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *a digital signature scheme. A* compiler for AKE-security, *denoted* C-A, *consists of an algorithm* INIT *and a protocol* A *defined as follows:*

INIT*: In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk_i', pk_i') *using* $\Sigma.\texttt{Gen}(1^\kappa)$*. This is in addition to any key pair* (sk_i, pk_i) *used in* P*.*

A*: An interactive protocol between the oracles* $\Pi_1^s, \ldots, \Pi_n^s$ *in* $\mathcal{G}$ *invoked prior to any operation execution of* P*. Each* Π_i^s *chooses a random* A nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \texttt{pid}_i^s$*. After* Π_i^s *receives* $U_j|r_j$ *from all* Π_j^s *with* $U_j \in \texttt{pid}_i^s$ *it computes* $\texttt{sid}_i^s := r_1| \ldots |r_n$*. Then it invokes the operation execution of* P *and proceeds as follows:*

- *If* Π_i^s *in the operation execution of* P *is supposed to output a message* $U_i|m$ *then in* C-A_P *it computes additionally* $\sigma_i := \Sigma.\texttt{Sign}(sk_i', m|\texttt{sid}_i^s|\texttt{pid}_i^s)$ *and outputs a modified message* $U_i|m|\sigma_i$*.*
- *If* Π_i^s *receives* $U_j|m|\sigma_j$ *from* Π_j^s *with* $U_j \in \texttt{pid}_i^s$ *it checks whether* $\Sigma.\texttt{Verify}(pk_j', m|\texttt{sid}_i^s|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$*. If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise it proceeds according to the specification of the executed operation of* P *upon receiving* $U_j|m$*.*
- *After* Π_i^s *computes the session group key* k_i^s *in the executed operation of* P *it accepts.*

There is one important difference between C-A and the compiler proposed by Katz and Yung. In our version we do not use sequence numbers as additional parameters for the sent messages. The compiler in [112] assumes that each sent message is of the form $U_i|t|m$ where t is a sequence number which starts with 0 and is incremented each time Π_i^s oracle sends a new message. Before any received message is processed by the original protocol P the compiler checks whether this message is expected or not with respect to the next expected sequence number. In our compiler we omit sequence numbers due to several reasons.

First, we assume that any GKE protocol which is secure against passive adversaries simply fails to process a message which is unexpected according to its natural specification. This is because in our model a passive adversary can drop messages and deliver them out of order whereas in the KY model [112] a passive adversary can only eavesdrop the protocol execution. In the following we show that consideration of a stronger passive adversary is in fact even necessary to talk about generic security of the Katz and Yung's compiler.

We consider the following version of the Diffie-Hellman key exchange protocol [79] between two oracles Π_1^s and Π_1^s. Note that although the following construction is artificial and would not be used in practice, it is still sufficient to illustrate that the above compiler is not really generic. Let g be a generator of the multiplicative group $\mathbb{G}$ of the prime order q in which the discrete logarithm problem is assumed to be hard, and let $a \in \mathbb{G}$ be some publicly known value. Assume that after being invoked for the protocol execution Π_1^s and Π_2^s start own timers. The natural specification of the protocol is that Π_1^s chooses own exponent $x_1 \in_R \mathbb{Z}_q^*$ and sends

$X_1 := g^{x_1}$ to Π_2^s. If Π_2^s receives X_1 within some specified time period δ_2 then Π_2^s replies with $X_2 := g^{x_2}$ for some randomly chosen $x_2 \in_R \mathbb{Z}_q^*$ and accepts with the Diffie-Hellman session key $g^{x_1 x_2}$. Similar if Π_1^s receives X_2 within some specified time period δ_1 then it accepts with $g^{x_1 x_2}$ too. However, if neither X_1 nor X_2 are received by Π_2^s and Π_1^s within δ_2 and δ_1, respectively, then both participants accept with a. Obviously, if the passive adversary is restricted to the eavesdropping only (as in the KY model) then the above protocol remains secure since X_1 and X_2 will always be delivered to Π_2^s and Π_1^s, respectively, so that both participants accept with the Diffie-Hellman key. However, an active adversary can drop both messages so that both participants accept with a which is also known to the adversary who can then easily answer own $Test$ query. In fact applying the original KY compiler to this protocol would not provide the required security since in the original protocol failures in the delivery of messages do not necessarily imply that both participants abort. To the contrary, in our model (with a stronger passive adversary) the above protocol would be already treated as insecure against passive adversaries.

Note also that in the Katz and Yung's compiler the sequence numbering restarts for each new protocol operation. Thus sequence numbers do not provide any additional security advantages for the protocol (e.g., they do not protect against replay attacks). Another reason for omitting sequence numbers is that in order to check whether a message is expected or not the compiler must explicitly know the total required number of messages for each protocol operation. Thus, these numbers should be additionally given as input to the compiler. This additional effort must be applied for each GKE protocol to be used with the compiler. Thus, compiler has to be configured each time a different GKE protocol is used. This can be considered as an additional inconvenience.

Additionally we note that the adversarial setting focused by the security proofs in [112] is comparable to $(\emptyset, \texttt{wcm})$ and $(\texttt{wfs}, \texttt{wcm-fs})$ using the terminology of our model. In the following we show that the compiled protocol $\texttt{C-A}_\texttt{P}$ is AKE-secure if the original protocol P is KE-secure. In contrast to the original proof in [112] we additionally consider the adversarial settings $(\texttt{wbs}, \texttt{wcm-bs})$ and $(\texttt{sfs}, \texttt{scm})$ claiming that C-A (and the original compiler in [112][1]) provides even stronger security properties since random nonces are chosen for each executed protocol session anew. Note that as mentioned in Section 8.4 we do not consider backward secrecy in the strong corruption model, i.e., $(\texttt{sbs}, \texttt{scm-bs})$. This requirement can be satisfied if the algorithm C-A.INIT is executed prior to each new execution of C-A.A. This, however, contradicts to the assumption that each key pair (sk_i', pk_i') is long-lived.

Theorem 9.3 (AKE-Security of Static $\texttt{C-A}_\texttt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$. For any **static** GKE-α protocol P if Σ is EUF-CMA then $\texttt{C-A}_\texttt{P}$ is AGKE-α, and*

- *if $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-A}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa),$$

- *if $(\alpha, \beta) \in \{(\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-A}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + q_s\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa),$$

where q_s is the total number of executed protocol sessions.

[1] if one forgets about different assumptions about passive adversaries in our model and the KY model

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 3$ and corresponding events $\mathsf{Win}_i^{\mathsf{ake}}$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}A_P}}(\kappa)$. Note that we do not consider $Join^+$ and $Leave^+$ queries in this proof since $\mathtt{P}$ is assumed to be static. Adversarial queries are answered by a simulator Δ. The (classical) idea behind the proof is to incrementally add changes to the game $\mathbf{G}_i$ in the definition of $\mathbf{G}_{i+1}$ and then relate the probabilities of the events $\mathsf{Win}_i^{\mathsf{ake}}$ and $\mathsf{Win}_{i+1}^{\mathsf{ake}}$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}A_P}}(\kappa)$ (see Definition 8.12) where Δ answers all queries of an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π_i^s. Keep in mind that on the test query the adversary receives either a random string or a session group key k_i^s.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulator fails and sets b' at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|m|\sigma$ such that σ is a valid signature on m that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery; such event is denoted Forge. Hence,

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_0^{\mathsf{ake}}]| \leq \Pr[\mathsf{Forge}]. \tag{9.1}$$

In order to estimate $\Pr[\mathsf{Forge}]$ we show that using $\mathcal{A}$ we can construct a EUF-CMA forger $\mathcal{F}$ against the signature scheme Σ as follows. $\mathcal{F}$ is given a public key pk and has access to the corresponding signing oracle. During the initialization of $\mathtt{C\text{-}A_P}$, $\mathcal{F}$ chooses uniformly at random a user $U_{i^*} \in \mathcal{U}$ and defines $pk'_{i^*} := pk$. All other key pairs, i.e., (sk'_i, pk'_i) for every $U_{i \neq i^*} \in \mathcal{U}$ are generated honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$. $\mathcal{F}$ generates also all key pairs (sk_i, pk_i) with $U_i \in \mathcal{U}$ if any are needed for the original execution of $\mathtt{P}$. The forger simulates all queries of $\mathcal{A}$ in a natural way by executing $\mathtt{C\text{-}A_P}$, and by obtaining the necessary signatures with respect to pk'_{i^*} from its signing oracle. This is a perfect simulation for $\mathcal{A}$ since by assumption no $Corrupt(U_{i^*})$ may occur (otherwise $\mathcal{F}$ would not be able to answer it). Assuming Forge occurs, $\mathcal{A}$ outputs a new valid message/signature pair with respect to some pk'_i; since i^* was randomly chosen and the simulation is perfect, $\Pr[i = i^*] = 1/N$. In that case $\mathcal{F}$ outputs this pair as its forgery. Its success probability is given by $\Pr[\mathsf{Forge}]/N$. This implies

$$\Pr[\mathsf{Forge}] \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa). \tag{9.2}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if an A nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. We call this event RepA. If q_s is the total number of protocol sessions, the probability that a randomly chosen A nonce r_i appears twice is $q_\mathsf{s}^2/2^\kappa$ for one particular user. Since there are at most N users we obtain

$$|\Pr[\mathsf{Win}_2^{\mathsf{ake}}] - \Pr[\mathsf{Win}_1^{\mathsf{ake}}]| \leq \Pr[\mathsf{RepA}] \leq \frac{Nq_\mathsf{s}^2}{2^\kappa} \tag{9.3}$$

This game implies that $\mathtt{sid}_i^s$ computed by any uncorrupted user's oracle Π_i^s remains unique for each new session. Note that $\mathtt{sid}_i^s$ is used to generate signatures in the $\mathtt{A}$ protocol of the compiler. Thus, this game prevents any replay attacks of $\mathcal{A}$.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q_\mathsf{s}^* \in [1, q_\mathsf{s}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q_s^*-th session then the simulation fails and bit b' is set at random. Let Q be the event that this guess is correct. Obviously, $\Pr[\mathsf{Q}] = 1/q_\mathsf{s}$. Thus, we get

$$\begin{aligned}\Pr[\mathsf{Win}_3^{\mathsf{ake}}] &= \Pr[\mathsf{Win}_3^{\mathsf{ake}} \wedge \mathsf{Q}] + \Pr[\mathsf{Win}_3^{\mathsf{ake}} \wedge \neg\mathsf{Q}] \\ &= \Pr[\mathsf{Win}_3^{\mathsf{ake}}|\mathsf{Q}]\Pr[\mathsf{Q}] + \Pr[\mathsf{Win}_3^{\mathsf{ake}}|\neg\mathsf{Q}]\Pr[\neg\mathsf{Q}] \\ &= \Pr[\mathsf{Win}_2^{\mathsf{ake}}]\frac{1}{q_\mathsf{s}} + \frac{1}{2}\left(1 - \frac{1}{q_\mathsf{s}}\right).\end{aligned}$$

This implies

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_\mathsf{s}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2}. \tag{9.4}$$

Having excluded forgeries and replay attacks we show that the probability of $\mathcal{A}$ to win in this game is upper-bounded by the probability of a passive adversary $\mathcal{A}'$ to win in $\mathsf{Game}^{\mathsf{ke}-b}_{\alpha,\beta,\mathtt{P}}(\kappa)$. In the following we describe the construction of $\mathcal{A}'$ which is similar to the one given in [112]. We focus on the construction w.r.t. the adversarial settings $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$ and specify a simpler construction for the settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ at the end of the proof. Note that in case of $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$ the active adversary $\mathcal{A}$ is given access to the $Corrupt$ query and according to Remark 8.10 can actively participate in the protocol execution via $Send$ queries in any session which is not α-fresh. Thus, if the guess of Δ is correct then no active participation of $\mathcal{A}$ in the q^*_s-th session is possible. Also none of the oracles participating in the q^*_s-th session can be asked for a $RevealState$ query, and after these oracles have accepted none of them cannot be asked for a $RevealKey$ query either. With these observations in mind we construct the passive adversary $\mathcal{A}'$ as follows.

Upon initialization $\mathcal{A}'$ corrupts every user $U_i \in \mathcal{U}$ to obtain the long-lived key pair (sk_i, pk_i) used in the original protocol $\mathtt{P}$ (if any such keys are defined). Then, $\mathcal{A}'$ generates all key pairs (sk'_i, pk'_i) honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk'_i, pk_i\}_{U_i \in \mathcal{U}}$. $\mathcal{A}'$ initializes the list $TList$ and runs $\mathcal{A}$ as a subroutine. The idea of the reduction is that in all sessions except for the q^*_s-th session $\mathcal{A}'$ executes the operation of $\mathtt{C\text{-}A_P}$ itself, whereas in the q^*_s-th session $\mathcal{A}'$ asks own $Setup$ query to obtain a transcript T for the operation execution of $\mathtt{P}$ which it extends to a transcript T' for the simulated operation execution of $\mathtt{C\text{-}A_P}$. An entry $(\mathtt{sid}, \perp)$ is saved in $TList$ for every session processed directly by $\mathcal{A}'$ whereas $(\mathtt{sid}, T')$ is saved for the q^*_s-th session. In both cases $\mathtt{sid}$ specifies the unique session id used in that session. We need also to consider that $\mathcal{A}$ can invoke the q^*_s-th session either via a $Setup$ or an appropriate $Send$ query. The queries of $\mathcal{A}$ are answered by $\mathcal{A}'$ as follows.

$Setup$ queries: If $\mathcal{A}$ invokes a protocol session via a $Setup(\mathcal{S})$ query and the invoked session is not the q^*_s-th session then $\mathcal{A}'$ executes $\mathtt{C\text{-}A_P}$ itself and saves $(\mathtt{sid}, \perp)$ in $TList$ where $\mathtt{sid}$ is the unique session id built by $\mathcal{A}$ for that session. $\mathcal{A}'$ can simulate the operation execution efficiently because it knows secret keys of all users. If the invoked session is the q^*_s-th session then $\mathcal{A}'$ forwards the received $Setup(\mathcal{S})$ query as its own query and obtains a transcript T for the execution of $\mathtt{P.Setup}$ between the oracles Π^s_i in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{S}$. The goal of $\mathcal{A}'$ is to extend T to a transcript T' for the corresponding execution of $\mathtt{C\text{-}A_P.Setup}$. Therefore, $\mathcal{A}'$ chooses a random nonce $r_i \in_R \{0,1\}^\kappa$ for each oracle $\Pi^s_i \in \mathcal{G}$ and specifies $\{U_i|r_i\}_{1\le i\le n}$ as the initial messages in T'. $\mathcal{A}'$ also builds the corresponding q^*_s-th session id $\mathtt{sid} := r_1|\ldots|r_n$. Furthermore, for each successive message $U_i|m$ in T the passive adversary $\mathcal{A}'$ computes a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, m|\mathtt{sid}|\mathtt{pid})$ and appends the modified message $U_i|m|\sigma_i$ to T'. $\mathcal{A}'$ saves $(\mathtt{sid}, T')$ in $TList$ and gives T' to $\mathcal{A}$.

$Send$ queries: Since $\mathtt{P}$ is static we are concerned only with the queries of the form $Send('\mathtt{setup}', \Pi^s_i, m)$. By $Send_0$ we define the query of the form $Send('\mathtt{setup}', \Pi^s_i, \mathcal{S})$ which invokes a new

operation execution for Π_i^s. Consequently the second $Send$ query to the same oracle Π_i^s should include messages of the form $U_j|r_j$ for each $U_{j\neq i}$ who holds an oracle in $\mathcal{S}$. We denote such second query as $Send_1(\texttt{'setup'}, \Pi_i^s, (U_1|r_1)|\ldots|(U_n|r_n))$ whereby each U_j that is part of $(U_j|r_j)$ is different from U_i. Note that Π_i^s must have received a $Send_0$ query before in order to be able to answer the $Send_1$ query.

On any $Send_0$ query asked to an oracle Π_i^s the adversary $\mathcal{A}'$ chooses a random nonce $r_i \in \{0,1\}^\kappa$ and answers with $U_i|r_i$.

If a $Send_1$ query is asked to an oracle Π_i^s in a session which is different from q_s^* then $\mathcal{A}'$ computes the session id $\texttt{sid}_i^s$ using nonces from the received query while the nonce r_i is already known after the previous $Send_0$ query, saves $(\texttt{sid}_i^s, \perp)$ in $TList$, and replies by executing the next step of the compiled protocol C-$\mathtt{A_P}$.Setup itself. Note that if no $Send_0$ query was previously asked to Π_i^s then $\mathcal{A}'$ replies with an empty string since the $Send_1$ query is unexpected.

If Π_i^s receives a $Send_1$ query in the q_s^*-th session then $\mathcal{A}'$ computes the session id $\texttt{sid}_i^s$ using nonces from the received query while the nonce r_i is already known after the previous $Send_0$ query and looks in $TList$ for the entry of the form $(\texttt{sid}_i^s, T')$. If such an entry exists then $\mathcal{A}'$ takes the appropriate response message from T' and gives it to $\mathcal{A}$. This means that $\mathcal{A}'$ has already asked own $Setup$ query in the q_s^*-th session and saved the obtained transcript in the "patched" form in $TList$. Note that in the q_s^*-th session $\mathcal{A}'$ is restricted to the actions of a passive adversary as mentioned in Remark 8.10. If no entry of the form $(\texttt{sid}_i^s, T')$ exists then $\mathcal{A}'$ asks own $Setup(\mathcal{S})$ query where $\mathcal{S}$ is composed of the unused oracles of the users whose identities are part of the $Send_1$ query, and obtains the transcript T of the operation execution of P.Setup. Similar to the description of the $Setup$ query for the q_s^*-th session above $\mathcal{A}'$ "patches" the transcript T with digital signatures to obtain the transcript T' for the corresponding operation execution of C-$\mathtt{A_P}$.Setup, saves $(\texttt{sid}_i^s, T')$ in $TList$, and replies to $\mathcal{A}$ with the appropriate message taken from T'.

On any other valid $Send$ query to an oracle Π_i^s the adversary $\mathcal{A}'$ looks in $TList$ for the entry of the form $(\texttt{sid}_i^s, T^*)$. Such an entry must exist since $TList$ contains such pairs for all session ids of the previously invoked sessions; otherwise the query cannot be valid due to the uniqueness of the session ids. If $\mathcal{A}'$ executes the operation for Π_i^s itself then $T^* = \perp$ must hold. In this case $\mathcal{A}'$ executes the next step of C-$\mathtt{A_P}$.Setup and replies accordingly. Otherwise, $\mathcal{A}'$ finds the appropriate message $U_i|m|\sigma_i$ in $T^* = T'$ and gives it to $\mathcal{A}$. Note that if $T^* \neq \perp$ then $T^* = T'$ must hold whereby T' corresponds to the "patched" transcript saved during the processing of the $Send_1$ query for the q_s^*-th session.

$Corrupt$ queries: If $\mathcal{A}$ asks a query of the form $Corrupt(U_i)$ then $\mathcal{A}'$ replies with (sk_i, sk_i).

$RevealState$ queries: If $\mathcal{A}$ asks a query of the form $RevealState(\Pi_i^s)$ then $\mathcal{A}'$ finds an entry $(\texttt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ it means that $\mathcal{A}'$ executes the protocol itself and is, therefore, able to answer this query directly. If $T^* = T'$ then $\mathcal{A}'$ asks its own $RevealState(\Pi_i^s)$ query and responds with whatever it obtains. We stress that this query is only available in the adversarial setting $(\texttt{sfs}, \texttt{scm})$.

$RevealKey$ queries: If $\mathcal{A}$ asks a query of the form $RevealKey(\Pi_i^s)$ then $\mathcal{A}'$ checks that Π_i^s has accepted; otherwise an empty string is returned. Next, $\mathcal{A}'$ finds an entry $(\texttt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ then $\mathcal{A}'$ is able to answer with k_i^s directly since the protocol execution with Π_i^s has been done by $\mathcal{A}'$. If $T^* = T'$ then the query is invalid since no $RevealKey$ queries are allowed to the oracles that have accepted in the q_s^*-th session.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. Therefore, $\mathcal{A}'$ forwards this query to an oracle activated by $\mathcal{A}'$ via its $Setup$ query and replies accordingly.

Since no forgeries and replay attacks occur in this game the described behavior of the passive adversary $\mathcal{A}'$ represents a perfect simulation for the active adversary $\mathcal{A}$ in case that $(\alpha, \beta) \in \{(\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$. Therefore, we get

$$\Pr[\mathsf{Win}_3^{\mathsf{ake}}] = \Pr[\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}-b}(\kappa) = b] \tag{9.5}$$

Considering Equations 9.2 to 9.5 we get

$$\begin{aligned}\Pr[\mathsf{Game}_{\alpha,\beta,\mathtt{C\text{-}A_P}}^{\mathsf{ake}-b}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\ &\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\ &= N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + q_{\mathsf{s}}\left(\Pr[\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}-b}(\kappa) = b] - \frac{1}{2}\right) + \frac{1}{2}\end{aligned}$$

This results in the desired inequality for the case $(\alpha, \beta) \in \{(\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$

$$\mathsf{Adv}_{\alpha,\beta,\mathtt{C\text{-}A_P}}^{\mathsf{ake}}(\kappa) \leq 2N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + q_{\mathsf{s}}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}}(\kappa).$$

On the other hand, if $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs})\}$ then $\mathcal{A}$ does not have access to the $Corrupt$ queries. Having excluded forgeries and replay attacks, the construction of a passive adversary $\mathcal{A}'$ in this case is much simpler since $\mathcal{A}'$ can answer all $Send$ queries from the predefined "patched" transcripts. More precisely, when $\mathcal{A}'$ asks for a $Setup(\mathcal{S})$ query $\mathcal{A}'$ first forwards it to obtain a transcript T of the execution of $\mathtt{P.Setup}$. Next, $\mathcal{A}'$ chooses random nonces $r_i \in_R \{0,1\}^{\kappa}$ for every oracle Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{G}$, computes $\texttt{sid}$ and specifies $\{U_i|r_i\}_{1\leq i\leq n}$ as the initial messages in T'. Then, for each successive message $U_i|m$ in T the passive adversary $\mathcal{A}'$ computes a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', m|\texttt{sid}|\texttt{pid})$ and appends the modified message $U_i|m|\sigma_i$ to T'. Finally, $\mathcal{A}'$ saves $(\texttt{sid}, T')$ in $TList$ and gives T' to $\mathcal{A}$.

In response to any $Send_0$ query for Π_i^s $\mathcal{A}'$ chooses a random nonce $r_i \in_R \{0,1\}\kappa$ and outputs $U_i|r_i$. If $\mathcal{A}$ asks a $Send_1$ query to an oracle Π_i^s then $\mathcal{A}'$ computes $\texttt{sid}_i^s$ whereby r_i is already known after the previous $Send_0$ query. Next, $\mathcal{A}'$ looks for an entry $(\texttt{sid}_i^s, T')$ in $TList$. If this entry exists then $\mathcal{A}'$ replies with the appropriate message from T'. Otherwise, $\mathcal{A}'$ asks own $Setup(\mathcal{S})$ query where $\mathcal{S}$ is composed of the unused oracles of the users whose identities are part of the $Send_1$ query, and obtains the transcript T of the execution of $\mathtt{P.Setup}$. Similar to the description of the $Setup$ query for the q_{s}^*-th session above $\mathcal{A}'$ "patches" the transcript T with digital signatures to obtain the transcript T' for the corresponding execution of $\mathtt{C\text{-}A_P.Setup}$, saves $(\texttt{sid}_i^s, T')$ in $TList$, and replies to $\mathcal{A}$ with the appropriate message taken from T'.

Al other $Send$ queries are answered from the predefined transcripts T' saved in $TList$.

All other queries, i.e., $RevealState$ (only in the setting $(\texttt{wbs}, \texttt{wcm-bs})$), $RevealKey$, and $Test$ asked by $\mathcal{A}$ are forwarded by $\mathcal{A}'$ as own queries to the appropriate oracles activated via its $Setup$ queries, and answered accordingly.

Furthermore, $\mathcal{A}'$ does not need to guess the q_{s}^*-th session in which the $Test$ query is asked because $\mathcal{A}'$ never executes protocol operations itself. Therefore, by omitting Equation 9.4 we obtain for the case $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs})\}$

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-A}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa).$$

□

The construction of the passive adversary $\mathcal{A}'$ given in the above proof uses the fact that any two different sessions of the static GKE protocol P use independent ephemeral secrets. Therefore, in all sessions except for the q_{s}^*-th session $\mathcal{A}'$ can execute the Setup operation of P itself and use its own $Setup$ query to obtain a transcript for the execution during the q_{s}^*-th session. However, when considering dynamic GKE protocols this approach fails since the q_{s}^*-th session may be also invoked for the operation $\texttt{Join}^+$ or $\texttt{Leave}^+$ which would depend on the previously executed operation. For example, in the $(q_{\mathsf{s}}^* - 1)$-th session $\mathcal{A}$ can participate on behalf of an oracle Π_i^s for some corrupted user U_i and invoke the q_{s}^*-th session as operation $\texttt{Leave}^+$ which should remove Π_i^s from the current group $\mathcal{G}$ such that all remaining oracles intended for the participation in the q_{s}^*-th session are α-fresh. Obviously, if $\mathcal{A}'$ would perform the operation execution in the $(q_{\mathsf{s}}^* - 1)$-th session itself then it cannot ask $Leave^+$ query in the q_{s}^*-th session. Nevertheless, we believe that C-A$_\texttt{P}$ preserves α-freshness of a dynamic (GKE-α) protocol due to the following arguments. First, for the adversarial setting $(\emptyset, \texttt{wcm})$ where no $Corrupt$ are allowed the construction of a passive adversary $\mathcal{A}'$ can be done similar to Theorem 9.3, that is by answering all queries of $\mathcal{A}$ from the predefined transcripts obtained via passive $Setup$, $Join^+$, and $Leave^+$ queries. Beside that C-A does not use any own ephemeral secrets, i.e., random nonces used to prevent replay attacks are public and chosen anew for every invoked operation of P. Therefore, we state the following conjecture for dynamic GKE protocols whereby excluding the adversarial setting $(\texttt{wbs}, \texttt{wcm-bs})$ as a consequence of Conjecture 8.31.

Conjecture 9.4 (AKE-Security of Dynamic C-A$_\texttt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \texttt{wcm}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$. For any **dynamic** GKE-α protocol P if Σ is EUF-CMA then C-A$_\texttt{P}$ is AGKE-α, and*

- *if $(\alpha, \beta) = (\emptyset, \texttt{wcm})$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-A}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa),$$

- *if $(\alpha, \beta) \in \{(\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-A}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

9.4 Compiler for MA-Security

As noted in Section 8.2.9 MA-security assures each protocol participant that all legitimate participants have computed the same session group key. However, sometimes this requirement is not needed if participants may notice the difference of computed group keys during the application. For example, an application of encrypted group communication may require from each party to send some encrypted test message to all participants. Obviously, if one of the parties is not able to properly decrypt the test message then not all parties hold the same group key. Therefore, it is reasonable to specify a compiler which can add MA-security to a GKE protocol.

Before proposing our compiler we briefly describe the compiler proposed by Katz and Shin [111]. Their compiler can be used to turn any AKE-secure GKE protocol into a GKE protocol which provides *security against insider attacks*. As already mentioned in Section 6.2.9 security against insider attacks subsumes two requirements: security against insider impersonation attacks and agreement. A protocol is said to be *secure against insider impersonation attacks* if there exists a user U_j and an oracle Π_i^s such that for any PPT adversary $\mathcal{A}$ the probability that $\mathcal{A}$ impersonates U_j to Π_i^s and neither U_j nor U_i are corrupted before Π_i^s accepts is negligible. The notion of *agreement* states that there exists no PPT adversary such that during the execution of the GKE protocol there are two oracles, Π_i^s and $\Pi_j^{s'}$, which are partnered and neither U_i nor U_j are corrupted but Π_i^s and $\Pi_j^{s'}$ have accepted with different session keys. From the definitional point of view security against insider attacks in [111] is related to our definition of MA-security. In the following we describe Katz and Shin's compiler using our notations for consistency.

Definition 9.5 (Compiler for Security against Insider Attacks by Katz and Shin [111]). *Let* $\mathtt{P}$ *be a GKE protocol from Definition 8.4,* $F := \{f_k\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ *a function ensemble with domain and range* $\{0,1\}^\kappa$*, and* $\mathtt{sid}_i^{s_i}$ *is a unique session id. A* compiler for security against insider attacks *consists of an initialization algorithm and a protocol defined as follows:*

`Initialization`*: In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk_i', pk_i') *using* $\Sigma.\mathtt{Gen}(1^\kappa)$*. This is in addition to any key pair* (sk_i, pk_i) *used in* $\mathtt{P}$*.*

The `protocol`*: After an oracle* Π_i^s *accepts with* $(k_i^s, \mathtt{pid}_i^s, \mathtt{sid}_i^s)$ *in* $\mathtt{P}$ *it computes* $\mu_i := f_{k_i^s}(v_0)$ *where* v_0 *is a constant public value and* $K_i^s := f_{k_i^s}(v_1)$ *where* $v_1 \neq v_0$ *is another constant public value. Next,* Π_i^s *erases its private information from* $\mathtt{state}_i^s$ *except for* K_i^s*. Then,* Π_i^s *computes a signature* $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mathtt{sid}_i^s|\mathtt{pid}_i^s|\mu_i)$ *and sends* $U_i|\sigma_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$*.*
After Π_i^s *receives* $U_j|\sigma_j$ *from* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it checks whether* $\Sigma.\mathtt{Verify}(pk_j', \mathtt{sid}_i^s|\mathtt{pid}_i^s|\mu_i, \sigma_j) \stackrel{?}{=} 1$*. If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it accepts with the session group key* K_i^s*.*

Note that Katz and Shin describe their compiler in the Universal Composability (UC)framework, thus considering composition of a GKE protocol with some higher-level application protocol. Therefore, they assume that unique session ids $\mathtt{sid}s$ are already provided by that application protocol. Note that this assumption is general for all protocols described within the UC framework. Nevertheless, the question which immediately arises is whether this protocol remains secure in case that no unique session ids are available? Obviously, leaving out session ids in the above compiler would allow replay attacks. In the following we investigate what consequences do these replay attacks have on the security of the compiled protocol.

On importance of unique session ids in Katz and Shin's compiler

In the following we show that if no unique session ids are available then there exists an adversary $\mathcal{A}$ that can enforce two participating oracles of uncorrupted members U_i and U_j accepting with different session group keys. The attack works as follows: $\mathcal{A}$ corrupts $n-2$ protocol participants in one of the previous sessions and behaves in that session honestly according to the specification of the protocol. Obviously, the adversary learns the key $\bar{k}$ returned by $\mathtt{P}$ in that session and the message $U_i|\bar{\sigma}_i$ sent by Π_i^s during the compiler round of that session. Note that $\bar{\sigma}_i$ was computed amongst other values on $\bar{\mu}_i = f_{\bar{k}}(v_0)$. After the execution of $\mathtt{P}'$ is completed

$\mathcal{A}$ invokes a new session with the same protocol participants (pid). It is legitimate to assume that $\mathcal{A}$ can influence $\Pi_j^{s'}$ to compute $k_j^{s'} = \bar{k}$ and $\Pi_i^{s'}$ to compute a different key $k_i^{s'} \neq \bar{k}$ with non-negligible probability. Then $\mathcal{A}$ intercepts and drops the original message $U_i|\sigma_i$ with σ_i on $\mu_i = f_{k_i^{s'}}(v_0)$ in the compiler round and sends $U_i|\bar{\sigma}_i$ to $\Pi_j^{s'}$ instead (as part of its $Send$ query). Obviously,

$$\mu_j = f_{k_j^{s'}}(v_0) = f_{\bar{k}}(v_0) = \bar{\mu}_i$$

holds so that $\Pi_j^{s'}$ verifies $\bar{\sigma}_i$ successfully but $k_j^{s'} \neq k_i^{s'}$ resulting in $K_j^{s'} \neq K_i^{s'}$. Thus uncorrupted oracles $\Pi_i^{s'}$ and $\Pi_j^{s'}$ accept with different session group keys, i.e., $\mathcal{A}$ succeeds.

Considering this attack it is reasonable to have a compiler which provides MA-security for any AKE-secure GKE protocol without relying on unique session ids given by another applications. Our compiler proposed in the following satisfies these requirements and computes session ids from nonces that prevent replay attacks. Obviously, this requires an additional communication round. Note that if unique session ids are provided by the application then omitting the first communication round in our compiler results in the original compiler from [111].

Definition 9.6 (Compiler for MA-Security C-MA). *Let P be a GKE protocol from Definition 8.4,* $F := \{f_k\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ *a function ensemble with domain and range* $\{0,1\}^\kappa$. *A* compiler for MA-security, *denoted C-MA, consists of an algorithm INIT and a protocol MA defined as follows:*

INIT*: In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk_i', pk_i') *using* $\Sigma.\mathtt{Gen}(1^\kappa)$. *This is in addition to any key pair* (sk_i, pk_i) *used in* P.

MA*: After an oracle* $\Pi_i^s \in \mathcal{G}$ *computes* k_i^s *in the execution of* P *it chooses a random* MA nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$.
After Π_i^s *receives* $U_j|r_j$ *from all* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it computes* $\mathtt{sid}_i^s := r_1|\ldots|r_n$, *a* MA token $\mu_i := f_{k_i^s}(v_0)$ *where* v_0 *is a constant public value, a signature* $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ *and sends* $U_i|\sigma_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$.
After Π_i^s *receives* $U_j|\sigma_j$ *from* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it checks whether* $\Sigma.\mathtt{Verify}(pk_j', \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ *holds. If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes the session group key* $K_i^s := f_{k_i^s}(v_1)$ *where* $v_1 \neq v_0$ *is another constant public value, erases every other private information from* $\mathtt{state}_i^s$ *(including* k_i^s*), and accepts with* K_i^s.

The following theorem shows that C-MA provides MA-security for any GKE protocol. A nice side effect of this proof is that the original compiler in [111] satisfies our definition of MA-security which can be, therefore, seen as alternative to the definitions concerning security against insider attacks proposed in [111].

Theorem 9.7 (MA-Security of C-MA$_\mathtt{P}$). *For any GKE protocol* P *if* Σ *is EUF-CMA and* F *is collision-resistant then* C-MA$_\mathtt{P}$ *is* MAGKE, *and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{C\text{-}MA_P}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}_i^{\mathsf{ma}}$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator Δ.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{C\text{-}MA_P}}(\kappa)$ played between Δ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π^s_i accepts with K^s_i and another user $U_j \in \mathtt{pid}^s_i$ that is uncorrupted at the time Π^s_i accepts and either does not have a corresponding oracle Π^s_j with $(\mathtt{pid}^s_j, \mathtt{sid}^s_j) = (\mathtt{pid}^s_i, \mathtt{sid}^s_i)$ or has such an oracle but this oracle accepts with $K^s_j \neq K^s_i$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation aborts if $\mathcal{A}$ asks a $Send$ query on a message $U_i|\sigma$ such that σ is a valid signature that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain,

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_1] - \Pr[\mathsf{Win}^{\mathsf{ma}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) \tag{9.6}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation aborts if a MA nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. Similar to Equation 9.3 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_2] - \Pr[\mathsf{Win}^{\mathsf{ma}}_1]| \leq \frac{Nq^2_{\mathsf{s}}}{2^{\kappa}} \tag{9.7}$$

Note that this prevents attacks where Π^s_i during any session of the MA protocol receives a replayed message of the form $U_j|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π^s_i does not accept unless it successfully verifies σ_j for all $U_j \in \mathtt{pid}^s_i$ in the MA protocol of C-MA. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}^s_i$ that is uncorrupted at the time Π^s_i accepts there exists a corresponding instance oracle Π^s_j with $(\mathtt{pid}^s_j, \mathtt{sid}^s_j) = (\mathtt{pid}^s_i, \mathtt{sid}^s_i)$. Thus, according to Definition 8.15 $\mathcal{A}$ wins in this game only if one of these oracles has accepted with $K^s_j \neq K^s_i$.

Assume that $\mathcal{A}$ wins in this game. Then the oracles Π^s_i and Π^s_j have accepted with $K^s_i = f_{k^s_i}(v_1)$ and $K^s_j = f_{k^s_j}(v_1)$ where k^s_i and k^s_j are corresponding keys computed during the execution of P, respectively. Having excluded forgeries and replay attacks on the messages exchanged between any two uncorrupted users' oracles we may assume in the following that Π^s_j has received in the MA protocol the original message $U_i|\sigma_i$ with σ_i computed on $\mu_i = f_{k^s_i}(v_0)$ and Π^s_i has received the original message $U_j|\sigma_j$ with σ_j computed on $\mu_j = f_{k^s_j}(v_0)$. Since both oracles have accepted (thus, signature verifications are successful) we follow that $f_{k^s_i}(v_0) = f_{k^s_j}(v_0)$ holds. Hence, the probability that $\mathcal{A}$ wins in this game (since there are at most q_{s} sessions) is given by

$$\Pr[K^s_i \neq K^s_j \wedge f_{k^s_i}(v_0) = f_{k^s_j}(v_0)] = \Pr[f_{k^s_i}(v_1) \neq f_{k^s_j}(v_1) \wedge f_{k^s_i}(v_0) = f_{k^s_j}(v_0)] \leq q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa).$$

Hence,

$$\Pr[\mathsf{Win}^{\mathsf{ma}}_2] \leq q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa). \tag{9.8}$$

Considering Equations 9.6 to 9.8 we get the desired inequality

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{C\text{-}MA_P}}(\kappa) = \Pr[\mathsf{Win}^{\mathsf{ma}}_0] \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq^2_{\mathsf{s}}}{2^{\kappa}} + q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa).$$

□

Additionally, we need to show that C-MA also preserves (A)KE-security of the original protocol. Note that messages exchanged between the oracles in the MA protocol are used to verify that shared keys already computed during the original protocol P are equal. In contrast to the AKE-security proof of C-A we are able to give a formal proof for both static and dynamic GKE protocols.

Theorem 9.8 (AKE-Security of $\texttt{C-MA}_\texttt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs})^2, (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm})\}$. For any $\texttt{AGKE-}\alpha$ protocol P if Σ is EUF-CMA and F is pseudo-random then $\texttt{C-MA}_\texttt{P}$ is $\texttt{AGKE-}\alpha$, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-MA}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{P}}(\kappa) + 4q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 6$ and corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-MA}_\texttt{P}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-MA}_\texttt{P}}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. We assume that the $Test$ query is asked to an α-fresh oracle Π^s_i. Keep in mind that on the $Test$ query the adversary receives either a random string or a session group key $K^s_i := f_{k^s_i}(v_0)$ where k^s_i is the session group key computed in P.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulator fails and sets b' at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \Pr[\mathsf{Win}^{\mathsf{ake}}_0]| \leq N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa). \tag{9.9}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulator fails and sets b' at random if a MA nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. According to Equation 9.3 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_2] - \Pr[\mathsf{Win}^{\mathsf{ake}}_1]| \leq \frac{Nq_\mathsf{s}^2}{2^\kappa}. \tag{9.10}$$

This game implies that $\texttt{sid}^s_i$ computed by any uncorrupted user's oracle Π^s_i remains unique for each new session. Note that $\texttt{sid}^s_i$ is used to generate signatures in the MA protocol of the compiler. This prevents any replay attacks of $\mathcal{A}$.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q^*_\mathsf{s} \in [1, q_\mathsf{s}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q^*_s-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}^{\mathsf{ake}}_2] = q_\mathsf{s}\left(\Pr[\mathsf{Win}^{\mathsf{ake}}_3] - \frac{1}{2}\right) + \frac{1}{2}. \tag{9.11}$$

Game $\mathbf{G}_4$. In this game we consider the simulator Δ as an active adversary against the AKE-security of P that participates in $\mathsf{Game}^{\mathsf{ake}-1}_{\alpha,\beta,\texttt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh

[2] Only in case of static protocols due to Conjecture 8.31.

oracle Π_i^s is answered with the real session group key k_i^s computed in P. In the following we show how Δ answers the queries of $\mathcal{A}$. Note that Δ and $\mathcal{A}$ operate in the same adversarial setting (α, β) and are both active. This is the main difference to the reduction described in the proof of Theorem 9.3. In fact, we never require Δ to execute operations of P itself but to forward each related query of $\mathcal{A}$ as its own query and respond accordingly. Δ itself performs only those additional computations that are necessary for C-MA.

The simulator Δ which is initialized with the public keys $\{pk_i'\}_{U_i \in \mathcal{U}}$ (if any are given in the original protocol P) generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. Then, Δ runs $\mathcal{A}$ as a subroutine and answers its queries.

$Setup$ queries: If $\mathcal{A}$ invokes a protocol session via a $Setup(\mathcal{S})$ query then Δ forwards this query as its own and obtains the transcript T of the $\mathtt{P.Setup}(\mathcal{S})$ execution. The goal of Δ is to extend T to a transcript T' for the corresponding execution of $\mathtt{C\text{-}MA_P.Setup}(\mathcal{S})$. Therefore, $\mathcal{A}$ chooses random nonces r_i for each Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{S}$ and computes $\mathtt{sid} := r_1|\ldots|r_n$. In order to build T' the simulator appends $\{U_i|r_i\}_{1 \le i \le n}$ to T (without any "patches" in contrast to the reduction in Theorem 9.3). Next, if the invoked session is the q_{s}^*-th session then Δ asks own $Test$ query to any oracle activated via the $Setup$ query and obtains (real) k. Otherwise, if the session is not the q_{s}^*-th session then Δ asks own $RevealKey$ query to any of the mentioned oracles and obtains real k. Hence, in any case Δ knows real k which it then uses to compute the MA token $\mu := f_k(v_0)$. Then, Δ computes $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu|\mathtt{sid}|\mathtt{pid})$ and appends messages of the form $\{U_i|\sigma_i\}_{1 \le i \le n}$ to T'. Then, Δ computes $K := f_k(v_1)$ and gives T' to $\mathcal{A}$.

$Join^+$ and $Leave^+$ queries: These queries are answered similar to the $Setup$ queries so that Δ is able to compute the extended transcript T' as well as K for the operation execution of $\mathtt{C\text{-}MA_P.Join}^+$ or $\mathtt{C\text{-}MA_P.Leave}^+$, respectively.

$Send$ queries: $Send$ queries: By $Send_{S0}$ we define the query of the form $Send('\mathtt{setup}', \Pi_i^s, \mathcal{S})$ which invokes a new execution of $\mathtt{C\text{-}MA_P.Setup}$ for Π_i^s. By $Send_{J0}$ we define the query of the form $Send('\mathtt{join}', \Pi_i^s, \mathcal{G}, \mathcal{J})$ which invokes a new execution of $\mathtt{C\text{-}MA_P.Join}^+$ for Π_i^s. By $Send_{L0}$ we define the query of the form $Send('\mathtt{leave}', \Pi_i^s, \mathcal{G}, \mathcal{L})$ which invokes a new execution of $\mathtt{C\text{-}MA_P.Leave}^+$ for Π_i^s. When $\mathcal{A}$ asks one of these queries to Π_i^s the simulator forwards it as its own query and replies with the received answer. All further $Send$ queries of $\mathcal{A}$ that belong to the operation execution of P are forwarded by Δ as own queries and answered accordingly.

By queries $Send_{SF}$, $Send_{JF}$, and $Send_{LF}$ to an oracle Π_i^s we define the *final* $Send$ queries of $\mathcal{A}$ concerning the execution of $\mathtt{P.Setup}$, $\mathtt{P.Join}^+$, or $\mathtt{P.Leave}^+$, respectively, which results in Π_i^s having computed k_i^s in the operation of P. This means that all further valid $Send$ queries to Π_i^s would be related to the communication rounds of the MA protocol. When $\mathcal{A}$ asks one these final $Send$ queries the simulator forwards it as its own query to an appropriate oracle Π_i^s implying the computation of k_i^s. Similar to the description of the $Setup$, $Join^+$, and $Leave^+$ queries above, Δ asks own $Test$ query (if the received $Send$ query is addressed to some participant of the q_{s}^*-th session) or $RevealKey$ query (in all other sessions) to obtain the real intermediate key k_i^s. Then, Δ chooses $r_i \in_R \{0,1\}^\kappa$ and responds with $U_i|r_i$.

By queries $Send_{SF+}$, $Send_{JF+}$, and $Send_{LF+}$ to an oracle Π_i^s we define the $Send$ queries of $\mathcal{A}$ of the form $Send(op, \Pi_i^s, (U_1|r_1)|\ldots|(U_n|r_n))$ where $op \in \{'\mathtt{setup}', '\mathtt{join}', '\mathtt{leave}'\}$ and n is the (updated) number of operation participants whereby $(U_i|r_i)$ is not part of the query

message. Note that Π_i^s must have received a $Send_{SF}$, $Send_{JF}$, or $Send_{LF}$ query before the corresponding $Send_{SF+}$, $Send_{JF+}$, or $Send_{LF+}$ query; otherwise the query is unexpected. When $\mathcal{A}$ asks one of these $Send$ queries the simulator computes the MA token $\mu_i := f_{k_i^s}(v_0)$, the signature $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ and responds with $U_i|\sigma_i$ to $\mathcal{A}$.

The last $Send$ query to Π_i^s is independent of the concrete operation op and has the form $Send(op, \Pi_i^s, (U_1|\sigma_1)|\ldots|(U_n|\sigma_n))$ whereby $(U_i|r_i)$ is not part of the query message. Δ checks all received signatures and computes $K_i^s := f_{k_i^s}(v_1)$.

$Corrupt$ queries: When $\mathcal{A}$ asks a $Corrupt(U_i)$ query Δ forwards own $Corrupt(U_i)$ query to obtain sk_i and replies with (sk_i, sk_i'). Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $Corrupt$ queries.

$RevealState$ queries: When $\mathcal{A}$ asks a $RevealState(\Pi_i^s)$ query Δ forwards own $RevealState(\Pi_i^s)$ query to obtain $\mathtt{state}_i^s$. If Π_i^s is waiting for the last $Send$ query of the form $Send(op,\Pi_i^s,(U_1|\sigma_1)|\ldots|(U_n|\sigma_n))$ then Δ inserts k_i^s (which is not erased yet) into $\mathtt{state}_i^s$ and returns it to $\mathcal{A}$; otherwise it simply forwards $\mathtt{state}_i^s$ to $\mathcal{A}$. Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $RevealState$ queries.

$RevealKey$ queries: When $\mathcal{A}$ asks a $RevealKey(\Pi_i^s)$ query Δ checks that Π_i^s has accepted; otherwise an empty string is returned. Then, Δ returns the session group key K_i^s. Note that Δ is always able to do this since it executes the last steps of the MA protocol itself, i.e., the computation of μ_i and K_i^s for every (honest) oracle Π_i^s.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. The simulator Δ already knows K_i^s since this value is computed by Δ for every (honest) oracle Π_i^s in the simulation. Thus, Δ chooses a random bit $b \in_R \{0,1\}$ and returns K_i^s if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This is a perfect simulation for $\mathcal{A}$. Since Δ uses the real k_i^s to derive K_i^s we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{9.12}$$

Game $\mathbf{G}_5$. In this game we consider the simulator Δ as an active adversary against the AKE-security of $\mathtt{P}$ that participates in $\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}-0}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}}(\kappa). \tag{9.13}$$

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session K and the MA token μ are replaced by random values. Recall that k used to compute K and μ is uniform according to Game $\mathbf{G}_5$. Obviously,

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.14}$$

Obviously, in this game $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_6^{\mathsf{ake}}] = \frac{1}{2} \tag{9.15}$$

Considering Equations 9.9 to 9.16 we obtain

$$\begin{aligned}\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}MA_P}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\ &= N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\ &\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa) + \frac{1}{2}.\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}MA_P}}(\kappa) \leq 2N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 4q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa).$$

□

Similar result can be obtained for any KE-secure GKE protocol. In this case $\mathcal{A}$ is passive and does not have access to the $Send$ queries. Hence, no forgeries and replay attacks need to be considered.

Theorem 9.9 (KE-Security of $\mathtt{C\text{-}MA_P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs})$[3]$, (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any $\mathtt{GKE}$-α protocol $\mathtt{P}$ if F is pseudo-random then $\mathtt{C\text{-}MA_P}$ is $\mathtt{GKE}$-α, and*

$$\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{C\text{-}MA_P}}(\kappa) \leq 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 4q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

9.5 Compiler for n-Contributiveness

As noted in Section 8.2.10 contributiveness deals with the trust relationship between participants of a GKE protocol such that none of them has any advantage concerning values of computed group keys. Some group applications may not need this kind of trust relationship between its users. Therefore, it is reasonable to specify a compiler which provides n-contributiveness for any GKE protocol and can be used optional if contributiveness is required.

Missing contributiveness in Katz and Shin's compiler

The compiler by Katz and Shin from Definition 9.5 does not provide contributiveness because if a malicious participant (or a subset of them) is able to predict or influence the computation of k_i^s for any uncorrupted oracle Π_i^s in any operation execution of $\mathtt{P}$ then it can also predict the value of K_i^s for the compiled protocol $\mathtt{P}'$ simply by computing $f_{k_i^s}(v_1)$.

One of the core points for successful adversarial actions is that K_i^s is derived based only on the session group key k_i^s and v_1 which is a constant public value known to all participants before the execution of the protocol. A solution to improve the above compiler by requiring that participants agree on v_1 during the protocol execution is not promising since it has to deal with attacks of malicious participants who will try to influence the choice of v_1 and therefore lead

[3] Only in case of static protocols due to Conjecture 8.31.

to non-standard collision requirements for pseudo-random functions in the security proof, i.e., collisions with respect to different inputs $f_k(v) = f_{k'}(v')$ and not only with respect to different keys $f_k(v) = f_{k'}(v)$ as required in Definition 5.5.

In the following we propose a compiler which can add n-contributiveness to any GKE protocol. There are two general ideas used in the construction. First, the group key derivation depends on the information which the participants learn before the execution of the protocol. Additionally, we use some information which is random (unpredictable) for each executed session. Second, a *proper* subset of group members is not be able to influence the randomness of this information.

Definition 9.10 (Compiler for n-Contributiveness C-CON). *Let* P *be a GKE protocol from Definition 8.4,* $\pi : \{0,1\}^\kappa \to \{0,1\}^\kappa$ *a permutation, and* $F := \left\{f_k\right\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ *a function ensemble with domain and range* $\{0,1\}^\kappa$. *A* compiler for n-contributiveness*, denoted* C-CON*, consists of the protocol* CON *defined as follows:*

CON*: After an oracle* $\Pi_i^s \in \mathcal{G}$ *computes* k_i^s *in the execution of* P *it chooses a random* CON nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$. *After* Π_i^s *receives* $U_j|r_j$ *from* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it checks whether* $|r_j| \stackrel{?}{=} \kappa$. *If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes* $\rho_1 := f_{k_i^s \oplus \pi(r_1)}(v_0)$ *and each* $\rho_l := f_{\rho_{l-1}\oplus\pi(r_l)}(v_0)$ *for all* $l = \{2,\ldots,n\}$, *where* v_0 *is a public value, computes the session group key* $K_i^s := \rho_n$, *erases every other private information from* $\mathtt{state}_i^s$ *(including* k_i^s *and all* ρ_l, $l \in [1,n]$*), and accepts with* K_i^s.

In the following proof we argue that possible ability of the adversary $\mathcal{A}$ to influence the session group key k computed in the operations of the original protocol P does not provide $\mathcal{A}$ with any additional advantage in its attack on influencing/predicting the session group key K in the compiled protocol C-CON$_\mathtt{P}$. We use the fact that every honest oracle $\Pi_i^s \in \mathcal{G}$, $i \in [1,n]$ computes the sequence $\rho_1,\ldots,\rho_n$ in order to accept with $K = \rho_n$ so that each ρ_l, $l \in [2,n]$ depends on the previously computed ρ_{l-1}. We consider the probability that $\mathcal{A}$ is able to influence an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$, $i^* \in [1,n]$ to accept some $\tilde{K}$ by considering its ability to influence $\Pi_{i^*}^s$ to compute any value ρ_l, $i^* \le l \le n$. This is equivalent to the ability of $\mathcal{A}$ in the prepare stage to output ρ_l which is then computed by $\Pi_{i^*}^s$ in some attack-ed session. On the other hand, applying Lemma 5.20 in our proof we do also consider the upper-bound for the success probability of the adversary in case that its strategy differs from influencing any value in $\rho_{i^*},\ldots,\rho_n$.

Theorem 9.11 (n-Contributiveness of C-CON$_\mathtt{P}$). *For any GKE protocol* P *if* π *is one-way and* F *is collision-resistant pseudo-random then* C-CON$_\mathtt{P}$ *is* n-CGKE*, and*

$$\mathsf{Succ}^{\mathsf{con}-n}_{\text{C-CON}_\mathtt{P}}(\kappa) \le \frac{Nq_s^2 + Nq_s + q_s}{2^\kappa} + (N+1)q_s\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_s\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_s\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Remark 9.12. Although this proof is put into the formal "sequence of games" framework some parts of it are kept intuitive. The main reason is that no formal reduction when discussing success probabilities in games $\mathbf{G}_2$ and $\mathbf{G}_3$ could be found. The reason is that from the perspective of the adversary all values are known, especially all PRF keys used to compute ρ_l, $l = 1,\ldots,n$. The classical approach where the outputs of a pseudo-random function are replaced by random values, fails here, because the adversary obtaining these values can easily distinguish between

the simulation and the real game. We also remark that in [32] no formal reductions for their (weaker) definition of contributiveness could be given either. It seems that proving this kind of requirements may require some additional techniques.

Proof (partially informal). In the following we consider an adversary $\mathcal{A}$ from Definition 8.18. Assume that $\mathcal{A}$ wins in $\mathsf{Game}^{\mathsf{con}-n}_{\mathsf{C\text{-}CON_P}}(\kappa)$ (which event we denote $\mathsf{Win}^{\mathsf{con}}$). Then at the end of the stage prepare it returned $\tilde{K}$ such that in the stage attack an honest oracle $\Pi^s_{i^*} \in \mathcal{G}$ accepted with $K^s_{i^*} = \tilde{K}$. According to the construction of $K^s_{i^*}$ the equation $\tilde{K} = \rho_n$ must hold. We consider the following games.

Game $\mathbf{G}_0$. This is the real game $\mathsf{Game}^{\mathsf{con}-n}_{\mathsf{C\text{-}CON_P}}(\kappa)$, in which the honest players are replaced by a simulator Δ.

Game $\mathbf{G}_1$. In this game we abort the simulation if the same CON nonce r_i is used by any honest Π^s_i in two different sessions. Considering r_i being uniform for any uncorrupted user's oracle Π^s_i, and since there are at most N users we have

$$\Pr[\mathsf{Win}^{\mathsf{con}}_0] - \Pr[\mathsf{Win}^{\mathsf{con}}_1] \leq \frac{Nq_{\mathsf{s}}^2}{2^\kappa}. \tag{9.16}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_{i^*} computed by $\Pi^s_{i^*}$ in any session of the attack stage.[4] According to Lemma 5.20 we need to estimate the probability of the opposite event, i.e., that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*}. We consider two cases: $i^* = 1$ and $i^* > 1$. Note that all other oracles except for $\Pi^s_{i^*}$ can be corrupted. An important observation for our argumentation in this game is that the random nonce r_{i^*} is chosen by $\Pi^s_{i^*}$ after the computation of $k^s_{i^*}$ in $\mathtt{P}$. In other words, when $\Pi^s_{i^*}$ chooses r_{i^*} the key $k^s_{i^*}$ is already defined and cannot be influenced (changed) any more, thus as soon as the compiler round starts $k^s_{i^*}$ can be considered as some fixed value.

Case $i^* = 1$: In any session of the attack stage honest oracle Π^s_1 computes $\rho_1 := f_{k^s_1 \oplus \pi(r_1)}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $k^s_1 \oplus \pi(r_1)$ (denoted R_1) in the prepare stage $\mathcal{A}$'s probability to output $\rho_1 = f_{R_1}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_1 in the prepare stage and tries to influence Π^s_1 computing exactly this value in some session of the attack stage. Note that as mentioned above k^s_1 is some fixed value at the time point when Π^s_1 chooses r_1 uniformly at random. Since π is a permutation the value $\pi(r_1)$ is uniform and fixed for every session in the attack stage. This implies that R_1 is also uniform and fixed, and unknown to $\mathcal{A}$ in the prepare stage. Hence, $\mathcal{A}$ cannot learn R_1 in the prepare stage better than by a random guess.

Case $i^* > 1$: In any session of the attack stage honest oracle $\Pi^s_{i^*}$ computes $\rho_{i^*} := f_{\rho_{i^*-1} \oplus \pi(r_{i^*})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*-1} \oplus \pi(r_{i^*})$ (denoted R_{i^*}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*} = f_{R_{i^*}}(v_0)$ in that stage is

[4] Note, in $\mathbf{G}_0$ and $\mathbf{G}_1$ the adversary only outputs a value for the resulting group key. In $\mathbf{G}_2$ we consider the additional (in)ability of the adversary to output the value for ρ_{i^*}. Since we are only interested in the probability of the adversarial success under this "condition event" (without changing the game in case that this event occurs; see also Section 5.6.1) the simulator does not need to detect whether $\mathcal{A}$ is able to output the correct value or not. In case that we would define this event as a "failure event" (and apply the corresponding transition type between $\mathbf{G}_1$ and $\mathbf{G}_2$) we would have to modify the original $\mathsf{Game}^{\mathsf{con}-n}_{\mathsf{C\text{-}CON_P}}(\kappa)$ by requiring that $\mathcal{A}$ also outputs a value for ρ_{i^*} so that the simulator is able to detect it since without detection we cannot ensure that $\mathbf{G}_2$ proceeds differently, e.g., aborts. The same considerations are applicable to $\mathbf{G}_3$ w.r.t. $K^s_{i^*}$.

bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*} in the prepare stage and tries to influence $\Pi_{i^*}^s$ computing exactly this value in some session of the attack stage. Since r_{i^*} is chosen by honest $\Pi_{i^*}^s$ at random in every attack-ed session and π is a permutation the value $\pi(r_{i^*})$ is uniform and fixed for each attack-ed session. Hence, the adversary must influence in the attack stage the oracle $\Pi_{i^*}^s$ to compute $\rho_{i^*-1} = R_{i^*} \oplus \pi(r_{i^*})$ which is fixed and uniformly distributed for each attack-ed session. Note that $\mathcal{A}$ learns the required ρ_{i^*-1} only after having received r_{i^*}, that is during the attack-ed session. Since U_{i^*} is uncorrupted its oracle computes ρ_{i^*-1} according to the protocol specification, that is $\rho_{i^*-1} := f_{\rho_{i^*-2} \oplus \pi(r_{i^*-1})}(v_0)$. Having excluded PRF collisions and random guesses we consider the PRF key $\rho_{i^*-2} \oplus \pi(r_{i^*-1})$ as a fixed value unknown to the adversary. The probability that $\mathcal{A}$ recovers it is intuitively bound by $\mathsf{Adv}_F^{\mathsf{prf}}(\kappa)$. This is because any adversary that is able to reveal the PRF key can act as a distinguisher for the pseudo-randomness of f.

Since there are at most q_s sessions we have

$$\Pr[\mathsf{Win}_1^{\mathsf{con}}] - \Pr[\mathsf{Win}_2^{\mathsf{con}}] \leq q_\mathsf{s}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_\mathsf{s}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}. \tag{9.17}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that we do not need to consider the values ρ_l, $l < i^*$ computed by $\Pi_{i^*}^s$ since in order to compute $K_{i^*}^s$ every honest oracle must compute the whole sequence $\rho_1, \ldots, \rho_n$. Thus, it is sufficient to argue that the probability of $\mathcal{A}$ influencing any ρ_l, $l \geq i^*$, computed by $\Pi_{i^*}^s$ in any attack-ed session is negligible.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $K_{i^*}^s = \rho_n$ computed by $\Pi_{i^*}^s$ in any session of the attack stage. Again, the simulator does not need to detect whether this event occurs since both games proceed identical in any case. According to Lemma 5.20 we need to estimate the probability of the opposite event that $\mathcal{A}$ being in the prepare stage is able to output $K_{i^*}^s$.

Based on the "hybrid technique" we define a sequence of auxiliary games $\mathbf{G}'_{3,l}$, $l = i^*, \ldots, n$. Each of these games is identical to the previous one in the sequence with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_l computed by $\Pi_{i^*}^s$ in any session of the attack stage. Obviously, $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$ and $\mathbf{G}'_{3,n} = \mathbf{G}_3$. According to Lemma 5.20 we need to estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_l.

Since $\rho_l := f_{\rho_{l-1} \oplus \pi(r_l)}(v_0)$ for all $l > i^*$ and each r_l is not chosen by $\Pi_{i^*}^s$ each two consecutive auxiliary games have the same difference. Hence, it is sufficient to compute this difference between any two consecutive auxiliary games. In the following we compute the difference between $\mathbf{G}'_{3,i^*+1}$ and $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$. We estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*+1} computed by $\Pi_{i^*}^s$ in any session of the attack stage.

We argue by intuition. Since $\Pi_{i^*}^s$ is honest, in the attack stage it computes $\rho_{i^*+1} := f_{\rho_{i^*} \oplus \pi(r_{i^*+1})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*} \oplus \pi(r_{i^*+1})$ (denoted R_{i^*+1}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*+1} = f_{R_{i^*+1}}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*+1} in the prepare stage and tries to influence $\Pi_{i^*}^s$ computing exactly this value in some session of the attack stage. Recall that $\mathcal{A}$ is allowed to corrupt any user $U_{l \neq i^*}$, and thus choose each nonce r_l, $l \neq i^*$. Since $\mathcal{A}$ learns ρ_{i^*} only in the attack-ed

session (as observed in Game $\mathbf{G}_2$) and having excluded PRF collisions and random guesses the probability that $\mathcal{A}$ is able to influence $\Pi^s_{i^*}$ computing R_{i^*+1} in the attack stage is bound by the probability that in the attack-ed session $\mathcal{A}$ computes the appropriate nonce r_{i^*+1} such that $\pi(r_{i^*+1}) = R_{i^*+1} \oplus \rho_{i^*}$ holds. Since π is one-way this probability is intuitively bound by $\mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa)$. Thus, $\mathcal{A}$ is able to output ρ_{i^*+1} while being in the prepare stage with the probability of at most $\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa) + \mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa) + 1/2^{\kappa}$. Since there are at most q_{s} sessions the total probability that $\mathcal{A}$ is able to do this is at most

$$q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa) + q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa) + \frac{q_{\mathsf{s}}}{2^{\kappa}}.$$

The above sum upper-bounds the difference between $\mathbf{G}'_{3,i^*+1}$ and $\mathbf{G}_2$. Since there are at most N auxiliary games (due to $n \leq N$) we obtain

$$\Pr[\mathsf{Win}^{\mathsf{con}}_{2}] - \Pr[\mathsf{Win}^{\mathsf{con}}_{3}] \leq Nq_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa) + Nq_{\mathsf{s}}\mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa) + \frac{Nq_{\mathsf{s}}}{2^{\kappa}}. \tag{9.18}$$

Since in this game $\mathcal{A}$ is not able to output $K^s_{i^*} = \rho_n$ in the prepare stage it does not output a correct value for $\tilde{K}$. Hence,

$$\Pr[\mathsf{Win}^{\mathsf{con}}_{3}] = 0 \tag{9.19}$$

Considering Equations 9.16 to 9.19 we obtain the desired inequality:

$$\begin{aligned}\mathsf{Succ}^{\mathsf{con}-n}_{\texttt{C-CON}_\texttt{P}}(\kappa) &= \Pr[\mathsf{Win}^{\mathsf{con}}_{0}] \\ &\leq \frac{Nq_{\mathsf{s}}^2 + Nq_{\mathsf{s}} + q_{\mathsf{s}}}{2^{\kappa}} + (N+1)q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa) + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa) + Nq_{\mathsf{s}}\mathsf{Succ}^{\mathsf{ow}}_{\pi}(\kappa).\end{aligned}$$

□

Additionally, we show that C-CON preserves (A)KE-security of the original protocol P. Similar to the proof of Theorem 9.8 we provide a formal proof for both static and dynamic GKE protocols.

Theorem 9.13 (AKE-Security of $\texttt{C-CON}_\texttt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \texttt{wcm}),\ (\texttt{wbs}, \texttt{wcm-bs})$[5]$,\ (\texttt{wfs}, \texttt{wcm-fs}),\ (\texttt{sfs}, \texttt{scm})\}$. For any $\texttt{AGKE-}\alpha$ protocol $\texttt{P}$ if F is pseudo-random then $\texttt{C-CON}_\texttt{P}$ is also $\texttt{AGKE-}\alpha$, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-CON}_\texttt{P}}(\kappa) \leq 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{P}}(\kappa) + 2Nq_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

Proof. Similar to the proof of Theorem 9.3 we define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 3$ and corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-CON}_\texttt{P}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-CON}_\texttt{P}}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. We assume that the $Test$ query is asked to an α-fresh oracle Π^s_i. Keep in mind that on the test query the adversary receives either a random string or a session group key $K^s_i := \rho_n$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ except that the following rule is added: Δ chooses $q^*_{\mathsf{s}} \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query

[5] Only in case of static protocols due to Conjecture 8.31.

$Test$. If this query does not occur in the q_{S}^*-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}_0^{\mathsf{ake}}] = q_{\mathsf{S}} \left(\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \frac{1}{2} \right) + \frac{1}{2}. \tag{9.20}$$

Game $\mathbf{G}_2$. In this game we consider the simulator Δ as an active adversary against the AKE-security of P that participates in $\mathsf{Game}^{\mathsf{ake}-1}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with the real session group key k_i^s computed in P. In the following we show how Δ answers the queries of $\mathcal{A}$. Note that Δ and $\mathcal{A}$ operate in the same adversarial setting (α, β) and are both active. We never require Δ to execute operations of P itself but to forward each related query of $\mathcal{A}$ as its own query and respond accordingly. Δ itself performs only those additional computations that are necessary for C-CON.

The simulator Δ which is initialized with the public keys $\{pk_i'\}_{U_i \in \mathcal{U}}$ (if any are given in the original protocol P) generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. Then, Δ runs $\mathcal{A}$ as a sub-routine and answers its queries.

$Setup$ queries: If $\mathcal{A}$ invokes a protocol session via a $Setup(\mathcal{S})$ query then Δ forwards this query as its own and obtains the transcript T of the $\mathtt{P}.\mathtt{Setup}(\mathcal{S})$ execution. The goal of Δ is to extend T to a transcript T' for the corresponding execution of $\mathtt{C\text{-}CON}_{\mathtt{P}}.\mathtt{Setup}(\mathcal{S})$. Therefore, $\mathcal{A}$ chooses random nonces r_i for each Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{S}$ and computes $\mathtt{sid} := r_1 | \ldots | r_n$. In order to build T' the simulator appends $\{U_i | r_i\}_{1 \le i \le n}$ to T (without any "patches" in contrast to the reduction in Theorem 9.3). Next, if the invoked session is the q_{S}^*-th session then Δ asks own $Test$ query to any oracle activated via the $Setup$ query and obtains (real) k. Otherwise, if the session is not the q_{S}^*-th session then Δ asks own $RevealKey$ query to any of the mentioned oracles and obtains real k. Hence, in any case Δ knows real k which it then uses to compute the sequence $\rho_1, \ldots, \rho_n$ (note that $K = \rho_n$). Then, Δ gives T' to $\mathcal{A}$.

$Join^+$ and $Leave^+$ queries: These queries are answered similar to the $Setup$ queries so that Δ is able to compute the extended transcript T' as well as K for the operation execution of $\mathtt{C\text{-}CON}_{\mathtt{P}}.\mathtt{Join}^+$ or $\mathtt{C\text{-}CON}_{\mathtt{P}}.\mathtt{Leave}^+$, respectively.

$Send$ queries: Similar to the proof of Theorem 9.8, by $Send_{S0}$, $Send_{J0}$, and $Send_{L0}$ we define the $Send$ query which invokes a new operation execution of $\mathtt{C\text{-}CON}_{\mathtt{P}}$. These and all further $Send$ queries of $\mathcal{A}$ that belong to the operation execution of P are forwarded by Δ as own queries and answered accordingly.

By queries $Send_{SF}$, $Send_{JF}$, and $Send_{LF}$ to an oracle Π_i^s we define the *final* $Send$ queries of $\mathcal{A}$ concerning the execution of $\mathtt{P}.\mathtt{Setup}$, $\mathtt{P}.\mathtt{Join}^+$, or $\mathtt{P}.\mathtt{Leave}^+$, respectively, which results in Π_i^s having computed k_i^s in the operation of P. This means that all further valid $Send$ queries to Π_i^s would be related to the communication rounds of the MA protocol. When $\mathcal{A}$ asks one these final $Send$ queries the simulator forwards it as its own query to an appropriate oracle Π_i^s implying the computation of k_i^s. Similar to the description of the $Setup$, $Join^+$, and $Leave^+$ queries above, Δ asks own $Test$ query (if the received $Send$ query is addressed to some participant of the q_{S}^*-th session) or $RevealKey$ query (in all other sessions) to obtain the real intermediate key k_i^s. Then, Δ chooses $r_i \in_R \{0,1\}^\kappa$ and responds with $U_i | r_i$.

By queries $Send_{SF+}$, $Send_{JF+}$, and $Send_{LF+}$ to an oracle Π_i^s we define the last $Send$ queries of $\mathcal{A}$ of the form $Send(op, \Pi_i^s, (U_1|r_1)|\ldots|(U_n|r_n))$ where $op \in \{'\texttt{setup}', '\texttt{join}', '\texttt{leave}'\}$ and n is the (updated) number of operation participants whereby $(U_i|r_i)$ is not part of the query message. Note that Π_i^s must have received a $Send_{SF}$, $Send_{JF}$, or $Send_{LF}$ query before the corresponding $Send_{SF+}$, $Send_{JF+}$, or $Send_{LF+}$ query; otherwise the query is unexpected. When $\mathcal{A}$ asks one of these $Send$ queries the simulator computes the sequence $\rho_1, \ldots, \rho_n$ and sets $K_i^s := \rho_n$.

$Corrupt$ queries: When $\mathcal{A}$ asks a $Corrupt(U_i)$ query Δ forwards own $Corrupt(U_i)$ query to obtain sk_i and replies with (sk_i, sk_i'). Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $Corrupt$ queries.

$RevealState$ queries: When $\mathcal{A}$ asks a $RevealState(\Pi_i^s)$ query Δ forwards own $RevealState(\Pi_i^s)$ query to obtain $\texttt{state}_i^s$ and gives it to $\mathcal{A}$. Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $RevealState$ queries.

$RevealKey$ queries: When $\mathcal{A}$ asks a $RevealKey(\Pi_i^s)$ query Δ checks that Π_i^s has accepted; otherwise an empty string is returned. Then, Δ returns the session group key K_i^s. Note that Δ is always able to do this since it executes the last step of the CON protocol itself, i.e., the computation of K_i^s for every (honest) oracle Π_i^s.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. The simulator Δ already knows K_i^s since this value is computed by Δ for every (honest) oracle Π_i^s in the simulation. Thus, Δ chooses a random bit $b \in_R \{0,1\}$ and returns K_i^s if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This is a perfect simulation for $\mathcal{A}$. Since Δ uses the real k_i^s to derive K_i^s we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = \Pr[\mathsf{Win}_1^{\mathsf{ake}}]. \tag{9.21}$$

Game $\mathbf{G}_3$. In this game we consider the simulator Δ as an active adversary against the AKE-security of P that participates in $\mathsf{Game}_{\alpha,\beta,\mathsf{P}}^{\mathsf{ake}-0}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_2$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \Pr[\mathsf{Win}_2^{\mathsf{ake}}]| \leq \mathsf{Adv}_{\alpha,\beta,\mathsf{P}}^{\mathsf{ake}}(\kappa). \tag{9.22}$$

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ except that in the q_{s}^*-th session each ρ_i, $i = 1, \ldots, n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that K is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{4,l}$, $l = 1, \ldots, n+1$ such that $\mathbf{G}'_{4,1} = \mathbf{G}_3$ and $\mathbf{G}'_{4,n+1} = \mathbf{G}_4$. That is, in the q_{s}^*-th session in each $\mathbf{G}'_{4,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the compiler whereas in $\mathbf{G}'_{4,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 1, \ldots, n-1$ by a random bit string implies uniform distribution of the PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that k used to compute ρ_1 is already uniform according to Game $\mathbf{G}_3$.

Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_4^{\mathsf{ake}}] - \Pr[\mathsf{Win}_3^{\mathsf{ake}}]| \leq N\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.23}$$

Note that since each ρ_i is erased at the end of each C-CON$_\mathtt{P}$ execution the adversary $\mathcal{A}$ cannot learn any ρ_i used in the α-fresh q_s^*-th session since the adversarial setting (α, β) disallows $RevealKey$ and $RevealState$ queries and prevents active participation of $\mathcal{A}$ on behalf of corrupted users in α-fresh sessions as mentioned in Remark 8.10.

Since K is uniformly distributed $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \frac{1}{2} \tag{9.24}$$

Considering Equation 9.20 to 9.25 we get

$$\begin{aligned}\Pr[\mathsf{Game}_{\alpha,\beta,\mathtt{C\text{-}CON_P}}^{\mathsf{ake}-b}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\ &= q_\mathsf{s}\left(\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\ &\leq q_\mathsf{s}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}}(\kappa) + Nq_\mathsf{s}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{1}{2}.\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}_{\alpha,\beta,\mathtt{C\text{-}CON_P}}^{\mathsf{ake}}(\kappa) \leq 2q_\mathsf{s}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}}(\kappa) + 2Nq_\mathsf{s}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa).$$

□

The same proof can also be applied to prove that C-CON preserves security of a GKE protocol which is secure against passive adversaries (KE-secure). In this case $\mathcal{A}$ does not have access to the $Send$ queries.

Theorem 9.14 (KE-security of C-CON$_\mathtt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs})$[6]$, (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any GKE-$\alpha$ protocol P if F is collision-resistant pseudo-random then C-CON$_\mathtt{P}$ is also GKE-α, and*

$$\mathsf{Adv}_{\alpha,\beta,\mathtt{C\text{-}CON_P}}^{\mathsf{ke}}(\kappa) \leq 2q_\mathsf{s}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}}(\kappa) + 2Nq_\mathsf{s}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa),$$

where q_s is the total number of executed protocol sessions.

9.6 Multi-Purpose Compilers

In this section we propose several compilers which add more than one security requirement to GKE protocols. Some of these compilers can be considered as a straightforward combination of a subset of the three previously described compilers.

[6] Only in case of static protocols due to Conjecture 8.31.

9.6.1 Compiler for AKE-Security and n-Contributiveness

In this section we present a compiler which can be used to add AKE-security and n-contributiveness for any KE-secure GKE protocol. The main idea behind this compiler is to use the same nonces for the authentication and key derivation.

Definition 9.15 (Compiler for AKE-Security and n-Contributiveness C-ACON). *Let* P *be a GKE protocol from Definition 8.4,* $\Sigma := (\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *a digital signature scheme,* $\pi : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$ *a permutation, and* $F := \{f_k\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ *a function ensemble with domain and range* $\{0,1\}^\kappa$. *A* compiler for AKE-security and n-contributiveness*, denoted* C-ACON*, consists of an algorithm* INIT *and a protocol* ACON *defined as follows:*

INIT*: In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk_i', pk_i') *using* $\Sigma.\texttt{Gen}(1^\kappa)$. *This is in addition to any key pair* (sk_i, pk_i) *used in* P.

ACON*: An interactive protocol between the oracles* $\Pi_1^s, \ldots, \Pi_n^s$ *in* $\mathcal{G}$ *invoked prior to any operation execution of* P. *Each* Π_i^s *chooses an* ACON nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \texttt{pid}_i^s$. *After* Π_i^s *receives* $U_j|r_j$ *from* $U_j \in \texttt{pid}_i^s$ *it checks* $|r_i| \stackrel{?}{=} \kappa$. *If this verification does not hold then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes* $\texttt{sid}_i^s := r_1|\ldots|r_n$. *Then it invokes the operation execution of* P *and proceeds as follows:*

- *If* Π_i^s *in* P *is supposed to output a message* $U_i|m$ *then in* $\texttt{C-ACON}_\texttt{P}$ *it computes additionally* $\sigma_i := \Sigma.\texttt{Sign}(sk_i', m|\texttt{sid}_i^s|\texttt{pid}_i^s)$ *and outputs a modified message* $U_i|m|\sigma_i$.
- *If* Π_i^s *receives a message* $U_j|m|\sigma_j$ *from* Π_j^s *with* $U_j \in \texttt{pid}_i^s$ *it checks whether* $\Sigma.\texttt{Verify}(pk_j', m|\texttt{sid}_i^s|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. *If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise it proceeds according to the specification of the executed operation of* P *upon receiving* $U_j|m$.
- *After an oracle* Π_i^s *computes* k_i^s *in the execution of* P *it computes* $\rho_0 := f_{k_i^s}(v_0)$, *and each* $\rho_l := f_{\rho_{l-1}\oplus\pi(r_l)}(v_0)$ *for all* $l = \{1,\ldots,n\}$, *where* v_0 *is a public value, computes the session group key* $K_i^s := \rho_n$, *erases every other private information from* $\texttt{state}_i^s$ *(including* k_i^s *and all* ρ_l, $l = \{0,\ldots,n\}$*), and accepts with* K_i^s.

Note that C-ACON differs in the computation of K_i^s from C-CON. In C-ACON each Π_i^s starts with the computation of $\rho_0 := f_{k_i^s}(v_0)$ which is then used to derive the pseudo-random function key for ρ_1, whereas in C-CON each user starts with the computation of ρ_1 using k_i^s directly for the derivation of the corresponding pseudo-random function key. This trick is applied in order to achieve contributiveness in the compiled protocol $\texttt{C-ACON}_\texttt{P}$ motivated by the fact that the honest participant's oracle Π_i^s chooses own ACON nonce before P is executed and k_i^s is returned whereas the CON nonce is chosen after k_i^s is computed.

Why do we need ρ_0*?*

In the following we describe a possible attack of an adversary against n-contributiveness of C-ACON in case that ρ_1 is computed using $k_i^{s_i}$ directly instead of ρ_0, i.e., if $\rho_1 := f_{k_i^{s_i}\oplus\pi(r_1)}(v_0)$ instead of $\rho_1 := f_{\rho_0\oplus\pi(r_1)}(v_0)$.

In the prepare stage the adversary $\mathcal{A}$ chooses some $R_1 \in \{0,1\}^\kappa$ and computes $\rho_1 := f_{R_1}(v_0)$. Then, for all $l > 1$ it computes $\rho_l := f_{\rho_{l-1}\oplus\pi(r_l)}(v_0)$ using a randomly chosen nonce r_l, and outputs $\tilde{K} := \rho_n$. Then in the attack stage $\mathcal{A}$ corrupts all users $U_{l\neq 1}$. After Π_1^s outputs

r_1 the adversary sends $U_l|r_l$, $l > 1$ with each r_l chosen in the prepare stage to Π_1^s (as part of its $Send$ query) and influences Π_1^s to compute $k_1^s = R_1 \oplus \pi(r_1)$. Since P is not expected to be contributory this may happen with non-negligible probability. It is clear that using nonces chosen by the adversary the oracle accepts with $K_1^s = \tilde{K}$, i.e., $\mathcal{A}$ succeeds.

In the following we discuss the n-contributiveness of C-ACON$_\mathtt{P}$ and argue that $\mathcal{A}$, despite of being able to influence the session group key k computed in the operations of the original protocol P, has a negligible success in its attack on influencing/predicting the session group key K in the compiled protocol C-ACON$_\mathtt{P}$. Similar to the proof of Theorem 9.11, we use the fact that every honest oracle $\Pi_i^s \in \mathcal{G}$, $i \in [1, n]$ computes the sequence $\rho_0, \ldots, \rho_n$ in order to accept with $K = \rho_n$ so that each ρ_l, $l \in [1, n]$ depends on the previously computed ρ_{l-1}. We consider the probability that $\mathcal{A}$ is able to influence an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$, $i^* \in [1, n]$ to accept some $\tilde{K}$ by considering its ability to influence $\Pi_{i^*}^s$ to compute any value ρ_l, $i^* \leq l \leq n$. This is equivalent to the ability of $\mathcal{A}$ in the prepare stage to output ρ_l which is then computed by $\Pi_{i^*}^s$ in some attack-ed session. On the other hand, applying Lemma 5.20 in our proof we do also consider the upper-bound for the success probability of the adversary in case that its strategy differs from influencing any value in $\rho_{i^*}, \ldots, \rho_n$.

Theorem 9.16 (n-Contributiveness of C-ACON$_\mathtt{P}$). *For any GKE protocol* P *if π is one-way and F is collision-resistant pseudo-random then* C-ACON$_\mathtt{P}$ *is n-*CGKE*, and*

$$\mathsf{Succ}^{\mathsf{con}-n}_{\text{C-ACON}_\mathtt{P}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + q_s}{2^\kappa} + (N+1)q_s\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_s\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_s\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa),$$

where q_s is the total number of executed protocol sessions.

Remark 9.17. Note that some arguments in this proof are intuitive for the same reasons as mentioned in Remark 9.12.

Proof (partially informal). In the following we consider an adversary $\mathcal{A}$ from Definition 8.18. Assume that $\mathcal{A}$ wins in $\mathsf{Game}^{\mathsf{con}-n}_{\text{C-ACON}_\mathtt{P}}(\kappa)$ (which event we denote $\mathsf{Win}^{\mathsf{con}}$). Then at the end of the stage prepare it returned $\tilde{K}$ such that in the stage attack an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$ accepted with $K_{i^*}^s = \tilde{K}$. According to the construction of $K_{i^*}^s$ we follow that $\tilde{K} = \rho_n$ computed by $\Pi_{i^*}^s$.

Game $\mathbf{G}_0$. This is the real game $\mathsf{Game}^{\mathsf{con}-n}_{\text{C-ACON}_\mathtt{P}}(\kappa)$, in which the honest players are replaced by a simulator Δ.

Game $\mathbf{G}_1$. In this game we abort the simulation if the same random nonce r_i is used by any honest Π_i^s in two different sessions. Considering r_i being uniform for every honest oracle Π_i^s, and since there are at most N users we have

$$\Pr[\mathsf{Win}^{\mathsf{con}}_0] - \Pr[\mathsf{Win}^{\mathsf{con}}_1] \leq \frac{Nq_s^2}{2^\kappa}. \tag{9.25}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage.[7] According to Lemma 5.20 we need to estimate the probability of the opposite event, i.e., that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*}. We consider two cases: $i^* = 1$ and $i^* > 1$. Note that all other oracles except for $\Pi_{i^*}^s$ can be corrupted.

[7] Note, in $\mathbf{G}_0$ and $\mathbf{G}_1$ the adversary only outputs a value for the resulting group key. In $\mathbf{G}_2$ we consider the additional (in)ability of the adversary to output the value for ρ_{i^*}. Since we are only interested in the probability of the adversarial success under this "condition event" (without changing the game in case that this event occurs; see also Section 5.6.1) the simulator does not need to detect whether $\mathcal{A}$ is able to output the correct value or not. The same considerations are applicable to $\mathbf{G}_3$ w.r.t. $K_{i^*}^s$.

Case $i^* = 1$: In any session of the attack stage honest oracle Π_1^s computes $\rho_1 := f_{\rho_0 \oplus \pi(r_1)}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_0 \oplus \pi(r_1)$ (denoted R_1) in the prepare stage $\mathcal{A}$'s probability to output $\rho_1 = f_{R_1}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_1 in the prepare stage and tries to influence Π_1^s computing exactly this value in some session of the attack stage. Note that the honest oracle Π_1^s chooses r_1 uniformly at random for every session in the attack stage. Since π is a permutation the value $\pi(r_1)$ is also uniform and fixed for every attack-ed session. Hence, the adversary must influence in the attack-ed session the oracle $\Pi_{i^*}^s$ to compute $\rho_0 = R_1 \oplus \pi(r_1)$ which is fixed and uniformly distributed for each such session. Note that $\mathcal{A}$ learns the required ρ_0 only after having received r_1, that is during the attack-ed session. Since U_{i^*} is uncorrupted its oracle computes ρ_0 according to the protocol specification, that is $\rho_0 := f_{k_1^s}(v_0)$. Having excluded PRF collisions and random guesses we consider the PRF key k_1^s (which is the session group key computed by Π_1^s in the original protocol P, and thus possibly influenceable by $\mathcal{A}$) as a fixed value unknown to $\mathcal{A}$. The probability that $\mathcal{A}$ recovers it is intuitively bound by $\mathsf{Adv}_F^{\mathsf{prf}}(\kappa)$. This is because any adversary that is able to reveal the PRF key can act as a distinguisher for the pseudo-randomness of f.

Case $i^* > 1$: In any session of the attack stage honest oracle $\Pi_{i^*}^s$ computes $\rho_{i^*} := f_{\rho_{i^*-1} \oplus \pi(r_{i^*})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*-1} \oplus \pi(r_{i^*})$ (denoted R_{i^*}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*} = f_{R_{i^*}}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*} in the prepare stage and tries to influence $\Pi_{i^*}^s$ computing exactly this value in some session of the attack stage. Since r_{i^*} is chosen by honest $\Pi_{i^*}^s$ at random in every attack-ed session and π is a permutation the value $\pi(r_{i^*})$ is uniform and fixed for each attack-ed session. Hence, the adversary must influence in the attack stage the oracle $\Pi_{i^*}^s$ to compute $\rho_{i^*-1} = R_{i^*} \oplus \pi(r_{i^*})$ which is fixed and uniformly distributed for each attack-ed session. Note that $\mathcal{A}$ learns the required ρ_{i^*-1} only after having received r_{i^*}, that is during the attack-ed session. Since U_{i^*} is uncorrupted its oracle computes ρ_{i^*-1} according to the protocol specification, that is $\rho_{i^*-1} := f_{\rho_{i^*-2} \oplus \pi(r_{i^*-1})}(v_0)$. Having excluded PRF collisions and random guesses we consider the PRF key $\rho_{i^*-2} \oplus \pi(r_{i^*-1})$ as a fixed value unknown to the adversary. The probability that $\mathcal{A}$ recovers it is intuitively bound by $\mathsf{Adv}_F^{\mathsf{prf}}(\kappa)$. This is because any adversary that is able to reveal the PRF key can act as a distinguisher for the pseudo-randomness of f.

Since there are at most q_s sessions we have

$$\Pr[\mathsf{Win}_1^{\mathsf{con}}] - \Pr[\mathsf{Win}_2^{\mathsf{con}}] \leq q_\mathsf{s}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_\mathsf{s}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}. \tag{9.26}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that we do not need to consider the values ρ_l, $l < i^*$ computed by $\Pi_{i^*}^s$ since in order to compute $K_{i^*}^s$ every honest oracle must compute the whole sequence $\rho_0, \ldots, \rho_n$. Thus, it is sufficient to argue that the probability of $\mathcal{A}$ influencing any ρ_l, $l \geq i^*$, computed by $\Pi_{i^*}^s$ in any attack-ed session is negligible.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $K_{i^*}^s := \rho_n$ computed by $\Pi_{i^*}^s$ in any session of the attack stage. Again, the simulator does not need to detect whether this event occurs since both games

proceed identical in any case. According to Lemma 5.20 we need to estimate the probability of the opposite event, i.e., that $\mathcal{A}$ being in the prepare stage is able to output $K^s_{i^*}$.

Similar to the proof of Theorem 9.11 based on the "hybrid technique" we define a sequence of auxiliary games $\mathbf{G}'_{3,l}$, $l = i^*, \ldots, n$. Each of these games is identical to the previous one in the sequence with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_l computed by $\Pi^s_{i^*}$ in any session of the attack stage. Obviously, $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$ and $\mathbf{G}'_{3,n} = \mathbf{G}_3$. According to Lemma 5.20 we need to estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_l.

Since $\rho_l := f_{\rho_{l-1} \oplus \pi(r_l)}(v_0)$ for all $l > i^*$ and each r_l is not chosen by $\Pi^s_{i^*}$ each two consecutive auxiliary games have the same difference. Hence, it is sufficient to compute this difference between any two consecutive auxiliary games. In the following we compute the difference between $\mathbf{G}'_{3,i^*+1}$ and $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$. We estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*+1} computed by $\Pi^s_{i^*}$ in any session of the attack stage.

We argue by intuition. Since $\Pi^s_{i^*}$ is honest, in the attack stage it computes $\rho_{i^*+1} := f_{\rho_{i^*} \oplus \pi(r_{i^*+1})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*} \oplus \pi(r_{i^*+1})$ (denoted R_{i^*+1}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*+1} = f_{R_{i^*+1}}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*+1} in the prepare stage and tries to influence $\Pi^s_{i^*}$ computing exactly this value in some session of the attack stage. Recall that $\mathcal{A}$ is allowed to corrupt any user $U_{l \neq i^*}$, and thus choose each nonce r_l, $l \neq i^*$. Since $\mathcal{A}$ learns ρ_{i^*} only in the attack-ed session (as observed in Game $\mathbf{G}_2$) and having excluded PRF collisions and random guesses the probability that $\mathcal{A}$ is able to influence $\Pi^s_{i^*}$ computing R_{i^*+1} in the attack stage is bound by the probability that in the attack-ed session $\mathcal{A}$ computes the appropriate nonce r_{i^*+1} such that $\pi(r_{i^*+1}) = R_{i^*+1} \oplus \rho_{i^*}$ holds. Since π is one-way this probability is intuitively bound by $\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa)$. Thus, $\mathcal{A}$ is able to output ρ_{i^*+1} while being in the prepare stage with the probability of at most $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + \mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + 1/2^\kappa$. Since there are at most q_s sessions the total probability that $\mathcal{A}$ is able to do this is at most

$$q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}.$$

The above sum upper-bounds the difference between $\mathbf{G}'_{3,i^*+1}$ and $\mathbf{G}_2$. Since there are at most N auxiliary games (due to $n \leq N$) we obtain

$$\Pr[\mathsf{Win}^{\mathsf{con}}_2] - \Pr[\mathsf{Win}^{\mathsf{con}}_3] \leq Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + \frac{Nq_\mathsf{s}}{2^\kappa}. \tag{9.27}$$

Since in this game $\mathcal{A}$ is not able to output $K^s_{i^*} = \rho_n$ in the prepare stage it does not output a correct value for $\tilde{K}$. Hence,

$$\Pr[\mathsf{Win}^{\mathsf{con}}_3] = 0. \tag{9.28}$$

Considering Equations 9.25 to 9.28 we obtain the desired inequality

$$\begin{aligned}\mathsf{Succ}^{\mathsf{con}-n}_{\texttt{C-ACON}_\mathsf{P}}(\kappa) &= \Pr[\mathsf{Win}^{\mathsf{con}}_0] \\ &\leq \frac{Nq_\mathsf{s}^2 + Nq_\mathsf{s} + q_\mathsf{s}}{2^\kappa} + (N+1)q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa).\end{aligned}$$

□

The following theorem shows that C-ACON adds AKE-security to any KE-secure static GKE protocol. As for the compiler C-A we do not consider (sbs, scm) as a possible adversarial setting.

Theorem 9.18 (AKE-Security of Static C-ACON$_{\mathtt{P}}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any **static** GKE-α protocol P if Σ is EUF-CMA and F is pseudo-random then C-ACON$_{\mathtt{P}}$ is AGKE-α, and*

- *if $(\alpha, \beta) \in \{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}ACON_P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 2(N+1)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

- *if $(\alpha, \beta) \in \{(\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}ACON_P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 2(N+1)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. Similar to the proof of Theorem 9.3 we define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 6$ and corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}ACON_P}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}ACON_P}}(\kappa)$ (see Definition 8.12) played between a simulator Δ and an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π^s_i. Keep in mind that on the test query the adversary receives either a random string or a session group key K^s_i.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|m|\sigma$ such that σ is a valid signature on m that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \Pr[\mathsf{Win}^{\mathsf{ake}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa). \tag{9.29}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulator fails and sets b' at random if an ACON nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. Similar to Equation 9.3 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_2] - \Pr[\mathsf{Win}^{\mathsf{ake}}_1]| \leq \frac{Nq_s^2}{2^{\kappa}}. \tag{9.30}$$

This game implies that $\mathtt{sid}^s_i$ computed by any uncorrupted user's oracle Π^s_i remains unique for each new session. Note that $\mathtt{sid}^s_i$ is used to generate signatures in the ACON protocol of the compiler. This prevents any replay attacks of $\mathcal{A}$.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q^*_s \in [1, q_s]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q^*_s-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}^{\mathsf{ake}}_2] = q_s\left(\Pr[\mathsf{Win}^{\mathsf{ake}}_3] - \frac{1}{2}\right) + \frac{1}{2}. \tag{9.31}$$

Game $\mathbf{G}_4$. In this game we consider the simulator Δ as a passive adversary against the KE-security of P that participates in $\mathsf{Game}^{\mathsf{ke}-1}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π^s_i is answered with the real session group key k^s_i. In the following we show how Δ

answers the queries of $\mathcal{A}$. We extend the simulation described in the proof of Theorem 9.3 by additional post-computations needed to derive the session group key K. Similar to the proof of Theorem 9.3 we focus on the adversarial settings $(\texttt{wfs}, \texttt{wcm-fs})$ and $(\texttt{sfs}, \texttt{scm})$.

Δ corrupts every user $U_i \in \mathcal{U}$ to obtain the long-lived key pair (sk_i, pk_i) used in the original protocol $\texttt{P}$ (if any such keys are defined). Then, Δ generates all key pairs (sk'_i, pk'_i) honestly using $\Sigma.\texttt{Gen}(1^\kappa)$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk'_i, pk_i\}_{U_i \in \mathcal{U}}$. Δ initializes the list $TList$ and runs $\mathcal{A}$ as a subroutine.

$Setup$ queries: These queries are processed as described in the proof of Theorem 9.3, i.e., Δ performs every operation execution itself saving $(\texttt{sid}, \perp)$ in the $TList$ except for the operation in the $q^*_{\texttt{s}}$-th session for which it asks own $Setup$ query. After Δ receives the execution transcript T of $\texttt{P.Setup}$ it computes T' for $\texttt{C-ACON}_{\texttt{P}}.\texttt{Setup}$ by extending T with the initial messages of the form $\{U_i | r_i\}_{1 \le i \le n}$ and digital signatures, saves $(\texttt{sid}, T')$ in $TList$, and gives T' to $\mathcal{A}$. Note that in $\texttt{C-ACON}_{\texttt{P}}$ no additional messages are sent after the computation of k so that T' can be considered as a transcript for the execution of $\texttt{C-ACON}_{\texttt{P}}$.

$Send$ queries: These queries are also processed as described in the proof of Theorem 9.3. All sessions invoked via a $Send$ query are executed by Δ itself except for the $q^*_{\texttt{s}}$-th session for which Δ asks own $Setup$ query. Similar to the description of the $Setup$ query for the $q^*_{\texttt{s}}$-th session above Δ "patches" the transcript T with digital signatures to obtain the transcript T' for the corresponding execution of $\texttt{C-ACON}_{\texttt{P}}.\texttt{Setup}$, saves $(\texttt{sid}^s_i, T')$ in $TList$, and replies to $\mathcal{A}$ with the appropriate message taken from T'. All other $Send$ queries related to the oracles that participate in the $q^*_{\texttt{s}}$-th session are answered from the predefined transcript T'.

$Corrupt$ queries: If $\mathcal{A}$ asks a query of the form $Corrupt(U_i)$ then Δ replies with (sk_i, sk_i).

$RevealState$ queries: If $\mathcal{A}$ asks a query of the form $RevealState(\Pi^s_i)$ then Δ finds an entry $(\texttt{sid}^s_i, T^*)$ in $TList$. If $T^* = \perp$ it means that Δ executes the protocol itself and is, therefore, able to answer this query directly. If $T^* = T'$ then Δ checks whether Π^s_i has already accepted. In this case Δ asks its own $RevealState$ query to obtain $\texttt{state}^s_i$ and replies accordingly. Note that if Π^s_i has not yet accepted in $\texttt{C-ACON}_{\texttt{P}}.\texttt{Setup}$ then an empty string is returned.

$RevealKey$ queries: If $\mathcal{A}$ asks a query of the form $RevealKey(\Pi^s_i)$ then Δ checks that Π^s_i has accepted; otherwise an empty string is returned. Next, Δ finds an entry $(\texttt{sid}^s_i, T^*)$ in $TList$. If $T^* = \perp$ then $\mathcal{A}'$ is able to answer with K^s_i directly since the protocol execution with Π^s_i has been done by Δ. If $T^* = T'$ then the query is invalid since no $RevealKey$ queries are allowed to the oracles that have accepted in the $q^*_{\texttt{s}}$-th session.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π^s_i that has participated in the $q^*_{\texttt{s}}$-th session. The simulator Δ asks own $Test$ query to an oracle which has been activated via the $Setup$ query and obtains k^s_i. Then, Δ computes the resulting session group key K^s_i using k^s_i and nonces in $\texttt{sid}^s_i$ as specified in $\texttt{C-ACON}_{\texttt{P}}$, i.e., via the computation of $\rho_0, \ldots, \rho_n$. The simulator chooses a random bit $b \in_R \{0,1\}$ and returns $K^s_i = \rho_n$ if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This provides a perfect simulation for $\mathcal{A}$. Since Δ uses the real k^s_i to derive K^s_i we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{9.32}$$

When dealing with the adversarial settings $(\emptyset, \texttt{wcm})$ and $(\texttt{wbs}, \texttt{wcm-bs})$ the above simulation can be simplified since no $Corrupt$ queries need to be considered. Similar to the proof of Theorem 9.3 Δ can answer all $Send$ queries of $\mathcal{A}$ from the predefined transcripts of $\mathtt{P}$ after having "patched" them with random nonces and digital signatures. In contrast to the above simulation Δ needs to perform post-computations also in case of the $RevealKey$ queries since by forwarding this query Δ obtains k but must return K. Note that since Δ does not execute any operation itself there is no need to guess the q_{s}^*-th session. Thus, in case of the adversarial settings $(\emptyset, \texttt{wcm})$ and $(\texttt{wbs}, \texttt{wcm-bs})$ Game $\mathbf{G}_3$ can be skipped.

Game $\mathbf{G}_5$. In this game we consider the simulator Δ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}^{\mathsf{ke}-0}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa). \tag{9.33}$$

As mentioned in Game $\mathbf{G}_4$ when considering $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wbs}, \texttt{wcm-bs})\}$ we can skip Game $\mathbf{G}_3$. Nevertheless Game $\mathbf{G}_3$ should be inserted prior to Game $\mathbf{G}_6$ which requires a correct guess for the q_{s}^*-th session in order to perform the simulation.

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session each ρ_i, $i = 0, \ldots, n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that K is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{6,l}$, $l = 0, \ldots, n+1$ such that $\mathbf{G}'_{6,0} = \mathbf{G}_5$ and $\mathbf{G}'_{6,n+1} = \mathbf{G}_6$. That is, in the q_{s}^*-th session in each $\mathbf{G}'_{6,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the compiler whereas in $\mathbf{G}'_{6,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 0, \ldots, n-1$ by a random bit string implies uniform distribution of the PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that k used to compute ρ_0 is already uniform according to Game $\mathbf{G}_5$.

Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq (N+1)\mathsf{Adv}^{\mathsf{prf}}_F(\kappa). \tag{9.34}$$

Since $K = \rho_n$ is uniformly distributed $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_6^{\mathsf{ake}}] = \frac{1}{2} \tag{9.35}$$

Considering Equations 9.29 to 9.36 we get for the case $(\alpha, \beta) \in \{(\texttt{wfs}, \texttt{wcm-fs}), (\texttt{sfs}, \texttt{scm-fs})\}$

$$\begin{aligned}
\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-ACON}_\mathtt{P}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&\leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\
&= N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + \\
&\quad (N+1)q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-ACON}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^{\kappa-1}} + 2q_\mathsf{s}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa) + 2(N+1)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa).$$

Similarly, for the case $(\alpha,\beta) \in \{(\emptyset,\texttt{wcm}),(\texttt{wbs},\texttt{wcm-bs})\}$ where Game $\mathbf{G}_3$ can be placed one position before Game $\mathbf{G}_6$ we get

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-ACON}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^{\kappa-1}} + 2\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa) + 2(N+1)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa).$$

□

The main problem when applying the reduction from Theorem 9.18 to dynamic GKE protocols occurs in Games $\mathbf{G}_4$ and $\mathbf{G}_5$ in the simulation of a passive adversary against the KE-security of the protocol. Considering the additional queries $Leave^+$ and $Join^+$ the above simulation fails for the similar reasons as described in the context of the compiler C-A. Nevertheless, we believe that C-ACON preserves the α-freshness of dynamic GKE protocols due to the following reasons. First, the above simulation can be easily extended to the dynamic case for the adversarial setting $(\emptyset,\texttt{wcm})$ where no $Corrupt$ queries have to be considered. Second, nonces used to prevent replay attacks and derive the session group key K are chosen anew for each operation execution of the protocol. Beside that the key k and all intermediate values ρ_i, $i = 0,\ldots,\rho_n$ are erased at the end of each operation execution protecting the secrecy of K against strong corruptions in later operations. Therefore, we state the following conjecture for dynamic GKE protocols whereby excluding the adversarial setting $(\texttt{wbs},\texttt{wcm-bs})$ as a consequence of Conjecture 8.31.

Conjecture 9.19 (AKE-Security of Dynamic C-ACON$_\texttt{P}$). *Let (α,β) be an adversarial setting sampled from $\{(\emptyset,\texttt{wcm}),(\texttt{wfs},\texttt{wcm-fs}),(\texttt{sfs},\texttt{scm})\}$. For any **dynamic** GKE-α protocol P if Σ is EUF-CMA and F is pseudo-random then C-ACON$_\texttt{P}$ is AGKE-α, and*

- *if $(\alpha,\beta) = (\emptyset,\texttt{wcm})$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-ACON}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^{\kappa-1}} + 2\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa) + 2(N+1)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

- *if $(\alpha,\beta) \in \{(\texttt{wfs},\texttt{wcm-fs}),(\texttt{sfs},\texttt{scm})\}$:*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-ACON}_\texttt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^{\kappa-1}} + 2q_\mathsf{s}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{P}}(\kappa) + 2(N+1)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

9.6.2 Compiler for AKE- and MA-Security

In this section we show that compiler for MA-security C-MA can be combined with the compiler for AKE-security C-A. The resulting compiler allows to save one communication round. The main idea behind this combination is to use the same nonces for the authentication and MA-security.

Definition 9.20 (Compiler for AKE-Security and MA-Security C-AMA). *Let P be a GKE protocol from Definition 8.4, $\Sigma := (\texttt{Gen},\texttt{Sign},\texttt{Verify})$ a digital signature scheme, $F := \{f_k\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ a function ensemble with domain and range $\{0,1\}^\kappa$. A* compiler for AKE-security and MA-security*, denoted C-AMA, consists of an algorithm INIT and a protocol AMA defined as follows:*

INIT: *In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk'_i, pk'_i) *using* $\Sigma.\mathtt{Gen}(1^\kappa)$. *This is in addition to any key pair* (sk_i, pk_i) *used in* P.

AMA: *An interactive protocol between the oracles* $\Pi_1^s, \ldots, \Pi_n^s$ *in* $\mathcal{G}$ *invoked prior to any operation of* P. *Each* Π_i^s *chooses a random* AMA nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$. *After* Π_i^s *receives* $U_j|r_j$ *from all* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it computes* $\mathtt{sid}_i^{s_i} := r_1| \ldots |r_n$. *Then it invokes the operation execution of* P *and proceeds as follows:*

- *If* Π_i^s *in* P *is supposed to output a message* $U_i|m$ *then in* C-AMA$_\mathtt{P}$ *it computes additionally* $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, m|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ *and outputs a modified message* $U_i|m|\sigma_i$.
- *If* Π_i^s *receives a message* $U_j|m|\sigma_j$ *from* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it checks whether* $\Sigma.\mathtt{Verify}(pk'_j, m|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. *If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise it proceeds according to the specification of the executed operation of* P *upon receiving* $U_j|m$.
- *After an oracle* Π_i^s *computes* k_i^s *in the execution of* P *it computes an* AMA token $\mu_i := f_{k_i^s}(v_0)$ *where* v_0 *is a constant public value, a signature* $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ *and sends* $U_i|\sigma_i$ *to every oracle* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$.
  After Π_i^s *receives* $U_j|\sigma_j$ *from* Π_j^s *with* $U_j \in \mathtt{pid}_i^s$ *it checks whether* $\Sigma.\mathtt{Verify}(pk'_j, \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. *If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes the session group key* $K_i^s := f_{k_i^s}(v_1)$ *where* $v_1 \neq v_0$ *is another constant public value, erases every other private information from* $\mathtt{state}_i^s$ *(including* k_i^s*), and accepts with* K_i^s.

First, we show that C-AMA adds AKE-security to any KE-secure GKE protocol. As for the compilers C-A and C-ACON we do not consider (sbs, scm) as a possible adversarial setting.

Theorem 9.21 (AKE-Security of Static C-AMA$_\mathtt{P}$). *Let* (α, β) *be an adversarial setting sampled from* $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. *For any* ***static*** GKE-α *protocol* P *if* Σ *is EUF-CMA and* F *is pseudo-random then* C-AMA$_\mathtt{P}$ *is* AGKE-α, *and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}AMA_P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 4q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Proof. Similar to the proofs of Theorems 9.3 and 9.18 we define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 6$ and corresponding events $\mathsf{Win}_i^{\mathsf{ake}}$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}AMA_P}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}AMA_P}}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π_i^s. Keep in mind that on the test query the adversary receives either a random string or a session group key K_i^s.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_0^{\mathsf{ake}}]| \leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa). \tag{9.36}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if an AMA nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. According to Equation 9.3 we get

$$|\Pr[\mathsf{Win}_2^{\mathsf{ake}}] - \Pr[\mathsf{Win}_1^{\mathsf{ake}}]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}}. \tag{9.37}$$

This game implies that $\mathtt{sid}_i^s$ computed by any uncorrupted user's oracle Π_i^s remains unique for each new session. Note that $\mathtt{sid}_i^s$ is used to generate signatures in the AMA protocol of the compiler. This prevents any replay attacks of $\mathcal{A}$.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q_{\mathsf{s}}^* \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q_{s}^*-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2}. \tag{9.38}$$

Game $\mathbf{G}_4$. In this game we consider the simulator Δ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}-1}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with the real session group key k_i^s. In the following we show how Δ answers the queries of $\mathcal{A}$. We extend the simulation described in the proof of Theorem 9.3 by additional post-computations needed to derive the session group key K. Similar to the proof of Theorem 9.3 we focus on the adversarial settings $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$.

Δ corrupts every user $U_i \in \mathcal{U}$ to obtain the long-lived key pair (sk_i, pk_i) used in the original protocol $\mathtt{P}$ (if any such keys are defined). Then, Δ generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^{\kappa})$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. Δ initializes the list $TList$ and runs $\mathcal{A}$ as a subroutine.

$Setup$ queries: These queries are processed as described in the proof of Theorem 9.3, i.e., Δ performs every operation execution itself saving $(\mathtt{sid}, \perp)$ in the $TList$ except for the operation in the q_{s}^*-th session for which it asks own $Setup$ query. After Δ receives the execution transcript T of $\mathtt{P}.\mathtt{Setup}$ in the q_{s}^*-th session it computes T' for the corresponding execution of $\mathtt{C\text{-}AMA_P}.\mathtt{Setup}$ by extending T with the initial messages of the form $\{U_i|r_i\}_{1\leq i\leq n}$ and digital signatures. In contrast to the proofs of Theorems 9.3 and 9.18 the simulator must extend T' with messages that include signatures on AMA tokens. For this purpose Δ asks own $Test$ query to any oracle activated via the $Setup$ query in the q_{s}^*-th session and obtains (real) k which it then uses to compute the AMA token $\mu := f_k(v_0)$. Then, Δ computes $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu|\mathtt{sid}|\mathtt{pid})$ and appends messages of the form $\{U_i|\sigma_i\}_{1\leq i\leq n}$ to T'. Finally, Δ saves $(\mathtt{sid}, T')$ in $TList$ and gives T' to $\mathcal{A}$.

$Send$ queries: These queries are also processed as described in the proof of Theorem 9.3. All sessions invoked via a $Send$ query are executed by Δ itself except for the q_{s}^*-th session for which Δ asks own $Setup$ query. Similar to the description of the $Setup$ query for the q_{s}^*-th session above Δ "patches" the transcript T with digital signatures and additional messages that contain digital signatures on the computed AMA tokens to obtain the transcript T' for the corresponding operation execution of $\mathtt{C\text{-}AMA_P}$ (in order to compute the AMA token Δ asks own $Test$ query), saves $(\mathtt{sid}_i^s, T')$ in $TList$, and replies to $\mathcal{A}$ with the appropriate message taken from T'.

All other $Send$ queries related to the oracles that participate in the q_{s}^*-th session are answered from the predefined transcript T'.

$Corrupt$ queries: If $\mathcal{A}$ asks a query of the form $Corrupt(U_i)$ then Δ replies with (sk_i, sk_i).

$RevealState$ queries: If $\mathcal{A}$ asks a query of the form $RevealState(\Pi_i^s)$ then Δ finds an entry $(\mathtt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ it means that Δ executes the protocol itself and is, therefore, able to answer this query directly. If $T^* = T'$ then Δ checks whether Π_i^s has already accepted. In this case Δ asks its own $RevealState$ query to obtain $\mathtt{state}_i^s$ and replies accordingly. Note that if Π_i^s has not yet accepted in $\mathtt{C\text{-}AMA_P.Setup}$ then an empty string is returned.

$RevealKey$ queries: If $\mathcal{A}$ asks a query of the form $RevealKey(\Pi_i^s)$ then Δ checks that Π_i^s has accepted; otherwise an empty string is returned. Next, Δ finds an entry $(\mathtt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ then Δ is able to answer with K_i^s directly since the protocol execution with Π_i^s has been done by Δ. If $T^* = T'$ then the query is invalid since no $RevealKey$ queries are allowed to the oracles that have accepted in the q_{s}^*-th session.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. The simulator Δ already knows k_i^s since it has already asked own $Test$ query to build the transcript T' straight after the invocation of the q_{s}^*-th session. Thus, Δ computes the resulting session group key $K_i^s := f_{k_i^s}(v_1)$ as specified in $\mathtt{C\text{-}AMA_P}$, chooses a random bit $b \in_R \{0,1\}$ and returns K_i^s if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This provides a perfect simulation for $\mathcal{A}$. Since Δ uses the real k_i^s to derive K_i^s we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{9.39}$$

When dealing with the adversarial settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ the above simulation can be simplified since no $Corrupt$ queries need to be considered. Similar to the proof of Theorem 9.3 Δ can answer all $Send$ queries of $\mathcal{A}$ from the predefined transcripts. However, note that although Δ can "patch" T with random nonces and digital signatures it cannot extend it with the messages that contain digital signatures on the AMA tokens without obtaining the session group key k needed to compute the AMA token μ. Thus, Δ must ask own $RevealKey$ query in order to build the complete transcript T'. Therefore, Δ must guess the q_{s}^*-th session. Otherwise, Δ risks that $\mathcal{A}$ asks its $Test$ query for the session which is already unfresh from the perspective of Δ. Thus, in contrast to the proof of Theorem 9.18 Game $\mathbf{G}_3$ cannot be skipped when dealing with $(\emptyset, \mathtt{wcm})$ or $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$.

Game $\mathbf{G}_5$. In this game we consider the simulator Δ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}^{\mathsf{ke}-0}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa). \tag{9.40}$$

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session K and the AMA token μ are replaced by random values sampled from $\{0,1\}^\kappa$. Recall that k used to compute K and μ is uniform according to Game $\mathbf{G}_5$. Hence,

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.41}$$

Obviously, in this game $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_6^{\mathsf{ake}}] = \frac{1}{2} \tag{9.42}$$

Considering Equations 9.36 to 9.42 we get:

$$\begin{aligned}
\Pr[\mathsf{Game}_{\alpha,\beta,\texttt{C-AMA}_\mathtt{P}}^{\mathsf{ake}-b}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\
&= N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}}(\kappa) + \\
&\quad 2q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}_{\alpha,\beta,\texttt{C-AMA}_\mathtt{P}}^{\mathsf{ake}}(\kappa) \leq 2N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}}(\kappa) + 4q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa).$$

□

Similar to the compiler C-ACON the main problem when applying the reduction from Theorem 9.21 to dynamic GKE protocols occurs in Games $\mathbf{G}_4$ and $\mathbf{G}_5$ in the simulation of a passive adversary against the KE-security of the protocol. Considering the additional queries $Leave^+$ and $Join^+$ the above simulation fails for the similar reasons as described in the context of the compiler C-A. Nevertheless, we believe that C-AMA preserves the α-freshness of dynamic GKE protocols due to the following reasons. First, the above simulation can be easily extended to the dynamic case for the adversarial setting $(\emptyset, \mathtt{wcm})$ where no $Corrupt$ queries have to be considered. Second, nonces used to prevent replay attacks are chosen anew for each operation execution of the protocol. Third, AMA tokens derived via a pseudo-random function do not reveal any information about the key k which is returned by the underlying protocol P and erased at the end of each operation execution protecting the secrecy of K against strong corruptions in later operations. Therefore, we state the following conjecture for dynamic GKE protocols whereby excluding the adversarial setting $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ as a consequence of Conjecture 8.31.

Conjecture 9.22 (AKE-Security of Dynamic C-AMA$_\mathtt{P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any **dynamic** GKE-α protocol P if Σ is EUF-CMA and F is pseudo-random then C-AMA$_\mathtt{P}$ is AGKE-α, and*

$$\mathsf{Adv}_{\alpha,\beta,\texttt{C-AMA}_\mathtt{P}}^{\mathsf{ake}}(\kappa) \leq 2N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ke}}(\kappa) + 4q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

Further we show that C-AMA adds MA-security to any GKE protocol.

Theorem 9.23 (MA-Security of C-AMA$_\mathtt{P}$). *For any GKE protocol* P *if* Σ *is EUF-CMA and* F *is collision-resistant then* C-AMA$_\mathtt{P}$ *is* MAGKE*, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{C\text{-}AMA_P}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}^{\mathsf{ma}}_i$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator Δ.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{C\text{-}AMA_P}}(\kappa)$ played between Δ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π_i^s accepts with K_i^s and another user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts and either does not have a corresponding oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$ or has such an oracle but this oracle accepts with $K_j^s \neq K_i^s$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails if $\mathcal{A}$ asks a $Send$ query on a message $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain,

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_1] - \Pr[\mathsf{Win}^{\mathsf{ma}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) \tag{9.43}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails if an AMA nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. Similar to Equation 9.3 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_2] - \Pr[\mathsf{Win}^{\mathsf{ma}}_1]| \leq \frac{Nq_s^2}{2^\kappa} \tag{9.44}$$

Note that this prevents attacks where Π_i^s during any session of the AMA protocol receives a replayed message of the form $U_j|m|\bar{\sigma}_j$ or $U_j|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π_i^s does not accept unless it successfully verifies all required σ_j for all $U_j \in \mathtt{pid}_i^s$ in the AMA protocol of C-AMA. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts there exists a corresponding instance oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$. Thus, according to Definition 8.15 $\mathcal{A}$ wins in this game only if any of these oracles has accepted with $K_j^s \neq K_i^s$.

Assume that $\mathcal{A}$ wins in this game. Then Π_i^s and Π_j^s have accepted with $K_i^s = f_{k_i^s}(v_1)$ resp. $K_j^s = f_{k_j^s}(v_1)$ where k_i^s resp. k_j^s are corresponding keys computed during the execution of P and $K_i^s \neq K_j^s$. Having eliminated forgeries and replay attacks between the oracles of any two oracles of uncorrupted users we follow that messages exchanged between Π_i^s and Π_j^s have been delivered without any modification. In particular, oracle Π_i^s received the signature σ_j computed on $\mu_j = f_{k_j^s}(v_0)$ and Π_j^s received the signature σ_i computed on $\mu_i = f_{k_i^s}(v_0)$. Since both oracles have accepted we have $\mu_i = \mu_j$; otherwise oracles cannot have accepted because signature verification would fail. The probability that $\mathcal{A}$ wins in this game is given by

$$\begin{aligned}\Pr[K_i^s \neq K_j^s \wedge f_{k_i^s}(v_0) = f_{k_j^s}(v_0)] \\ = \Pr[f_{k_i^s}(v_1) \neq f_{k_j^s}(v_1) \wedge f_{k_i^s}(v_0) = f_{k_j^s}(v_0)] \leq q_s\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa).\end{aligned}$$

Thus

$$\Pr[\mathsf{Win}^{\mathsf{ma}}_2] \leq q_s\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa). \tag{9.45}$$

Considering Equations 9.43 to 9.45 we get the desired inequality

$$\begin{aligned}\mathsf{Succ}^{\mathsf{ma}}_{\texttt{C-AMA}_\texttt{P}}(\kappa) &= \Pr[\mathsf{Win}^{\mathsf{ma}}_0] \\ &\leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^\kappa} + q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa).\end{aligned}$$

□

9.6.3 Compiler for MA-Security and n-Contributiveness

In this section we show that compiler for MA-security C-MA can be combined with the compiler for contributiveness C-CON. The resulting compiler allows to save one communication round since the same nonces can be used for the MA-security and contributiveness.

Definition 9.24 (Compiler for MA-Security and n-Contributiveness C-MACON). *Let* P *be a GKE protocol from Definition 8.4,* $\pi : \{0,1\}^\kappa \to \{0,1\}^\kappa$ *a permutation,* $F := \left\{f_k\right\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ *a function ensemble with domain and range* $\{0,1\}^\kappa$*, and* $\Sigma := (\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *a digital signature scheme. A* compiler for MA-security and n-contributiveness*, denoted* $\texttt{C-MACON}_\texttt{P}$*, consists of the algorithm* INIT *and a two-round protocol* MACON *defined as follows:*

INIT*: In the initialization phase each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk_i', pk_i') *using* $\Sigma.\texttt{Gen}(1^\kappa)$*. This is in addition to any key pair* (sk_i, pk_i) *used in* P*.*

MACON*: After an oracle* $\Pi_i^s \in \mathcal{G}$ *computes* k_i^s *in the execution of* P *it chooses a random* MACON nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to every oracle* Π_j^s *with* $U_j \in \texttt{pid}_i^s$*. After* Π_i^s *receives* $U_j|r_j$ *from* Π_j^s *with* $U_j \in \texttt{pid}_i^s$ *it checks whether* $|r_j| \stackrel{?}{=} \kappa$*. If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes* $\rho_1 := f_{k_i^s \oplus \pi(r_1)}(v_0)$ *and each* $\rho_l := f_{\rho_{l-1}\oplus\pi(r_l)}(v_0)$ *for all* $l = \{2,\ldots,n\}$ *where* v_0 *is a public value, defines the intermediate key* $K_i^s := \rho_n$ *and* $\texttt{sid}_i^s := r_1|\ldots|r_n$*, computes a* MACON token $\mu_i := f_{K_i^s}(v_1)$ *where* v_1 *is a public value, a signature* $\sigma_i := \Sigma.\texttt{Sign}(sk_i', \mu_i|\texttt{sid}_i^s|\texttt{pid}_i^s)$ *and sends* $U_i|\sigma_i$ *to every oracle* Π_j^s *with* $U_j \in \texttt{pid}_i^s$*. Then it erases every other private information from* $\texttt{state}_i^s$ *(including* k_i^s *and each* ρ_l*,* $l \in [1,n]$*) except for* K_i^s*.*
After oracle Π_i^s *receives* $U_j|\sigma_j$ *from* Π_j^s *with* $U_j \in \texttt{pid}_i^s$ *it checks whether* $\Sigma.\texttt{Verify}(pk_j',$ $\mu_i|\texttt{sid}_i^s|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$*. If this verification fails then* Π_i^s *turns into a stand-by state without accepting; otherwise after* Π_i^s *has received and verified these messages from all other partnered oracles it computes the session group key* $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ *where* $v_2 \neq v_1$ *is another public value, erases every other private information from* $\texttt{state}_i^s$ *(including* K_i^s*), and accepts with* $\mathbf{K}_i^s$*.*

Intuitively, MACON nonces are supposed to randomize and ensure contributiveness for the intermediate value K which is in turn used to derive the resulting session group key $\mathbf{K}$. MACON tokens are then used for the purpose of key confirmation.

Theorem 9.25 (MA-Security of $\texttt{C-MACON}_\texttt{P}$). *For any GKE protocol* P *if* Σ *is EUF-CMA and* F *is collision-resistant then* $\texttt{C-MACON}_\texttt{P}$ *is* MAGKE*, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\texttt{C-MACON}_\texttt{P}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^\kappa} + q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa),$$

where q_s *is the total number of executed protocol sessions.*

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}_i^{\mathsf{ma}}$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator Δ.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\texttt{C-MACON}_\texttt{P}}(\kappa)$ played between Δ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π_i^s accepts with $\mathbf{K}_i^s$ and another user $U_j \in \texttt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts and either does not have a corresponding oracle Π_j^s with $(\texttt{pid}_j^s, \texttt{sid}_j^s) = (\texttt{pid}_i^s, \texttt{sid}_i^s)$ or has such an oracle but this oracle accepts with $\mathbf{K}_i^s \neq \mathbf{K}_j^s$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ except that the simulation fails if a MACON nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. According to Equation 9.3 we get

$$|\Pr[\mathsf{Win}_1^{\mathsf{ma}}] - \Pr[\mathsf{Win}_0^{\mathsf{ma}}]| \leq \frac{Nq_{\mathsf{s}}^2}{2^\kappa}. \tag{9.46}$$

This game implies that $\texttt{sid}_i^s$ computed by any uncorrupted user's oracle Π_i^s remains unique for each new session. Note that $\texttt{sid}_i^s$ is used to generate the signature σ_i in the MACON protocol of the compiler.

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the only exception that the simulation fails if $\mathcal{A}$ asks a $Send$ query on a message $U_i|\sigma$ such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain,

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq N\mathsf{Succ}_{\Sigma}^{\texttt{euf-cma}}(\kappa). \tag{9.47}$$

Note that this prevents attacks where Π_i^s during any session of the MACON protocol receives a replayed message of the form $U_j|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π_i^s does not accept unless it successfully verifies σ_j for all $U_j \in \texttt{pid}_i^s$ in the MACON protocol of C-MACON. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \texttt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts there exists a corresponding instance oracle Π_j^s with $(\texttt{pid}_j^s, \texttt{sid}_j^s) = (\texttt{pid}_i^s, \texttt{sid}_i^s)$. Thus, according to Definition 8.15 $\mathcal{A}$ wins in this game only if any of these oracles has accepted with $\mathbf{K}_i^s \neq \mathbf{K}_j^s$.

Assume that $\mathcal{A}$ wins in this game. Then, Π_i^s and Π_j^s have accepted with $\mathbf{K}_i^s = f_{K_i^s}(v_2)$ resp. $\mathbf{K}_i^s = f_{K_j^s}(v_2)$ where K_i^s resp. K_j^s are corresponding temporary keys computed during the execution of MACON, and $\mathbf{K}_i^s \neq \mathbf{K}_j^s$. Having eliminated forgeries and replay attacks between the oracles of any two oracles of uncorrupted users we follow that messages exchanged between Π_i^s and Π_j^s have been delivered without any modification. In particular, oracle Π_i^s received the signature σ_j computed on $\mu_j = f_{K_j^s}(v_1)$ and Π_j^s received the signature σ_i computed on $\mu_i = f_{K_i^s}(v_1)$. Since both oracles have accepted we have $\mu_i = \mu_j$; otherwise oracles cannot have accepted because signature verification would fail. The probability that $\mathcal{A}$ wins in this game is given by

$$\Pr[\mathbf{K}_i^s \neq \mathbf{K}_j^s \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] = \Pr[f_{K_i^s}(v_2) \neq f_{K_j^s}(v_2) \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).$$

Hence,

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa). \tag{9.48}$$

Considering Equations 9.47 to 9.48 we get the desired inequality

$$\mathsf{Succ}^{\mathsf{ma}}_{\texttt{C-MACON}_\texttt{P}}(\kappa) = \Pr[\mathsf{Win}_0^{\mathsf{ma}}] \leq N\mathsf{Succ}_{\Sigma}^{\texttt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).$$

□

In the following we argue that the compiler C-MACON provides n-contributiveness for the compiled protocol. The proof strategy is the same as in Theorem 9.11, i.e., we consider the probability that $\mathcal{A}$ is able to influence an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$, $i^* \in [1,n]$ to accept some $\tilde{K}$ by considering its ability to influence $\Pi_{i^*}^s$ to compute any value ρ_l, $i^* \leq l \leq n$.

Theorem 9.26 (n-Contributiveness of $\mathtt{C\text{-}MACON}_\mathtt{P}$). *For any GKE protocol* P *if π is one-way and F is collision-resistant pseudo-random then $\mathtt{C\text{-}MACON}_\mathtt{P}$ is n-CGKE, and*

$$\mathsf{Succ}^{\mathsf{con}-n}_{\mathtt{C\text{-}MACON}_\mathtt{P}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N+2)q_s\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_s\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_s\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa),$$

where q_s is the total number of executed protocol sessions.

Remark 9.27. Note that some arguments in this proof are intuitive for the same reasons as mentioned in Remark 9.12.

Proof (partially informal). In the following we consider an adversary $\mathcal{A}$ from Definition 8.18. Assume that $\mathcal{A}$ wins in $\mathsf{Game}^{\mathsf{con}-n}_{\mathtt{C\text{-}MACON}_\mathtt{P}}(\kappa)$ (which event we denote $\mathsf{Win}^{\mathsf{con}}$). Then at the end of the stage prepare it has returned $\tilde{\mathbf{K}}$ such that in the stage attack an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$ accepted with $\mathbf{K}_{i^*}^s = \tilde{\mathbf{K}}$. According to the construction of $\mathbf{K}_{i^*}^s$ we follow that $\tilde{\mathbf{K}} = f_{K_{i^*}^s}(v_2)$ computed by $\Pi_{i^*}^s$, and consider the following games.

Game $\mathbf{G}_0$. This is the real game $\mathsf{Game}^{\mathsf{con}-n}_{\mathtt{C\text{-}MACON}_\mathtt{P}}(\kappa)$, in which the honest players are replaced by a simulator.

Game $\mathbf{G}_1$. In this game we abort the simulation if the same MACON nonce r_i is used by any honest oracle Π_i^s in two different sessions. Considering r_i being uniform for every honest oracle Π_i^s and since there are at most N users, we have

$$\Pr[\mathsf{Win}^{\mathsf{con}}_0] - \Pr[\mathsf{Win}^{\mathsf{con}}_1] \leq \frac{Nq_s^2}{2^\kappa}. \tag{9.49}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage.[8] We can argue by intuition exactly as in the proof of Theorem 9.11. According to Equation 9.17 we have

$$\Pr[\mathsf{Win}^{\mathsf{con}}_1] - \Pr[\mathsf{Win}^{\mathsf{con}}_2] \leq q_s\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_s\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + \frac{q_s}{2^\kappa}. \tag{9.50}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that we do not need to consider the values ρ_l, $l < i^*$ computed by $\Pi_{i^*}^s$ since in order to compute $\mathbf{K}_{i^*}^s$ every honest oracle must first compute the whole sequence $\rho_1, \ldots, \rho_n$. Thus, it is sufficient to argue that the probability of $\mathcal{A}$ influencing any ρ_l, $l \geq i^*$, computed by $\Pi_{i^*}^s$ in any attack-ed session is negligible.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $K_{i^*}^s = \rho_n$ computed by $\Pi_{i^*}^s$ in any session of the attack

[8] Note, in $\mathbf{G}_0$ and $\mathbf{G}_1$ the adversary only outputs a value for the resulting group key. In $\mathbf{G}_2$ we consider the additional (in)ability of the adversary to output the value for ρ_{i^*}. Since we are only interested in the probability of the adversarial success under this "condition event" (without changing the game in case that this event occurs; see also Section 5.6.1) the simulator does not need to detect whether $\mathcal{A}$ is able to output the correct value or not. The same considerations are applicable to $\mathbf{G}_3$ w.r.t. $K_{i^*}^s$, and $\mathbf{G}_4$ w.r.t. $\mathbf{K}_{i^*}^s$.

stage. Again, the simulator does not need to detect whether this event occurs since both games proceed identical in any case.

Using the "hybrid technique" similar to the proof of Theorem 9.11 we obtain according to Equation 9.18

$$\Pr[\mathsf{Win}_2^{\mathsf{con}}] - \Pr[\mathsf{Win}_3^{\mathsf{con}}] \leq Nq_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + Nq_{\mathsf{s}}\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa) + \frac{Nq_{\mathsf{s}}}{2^\kappa}. \tag{9.51}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output $K_{i^*}^s$ computed by $\Pi_{i^*}^s$ in any session of the attack stage.

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $\mathbf{K}_{i^*}^s$ computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that in every attack-ed session, the honest oracle $\Pi_{i^*}^s$ computes $\mathbf{K}_{i^*}^s := f_{K_{i^*}^s}(v_2)$. Intuitively, since in the prepare stage $K_{i^*}^s$ is unknown to $\mathcal{A}$ (as observed in the previous game), $\mathcal{A}$'s probability to output $\mathbf{K}_{i^*}^s$ in that stage is bound by the probability that $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Hence,

$$\Pr[\mathsf{Win}_3^{\mathsf{con}}] - \Pr[\mathsf{Win}_4^{\mathsf{con}}] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + \frac{q_{\mathsf{s}}}{2^\kappa}. \tag{9.52}$$

Obviously, the probability of $\mathsf{Win}_4^{\mathsf{con}}$ is 0, meaning that the adversary did not output a correct value of $\tilde{\mathbf{K}}$ in the prepare stage.

Considering Equations 9.49 to 9.52 we obtain the desired inequality

$$\begin{aligned}\mathsf{Succ}_{\mathtt{C\text{-}MACON_P}}^{\mathsf{con}-n}(\kappa) &= \Pr[\mathsf{Win}_0^{\mathsf{con}}] \\ &\leq \frac{Nq_{\mathsf{s}}^2 + Nq_{\mathsf{s}} + 2q_{\mathsf{s}}}{2^\kappa} + (N+2)q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + Nq_{\mathsf{s}}\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa).\end{aligned}$$

□

Additionally, we show that C-MACON preserves (A)KE-security of the compiled protocol without considering the adversarial setting $(\mathtt{sbs}, \mathtt{scm})$.

Theorem 9.28 (AKE-Security of $\mathtt{C\text{-}MACON_P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs})$[9]$, (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any $\mathtt{AGKE}$-α protocol $\mathtt{P}$ if F is pseudo-random then $\mathtt{C\text{-}MACON_P}$ is also a $\mathtt{AGKE}$-α protocol, and*

$$\mathsf{Adv}_{\alpha,\beta,\mathtt{C\text{-}MACON_P}}^{\mathsf{ake}}(\kappa) \leq 2N\mathsf{Succ}_\Sigma^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}}(\kappa) + 2(N+2)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 7$ and corresponding events $\mathsf{Win}_i^{\mathsf{ake}}$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}_{\alpha,\beta,\mathtt{C\text{-}MACON_P}}^{\mathsf{ake}-b}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}_{\alpha,\beta,\mathtt{C\text{-}MACON_P}}^{\mathsf{ake}-b}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. We assume that the $Test$ query is asked to an α-fresh oracle Π_i^s.

[9] Only in case of static protocols due to Conjecture 8.31.

Keep in mind that on the test query the adversary receives either a random string or a session group key $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ computed by Π_i^s.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulator fails and sets b' at random if $\mathcal{A}'$ asks a $Send$ query on some $U_i|m|\sigma$ such that σ is a valid signature on m that has not been previously output by an honest oracle Π_i^s before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}'$ outputs a successful forgery. According to Equation 9.2 we obtain

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_0^{\mathsf{ake}}]| \leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa). \tag{9.53}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulator fails and sets b' at random if a MACON nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. According to Equation 9.3 we obtain

$$|\Pr[\mathsf{Win}_2^{\mathsf{ake}}] - \Pr[\mathsf{Win}_1^{\mathsf{ake}}]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} \tag{9.54}$$

Note that this game excludes replay attacks in the MACON protocol because $\mathtt{sid}_i^s$ is unique for each new session.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q_{\mathsf{s}}^* \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q_{s}^*-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2}. \tag{9.55}$$

Game $\mathbf{G}_4$. In this game we consider the simulator Δ as an active adversary against the AKE-security of $\mathtt{P}$ that participates in $\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}-1}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with the real session group key k_i^s computed in $\mathtt{P}$. In the following we show how Δ answers the queries of $\mathcal{A}$. Note that Δ and $\mathcal{A}$ operate in the same adversarial setting (α, β) and are both active. In fact, we never require Δ to execute operations of $\mathtt{P}$ itself but to forward each related query of $\mathcal{A}$ as its own query and respond accordingly. Δ itself performs only those additional computations that are necessary for C-MACON.

The simulator Δ which is initialized with the public keys $\{pk_i'\}_{U_i \in \mathcal{U}}$ (if any are given in the original protocol $\mathtt{P}$) generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^{\kappa})$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. Then, Δ runs $\mathcal{A}$ as a sub-routine and answers its queries.

$Setup$ queries: If $\mathcal{A}$ invokes a protocol session via a $Setup(\mathcal{S})$ query then Δ forwards this query as its own and obtains the transcript T of the $\mathtt{P}.\mathtt{Setup}(\mathcal{S})$ execution. The goal of Δ is to extend T to a transcript T' for the corresponding execution of $\mathtt{C\text{-}MACON}_{\mathtt{P}}.\mathtt{Setup}(\mathcal{S})$. Therefore, $\mathcal{A}$ chooses random nonces r_i for each Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{S}$ and computes $\mathtt{sid} := r_1|\ldots|r_n$. In order to build T' the simulator appends $\{U_i|r_i\}_{1 \leq i \leq n}$ to T. Next, if the invoked session is the q_{s}^*-th session then Δ asks own $Test$ query to any oracle activated via the $Setup$ query and obtains (real) k. Otherwise, if the session is not the q_{s}^*-th session then Δ asks own $RevealKey$ query to any of the mentioned oracles and obtains real k. Hence, in any case Δ knows real k which it then uses to compute the sequence $\rho_1, \ldots, \rho_n$ (note that $K = \rho_n$) and the MACON token $\mu := f_K(v_1)$. Then, Δ computes $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu|\mathtt{sid}|\mathtt{pid})$ and appends messages of the form $\{U_i|\sigma_i\}_{1 \leq i \leq n}$ to T'. Then, Δ computes $\mathbf{K} := f_K(v_2)$ and gives

T' to $\mathcal{A}$.

$Join^+$ and $Leave^+$ queries: These queries are answered similar to the $Setup$ queries so that Δ is able to compute the extended transcript T' as well as μ and $\mathbf{K}$ for the operation execution of $\texttt{C-MACON}_\texttt{P}.\texttt{Join}^+$ or $\texttt{C-MACON}_\texttt{P}.\texttt{Leave}^+$, respectively.

$Send$ queries: Similar to the proof of Theorem 9.8, by $Send_{S0}$, $Send_{J0}$, and $Send_{L0}$ we define the $Send$ query which invokes a new operation execution of $\texttt{C-MACON}_\texttt{P}$. These and all further $Send$ queries related to the execution of the operations of P are forwarded by Δ as own queries and answered accordingly.

By queries $Send_{SF}$, $Send_{JF}$, and $Send_{LF}$ to an oracle Π_i^s we define the *final* $Send$ queries of $\mathcal{A}$ concerning the execution of $\texttt{P.Setup}$, $\texttt{P.Join}^+$, or $\texttt{P.Leave}^+$, respectively, which results in Π_i^s having computed k_i^s in the operation of P. This means that all further valid $Send$ queries to Π_i^s would be related to the additional communication rounds of the MACON protocol. When $\mathcal{A}$ asks one these final $Send$ queries the simulator forwards it as its own query to an appropriate oracle Π_i^s implying the computation of k_i^s. Similar to the description of the $Setup$, $Join^+$, and $Leave^+$ queries above, Δ asks own $Test$ query (if the received $Send$ query is addressed to some participant of the $q_\texttt{S}^*$-th session) or $RevealKey$ query (in all other sessions) to obtain the real intermediate key k_i^s. Then, Δ chooses $r_i \in_R \{0,1\}^\kappa$ and responds with $U_i|r_i$.

By queries $Send_{SF+}$, $Send_{JF+}$, and $Send_{LF+}$ to an oracle Π_i^s we define the $Send$ queries of $\mathcal{A}$ of the form $Send(op, \Pi_i^s, (U_1|r_1)|\ldots|(U_n|r_n))$ where $op \in \{\texttt{'setup'}, \texttt{'join'}, \texttt{'leave'}\}$ and n is the (updated) number of operation participants whereby $(U_i|r_i)$ is not part of the query message. Note that Π_i^s must have received a $Send_{SF}$, $Send_{JF}$, or $Send_{LF}$ query before the corresponding $Send_{SF+}$, $Send_{JF+}$, or $Send_{LF+}$ query; otherwise the query is unexpected. When $\mathcal{A}$ asks one of these $Send$ queries the simulator computes the sequence $\rho_1, \ldots, \rho_n$, defines $K_i^s := \rho_n$, computes the MACON token $\mu_i := f_{K_i^s}(v_1)$, the signature $\sigma_i := \Sigma.\texttt{Sign}(sk_i', \mu_i|\texttt{sid}_i^s|\texttt{pid}_i^s)$ and responds with $U_i|\sigma_i$ to $\mathcal{A}$.

The last $Send$ query to Π_i^s is independent of the concrete operation op and has the form $Send(op, \Pi_i^s, (U_1|\sigma_1)|\ldots|(U_n|\sigma_n))$ whereby $(U_i|r_i)$ is not part of the query message. Δ checks all received signatures and computes $\mathbf{K}_i^s := f_{K_i^s}(v_2)$.

$Corrupt$ queries: When $\mathcal{A}$ asks a $Corrupt(U_i)$ query Δ forwards own $Corrupt(U_i)$ query to obtain sk_i and replies with (sk_i, sk_i'). Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $Corrupt$ queries.

$RevealState$ queries: When $\mathcal{A}$ asks a $RevealState(\Pi_i^s)$ query Δ forwards own $RevealState(\Pi_i^s)$ query to obtain $\texttt{state}_i^s$. If Π_i^s is waiting for the last $Send$ query of the form $Send(op, \Pi_i^s, (U_1|\sigma_1)|\ldots|(U_n|\sigma_n))$ then Δ inserts K_i^s (which is not erased yet) into $\texttt{state}_i^s$ and returns it to $\mathcal{A}$; otherwise it simply forwards $\texttt{state}_i^s$ to $\mathcal{A}$. Note that Δ and $\mathcal{A}$ have identical restrictions concerning the $RevealState$ queries.

$RevealKey$ queries: When $\mathcal{A}$ asks a $RevealKey(\Pi_i^s)$ query Δ checks that Π_i^s has accepted; otherwise an empty string is returned. Then, Δ returns the session group key $\mathbf{K}_i^s$. Note that Δ is always able to do this since it executes the last steps of the MACON protocol itself, i.e., the computation of $\rho_1, \ldots, \rho_n$, μ_i, and $\mathbf{K}_i^s$ for every (honest) oracle Π_i^s.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s

that has participated in the q_{s}^*-th session. The simulator Δ already knows K_i^s since this value is computed by Δ for every (honest) oracle Π_i^s in the simulation. Thus, Δ chooses a random bit $b \in_R \{0,1\}$ and returns K_i^s if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This is a perfect simulation for $\mathcal{A}$. Since Δ uses the real k_i^s to derive $\mathbf{K}_i^s$ we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{9.56}$$

Game $\mathbf{G}_5$. In this game we consider the simulator Δ as an active adversary against the AKE-security of $\mathtt{P}$ that participates in $\mathsf{Game}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}-0}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}_{\alpha,\beta,\mathtt{P}}^{\mathsf{ake}}(\kappa). \tag{9.57}$$

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session each ρ_i, $i = 1,\ldots,n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that K is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{6,l}$, $l = 1,\ldots,n+1$ such that $\mathbf{G}'_{6,1} = \mathbf{G}_5$ and $\mathbf{G}'_{6,n+1} = \mathbf{G}_6$. That is, in the q_{s}^*-th session in each $\mathbf{G}'_{6,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the compiler whereas in $\mathbf{G}'_{6,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 1,\ldots,n-1$ by a random bit string implies uniform distribution of the PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that k used to compute ρ_1 is already uniform according to Game $\mathbf{G}_5$.

Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq N\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.58}$$

Note that since each ρ_i is erased at the end of each $\mathtt{C\text{-}MACON}_\mathtt{P}$ execution the adversary $\mathcal{A}$ cannot learn any ρ_i used in the α-fresh q_{s}^*-th session since the adversarial setting (α,β) disallows $RevealState$ queries and prevents active participation of $\mathcal{A}$ on behalf of corrupted users in α-fresh sessions as mentioned in Remark 8.10.

Game $\mathbf{G}_7$. This game is identical to Game $\mathbf{G}_6$ except that in the q_{s}^*-th session $\mathbf{K}$ and the MACON token μ are replaced by random values sampled from $\{0,1\}^\kappa$. Recall that K used to compute $\mathbf{K}$ and μ is uniform according to Game $\mathbf{G}_6$. Notice that this implies that $\mathbf{K}$ is uniformly distributed in this game. Obviously,

$$|\Pr[\mathsf{Win}_7^{\mathsf{ake}}] - \Pr[\mathsf{Win}_6^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.59}$$

Since $\mathbf{K}$ is uniformly distributed $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_7^{\mathsf{ake}}] = \frac{1}{2} \tag{9.60}$$

Considering Equations 9.53 to 9.60 we get

$$\begin{aligned}\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-MACON}_\mathtt{P}}(\kappa)=b] &= \Pr[\mathsf{Win}^{\mathsf{ake}}_0]\\ &= N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa)+\frac{Nq_\mathsf{s}^2}{2^\kappa}+q_\mathsf{s}\left(\Pr[\mathsf{Win}^{\mathsf{ake}}_1]-\frac{1}{2}\right)+\frac{1}{2}\\ &\leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa)+\frac{Nq_\mathsf{s}^2}{2^\kappa}+q_\mathsf{s}\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{P}}(\kappa)+(N+2)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa)+\frac{1}{2}.\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-MACON}_\mathtt{P}}(\kappa)\leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa)+\frac{Nq_\mathsf{s}^2}{2^{\kappa-1}}+2q_\mathsf{s}\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{P}}(\kappa)+2(N+2)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa).$$

□

The same proof can also be used to show that C-MACON preserves KE-security of the original GKE protocol. In this case $\mathcal{A}$ is passive and does not have access to the $Send$ queries. Hence, no forgeries and replay attacks need to be considered.

Theorem 9.29 (KE-Security of $\texttt{C-MACON}_\mathtt{P}$). *Let (α,β) be an adversarial setting sampled from $\{(\emptyset,\texttt{wcm}),\ (\texttt{wbs},\texttt{wcm-bs})$[10]$,\ (\texttt{wfs},\texttt{wcm-fs}),\ (\texttt{sfs},\texttt{scm})\}$. For any GKE-$\alpha$ protocol P if F is pseudo-random then $\texttt{C-MACON}_\mathtt{P}$ is also a GKE-α protocol, and*

$$\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\texttt{C-MACON}_\mathtt{P}}(\kappa)\leq 2q_\mathsf{s}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa)+2(N+2)q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa),$$

where q_s is the total number of executed protocol sessions.

9.6.4 Compiler for AKE-, MA-Security and n-Contributiveness

In this section we show that it is possible to use ideas of the compilers C-A, C-MA and C-CON to design a single compiler which provides all three security requirements described by our model, that is, AKE-, MA-security and n-contributiveness for any KE-secure GKE protocol. This compiler uses the same nonces for the authentication, MA-security and contributiveness.

Definition 9.30 (Compiler for AKE-, MA-Security and n-Contributiveness C-AMACON). *Let P be a GKE protocol from Definition 8.4, $\Sigma:=(\texttt{Gen},\texttt{Sign},\texttt{Verify})$ a digital signature scheme, $\pi:\{0,1\}^\kappa\to\{0,1\}^\kappa$ a permutation, $F:=\{f_k\}_{k\in\{0,1\}^\kappa}$, $\kappa\in\mathbb{N}$ a function ensemble with domain and range $\{0,1\}^\kappa$. A* compiler for AKE-, MA-security and n-contributiveness*, denoted C-AMACON, consists of an algorithm INIT and a protocol AMACON defined as follows:*

> INIT*: In the initialization phase each $U_i\in\mathcal{U}$ generates own private/public key pair (sk_i',pk_i') using $\Sigma.\texttt{Gen}(1^\kappa)$. This is in addition to any key pair (sk_i,pk_i) used in* P.
>
> AMACON*: An interactive protocol between the oracles $\Pi_1^s,\ldots,\Pi_n^s$ in $\mathcal{G}$ invoked prior to any operation of* P*. Each Π_i^s chooses an* AMACON nonce $r_i\in_R\{0,1\}^\kappa$ *and sends $U_i|r_i$ to every oracle Π_j^s with $U_j\in\texttt{pid}_i^s$. After Π_i^s receives $U_j|r_j$ from $U_j\in\texttt{pid}_i^s$ it checks whether $|r_j|\stackrel{?}{=}\kappa$. If this verification fails then Π_i^s turns into a stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it computes $\texttt{sid}_i^s:=r_1|\ldots|r_n$. Then, it invokes the operation execution of* P *and proceeds as follows:*

[10] Only in case of static protocols due to Conjecture 8.31.

- *If Π_i^s in* $\mathtt{P}$ *is supposed to output a message $U_i|m$ then in* $\mathtt{C\text{-}AMACON_P}$ *it computes additionally $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', m|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ and outputs a modified message $U_i|m|\sigma_i$.*
- *If Π_i^s receives a message $U_j|m|\sigma_j$ from Π_j^s with $U_j \in \mathtt{pid}_i^s$ it checks whether $\Sigma.\mathtt{Verify}(pk_j', m|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If this verification fails then Π_i^s turns into a stand-by state without accepting; otherwise it proceeds according to the specification of the executed operation of* $\mathtt{P}$ *upon receiving $U_j|m$.*
- *After Π_i^s computes k_i^s in the execution of* $\mathtt{P}$ *it computes $\rho_0 := f_{k_i^s}(v_0)$, and each $\rho_l := f_{\rho_{l-1}\oplus\pi(r_l)}(v_0)$ for all $l = \{1,\ldots,n\}$, where v_0 is a public value, defines the intermediate key $K_i^s := \rho_n$, computes an* AMACON token *$\mu_i := f_{K_i^s}(v_1)$ where v_1 is a public value, a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ and sends $U_i|\sigma_i$ to every oracle Π_j^s with $U_j \in \mathtt{pid}_i^s$. Then it erases every other private information from $\mathtt{state}_i^s$ (including k_i^s and each ρ_l, $l = \{0,\ldots,n\}$) except for K_i^s.*
  After Π_i^s receives $U_j|\sigma_j$ from Π_j^s with $U_j \in \mathtt{pid}_i^s$ it checks whether $\Sigma.\mathtt{Verify}(pk_j', \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If this verification fails then Π_i^s turns into a stand-by state without accepting; otherwise after Π_i^s has received and verified these messages from all other partnered oracles it computes the session group key $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ where $v_2 \neq v_1$ is another public value, erases every other private information from $\mathtt{state}_i^s$ (including K_i^s), and accepts with $\mathbf{K}_i^s$.

We start with the AKE-security of C-AMACON. As for the compilers C-A and C-AMA we do not consider $(\mathtt{sbs}, \mathtt{scm})$ as a possible adversarial setting.

Theorem 9.31 (AKE-Security of Static $\mathtt{C\text{-}AMACON_P}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any* ***static*** $\mathtt{GKE}$-α *protocol* $\mathtt{P}$ *if Σ is EUF-CMA and F is pseudo-random then* $\mathtt{C\text{-}AMACON_P}$ *is* $\mathtt{AGKE}$-α*, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}AMACON_P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 2(N+3)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0,\ldots,7$ and corresponding events $\mathsf{Win}_i^{\mathsf{ake}}$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}AMACON_P}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}AMACON_P}}(\kappa)$ (see Definition 8.12) played between a simulator Δ and an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π_i^s. Keep in mind that in response to the $Test$ query the adversary receives either a random string or a session group key $\mathbf{K}_i^s$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_0^{\mathsf{ake}}]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa). \tag{9.61}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if an AMACON nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. According to Equation 9.3 we get

$$|\Pr[\mathsf{Win}_2^{\mathsf{ake}}] - \Pr[\mathsf{Win}_1^{\mathsf{ake}}]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} \tag{9.62}$$

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q_{\mathsf{s}}^* \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q_{s}^*-th session then the simulation fails and bit b' is set at random. Similar to Equation 9.4 we get

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_{\mathsf{s}} \left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2} \right) + \frac{1}{2}. \tag{9.63}$$

Game $\mathbf{G}_4$. In this game we consider the simulator Δ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}^{\mathsf{ke}-1}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with the proper session group key k_i^s. In the following we show how Δ answers the queries of $\mathcal{A}$. We extend the simulation described in the proof of Theorem 9.3 by additional post-computations needed to derive the session group key K. Similar to the proof of Theorem 9.3 we focus on the adversarial settings $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$.

Δ corrupts every user $U_i \in \mathcal{U}$ to obtain the long-lived key pair (sk_i, pk_i) used in the original protocol $\mathtt{P}$ (if any such keys are defined). Then, Δ generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^{\kappa})$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. Δ initializes the list $TList$ and runs $\mathcal{A}$ as a subroutine.

$Setup$ queries: These queries are processed as described in the proof of Theorem 9.3, i.e., Δ performs every operation execution itself saving $(\mathtt{sid}, \perp)$ in the $TList$ except for the operation in the q_{s}^*-th session for which it asks own $Setup$ query. After Δ receives the execution transcript T of $\mathtt{P}.\mathtt{Setup}$ in the q_{s}^*-th session it computes T' for the corresponding execution of $\mathtt{C\text{-}AMACON}_{\mathtt{P}}.\mathtt{Setup}$ by extending T with the initial messages of the form $\{U_i|r_i\}_{1 \leq i \leq n}$ and digital signatures. As in the proof of Theorem 9.21 the simulator must extend T' with the messages that include signatures on the AMACON tokens. For this purpose Δ asks own $Test$ query to any oracle activated via the $Setup$ query in the q_{s}^*-th session and obtains (real) k. Then, Δ computes $\rho_0, \ldots, \rho_n$ and obtains K which it in turn uses to compute $\mu := f_K(v_1)$. Then, Δ computes $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu|\mathtt{sid}|\mathtt{pid})$ and appends messages of the form $\{U_i|\sigma_i\}_{1 \leq i \leq n}$ to T'. Finally, Δ saves $(\mathtt{sid}, T')$ in $TList$ and gives T' to $\mathcal{A}$.

$Send$ queries: These queries are also processed as described in the proof of Theorem 9.3. All sessions invoked via a $Send$ query are executed by Δ itself except for the q_{s}^*-th session for which Δ asks own $Setup$ query. Similar to the description of the $Setup$ query for the q_{s}^*-th session above Δ "patches" the transcript T with digital signatures and additional messages that contain digital signatures on the computed AMACON tokens to obtain the transcript T' for the corresponding operation execution of $\mathtt{C\text{-}AMACON}_{\mathtt{P}}.\mathtt{Setup}$ (note that in order to compute the AMACON token Δ asks own $Test$ query to obtain k_i^s), saves $(\mathtt{sid}_i^s, T')$ in $TList$, and replies to $\mathcal{A}$ with the appropriate message taken from T'. All other $Send$ queries related to the oracles that participate in the q_{s}^*-th session are answered from the predefined transcript T'.

$Corrupt$ queries: If $\mathcal{A}$ asks a query of the form $Corrupt(U_i)$ then Δ replies with (sk_i, sk_i).

$RevealState$ queries: If $\mathcal{A}$ asks a query of the form $RevealState(\Pi_i^s)$ then Δ finds an entry $(\mathtt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ it means that Δ executes the protocol itself and is, therefore, able to answer this query directly. If $T^* = T'$ then Δ checks whether Π_i^s has already accepted.

In this case Δ asks its own $RevealState$ query to obtain $\mathtt{state}_i^s$ and replies accordingly. Note that if Π_i^s has not yet accepted in $\mathtt{C\text{-}AMACON}_{\mathtt{P}}.\mathtt{Setup}$ then an empty string is returned.

$RevealKey$ queries: If $\mathcal{A}$ asks a query of the form $RevealKey(\Pi_i^s)$ then Δ checks that Π_i^s has accepted; otherwise an empty string is returned. Next, Δ finds an entry $(\mathtt{sid}_i^s, T^*)$ in $TList$. If $T^* = \perp$ then $\mathcal{A}'$ is able to answer with $\mathbf{K}_i^s$ directly since the protocol execution with Π_i^s has been done by Δ. If $T^* = T'$ then the query is invalid since no $RevealKey$ queries are allowed to the oracles that have accepted in the q_{s}^*-th session.

$Test$ query: Note that in this game we are dealing with the $Test$ query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. Similar to the proof of Theorem 9.21 the simulator Δ already knows k_i^s since it has already asked own $Test$ query to build the transcript T' straight after the invocation of the q_{s}^*-th session. Thus, Δ computes the sequence $\rho_0, \ldots, \rho_n$, sets $K_i^s = \rho_n$, and derives the resulting session group key $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ as specified in $\mathtt{C\text{-}AMACON}_{\mathtt{P}}$. Finally, Δ chooses a random bit $b \in_R \{0,1\}$ and returns $\mathbf{K}_i^s$ if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This provides a perfect simulation for $\mathcal{A}$. Since Δ uses the real k_i^s to derive $\mathbf{K}_i^s$ we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{9.64}$$

When dealing with the adversarial settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ the above simulation can be simplified since no $Corrupt$ queries need to be considered. Similar to the proof of Theorem 9.3 Δ can forward all queries of $\mathcal{A}$ and obtain a transcript T for the operation execution of $\mathtt{P}$. Although, Δ can "patch" T with random nonces and digital signatures it cannot extend it with messages that contain digital signatures on the AMACON tokens without obtaining the session group key k needed to compute K and consequently the AMACON token μ. Thus, Δ must ask own $RevealKey$ query in order to build the complete transcript T'. Therefore, Δ must guess the q_{s}^*-th session. Otherwise, Δ risks that $\mathcal{A}$ asks its $Test$ query for the session which is already unfresh from the perspective of Δ. Thus, in contrast to the proof of Theorem 9.18 but similar to the proof of Theorem 9.21 Game $\mathbf{G}_3$ cannot be skipped when dealing with $(\emptyset, \mathtt{wcm})$ or $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$.

Game $\mathbf{G}_5$. In this game we consider the simulator Δ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}^{\mathsf{ke}-0}_{\alpha,\beta,\mathtt{P}}(\kappa)$, i.e., the $Test$ query of Δ to an accepted α-fresh oracle Π_i^s is answered with a random bit string instead of the real key k_i^s. Δ answers all queries of $\mathcal{A}$ exactly as described in Game $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa). \tag{9.65}$$

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session each ρ_i, $i = 0, \ldots, n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that $K = \rho_n$ is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{6,l}$, $l = 0, \ldots, n+1$ such that $\mathbf{G}'_{6,0} = \mathbf{G}_5$ and $\mathbf{G}'_{6,n+1} = \mathbf{G}_6$. That is, in the q_{s}^*-th session in each $\mathbf{G}'_{6,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the compiler whereas in $\mathbf{G}'_{6,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 0, \ldots, n-1$ by a random bit string implies uniform distribution of the

PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that k used to compute ρ_0 is already uniform according to Game $\mathbf{G}_5$.

Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq (N+1)\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.66}$$

Game $\mathbf{G}_7$. This game is identical to Game $\mathbf{G}_6$ except that in the q_{s}^*-th session $\mathbf{K}$ and the AMACON token μ are replaced by random values sampled from $\{0,1\}^\kappa$. Notice, this implies that $\mathbf{K}$ is uniformly distributed in these sessions. Recall that K used to compute $\mathbf{K}$ and μ is uniform according to Game $\mathbf{G}_6$. Obviously,

$$|\Pr[\mathsf{Win}_7^{\mathsf{ake}}] - \Pr[\mathsf{Win}_6^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{9.67}$$

Since $\mathbf{K}$ is uniformly distributed $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_7^{\mathsf{ake}}] = \frac{1}{2} \tag{9.68}$$

Considering Equations 9.61 to 9.68 we get:

$$\begin{aligned}
\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-AMACON}_\mathsf{P}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&\leq N\mathsf{Succ}_\Sigma^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\
&= N\mathsf{Succ}_\Sigma^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq N\mathsf{Succ}_\Sigma^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathsf{P}}(\kappa) + \\
&\quad (N+3)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-AMACON}_\mathsf{P}}(\kappa) \leq 2N\mathsf{Succ}_\Sigma^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathsf{P}}(\kappa) + 2(N+3)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa).$$

□

Similar to the compilers C-ACON and C-AMA the main problem when applying the reduction from Theorem 9.31 to dynamic GKE protocols occurs in Games $\mathbf{G}_4$ and $\mathbf{G}_5$ in the simulation of a passive adversary against the KE-security of the protocol. Considering the additional queries $Leave^+$ and $Join^+$ the above simulation fails for the similar reasons as described in the context of the compiler C-A. Nevertheless, we believe that C-AMACON preserves the α-freshness of dynamic GKE protocols due to the following reasons. First, the above simulation can be easily extended to the dynamic case for the adversarial setting $(\emptyset, \texttt{wcm})$ where no $Corrupt$ queries have to be considered. Second, nonces used to prevent replay attacks are chosen anew for each operation execution of the protocol. Third, the key k and all intermediate values ρ_i, $i = 0, \ldots, \rho_n$ (and consequently K) are erased at the end of each operation execution protecting the secrecy of $\mathbf{K}$ against strong corruptions in later operations. Note also that the AMACON tokens are derived via a pseudo-random function and do not reveal any information about the intermediate key K used to compute them. Therefore, we state the following conjecture for dynamic GKE protocols whereby excluding the adversarial setting $(\texttt{wbs}, \texttt{wcm-bs})$ as a consequence of Conjecture 8.31.

Conjecture 9.32 (AKE-Security of Dynamic C-AMACON$_{\mathtt{P}}$). *Let (α, β) be an adversarial setting sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$. For any **dynamic** GKE-α protocol P if Σ is EUF-CMA and F is pseudo-random then C-AMACON$_{\mathtt{P}}$ is AGKE-α, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}AMACON_P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 2(N+3)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Further, we show that C-AMACON provides MA-security for any GKE protocol.

Theorem 9.33 (MA-Security of C-AMACON$_{\mathtt{P}}$). *For any GKE protocol P if Σ is EUF-CMA and F is collision-resistant then C-AMACON$_{\mathtt{P}}$ is MAGKE, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{C\text{-}AMACON_P}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa}} + q_s\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}^{\mathsf{ma}}_i$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator Δ.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{C\text{-}AMACON_P}}(\kappa)$ played between a simulator Δ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π_i^s accepts with $\mathbf{K}_i^s$ and another user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts and either does not have a corresponding oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$ or has such an oracle but this oracle accepts with $\mathbf{K}_i^s \neq \mathbf{K}_j^s$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation aborts if $\mathcal{A}$ asks a $Send$ query on a message $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $Corrupt(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 9.2 we obtain,

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_1] - \Pr[\mathsf{Win}^{\mathsf{ma}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) \tag{9.69}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails if an AMACON nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. Similar to Equation 9.3 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_2] - \Pr[\mathsf{Win}^{\mathsf{ma}}_1]| \leq \frac{Nq_s^2}{2^{\kappa}} \tag{9.70}$$

Note that this eliminates attacks where Π_i^s during any session of the AMACON protocol receives a replayed message of the form $U_j|m|\bar{\sigma}_j$ or $U_j|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π_i^s does not accept unless it successfully verifies all required σ_j for all $U_j \in \mathtt{pid}_i^s$ in the AMACON protocol of C-AMACON. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts there exists a corresponding instance oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$. Thus, according to Definition 8.15 $\mathcal{A}$ wins in this game only if any of these oracles has accepted with $\mathbf{K}_i^s \neq \mathbf{K}_j^s$.

Assume that $\mathcal{A}$ wins in this game. Then Π_i^s and Π_j^s have accepted with $\mathbf{K}_i^s = f_{K_i^s}(v_2)$ resp. $\mathbf{K}_i^s = f_{K_j^s}(v_2)$ where K_i^s resp. K_j^s are corresponding temporary keys computed during the execution of AMACON, and $\mathbf{K}_i^s \neq \mathbf{K}_j^s$. Having eliminated forgeries and replay attacks between the oracles of any uncorrupted users we follow that messages exchanged between Π_i^s and Π_j^s

have been delivered without any modification. In particular, oracle Π_i^s received the signature σ_j computed on $\mu_j = f_{K_j^s}(v_1)$ and Π_j^s received the signature σ_i computed on $\mu_i = f_{K_i^s}(v_1)$. Since both oracles have accepted we have $\mu_i = \mu_j$; otherwise oracles cannot have accepted because signature verification would fail. The probability that $\mathcal{A}$ wins in this game is given by

$$\Pr[\mathbf{K}_i^s \neq \mathbf{K}_j^s \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] = \Pr[f_{K_i^s}(v_2) \neq f_{K_j^s}(v_2) \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).$$

Hence,

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa). \tag{9.71}$$

Considering Equations 9.69 to 9.71 we get the desired inequality

$$\mathsf{Succ}_{\mathtt{C\text{-}AMACON}_{\mathtt{P}}}^{\mathsf{ma}}(\kappa) = \Pr[\mathsf{Win}_0^{\mathsf{ma}}] \leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).$$

□

Finally, we argue that C-AMACON provides n-contributiveness for any GKE protocol. The proof strategy is the same as in Theorems 9.16 and 9.26, i.e., we consider the probability that $\mathcal{A}$ is able to influence an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$, $i^* \in [1, n]$ to accept some $\tilde{K}$ by considering its ability to influence $\Pi_{i^*}^s$ to compute any value ρ_l, $i^* \leq l \leq n$.

Theorem 9.34. *For any GKE protocol* P *if* π *is one-way and* F *is collision-resistant pseudo-random then* $\mathtt{C\text{-}AMACON}_{\mathtt{P}}$ *is* n-CGKE*, and*

$$\mathsf{Succ}_{\mathtt{C\text{-}AMACON}_{\mathtt{P}}}^{\mathsf{con}-n}(\kappa) \leq \frac{Nq_{\mathsf{s}}^2 + Nq_{\mathsf{s}} + 2q_{\mathsf{s}}}{2^\kappa} + (N+2)q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + Nq_{\mathsf{s}}\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa),$$

where q_{s} *is the total number of executed protocol sessions.*

Remark 9.35. Note that some arguments in this proof are intuitive for the same reasons as mentioned in Remark 9.12.

Proof (partially informal). In the following we consider an adversary $\mathcal{A}$ from Definition 8.18. Assume that $\mathcal{A}$ wins in $\mathsf{Game}_{\mathtt{C\text{-}AMACON}_{\mathtt{P}}}^{\mathsf{con}-n}(\kappa)$ (which event we denote $\mathsf{Win}^{\mathsf{con}}$). Then at the end of the stage prepare it has returned $\tilde{K}$ such that in the stage attack an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$ accepted with $\mathbf{K}_{i^*}^s = \tilde{K}$. According to the construction of $\mathbf{K}_{i^*}^s$ we follow that $\tilde{K} = f_{K_{i^*}^s}(v_2)$ with $K_{i^*}^s = \rho_n$ computed by $\Pi_{i^*}^s$, and consider the following games.

Game $\mathbf{G}_0$. This is the real game $\mathsf{Game}_{\mathtt{C\text{-}AMACON}_{\mathtt{P}}}^{\mathsf{con}-n}(\kappa)$, in which the honest players are replaced by a simulator Δ.

Game $\mathbf{G}_1$. In this game we abort the simulation if the same AMACON nonce r_i is used by any honest oracle Π_i^s in two different sessions. Considering r_i being uniform for every honest oracle Π_i^s and that there are at most N users, we have

$$\Pr[\mathsf{Win}_0^{\mathsf{con}}] - \Pr[\mathsf{Win}_1^{\mathsf{con}}] \leq \frac{Nq_{\mathsf{s}}^2}{2^\kappa}. \tag{9.72}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack

stage.[11] Arguing intuitively similar to the proof of Theorem 9.16 and considering Equation 9.26 we obtain

$$\Pr[\mathsf{Win}_1^{\mathsf{con}}] - \Pr[\mathsf{Win}_2^{\mathsf{con}}] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{q_{\mathsf{s}}}{2^\kappa}. \tag{9.73}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that we do not need to consider the values ρ_l, $l < i^*$ computed by $\Pi_{i^*}^s$ since in order to compute $\mathbf{K}_{i^*}^s$ every honest oracle must first compute the whole sequence $\rho_0, \ldots, \rho_n$. Thus, it is sufficient to argue that the probability of $\mathcal{A}$ influencing any ρ_l, $l \geq i^*$, computed by $\Pi_{i^*}^s$ in any attack-ed session is negligible.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $K_{i^*}^s = \rho_n$ computed by $\Pi_{i^*}^s$ in any session of the attack stage. Again, the simulator does not need to detect whether this event occurs since both games proceed identical in any case.

Using the "hybrid technique" similar to the proof of Theorem 9.16 we obtain according to Equation 9.27

$$\Pr[\mathsf{Win}_2^{\mathsf{con}}] - \Pr[\mathsf{Win}_3^{\mathsf{con}}] \leq N q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + N q_{\mathsf{s}}\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa) + \frac{N q_{\mathsf{s}}}{2^\kappa}. \tag{9.74}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output $K_{i^*}^s$ computed by $\Pi_{i^*}^s$ in any session of the attack stage.

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $\mathbf{K}_{i^*}^s$ computed by $\Pi_{i^*}^s$ in any session of the attack stage. Note that in every attack-ed session, the honest oracle $\Pi_{i^*}^s$ computes $\mathbf{K}_{i^*}^s := f_{K_{i^*}^s}(v_2)$. Intuitively, since in the prepare stage $K_{i^*}^s$ is unknown to $\mathcal{A}$ (as observed in the previous game), $\mathcal{A}$'s probability to output $\mathbf{K}_{i^*}^s$ in that stage is bound by the probability that $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Hence,

$$\Pr[\mathsf{Win}_3^{\mathsf{con}}] - \Pr[\mathsf{Win}_4^{\mathsf{con}}] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + \frac{q_{\mathsf{s}}}{2^\kappa}. \tag{9.75}$$

Obviously the probability of $\mathsf{Win}_4^{\mathsf{con}}$ is 0, meaning that the adversary did not output a correct value of $\tilde{\mathbf{K}}$ in the prepare stage.

Considering Equations 9.72 to 9.75 we obtain the desired inequality

$$\begin{aligned}\mathsf{Succ}_{\mathtt{C\text{-}AMACON}_{\mathtt{P}}}^{\mathsf{con}-n}(\kappa) &= \Pr[\mathsf{Win}_0^{\mathsf{con}}] \\ &\leq \frac{N q_{\mathsf{s}}^2 + N q_{\mathsf{s}} + 2 q_{\mathsf{s}}}{2^\kappa} + (N+2) q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + N q_{\mathsf{s}}\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa).\end{aligned}$$

□

[11] Note, in $\mathbf{G}_0$ and $\mathbf{G}_1$ the adversary only outputs a value for the resulting group key. In $\mathbf{G}_2$ we consider the additional (in)ability of the adversary to output the value for ρ_{i^*}. Since we are only interested in the probability of the adversarial success under this "condition event" (without changing the game in case that this event occurs; see also Section 5.6.1) the simulator does not need to detect whether $\mathcal{A}$ is able to output the correct value or not. The same considerations are applicable to $\mathbf{G}_3$ w.r.t. K_i^s, and $\mathbf{G}_4$ w.r.t. $\mathbf{K}_{i^*}^s$.

9.7 Summary

In this chapter we have proposed seven generic security-enhancing compilers (C-A, C-MA, C-CON, C-ACON, C-AMA, C-MACON, and C-AMACON) for GKE protocols with respect to the three major requirements (and combinations thereof) specified in our model in Chapter 8, i.e., the requirements of (A)KE- and MA-security, and n-contributiveness. Table 9.7 summarizes main security results and lists the number of the required communication rounds in addition to those of the

Table 9.1. Summary of the Generic Security-Enhancing Compilers for GKE Protocols

Compiler	(A)KE	MA	n-CON	Additional Rounds
C-A	Theo. 9.3 Conj. 9.4	–	–	1
C-MA	Theo. 9.8 Theo. 9.9	Theo. 9.7	–	2
C-CON	Theo. 9.13 Theo. 9.14	–	Theo. 9.11	1
C-ACON	Theo. 9.18 Conj. 9.19	–	Theo. 9.16	1
C-AMA	Theo. 9.21 Conj. 9.22	Theo. 9.23	–	2
C-MACON	Theo. 9.28 Theo. 9.29	Theo. 9.25	Theo. 9.26	2
C-AMACON	Theo. 9.31 Conj. 9.32	Theo. 9.33	Theo. 9.34	2

underlying GKE protocol. Obviously, every compiler requires a constant number of additional communication rounds. The first round of each compiler, in which all participants exchange their randomly chosen session nonces, is identical. Note that the concatenation of these nonces represents the unique session id since nonces are chosen anew for each invoked protocol operation. In compilers C-CON, C-ACON, C-MACON, and C-AMACON these nonces are additionally used for the derivation of the session group key ensuring the n-contributiveness of the compiled protocol.

Observe, that all described compilers are generic. Therefore, when applied to concrete GKE protocols, further efficiency optimizations may become possible, e.g., some messages of the compiler can possibly be interleaved with those of the underlying GKE protocol. For example, in our GKE protocol TDH1 described in the next chapter the first protocol message of every participant includes a random nonce and an ephemeral public value which is necessary for the computation of the session group key.

Chapter 10

Constant-Round GKE Protocol TDH1 Secure Against Strong Corruptions

In this chapter we propose two versions (static and dynamic) of our constant-round group key exchange protocol TDH1 (based on the informal ideas from [118, 171]) and prove its security under standard cryptographic assumptions using the computational model from Chapter 8.

10.1 Number-Theoretic Assumptions

Security of our TDH1 protocol bases on the Tree Decisional Diffie-Hellman (TDDH) assumption (Section 10.1.2) which requires a special algebraic group $\mathbb{G}$ with generator g where the exponentiation operation g^x, $x \in_R \mathbb{G}$ is a bijection from $\mathbb{G}$ to $\mathbb{G}$ preserving uniformity and randomness. In the next section we describe an example construction of $\mathbb{G}$.

10.1.1 Algebraic Group

Mathematical operations in our protocol are carried out in the algebraic group $\mathbb{G}$ which is described in the following (and also used in GKE protocols in [118, 119] and for the randomness extraction in [69, 70]). Let p be a safe prime, i.e., $p = 2q + 1$, with q a κ-bit prime where $\kappa \in \mathbb{N}$ is the security parameter. To specify $\mathbb{G}$ we require an auxiliary group $\hat{\mathbb{G}}$, which is a multiplicative group of quadratic residues modulo p. $\hat{\mathbb{G}}$ has order q and is generated by a quadratic residue $g = \hat{g}^2 \mod p$, with $g < q$ and $\hat{g}$ a primitive element of the multiplicative group $\mathbb{Z}_p^*$. Let a function $\hat{u} : \mathbb{Z}_q \to \mathbb{Z}_p$ be defined as $\hat{u}(x) := g^x \bmod p$ and $\hat{\mathbb{G}} = \{\hat{u}(i) \mid i \in \mathbb{Z}_q\}$. Let a function $u : \mathbb{Z}_p^* \to \mathbb{Z}_q$ be defined as

$$u(x) := \begin{cases} x \mod q & \text{if } x \leq q \\ p - x \mod q & \text{if } q < x < p \end{cases}$$

This function is used to define the group $\mathbb{G} := \{u(\hat{u}(i)) \mid i \in \mathbb{Z}_q\}$. The elements of $\mathbb{G}$ are integers from 0 to $q-1$. The multiplicative group operation $\cdot$ on $\mathbb{G}$ is defined as $a \cdot b := u(ab \pmod p)$ for all $a, b \in \mathbb{G}$. The exponentiation is therefore $a^b := u(a^b \bmod p)$ for all $a, b \in \mathbb{G}$.

Lemma 10.1. *Let $f(x) := u(\hat{u}(x))$. For all $x \in \mathbb{G}$ $f(x)$ is a bijection from $\mathbb{G}$ to $\mathbb{G}$.*

Proof. First, we show that f is injective, that is for any $x, y \in \mathbb{G}$ if $f(x) = f(y)$ then $x = y$. The equation $f(x) = f(y)$ is equivalent to $u(\hat{u}(x)) = u(\hat{u}(y))$. We denote $X := \hat{u}(x)$ and $Y := \hat{u}(y)$, and assume the following four cases.
CASE $X \leq q \wedge Y \leq q$: In this case $u(X) = X$, $u(Y) = Y$, and hence $X = Y$. Considering definition of $\hat{u}$ the equation $g^{x-y} \mod p = 1$ must hold. Consequently $q | x - y$ must hold because q is the order of g. Since $x, y \in \mathbb{G}$ and, therefore, $0 < x, y \leq q$ the previous equation holds only if $x - y = 0$ resulting in $x = y$.
CASE $X > q \wedge Y > q$: In this case $u(X) = p - X$, $u(Y) = p - Y$, and hence $X = Y$. Similar arguments as in the previous case lead to $x = y$.
CASE $X \leq q \wedge Y > q$: This case is impossible as explained in the following. Since $u(X) = X$, $u(Y) = p - Y$ so that $X = p - Y = -Y = (-1)Y$ must hold. However, $X, Y \in \hat{\mathbb{G}}$ (are quadratic residues) but $(-1) \notin \mathbb{G}$.
CASE $X > q \wedge Y \leq q$: This case is also impossible for the same reason as the previous one.

From the equality of the domain and image of f we follow that f is bijective. □

The following corollary follows directly from Lemma 10.1.

Corollary 10.2. *If x is chosen uniformly at random in $\mathbb{G}$ then $f(x)$ is also uniform and random in $\mathbb{G}$.*

Remark 10.3. Note that since $\mathbb{G}$ is equivalent to $\mathbb{Z}_q$ the function f preserves uniformity in $\mathbb{Z}_q$.

As a consequence of Corollary 10.2 the exponentiation function preserves uniformity in $\mathbb{G}$, i.e., if $x \in \mathbb{G}$ then $g^x = u(g^x \mod p) \in \mathbb{G}$.

Further, it is believed that the DDH assumption from Definition 5.11 holds in $\mathbb{G}$.

10.1.2 Tree Decisional Diffie-Hellman Assumption

Let $\mathcal{T}_n$ be *a set of all binary trees with n leaf nodes*[1] and $T_n \in \mathcal{T}_n$ *a binary tree* defined over the set of all its nodes (identified via *labels* $\langle l, v \rangle$) $T_n := \{\langle l, v \rangle \mid 0 \leq l \leq d_{T_n}, 0 \leq v \leq v_{\max} \leq 2^l - 1\}$ where l is a *level* of the node, v its *position* within this level ($v_{\max}$ is the maximal possible position depending on the tree structure, for example in TDH1 we use linear trees with $v_{\max} = 1$), and d_{T_n} the *depth* of T_n. The *set of leaf nodes* of T_n is defined as

$$LN_{T_n} := \{\langle l, v \rangle \mid \langle l, v \rangle \in T_n, \langle l+1, 2v \rangle \notin T_n, \langle l+1, 2v+1 \rangle \notin T_n\},$$

meaning that leaf nodes do not have any child nodes, and the *set of internal nodes* of T_n excluding the root node $\langle 0, 0 \rangle$ is defined as

$$IN_{T_n} := \{\langle l, v \rangle \mid \langle l, v \rangle \in T_n / \langle 0, 0 \rangle, \langle l+1, 2v \rangle \in T_n, \langle l+1, 2v+1 \rangle \in T_n\}.$$

[1] We consider only binary trees where each node is either a leaf or a parent of two child nodes.

Note that $T_n \in \mathcal{T}_n$ is not necessarily balanced or complete. Thus, not every label $\langle l, v\rangle$ with $0 \le l \le d_{T_n}, 0 \le v \le 2^l - 1$ may be part of T_n. In this case we say that the label $\langle l, v\rangle$ is *empty*.

For a set of n uniformly distributed variables $x_{\langle l,v\rangle} \in_R \mathbb{G}$ we specify the Tree Diffie-Hellman distribution (TDH-distribution) given by a set of pairs each consisting of a node's label $\langle l, v\rangle$ and a value $g^{x_{\langle l,v\rangle}} \in \mathbb{G}$ as

$$\mathtt{TDH}_{T_n} := \Big(g, \big\{(\langle l, v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in LN_{T_n}} \mid x_{\langle l,v\rangle} \in_R \mathbb{G}\big\}, \\ \big\{(\langle l, v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in IN_{T_n}} \mid x_{\langle l,v\rangle} = g^{x_{\langle l+1,2v\rangle} x_{\langle l+1,2v+1\rangle}}\big\}\Big).$$

Further we define two additional distributions from the TDH-distribution and a uniformly distributed $r \in_R \mathbb{G}$ (note that if $n = 2$ we get classical DH distributions):

$$\mathtt{TDDH}^{\star}_{T_n} := \big(\mathtt{TDH}_{T_n}, (\langle 0, 0\rangle, x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}})\big) \text{ and } \mathtt{TDDH}^{\$}_{T_n} := \big(\mathtt{TDH}_{T_n}, (\langle 0, 0\rangle, x_{\langle 0,0\rangle} := g^{r})\big).$$

Definition 10.4 (TDDH Assumption). *A* $\mathtt{TDDH}_{T_n}$ distinguisher for $\mathbb{G}$ *is a PPT algorithm* $\mathcal{A}$ *whose advantage probability defined as*

$$\mathsf{Adv}^{\mathtt{TDDH}}_{T_n,\mathbb{G}}(\kappa) := |\Pr[\mathcal{A}(\mathtt{TDDH}^{\star}_{T_n}) = 1] - \Pr[\mathcal{A}(\mathtt{TDDH}^{\$}_{T_n}) = 1]|$$

is non negligible. The $\mathtt{TDDH}_{T_n}$ *problem is* intractable *if there exists no* $\mathtt{TDDH}_{T_n}$ *distinguisher for* $\mathbb{G}$*. The* TDDH *assumption states that this is the case for all* $n > 1$*, all* $T_n \in \mathcal{T}_n$*, and all sufficiently large* κ*.*

In the following we show the equivalence between the TDDH and DDH assumptions. This allows us to treat TDDH as a standard cryptographic assumption.

Theorem 10.5 (DDH $\iff$ TDDH). *The* DDH *problem is polynomial time reducible to the* TDDH *problem and the* TDDH *problem is polynomial time reducible to the* DDH *problem, and*

$$\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) \le \mathsf{Adv}^{\mathtt{TDDH}}_{T_n,\mathbb{G}}(\kappa) \le (2n-3)\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa).$$

Proof. CASE $\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) \le \mathsf{Adv}^{\mathtt{TDDH}}_{T_n,\mathbb{G}}(\kappa)$: Assuming that there exists a DDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}'$ we construct a $\mathtt{TDDH}_{T_n}$ distinguisher for $\mathbb{G}$ denoted $\mathcal{A}$ as follows. On input a TDDH distribution $\mathtt{D}_{T_n}$ (either $\mathtt{TDDH}^{\star}_{T_n}$ or $\mathtt{TDDH}^{\$}_{T_n}$) $\mathcal{A}$ calls $\mathcal{A}'$ on input $(g, g^{x_{\langle 1,0\rangle}}, g^{x_{\langle 1,1\rangle}}, x_{\langle 0,0\rangle})$ such that $(\langle 1, 0\rangle, g^{x_{\langle 1,0\rangle}}) \in \mathtt{D}_{T_n}$, $(\langle 1, 1\rangle, g^{x_{\langle 1,1\rangle}}) \in \mathtt{D}_{T_n}$, $(\langle 0, 0\rangle, x_{\langle 0,0\rangle}) \in \mathtt{D}_{T_n}$, and returns its output. Note that if $\mathcal{A}$ has received $\mathtt{TDDH}^{\star}_{T_n}$ then $x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$ and $(g, g^{x_{\langle 1,0\rangle}}, g^{x_{\langle 1,1\rangle}}, x_{\langle 0,0\rangle})$ corresponds to the distribution $\mathtt{DDH}^{\star}$. On the other hand, if $\mathcal{A}$ has received $\mathtt{TDDH}^{\$}_{T_n}$ then $x_{\langle 0,0\rangle} := g^r$, $r \in_R \mathbb{G}$ and $(g, g^{x_{\langle 1,0\rangle}}, g^{x_{\langle 1,1\rangle}}, x_{\langle 0,0\rangle})$ corresponds to the distribution $\mathtt{DDH}^{\$}$. Thus, $\Pr[\mathcal{A}(\mathtt{TDDH}^{\star}_{T_n}) = 1] = \Pr[\mathcal{A}'(\mathtt{DDH}^{\star}) = 1]$ and $\Pr[\mathcal{A}(\mathtt{TDDH}^{\$}_{T_n}) = 1] = \Pr[\mathcal{A}'(\mathtt{DDH}^{\$}) = 1]$. This implies the desired inequality $\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) \le \mathsf{Adv}^{\mathtt{TDDH}}_{T_n,\mathbb{G}}(\kappa)$.

CASE $\mathsf{Adv}^{\mathtt{TDDH}}_{T_n,\mathbb{G}}(\kappa) \le (2n-3)\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa)$: We apply the "sequence of games" technique where simulator Δ interacts with the $\mathtt{TDDH}_{T_n}$ distinguisher for $\mathbb{G}$ denoted $\mathcal{A}$.

General Idea: The simulator is responsible for the choice of the TDDH distributions. We start with the game $\mathbf{G}_0$ where Δ chooses a random tree $T_n \in_R \mathcal{T}_n$ and constructs a distribution $\mathtt{D}_{0,b}$, $b \in_R \{0, 1\}$ such that $\mathtt{D}_{0,0} = \mathtt{TDDH}^{\star}_{T_n}$ and $\mathtt{D}_{0,1} = \mathtt{TDDH}^{\$}_{T_n}$. The event $\mathsf{Win}^{\mathtt{TDDH}}_0$ is the event that $\mathcal{A}$ on input the distribution $\mathtt{D}_{0,b}$ correctly guesses the bit b chosen by the simulator. In the second game $\mathbf{G}_1$ (and this is also the last game in the sequence) the simulator constructs distribution $\mathtt{D}_{1,b}$ by computing all values $x_{\langle l,v\rangle}$, $\langle l, v\rangle \in T_n \setminus \langle 0, 0\rangle$ as random independent values whereby in $\mathtt{D}_{1,0}$

the simulator defines $x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$ and in $\mathtt{D}_{1,1}$ it defines $x_{\langle 0,0\rangle} := g^r$, $r \in_R \mathbb{G}$. The event $\mathsf{Win}_1^{\mathtt{TDDH}}$ is the event that $\mathcal{A}$ on input the distribution $\mathtt{D}_{1,b}$ correctly guesses the bit b chosen by the simulator. Since all $x_{\langle l,v\rangle}$, $\langle l,v\rangle \in T_n \setminus \langle 0,0\rangle$ are random and independent we can upper-bound $\Pr[\mathsf{Win}_1^{\mathtt{TDDH}}]$ by the probability that a DDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}'$ correctly guesses b on input the distribution $\mathtt{D}'_{1,b}$ such that $\mathtt{D}'_{1,0} = \mathtt{DDH}^{\star}$ and $\mathtt{D}'_{1,1} = \mathtt{DDH}^{\$}$. In order to upper-bound $|\Pr[\mathsf{Win}_0^{\mathtt{TDDH}}] - \Pr[\mathsf{Win}_1^{\mathtt{TDDH}}]|$ we apply the "hybrid technique" and specify auxiliary games with corresponding hybrid distributions $\mathtt{D}_{1,b}^{l,v}$, one for each internal node $\langle l,v\rangle \in IN_{T_n}$ (there are $n-2$ such nodes). The neighboring hybrid distributions differ in the choice of *one particular* $x_{\langle l,v\rangle}$: in the preceding hybrid distribution the simulator computes $x_{\langle l,v\rangle} := g^{x_{\langle l+1,2v\rangle} x_{\langle l+1,2v+1\rangle}}$ whereas in the succeeding hybrid distribution it computes $x_{\langle l,v\rangle} := g^{r_{\langle l,v\rangle}}$, $r_{\langle l,v\rangle} \in_R \mathbb{G}$. We start building hybrids from the deepest leftmost internal node $\langle l,v\rangle$ (with level $l = d_{T_n} - 1$) and move up to the root only after having built hybrid distributions for the rightmost internal node located at the same level. Thus, in each hybrid distribution $\mathtt{D}_{1,b}^{l,v}$ every secret value $x_{\langle l_i,v_i\rangle}$ with $l_i \le l$, $v_i < v$ is computed as a random independent value. This allows us to upper-bound the computational distinguishability between any two subsequent auxiliary games using the advantage probability of the DDH distinguisher $\mathcal{A}'$.

Detailed Proof: **Game $\mathbf{G}_0$.** This game is an interaction between the simulator Δ and the $\mathtt{TDDH}_{T_n}$ distinguisher for $\mathbb{G}$ denoted $\mathcal{A}$. The simulator chooses $n \in_R \mathbb{N}$ and $T_n \in_R \mathcal{T}_n$, and constructs a distribution $\mathtt{D}_0$ as

$$\mathtt{D}_0 := \Big(g, \big\{(\langle l,v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in LN_{T_n}} \mid x_{\langle l,v\rangle} \in_R \mathbb{G}\big\}, \\ \big\{(\langle l,v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in IN_{T_n}} \mid x_{\langle l,v\rangle} = g^{x_{\langle l+1,2v\rangle} x_{\langle l+1,2v+1\rangle}}\big\}\Big).$$

Note that $\mathtt{D}_0 \equiv \mathtt{TDH}_{T_n}$.

Then, Δ chooses a random bit $b \in_R \{0,1\}$ and defines distribution $\mathtt{D}_{0,b}$ as follows. If $b = 0$ then $\mathtt{D}_{0,0} = \big(\mathtt{D}_0, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}})\big)$, else if $b = 1$ then $\mathtt{D}_{0,1} = \big(\mathtt{D}_0, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^r)\big)$ with $r \in_R \mathbb{G}$.

Note that $\mathtt{D}_{0,0} \equiv \mathtt{TDDH}^{\star}_{T_n}$ and $\mathtt{D}_{0,1} \equiv \mathtt{TDDH}^{\$}_{T_n}$.

The simulator calls then $\mathcal{A}$ on input $\mathtt{D}_{0,b}$ and obtains bit b'. By $\mathsf{Win}_0^{\mathtt{TDDH}}$ we denote the event $b' = b$. Thus,

$$\Pr[\mathsf{Win}_0^{\mathtt{TDDH}}] = \Pr[\mathcal{A}(\mathtt{D}_{0,b}) = b]. \tag{10.1}$$

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ except that instead of constructing $\mathtt{D}_0$ as above the simulator constructs

$$\mathtt{D}_1 := \Big(g, \big\{(\langle l,v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in LN_{T_n}} \mid x_{\langle l,v\rangle} \in_R \mathbb{G}\big\}, \\ \big\{(\langle l,v\rangle, g^{x_{\langle l,v\rangle}})_{\langle l,v\rangle \in IN_{T_n}} \mid x_{\langle l,v\rangle} = g^{r_{\langle l,v\rangle}}, r_{\langle l,v\rangle} \in_R \mathbb{G}\big\}\Big).$$

Note that in this distribution all secret values $x_{\langle l,v\rangle}$, $\langle l,v\rangle \in T_n \setminus \langle 0,0\rangle$ are random and independent of each other.

Then Δ chooses a random bit $b \in_R \{0,1\}$ and defines distributions

$$\mathtt{D}_{1,0} := \big(\mathtt{D}_1, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}})\big), \text{ and} \\ \mathtt{D}_{1,1} := \big(\mathtt{D}_1, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^r)\big), \text{ where } r \in_R \mathbb{G}.$$

The simulator calls then $\mathcal{A}$ on input $\mathtt{D}_{1,b}$ and obtains bit b'. By $\mathsf{Win}_1^{\mathtt{TDDH}}$ we denote the event $b' = b$ (this is the same event as $\mathsf{Win}_0^{\mathtt{TDDH}}$). Thus,

$$\Pr[\mathsf{Win}_1^{\mathtt{TDDH}}] = \Pr[\mathcal{A}(\mathtt{D}_{1,b}) = b].$$

Since all secret values $x_{\langle l,v\rangle}$ except for the root value $x_{\langle 0,0\rangle}$ are random and independent the probability that $\mathcal{A}$ correctly guesses b is no greater than the probability that a DDH distinguisher for $\mathbb{G}$ denoted $\mathcal{A}'$ is able to distinguish between the distributions $\mathtt{D}'_{1,0} := (g, g^{x_{\langle 1,0\rangle}}, g^{x_{\langle 1,1\rangle}}, x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}})$ and $\mathtt{D}'_{1,1} := (g, g^{x_{\langle 1,0\rangle}}, g^{x_{\langle 1,1\rangle}}, x_{\langle 0,0\rangle} := g^r)$, where $r \in_R \mathbb{G}$. Note that $g^{x_{\langle 1,0\rangle}}$, $g^{x_{\langle 1,1\rangle}}$, and $x_{\langle 0,0\rangle}$ are sampled from the corresponding distributions $\mathtt{D}_{1,0}$ respectively $\mathtt{D}_{1,1}$. Note also that $\mathtt{D}'_{1,0} \equiv \mathtt{DDH}^{\star}$ and $\mathtt{D}'_{1,1} \equiv \mathtt{DDH}^{\$}$. Thus,

$$\Pr[\mathsf{Win}_1^{\mathtt{TDDH}}] = \Pr[\mathcal{A}(\mathtt{D}_{1,b}) = b] \leq \Pr[\mathcal{A}'(\mathtt{D}'_{1,b}) = b]. \tag{10.2}$$

Further, we estimate the difference $|\Pr[\mathsf{Win}_1^{\mathtt{TDDH}}] - \Pr[\mathsf{Win}_0^{\mathtt{TDDH}}]|$. For this purpose we apply the "hybrid technique". We consider a sequence of auxiliary games $\mathbf{G}_1^{l,v}$ for all $l = d_{T_n} - 1$ down to 0, $0 \leq v \leq v_{\mathtt{max}} \leq 2^l - 1$, $\langle l, v\rangle \in T_n \setminus LN_{T_n}$. Each $\mathbf{G}_1^{l,v}$ is identical to Game $\mathbf{G}_0$ except that instead of constructing $\mathtt{D}_0$ as in Game $\mathbf{G}_0$ the simulator constructs

$$\begin{aligned}\mathtt{D}_1^{l,v} := \Big(& g, \big\{(\langle l_i, v_i\rangle, g^{x_{\langle l_i,v_i\rangle}})_{\langle l_i,v_i\rangle \in LN_{T_n}} \mid x_{\langle l_i,v_i\rangle} \in_R \mathbb{G}\big\},\\ & \big\{(\langle l_i, v_i\rangle, g^{x_{\langle l_i,v_i\rangle}})_{\langle l_i,v_i\rangle \in IN_{T_n}, l_i \leq l, v_i < v} \mid x_{\langle l_i,v_i\rangle} := g^{r_{\langle l_i,v_i\rangle}}, r_{\langle l_i,v_i\rangle} \in_R \mathbb{G}\big\},\\ & \big\{(\langle l_i, v_i\rangle, g^{x_{\langle l_i,v_i\rangle}})_{\langle l_i,v_i\rangle \in IN_{T_n}, l_i \geq l, v_i \geq v} \mid x_{\langle l_i,v_i\rangle} := g^{x_{\langle l_i+1,2v_i\rangle} x_{\langle l_i+1,2v_i+1\rangle}}\big\}\Big).\end{aligned}$$

Note that if $l = d_{T_n} - 1$ and v_{min} is the minimal position of all non-empty labels with level $d_{T_n} - 1$ then $\mathtt{D}_1^{d_{T_n}-1,v_{\mathsf{min}}} \equiv \mathtt{D}_0$ whereas if $l = 0$ and $v = 0$ then $\mathtt{D}_1^{0,0} \equiv \mathtt{D}_1$.

Then, instead of constructing the distribution $\mathtt{D}_{0,b}$ as in Game $\mathbf{G}_0$ the simulator constructs

$$\begin{aligned}\mathtt{D}_{1,0}^{l,v} &= \big(\mathtt{D}_1^{l,v}, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}})\big), \text{ and}\\ \mathtt{D}_{1,1}^{l,v} &= \big(\mathtt{D}_1^{l,v}, (\langle 0,0\rangle, x_{\langle 0,0\rangle} := g^{r})\big), \text{ where } r \in_R \mathbb{G},\end{aligned}$$

and proceeds as in Game $\mathbf{G}_0$. Note that if $l = d_{T_n} - 1$ and v_{min} is the minimal position of all non-empty labels with level $d_{T_n} - 1$ then $\mathtt{D}_{1,b}^{d_{T_n}-1,v_{\mathsf{min}}} \equiv \mathtt{D}_{0,b}$ whereas if $l = 0$ and $v = 0$ then $\mathtt{D}_{1,b}^{0,0} \equiv \mathtt{D}_{1,b}$. Let $\mathsf{Win}_{l,v}^{\mathtt{TDDH}}$ denote the event that Win_0 (or equally Win_1) occurs in $\mathbf{G}_1^{l,v}$.

The only difference between two neighboring auxiliary games $\mathbf{G}_1^{l_i,v_i}$ and $\mathbf{G}_1^{l_j,v_j}$ is how Δ constructs the value $x_{\langle l_i,v_i\rangle}$ in the distributions $\mathtt{D}_{1,b}^{l_i,v_i}$ and $\mathtt{D}_{1,b}^{l_j,v_j}$: in $\mathtt{D}_{1,b}^{l_i,v_i}$ the simulator computes $x_{\langle l_i,v_i\rangle} := g^{x_{\langle l_i+1,2v_i\rangle} x_{\langle l_i+1,2v_i+1\rangle}}$ whereas in $\mathtt{D}_{1,b}^{l_j,v_j}$ it computes $x_{\langle l_i,v_i\rangle} := g^{r_{\langle l_i,v_i\rangle}}$, $r_{\langle l_i,v_i\rangle} \in_R \mathbb{G}$. Therefore, $(g, g^{x_{\langle l_i+1,2v_i\rangle}}, g^{x_{\langle l_i+1,2v_i\rangle}}, g^{x_{\langle l_i+1,2v_i\rangle} x_{\langle l_i+1,2v_i+1\rangle}})$ consisting of values taken from distribution $\mathtt{D}_{1,b}^{l_i,v_i}$ corresponds to the distribution $\mathtt{DDH}^{\star}$ whereas $(g, g^{x_{\langle l_i+1,2v_i\rangle}}, g^{x_{\langle l_i+1,2v_i\rangle}}, g^{r_{\langle l_i,v_i\rangle}})$ consisting of values taken from distribution $\mathtt{D}_{1,b}^{l_j,v_j}$ corresponds to the distribution $\mathtt{DDH}^{\$}$. This allows us to upper-bound the difference between $\mathbf{G}_1^{l_j,v_j}$ and $\mathbf{G}_1^{l_i,v_i}$ with the advantage of the DDH distinguisher $\mathcal{A}'$, i.e.,

$$|\Pr[\mathsf{Win}_{l_j,v_j}^{\mathtt{TDDH}}] - \Pr[\mathsf{Win}_{l_i,v_i}^{\mathtt{TDDH}}]| \leq \mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa).$$

Note that $\mathbf{G}_1^{d_{T_n}-1,v_{\mathsf{min}}} = \mathbf{G}_0$, where v_{min} is the minimal position of all non-empty labels with level $d_{T_n} - 1$, and that $\mathbf{G}_1^{0,0} = \mathbf{G}_1$. In any binary tree $T_n \in \mathcal{T}_n$ there are exactly $n-2$ intermediate nodes excluding the root node, that is $|T_n \setminus \{LN_{T_n} \cup \langle 0,0\rangle\}| = n - 2$. Hence,

$$|\Pr[\mathsf{Win}_1^{\mathtt{TDDH}}] - \Pr[\mathsf{Win}_0^{\mathtt{TDDH}}]| \leq (n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa). \tag{10.3}$$

This implies

$$\Pr[\mathsf{Win}_0^{\mathtt{TDDH}}] \leq (n-1)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) + \Pr[\mathsf{Win}_1^{\mathtt{TDDH}}].$$

Considering Equations 10.1 and 10.2 we obtain

$$\Pr[\mathcal{A}(\mathtt{D}_{0,b}) = b] \leq (n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) + \Pr[\mathcal{A}'(\mathtt{D}'_{1,b}) = b],$$

that is equivalent to

$$2\Pr[\mathcal{A}(\mathtt{D}_{0,b}) = b] - 1 \leq 2(n-2)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) + 2\Pr[\mathcal{A}'(\mathtt{D}'_{1,b}) = b] - 1. \tag{10.4}$$

In the following we argue that the $\mathtt{TDDH}_{T_n}$ distinguisher $\mathcal{A}$ has the advantage $\mathsf{Adv}_{T_n,\mathbb{G}}^{\mathtt{TDDH}}(\kappa) := |2\Pr[\mathcal{A}(\mathtt{TDDH}_b) = b] - 1|$, where b is a uniformly chosen bit, and $\mathtt{TDDH}_0 \equiv \mathtt{TDDH}^{\star}_{T_n}$ resp. $\mathtt{TDDH}_1 \equiv \mathtt{TDDH}^{\$}_{T_n}$. Recall that in Definition 10.4 we have defined $\mathsf{Adv}_{T_n,\mathbb{G}}^{\mathtt{TDDH}}(\kappa) := |\Pr[\mathcal{A}(\mathtt{TDDH}^{\star}_{T_n}) = 1] - \Pr[\mathcal{A}(\mathtt{TDDH}^{\$}_{T_n}) = 1]|$. The following simple computation shows that both definitions are equivalent:

$$\begin{aligned}
&|2\Pr[\mathcal{A}(\mathtt{TDDH}_b) = b] - 1| = \\
&\quad = |2(\Pr[\mathcal{A}(\mathtt{TDDH}_b) = b | b = 0]\Pr[b = 0] + \Pr[\mathcal{A}(\mathtt{TDDH}_b) = b | b = 1]\Pr[b = 1]) - 1| \\
&\quad = |2\frac{1}{2}(\Pr[\mathcal{A}(\mathtt{TDDH}_0) = 0] + \Pr[\mathcal{A}(\mathtt{TDDH}_1) = 1]) - 1| \\
&\quad = |((1 - \Pr[\mathcal{A}(\mathtt{TDDH}_0) = 1]) + \Pr[\mathcal{A}(\mathtt{TDDH}_1) = 1]) - 1| \\
&\quad = |\Pr[\mathcal{A}(\mathtt{TDDH}_1) = 1] - \Pr[\mathcal{A}(\mathtt{TDDH}_0) = 1]| \\
&\quad = |\Pr[\mathcal{A}(\mathtt{TDDH}^{\$}) = 1] - \Pr[\mathcal{A}(\mathtt{TDDH}^{\star}) = 1]|
\end{aligned}$$

The same approach can also be applied in order to show the equivalence between both definitions $\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) := |2\Pr[\mathcal{A}'(\mathtt{DDH}_b) = b] - 1|$, where b is a uniformly chosen bit, $\mathtt{DDH}_0 \equiv \mathtt{DDH}^{\star}$ resp. $\mathtt{DDH}_1 \equiv \mathtt{DDH}^{\$}$, and $\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) := |\Pr[\mathcal{A}'(\mathtt{DDH}^{\star}) = 1] - \Pr[\mathcal{A}(\mathtt{DDH}^{\$}) = 1]|$. Recall that $\mathtt{D}_{0,0} \equiv \mathtt{TDDH}^{\star}_{T_n}$ resp. $\mathtt{D}_{0,1} \equiv \mathtt{TDDH}^{\$}_{T_n}$ as noted in Game $\mathbf{G}_0$, and $\mathtt{D}'_{1,0} \equiv \mathtt{DDH}^{\star}$ and $\mathtt{D}'_{1,1} \equiv \mathtt{DDH}^{\$}$ as noted in Game $\mathbf{G}_1$. Thus, considering Equation 10.4 we obtain the desired inequality $\mathsf{Adv}_{T_n,\mathbb{G}}^{\mathtt{TDDH}}(\kappa) \leq (2n-3)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa)$. □

10.2 Short Overview of TDH1

In the following we highlight the main idea of the static and dynamic TDH1 protocols. Similar to Section 9.2.2 by $\Pi_i^{s_i} \in \mathcal{G}$ we denote the i-th oracle involved in the group $\mathcal{G}$ assuming that there exists an index $j \in [1, N]$ such that some $U_j \in \mathcal{U}$ owns $\Pi_i^{s_i}$. Further, in order to simplify the description of the protocol we assume that $s := s_1 = \ldots = s_n$ for all oracles $\Pi_i^{s_i} \in \mathcal{G}$ and use the notation Π_i^s instead.

Assume that the initial group $\mathcal{G}$ consists of n oracles $\Pi_1^s, \ldots, \Pi_n^s$ and each of them knows this order implicitly. Every Π_i^s is assigned to a leaf node $\langle l_i, v_i \rangle$ of a *linear* binary tree T_n (in this context linear means that $d_{T_n} = n - 1$ and $v_i \in \{0, 1\}$, e.g., Figure 10.1).

Every sent message in TDH1 is authenticated via an attached digital signature. In order to prevent replay attacks each signature is generated on a message and the session id $\mathtt{sid} = r_1 | \ldots | r_n$ where each nonce r_i is chosen by Π_i^s fresh for every new session, i.e., session ids are unique.

Each operation of TDH1 requires three communication rounds independent of the number of participants.

10.2.1 Static and Dynamic TDH1.Setup

In order to compute the session group key in the static version of TDH1 each oracle with position $\langle l_i, v_i \rangle$ in T_n chooses own secret exponent $x_{\langle l_i,v_i \rangle} \in_R \mathbb{G}$ and broadcasts[2] $y_{\langle l_i,v_i \rangle} := g^{x_{\langle l_i,v_i \rangle}}$ (note that $y_{\langle l_i,v_i \rangle} \in \mathbb{G}$). Π_1^s which is assigned to the deepest leftmost leaf node of T_n is the first to build a set X_1 of secret values (of nodes) $x_{\langle l,v \rangle}$ in its path up to the root $\langle 0, 0 \rangle$. Note, each internal node $x_{\langle l,0 \rangle} = (y_{\langle l+1,0 \rangle})^{x_{\langle l+1,1 \rangle}} = (y_{\langle l+1,1 \rangle})^{x_{\langle l+1,0 \rangle}} = g^{x_{\langle l+1,0 \rangle} x_{\langle l+1,1 \rangle}}$. Thus, Π_1^s is the first to learn $x_{\langle 0,0 \rangle}$. To ensure that also other oracles compute $x_{\langle 0,0 \rangle}$, Π_1^s computes the set $\hat{Y}$ consisting of $y_{\langle l,0 \rangle} := g^{x_{\langle l,0 \rangle}}$ for each previously computed internal node's secret value $x_{\langle l,0 \rangle} \in X_1$ but without computing $y_{\langle 0,0 \rangle}$, and broadcasts $\hat{Y}$. Upon receiving $\hat{Y}$ every $\Pi_{i \neq 1}^s$ is able to compute own set X_i consisting of all secret values $x_{\langle l,v \rangle}$ in its path up to the root. Thus, every $\Pi_{i \neq 1}^s$ learns $x_{\langle 0,0 \rangle}$ too. The protocol ensures that only valid participants learn $x_{\langle 0,0 \rangle}$ which is then used to derive the intermediate value K. In order to compute the intermediate value K each participant iteratively computes values $\rho_0, \ldots, \rho_n$ as outputs of the pseudo-random function f on the public input value v_0 whereby the key (secret seed) of f for the computation of each ρ_i is build as the XOR sum $\rho_{i-1} \oplus \pi(r_i)$ where π is a one-way permutation (note that $\rho_0 := f_{x_{\langle 0,0 \rangle \lfloor \kappa}}(v_0)$ whereby $x_{\langle 0,0 \rangle \lfloor \kappa}$ denotes $x_{\langle 0,0 \rangle}$ truncated to its κ rightmost bits). This successive evaluation of f embeds every random nonce r_i into the computation of $K := \rho_n$. Intuitively, the one-way permutation π is used to prevent that malicious participants choose own nonces such that they can influence values of the pseudo-random function secret seeds (note that if malicious participants are able to do this then they can also influence K). Thus, random nonces and successive evaluation of f as described above are used to ensure n-contributiveness for K, which is then used as a seed for the pseudo-random function f to derive the key confirmation token μ (on input a constant public value v_1) and the actual session group key $\mathbf{K}$ (on input another constant public value $v_2 \neq v_1$). Note that prior to accepting $\mathbf{K}$ participants exchange and verify signatures on the computed key confirmation tokens ensuring MA-security of the protocol.

The dynamic version of TDH1 setup requires additional computations described in the following. In order to ensure strong forward secrecy every Π_i^s must erase[3] all secret information which can be used to obtain $\mathbf{K}$ (this information includes ephemeral secrets X_i and $\rho_0, \ldots, \rho_n$, and K) but in order to proceed with dynamic changes efficiently each Π_i^s saves X_i' which is composed of $x'_{\langle l,v \rangle} := g^{x^2_{\langle l,v \rangle}}$ for each $x_{\langle l,v \rangle}$ from X_i. Intuitively, the knowledge of any $x_{\langle l,v \rangle}$ in X_i would enable the adversary to compute $\mathbf{K}$ (if the sent protocol messages are stored). We show that this kind of mapping prevents the adversary from recovering any $x_{\langle l,v \rangle}$ in later sessions.

10.2.2 TDH1.Join$^+$

In order to handle join events the initial tree is updated, i.e., one new leaf node on top of the initial tree is added for each joining oracle from $\mathcal{J}$ (Figure 10.3). Before all oracles can compute the updated session group key $\mathbf{K}$ they need to compute the updated secret value $x_{\langle 0,0 \rangle}$ at the root node of the updated tree. The computation process of $x_{\langle 0,0 \rangle}$ is like in the initialization procedure but more efficient since initial participants do not need to compute all secret values $x_{\langle l,v \rangle}$ in their

[2] By broadcast we abstractly mean that a message is sent to all participants. This can be also realized via multicast and unicast connections, i.e., as a sequence of equal messages to different receivers. Note that the adversary can treat each of these messages independently.

[3] The most challenging task when using the *(secure) erasure technique* [76] to achieve strong forward secrecy in dynamic GKE protocols is to ensure that some ephemeral information remains in order to provide efficient dynamic operations. Note that previously proposed dynamic GKE protocols, e.g., [114], could not achieve this and provide security under standard cryptographic assumptions at the same time.

paths up to the root (that is not the whole set X_i as in the setup operation) but only those starting from the level of the first joined oracle. For this purpose an oracle Π_γ^s broadcasts a message that contains updated $\hat{Y}$ whereby $\langle l_\gamma, v_\gamma \rangle$ is the leaf node below the leaf node of the first joined oracle. The same derivation process for $\mathbf{K}$ and the same erasure technique as in the setup is applied to ensure contributiveness and strong forward secrecy, respectively.

10.2.3 TDH1.Leave$^+$

In order to handle leave events the initial tree is updated, i.e., leaf nodes of leaving oracles from $\mathcal{L}$ are deleted from the tree (Figure 10.4). All remaining oracles first update $x_{\langle 0,0 \rangle}$. Similar to setup and join they start updating secret values $x_{\langle l,v \rangle}$ in their paths up to the root from a certain level which depends on the deepest removed leaf node (thus more efficiently compared to setup). For this purpose an oracle Π_γ^s broadcasts a message that contains updated $\hat{Y}$ whereby $\langle l_\gamma, v_\gamma \rangle$ is the leaf node below the deepest removed leaf node in the updated tree. The updated session group key $\mathbf{K}$ is derived similar to setup and join, and the same erasure technique is applied at the end of the operation.

10.3 Static TDH1

In this section we propose the static version of the TDH1 protocol and prove its security using definitions of our model from Chapter 8. In order to simplify the description of the protocol and to provide feeling for its implementation we first specify some auxiliary functions.

10.3.1 Authentication Functions

Authentication in TDH1 is achieved using a digital signature scheme $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$. We assume that each user $U_i \in \mathcal{U}$ holds a long-term key pair $(sk_i, pk_i) := \Sigma.\mathtt{Gen}(1^\kappa)$. The authentication mechanism of TDH1 consists of the functions:

- $\mathtt{Auth\_Sig}(U_i, m)$. This function invokes $\Sigma.\mathtt{Sign}(sk_i, m)$ to obtain signature σ, which is returned.
- $\mathtt{Auth\_Ver}(U_i, m, \sigma)$. This function invokes $\Sigma.\mathtt{Verify}(pk_i, m, \sigma)$ and returns its output.
- $\mathtt{Auth\_Nonce}(1^\kappa)$. This function generates and returns a random *nonce* $r \in_R \{0,1\}^\kappa$.

10.3.2 Tree Management Function

In the TDH1 protocol each participating oracle $\Pi_i^{s_i}$ is logically assigned to a leaf node $\langle l_i, v_i \rangle$ of a binary tree T_n which is initialized during the setup protocol. We sometimes call $\langle l_i, v_i \rangle$ the *position of* $\Pi_i^{s_i}$ in T_n. The TDH1 protocol uses linear binary trees such that $d_{T_n} = n - 1$ and $v \in \{0,1\}$, e.g., in Figure 10.1. The following function is responsible for the logical assignment of n instance oracles to the binary tree T_n.

- $\mathtt{Tree\_Init}(\mathcal{G})$. This function creates an empty tree structure T_n and assigns Π_1^s to the position $\langle n-1, 0 \rangle$ and each Π_i^s, $i = 2, \ldots, n$ to $\langle n+1-i, 1 \rangle$ in T_n. The function returns T_n. (see Figure 10.1 for an example)

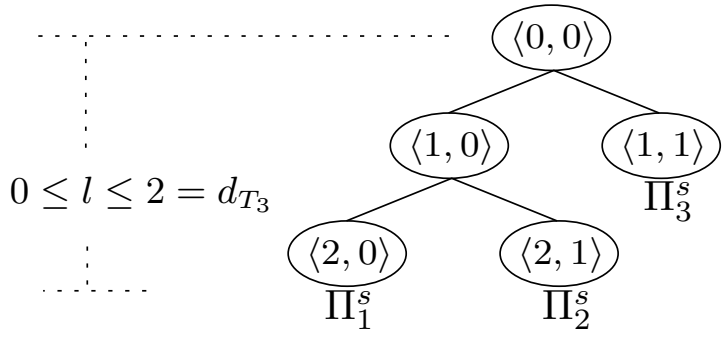

Fig. 10.1. Example: `Tree_Init`. Linear Binary Tree T_4. This Tree is Returned by `Tree_Init` on Input $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s, \Pi_4^s.\}$

10.3.3 Key Exchange Functions

For the description of the static TDH1 protocol we require the following key exchange functions where g is a generator of $\mathbb{G}$ from Section 10.1.1:

- $\texttt{TDH1\_Exp}(x_{\langle l,v\rangle})$. The function returns $y_{\langle l,v\rangle} := g^{x_{\langle l,v\rangle}}$.
- $\texttt{TDH1\_Pick}(1^\kappa)$. The function picks $x_{\langle l,v\rangle} \in_R \mathbb{G}$ and returns $(x_{\langle l,v\rangle}, y_{\langle l,v\rangle} := \texttt{TDH1\_Exp}(x_{\langle l,v\rangle}))$.
- $\texttt{TDH1\_Up}(l, v, x_{\langle l,v\rangle}, y_{\langle l,1-v\rangle}, Y)$ with $Y := \{y_{\langle j,1\rangle} | \; j = l-1, \ldots, 1\}$. The function computes $x_{\langle l-1,0\rangle} := (y_{\langle l,1-v\rangle})^{x_{\langle l,v\rangle}}$, and returns

$$X := \{x_{\langle l,v\rangle}, x_{\langle l-1,0\rangle}\} \bigcup \{x_{\langle j,0\rangle} := (y_{\langle j+1,1\rangle})^{x_{\langle j+1,0\rangle}} \mid y_{\langle j+1,1\rangle} \in Y, \forall j = l-2, \ldots, 0\}.$$

- $\texttt{TDH1\_Exp}^\star(l, X)$. The function returns $Y := \{y_{\langle j,0\rangle} := \texttt{TDH1\_Exp}(x_{\langle j,0\rangle}) \mid x_{\langle j,0\rangle} \in X, \forall j = l, \ldots, 1\}$.

10.3.4 Key Confirmation and Derivation Functions

Let $F := \left\{f_k\right\}_{k\in\{0,1\}^\kappa}$, $\kappa \in \mathbb{N}$ be a collision-resistant pseudo-random function ensemble with domain and range $\{0,1\}^\kappa$ (Definitions 5.4 and 5.5) and $\pi : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$ a one-way permutation (Definition 5.2). The following function is used to compute the *intermediate value* K and the *key confirmation token* μ.

- $\texttt{TDH1\_Con}(x_{\langle 0,0\rangle}, r_1 | \ldots | r_n)$. The function computes $x_{\langle 0,0\rangle \lfloor \kappa}$ which is a κ-bit string composed of κ least significant bits of $x_{\langle 0,0\rangle}$. Then the function computes $\rho_0 := f_{x_{\langle 0,0\rangle \lfloor \kappa}}(v_0)$, and each $\rho_l := f_{\rho_{l-1} \oplus \pi(r_l)}(v_0)$ for all $l = \{1, \ldots, n\}$, where v_0 is a constant public value. Let $K := \rho_n$. Finally, the function computes $\mu := f_K(v_1)$ where v_1 is a constant public value, and returns (K, μ).

Note, this function has certain similarity to the computation process applied in our compiler C-MACON from Chapter 9. Indeed, n-contributiveness and key confirmation for the TDH1 protocol is achieved in the same way as in the mentioned compiler. However, we need to truncate $x_{\langle 0,0\rangle}$ to a bit string of length κ in order to allow any pseudo-random function f to be used. Note that according to Definition 5.4 a PRF key is a randomly chosen string of κ bits.

The actual session group key $\mathbf{K}$ in TDH1 is derived using the following function:

- $\texttt{TDH1\_Key}(K)$. The function returns $\mathbf{K} := f_K(v_2)$ where $v_2 \neq v_1$ is a constant public value.

10.3.5 Protocol Execution

Having specified protocol functions for authentication, tree management, key exchange, and key derivation we are now ready to describe the setup operation TDH1.Setup. Note that in the static version of TDH1 this is the only operation to be considered.

Operation TDH1.Setup **(Static Case)**

Figure 10.2 shows an example of the TDH1.Setup operation with three participating oracles Π_1^s, Π_2^s, and Π_3^s. Each oracle computes the same tree structure T_3 (Figure 10.1). Each ora-

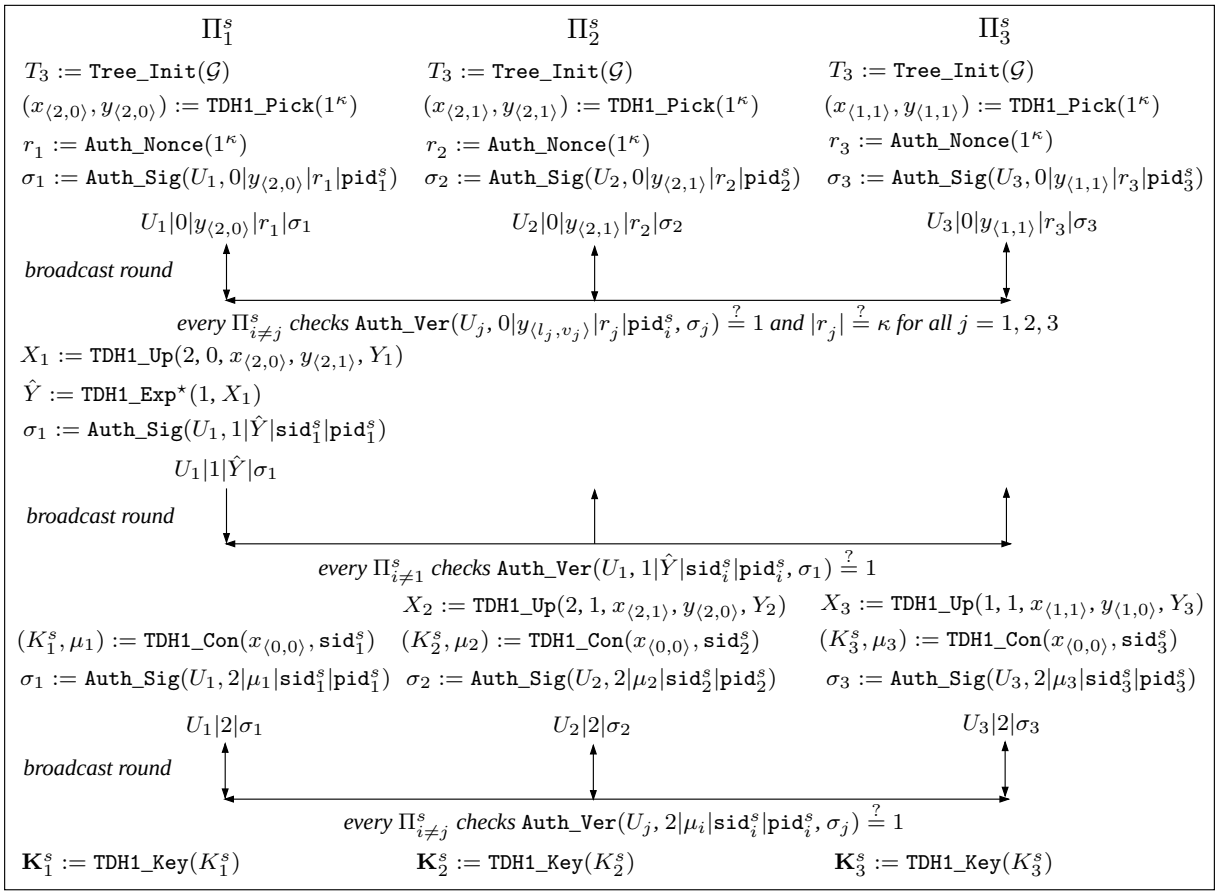

Fig. 10.2. Example: Operation TDH1.Setup (Static Case) with $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s\}$. Public values: $\texttt{sid}_i^s = r_1|r_2|r_3$, $Y_1 = Y_2 = \{y_{\langle 1,1\rangle}\}$, $Y_3 = \emptyset$, $\hat{Y} = \{y_{\langle 1,0\rangle}\}$, where $y_{\langle 1,1\rangle} = g^{x_{\langle 1,1\rangle}}$, $y_{\langle 1,0\rangle} = g^{x_{\langle 1,0\rangle}}$. Secret values: $X_1 = \{x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_2 = \{x_{\langle 2,1\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_3 = \{x_{\langle 1,1\rangle}, x_{\langle 0,0\rangle}\}$, where $x_{\langle 0,0\rangle} = g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$, $x_{\langle 1,0\rangle} = g^{x_{\langle 2,0\rangle} x_{\langle 2,1\rangle}}$ and $x_{\langle 2,0\rangle}$, $x_{\langle 2,1\rangle}$, $x_{\langle 1,1\rangle} \in_R \mathbb{G}$.

cle Π_i^s chooses own secret exponent $x_{\langle l_i,v_i\rangle}$ and sends $y_{\langle l_i,v_i\rangle} := g^{x_{\langle l_i,v_i\rangle}}$ together with own randomly chosen nonce r_i to each other oracle. Hence, each Π_i^s is able to build Y_i and $\texttt{sid}_i^s := r_1|\ldots|r_3$. Oracle Π_1^s builds X_1 composed of the previously chosen $x_{\langle 2,0\rangle}$, and computed $x_{\langle 1,0\rangle} := (y_{\langle 2,1\rangle})^{x_{\langle 2,0\rangle}}$, and $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$ (note that $y_{\langle 2,1\rangle}$ and $y_{\langle 1,1\rangle}$ are received in the previous round). Then Π_1^s builds $\hat{Y}$ composed of $y_{\langle 1,0\rangle} := g^{x_{\langle 1,0\rangle}}$ and sends it to Π_2^s and Π_3^s. Oracle Π_2^s builds X_2 composed of the previously chosen $x_{\langle 2,1\rangle}$, and computed $x_{\langle 1,0\rangle} := (y_{\langle 2,0\rangle})^{x_{\langle 2,1\rangle}}$, and $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$ (note that $y_{\langle 2,0\rangle}$ and $y_{\langle 1,1\rangle}$ are received in the first round). Oracle Π_3^s builds X_3 composed of the previously chosen $x_{\langle 1,1\rangle}$, and computed

$x_{\langle 0,0\rangle} := (y_{\langle 1,0\rangle})^{x_{\langle 1,1\rangle}}$ (note that $y_{\langle 1,0\rangle}$ is part of the previously received $\hat{Y}$). Thus, every oracle is in possession of $x_{\langle 0,0\rangle}$ and $\mathtt{sid}_i^s = r_1|\ldots|r_3$, and is therefore able to compute the intermediate value K, i.e., $\rho_0 := f_{x_{\langle 0,0\rangle\lfloor\kappa}}(v_0)$, $\rho_1 := f_{\rho_0\oplus\pi(r_1)}(v_0)$, $\rho_2 := f_{\rho_1\oplus\pi(r_2)}(v_0)$, and $K = \rho_3 := f_{\rho_2\oplus\pi(r_2)}(v_0)$, and the key confirmation token $\mu := f_K(v_1)$. The latter is used to compute the corresponding signature, which is then sent to and verified by all other group members prior to the acceptance with the session group key $\mathbf{K}$.

In the following we give the formal description of the static TDH1.Setup operation.

Round 1. Each oracle $\Pi_i^s \in \mathcal{G}$ computes $T_n := \mathtt{Tree\_Init}(\mathcal{G})$, $(x_{\langle l_i,v_i\rangle}, y_{\langle l_i,v_i\rangle}) := \mathtt{TDH1\_Pick}(1^\kappa)$, $r_i := \mathtt{Auth\_Nonce}(1^\kappa)$, $\sigma_i := \mathtt{Auth\_Sig}(U_i, 0|y_{\langle l_i,v_i\rangle}|r_i|\mathtt{pid}_i^s)$, and broadcasts $U_i|0|y_{\langle l_i,v_i\rangle}|r_i|\sigma_i$.

Round 2. When $\Pi_i^s \in \mathcal{G}$ receives $U_j|0|y_{\langle l_j,v_j\rangle}|r_j|\sigma_j$ from each $\Pi_{j\neq i}^s \in \mathcal{G}$ it verifies $\mathtt{Auth\_Ver}(U_j, 0|y_{\langle l_j,v_j\rangle}|r_j|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ and checks whether $|r_j| \stackrel{?}{=} \kappa$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi_i^s \in \mathcal{G}$ defines $\mathtt{sid}_i^s := r_1|\ldots|r_n$ and $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_i - 1,\ldots,1\}$. Then, Π_1^s computes $X_1 := \mathtt{TDH1\_Up}(n-1, 0, x_{\langle n-1,0\rangle}, y_{\langle n-1,1\rangle}, Y_1)$, $\hat{Y} := \mathtt{TDH1\_Exp}^\star(n-2, X_1)$, $\sigma_1 := \mathtt{Auth\_Sig}(U_1, 1|\hat{Y}|\mathtt{sid}_1^s|\mathtt{pid}_1^s)$, and broadcasts $U_1|1|\hat{Y}|\sigma_1$.

Round 3. When $\Pi_{i\neq 1}^s \in \mathcal{G}$ receives $U_1|1|\hat{Y}|\sigma_1$ it verifies $\mathtt{Auth\_Ver}(U_1, 1|\hat{Y}|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_1) \stackrel{?}{=} 1$. If this verification fails then the protocol outputs a failure. Else $\Pi_{i\neq 1}^s \in \mathcal{G}$ obtains $X_i := \mathtt{TDH1\_Up}(l_i, v_i, x_{\langle l_i,v_i\rangle}, y_{\langle l_i,1-v_i\rangle}, Y_i)$. Each $\Pi_i^s \in \mathcal{G}$ computes $(K_i^s, \mu_i) := \mathtt{TDH1\_Con}(x_{\langle 0,0\rangle}, \mathtt{sid}_i^s)$ where $x_{\langle 0,0\rangle} \in X_i$ and erases every other private information from $\mathtt{state}_i^s$ (including all $x_{\langle l,v\rangle}$, and $\rho_0,\ldots,\rho_n$) except for K_i^s. Further, it computes $\sigma_i := \mathtt{Auth\_Sig}(U_i, 2|\mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$, and broadcasts $U_i|2|\sigma_i$.

Key Computation. When $\Pi_i^s \in \mathcal{G}$ receives $U_j|2|\sigma_j$ from each $\Pi_{j\neq i}^s \in \mathcal{G}$ it verifies $\mathtt{Auth\_Ver}(U_j, 2|\mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi_i^s \in \mathcal{G}$ computes $\mathbf{K}_i^s := \mathtt{TDH1\_Key}(K_i^s)$, erases every other private information from $\mathtt{state}_i^s$ (including K_i^s), and accepts with $\mathbf{K}_i^s$.

Note that TDH1.Setup requires three communication rounds. Note also that the intermediate value K_i^s is derived from $x_{\langle 0,0\rangle}$ and that both these values are never sent over the public communication channel but computed locally by all participants upon receiving enough information.

10.3.6 Security Analysis of Static TDH1

Similar to the compilers in Chapter 9 our TDH1 protocol uses digital signature for the purpose of authentication. Hence, following the discussion in Section 9.2.1 we assume that internal states of participating oracles are independent from the long-lived keys of the corresponding users in order to allow curios behavior of the adversary without necessarily corruption the user. Thus, at any time during the execution of static TDH1 the internal state information $\mathtt{state}_U^s$ is independent of LL_U.

The following theorem shows that the static TDH1 protocol is AKE-secure with respect to the requirements of backward secrecy in the weak-corruption model and forward secrecy in the strong-corruption model.

Theorem 10.6 (AKE-Security of Static TDH1). *Let (α, β) be an adversarial setting sampled from $\{(\texttt{wbs}, \texttt{wcm-bs}), (\texttt{sfs}, \texttt{scm})\}$. If Σ is EUF-CMA, F is pseudo-random, and the* TDDH *assumption holds in $\mathbb{G}$ then* TDH1 *is* AGKE-α*, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{TDH1}}(\kappa) \leq 2N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s\mathsf{Adv}^{\texttt{TDDH}}_{T_N,\mathbb{G}}(\kappa) + 2(N+3)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 8$ and corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{TDH1}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{TDH1}}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π^s_i. Keep in mind that on the test query the adversary receives either a random string or a session group key $\mathbf{K}^s_i$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|\texttt{seqn}|m|\sigma$ (or $U_i|\texttt{seqn}|\sigma$) with the sequence number $\texttt{seqn} \in \{0, 1, 2\}$ such that σ is a valid signature on m (or on μ_i) that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery; such event is denoted Forge. Hence,

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \Pr[\mathsf{Win}^{\mathsf{ake}}_0]| \leq \Pr[\mathsf{Forge}]. \tag{10.5}$$

In order to estimate $\Pr[\mathsf{Forge}]$ we show that using $\mathcal{A}$ we can construct a EUF-CMA forger $\mathcal{F}$ against the signature scheme Σ as follows. $\mathcal{F}$ is given a public key pk and has access to the corresponding signing oracle. $\mathcal{F}$ chooses uniformly at random a user $U_{i^*} \in \mathcal{U}$ and defines $pk_{i^*} := pk$. All other key pairs, i.e., (sk_i, pk_i) for every $U_{i \neq i^*} \in \mathcal{U}$ are generated honestly using $\Sigma.\texttt{Gen}(1^\kappa)$. The forger simulates all queries of $\mathcal{A}$ in a natural way by executing TDH1 operations by itself, and by obtaining the necessary signatures with respect to pk_{i^*} from its signing oracle. This is a perfect simulation for $\mathcal{A}$ since by assumption no $Corrupt(U_{i^*})$ may occur (otherwise $\mathcal{F}$ would not be able to answer it). Assuming Forge occurs, $\mathcal{A}$ outputs a new valid message/signature pair with respect to some pk_i; since i^* was randomly chosen and the simulation is perfect, $\Pr[i = i^*] = 1/N$. In that case $\mathcal{F}$ outputs this pair as its forgery. Its success probability is given by $\Pr[\mathsf{Forge}]/N$. This implies $\Pr[\mathsf{Forge}] \leq N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa)$. Hence,

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \Pr[\mathsf{Win}^{\mathsf{ake}}_0]| \leq N\mathsf{Succ}^{\texttt{euf-cma}}_{\Sigma}(\kappa). \tag{10.6}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if a nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. If q_s is the total number of protocol sessions, the probability that a randomly chosen nonce r_i appears twice is bound by $q_\mathsf{s}^2/2^\kappa$ for one given user. Since there are at most N users we obtain

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_2] - \Pr[\mathsf{Win}^{\mathsf{ake}}_1]| \leq \frac{Nq_\mathsf{s}^2}{2^\kappa}. \tag{10.7}$$

Note that this game excludes replay attacks in operations of TDH1.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q^*_\mathsf{s} \in [1, q_\mathsf{s}]$ as a guess for the number of sessions invoked before $\mathcal{A}$ asks the query

$Test$. If this query does not occur in the q_{s}^*-th session then the simulation fails and bit b' is set at random. Let Q be the event that this guess is correct. Obviously, $\Pr[\mathsf{Q}] = 1/q_{\mathsf{s}}$. Then we get

$$\begin{aligned}\Pr[\mathsf{Win}_3^{\mathsf{ake}}] &= \Pr[\mathsf{Win}_3^{\mathsf{ake}} \wedge \mathsf{Q}] + \Pr[\mathsf{Win}_3^{\mathsf{ake}} \wedge \neg\mathsf{Q}] \\ &= \Pr[\mathsf{Win}_3^{\mathsf{ake}}|\mathsf{Q}]\Pr[\mathsf{Q}] + \Pr[\mathsf{Win}_3^{\mathsf{ake}}|\neg\mathsf{Q}]\Pr[\neg\mathsf{Q}] \\ &= \Pr[\mathsf{Win}_2^{\mathsf{ake}}]\frac{1}{q_{\mathsf{s}}} + \frac{1}{2}\left(1 - \frac{1}{q_{\mathsf{s}}}\right).\end{aligned}$$

This implies

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2}. \tag{10.8}$$

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ except that Δ is given a tuple from the real $\mathtt{TDDH}^{\star}_{T_N}$ distribution (as specified in Section 10.1.2) where T_N is a linear tree. In all sessions except for the q_{s}^*-th session Δ computes all secret values $x_{\langle l,v\rangle}$ on behalf of honest participants as specified in the protocol description. In the q_{s}^*-th session with n participants Δ injects public values $g^{x_{\langle l,v\rangle}}$ from $\mathtt{TDDH}^{\star}_{T_N}$ into the protocol execution. Note that in the q_{s}^*-th session the group size n might be smaller than N, thus the simulator considers only the tree T_n which is composed of T_N's nodes from level 0 to level $n-1$. The idea for the simulation is to assign Π_1^s to the (internal) node $\langle n-1, 0\rangle$, Π_2^s to the leaf node $\langle n-1, 1\rangle$, ..., Π_n^s to the leaf node $\langle 1, 1\rangle$, and to use for each node $\langle l, v\rangle \in T_n \setminus \langle 0, 0\rangle$ public values $g^{x_{\langle l,v\rangle}}$ taken from the given $\mathtt{TDDH}^{\star}_{T_N}$ distribution. The secret value $x_{\langle 0,0\rangle}$ used for the function `TDH1_Con` in the q_{s}^*-th session for all honest participants is also taken from the $\mathtt{TDDH}^{\star}_{T_N}$ distribution (thus $x_{\langle 0,0\rangle} := g^{x_{\langle 1,0\rangle}x_{\langle 1,0\rangle}}$). Since the session where the $Test$ query is asked in must be α-fresh, we can deduce that no $RevealState$ queries to Π_i^s or to any of its partners have been asked in the q_{s}^*-th session (assuming the guess of Δ is correct); otherwise Δ would not be able to answer them since it does not known secret values $x_{\langle l,v\rangle}$ of internal and leaf nodes of T_n with respect to $\mathtt{TDDH}^{\star}_{T_N}$. However, in all other sessions where Δ computes all $x_{\langle l,v\rangle}$ on behalf of honest participants as specified in the protocol description $RevealState$ queries can be easily answered. Note that as soon as honest oracles accept in the q_{s}^*-th session $x_{\langle 0,0\rangle}$ is erased, and thus remains unknown to $\mathcal{A}$.

Since $\mathtt{TDDH}^{\star}_{T_N}$ is a real distribution we conclude that this game is a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{10.9}$$

Game $\mathbf{G}_5$. This game is identical to Game $\mathbf{G}_4$ except that Δ is given a tuple from the random $\mathtt{TDDH}^{\$}_{T_N}$ distribution where T_N is a linear tree. Similar to Game $\mathbf{G}_4$, in all sessions except for the q_{s}^*-th session Δ computes all secret values $x_{\langle l,v\rangle}$ on behalf of honest participants as specified in the protocol description. In the q_{s}^*-th session with n participants Δ injects $g^{x_{\langle l,v\rangle}}$ values from $\mathtt{TDDH}^{\$}_{T_N}$ into the protocol execution. Again, since in the q_{s}^*-th session the group size n might be smaller than N the simulator considers only the tree T_n which is composed of T_N's nodes from level 0 to level $n-1$ (the same assignment idea as above). Thus, for each node $\langle l, v\rangle \in T_n \setminus \langle 0, 0\rangle$ the simulator uses public values $g^{x_{\langle l,v\rangle}}$ taken from the given $\mathtt{TDDH}^{\$}_{T_N}$ distribution. Thus, the secret value $x_{\langle 0,0\rangle}$ used for the function `TDH1_Con` in the q_{s}^*-th session for all honest participants is also taken from $\mathtt{TDDH}^{\$}_{T_N}$ implying that $x_{\langle 0,0\rangle} := g^r$, $r \in_R \mathbb{G}$. Since the session where the $Test$ query is asked in must be α-fresh, we can deduce that no $RevealState$ queries to Π_i^s or to any of its partners have been asked in the q_{s}^*-th session (assuming the guess of Δ is correct); otherwise Δ would not be able to answer them. However, in all other sessions where Δ computes all $x_{\langle l,v\rangle}$ on

behalf of honest participants as specified in the protocol description $RevealState$ queries can be easily answered. Note that as soon as honest oracles accept in the q_{s}^*-th session the random $x_{\langle 0,0\rangle}$ is erased, and thus remains unknown to $\mathcal{A}$. By a "hybrid argument" we get

$$|\Pr[\mathsf{Win}_5^{\mathsf{ake}}] - \Pr[\mathsf{Win}_4^{\mathsf{ake}}]| \leq \mathsf{Adv}_{T_N,\mathbb{G}}^{\mathtt{TDDH}}(\kappa). \tag{10.10}$$

Note that in this game $x_{\langle 0,0\rangle} = g^{r_{\langle 0,0\rangle}}$ is a uniformly distributed value in $\mathbb{G}$ (or equivalently in $\mathbb{Z}_q$).

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session a random bit string sampled from $\{0,1\}^\kappa$ is used instead of the truncated value $x_{\langle 0,0\rangle\lfloor\kappa}$ inside the function TDH1_Con. Since in Game $\mathbf{G}_5$ the entire value $x_{\langle 0,0\rangle}$ is already random in $\mathbb{G}$ (and according to Corollary 10.2 and Remark 10.3 also uniform in $\mathbb{Z}_q$) both games are identical. Thus, replacing $x_{\langle 0,0\rangle}$ with a randomly sampled bit string can be seen as a "bridging step" so that

$$\Pr[\mathsf{Win}_6^{\mathsf{ake}}] = \Pr[\mathsf{Win}_5^{\mathsf{ake}}]. \tag{10.11}$$

Game $\mathbf{G}_7$. This game is identical to Game $\mathbf{G}_6$ except that in the q_{s}^*-th session each ρ_i, $i = 0,\ldots,n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that K is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{7,l}$, $l = 0,\ldots,n+1$ such that $\mathbf{G}'_{7,0} = \mathbf{G}_6$ and $\mathbf{G}'_{7,n+1} = \mathbf{G}_7$. That is, in the q_{s}^*-th session in each $\mathbf{G}'_{7,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the protocol (as output of the pseudo-random function f) whereas in $\mathbf{G}'_{7,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 0,\ldots,n-1$ by a random bit string implies uniform distribution of the PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that $x_{\langle 0,0\rangle\lfloor\kappa}$ used to compute ρ_0 is already uniform according to Game $\mathbf{G}_6$.

Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_7^{\mathsf{ake}}] - \Pr[\mathsf{Win}_6^{\mathsf{ake}}]| \leq (N+1)\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{10.12}$$

Game $\mathbf{G}_8$. This game is identical to Game $\mathbf{G}_7$ except that in the q_{s}^*-th session the confirmation token μ and the session group key $\mathbf{K}$ are replaced by two different random values sampled from $\{0,1\}^\kappa$. This can be done since the intermediate value K is uniformly distributed according to Game $\mathbf{G}_7$. Obviously,

$$|\Pr[\mathsf{Win}_8^{\mathsf{ake}}] - \Pr[\mathsf{Win}_7^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{10.13}$$

Since $\mathbf{K}$ is uniformly distributed $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_8^{\mathsf{ake}}] = \frac{1}{2}. \tag{10.14}$$

Considering Equations 10.6 to 10.14 we get

$$\begin{aligned}
\Pr[\mathsf{Game}_{\alpha,\beta,\mathtt{TDH1}}^{\mathsf{ake}-b}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\
&= N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_8^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Adv}_{T_N,\mathbb{G}}^{\mathtt{TDDH}}(\kappa) + (N+3)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{TDH1}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}^{\mathtt{TDDH}}_{T_N,\mathbb{G}}(\kappa) + 2(N+3)q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa).$$

□

In the following we show that TDH1 satisfies the requirement of MA-security.

Theorem 10.7 (MA-Security of TDH1). *If Σ is EUF-CMA and F is collision-resistant then* TDH1 *is* MAGKE*, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{TDH1}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}^{\mathsf{ma}}_i$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator Δ.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{TDH1}}(\kappa)$ played between a simulator Δ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π^s_i accepts with $\mathbf{K}^s_i$ and another user $U_j \in \mathtt{pid}^s_i$ that is uncorrupted at the time Π^s_i accepts and either does not have a corresponding oracle Π^s_j with $(\mathtt{pid}^s_j, \mathtt{sid}^s_j) = (\mathtt{pid}^s_i, \mathtt{sid}^s_i)$ or has such an oracle but this oracle accepts with $\mathbf{K}^s_i \neq \mathbf{K}^s_j$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation aborts if $\mathcal{A}$ asks a $Send$ query on some $U_i|\mathtt{seqn}|m|\sigma$ (or on $U_i|\mathtt{seqn}|\sigma$) with the sequence number $\mathtt{seqn} \in \{0,1,2\}$ such that σ is a valid signature on m (on μ_i) that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 10.6 we obtain,

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_1] - \Pr[\mathsf{Win}^{\mathsf{ma}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) \tag{10.15}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails if a nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. Similar to Equation 10.7 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ma}}_2] - \Pr[\mathsf{Win}^{\mathsf{ma}}_1]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} \tag{10.16}$$

Note that this eliminates attacks where Π^s_i during any session of the TDH1 protocol receives a replayed message of the form $U_j|\mathtt{seqn}|m|\bar{\sigma}_j$ or $U_j|\mathtt{seqn}|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π^s_i does not accept unless it successfully verifies all required σ_j for all $U_j \in \mathtt{pid}^s_i$. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}^s_i$ that is uncorrupted at the time Π^s_i accepts there exists a corresponding instance oracle Π^s_j with $(\mathtt{pid}^s_j, \mathtt{sid}^s_j) = (\mathtt{pid}^s_i, \mathtt{sid}^s_i)$. Thus, according to Definition 8.15 $\mathcal{A}$ wins in this game only if any of these oracles has accepted with $\mathbf{K}^s_i \neq \mathbf{K}^s_j$.

Assume that $\mathcal{A}$ wins in this game. Then Π^s_i and Π^s_j have accepted with $\mathbf{K}^s_i = f_{K^s_i}(v_2)$ resp. $\mathbf{K}^s_i = f_{K^s_j}(v_2)$ where K^s_i resp. K^s_j are corresponding intermediate values, and $\mathbf{K}^s_i \neq \mathbf{K}^s_j$. Having eliminated forgeries and replay attacks between the oracles of any two uncorrupted users we follow that messages exchanged between Π^s_i and Π^s_j have been delivered without any modification. In particular, oracle Π^s_i received the signature σ_j computed on $\mu_j = f_{K^s_j}(v_1)$ and Π^s_j received the signature σ_i computed on $\mu_i = f_{K^s_i}(v_1)$. Since both oracles have accepted we

have $\mu_i = \mu_j$; otherwise oracles cannot have accepted because signature verification would fail. The probability that $\mathcal{A}$ wins in this game is given by

$$\Pr[\mathbf{K}_i^s \neq \mathbf{K}_j^s \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] = \Pr[f_{K_i^s}(v_2) \neq f_{K_j^s}(v_2) \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).$$

Hence,

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa). \tag{10.17}$$

Considering Equations 10.15 to 10.17 we get the desired inequality

$$\begin{aligned}\mathsf{Succ}_{\mathtt{TDH1}}^{\mathsf{ma}}(\kappa) &= \Pr[\mathsf{Win}_0^{\mathsf{ma}}] \\ &\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa).\end{aligned}$$

□

In the following we focus on several aspects concerning the n-contributiveness of TDH1. Recall that our Definition 8.18 ensures that the adversary $\mathcal{A}$ fails to influence some honest participant Π_U^s to accept the session group key chosen by $\mathcal{A}$. In particular, we mentioned that it prevents attacks where a session group key repeats in two different sessions due to the adversarial actions. We also mentioned that in the TDH1 protocol the random nonces r_i and the function $\mathtt{TDH1\_Con}(x_{\langle 0,0\rangle}, r_1|\ldots|r_n)$ are used to provide contributiveness. Note that in TDH1 the adversary is able to influence the resulting value for $x_{\langle 0,0\rangle}$ (as shown below). Therefore, in our proof we show that despite being able to influence $x_{\langle 0,0\rangle}$ the adversary is still unable to influence the resulting session group key $\mathbf{K}$.

How can $\mathcal{A}$ influence $x_{\langle 0,0\rangle}$ in TDH1?

Before proceeding with the proof of n-contributiveness of TDH1 in Theorem 10.8, we would like to show how $\mathcal{A}$ can influence $x_{\langle 0,0\rangle}$, more precisely the goal of $\mathcal{A}$ is to influence that the same $x_{\langle 0,0\rangle}$ is computed by the oracles of some uncorrupted user in two different sessions. For simplicity we assume three participants: Π_1^s, Π_2^s, and Π_3^s, and consider that Π_1^s and Π_3^s are malicious (controlled by $\mathcal{A}$) whereas Π_2^s is honest. We consider two different sessions: session A and session B, whereby session B takes place later than A. In both sessions the tree is as in Figure 10.1. We assume that in session A all oracles behave as specified in the protocol except that neither Π_1^s nor Π_3^s erase their states. At some point before session B is started, the adversary $\mathcal{A}$ (that controls Π_1^s and Π_3^s) computes $z := x_{\langle 1,0\rangle}x_{\langle 1,1\rangle}$ where $x_{\langle 1,0\rangle}$ is a value computed by Π_1^s and $x_{\langle 1,1\rangle}$ is the exponent chosen by Π_3^s in session A. Obviously, g^z equals to $x_{\langle 0,0\rangle}$ computed by all oracles in session A. The goal of $\mathcal{A}$ is to influence honest $\Pi_2^{s'}$ to compute the same $x_{\langle 0,0\rangle}$ in session B. In session B, the exponent $x_{\langle 2,1\rangle}$ used by honest $\Pi_2^{s'}$ is likely to be different compared to session A. To proceed with the attack $\mathcal{A}$ waits for $\Pi_2^{s'}$ to broadcast $y_{\langle 2,1\rangle} = g^{x_{\langle 2,1\rangle}}$ in session B (note the communication is asymmetric). Then, the adversary shows its curiosity by revealing $x_{\langle 2,1\rangle}$ (via the $RevealState(\Pi_2^{s'})$ query); chooses $x_{\langle 2,0\rangle}$ truly at random on behalf of $\Pi_1^{s'}$, computes $x_{\langle 1,0\rangle} := g^{x_{\langle 2,0\rangle}x_{\langle 2,1\rangle}}$ and $x_{\langle 1,1\rangle} := z/x_{\langle 1,0\rangle}$. To complete the attack, $\mathcal{A}$ broadcasts $y_{\langle 2,0\rangle} := g^{x_{\langle 2,0\rangle}}$ and $y_{\langle 1,1\rangle} := g^{x_{\langle 1,1\rangle}}$ on behalf of $\Pi_1^{s'}$ and $\Pi_3^{s'}$, respectively. It is easy to check that $\Pi_2^{s'}$ computes $x_{\langle 0,0\rangle} = g^z$ in session B.

In the following theorem we argue that the ability to influence $x_{\langle 0,0\rangle}$ does not provide the adversary $\mathcal{A}$ with any additional advantage in its attack on influencing/predicting the session group

key $\mathbf{K}$. Similar to the argumentation in Theorem 9.34 we discuss the probability that for an honest participant $\Pi_{i^*}^s$ the adversary $\mathcal{A}$ is able to influence/predict any of the values $\rho_{i^*}, \ldots, \rho_n$ (note that $K = \rho_n$), or $\mathbf{K}$ computed by the collision-resistant pseudo-random function f. This is equivalent to the event that in the prepare stage $\mathcal{A}$ is able to output any $\rho_{i^*}, \ldots, \rho_n$, or $\mathbf{K}$ which $\Pi_{i^*}^s$ computes in any session of the attack stage. Note that according to Lemma 5.20 in our proof we do also consider the upper-bound for the success probability of the adversary in case that its strategy differs from influencing any value in $\rho_{i^*}, \ldots, \rho_n$.

Theorem 10.8 (n-Contributiveness of Static TDH1). *If F is collision-resistant pseudo-random and π is one-way then static TDH1 is a n-CGKE protocol, and*

$$\mathsf{Succ}_{\mathtt{TDH1}}^{\mathsf{con}-n}(\kappa) \leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N+2)q_s\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + q_s\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + Nq_s\mathsf{Succ}_\pi^{\mathsf{ow}}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Remark 10.9. Note that some arguments in this proof are intuitive for the same reasons as mentioned in Remark 9.12.

Proof (partially informal). In the following we consider an adversary $\mathcal{A}$ from Definition 8.18. Assume that $\mathcal{A}$ wins in $\mathsf{Game}_{\mathtt{TDH1}}^{\mathsf{con}-n}(\kappa)$ (which event we denote $\mathsf{Win}^{\mathsf{con}}$). Then at the end of the stage prepare it returned $\tilde{\mathbf{K}}$ such that in the stage attack an honest oracle $\Pi_{i^*}^s \in \mathcal{G}$ accepted with $\mathbf{K}_{i^*}^s = \tilde{\mathbf{K}}$. According to the construction of $\mathbf{K}_{i^*}^s$ we follow that $\tilde{\mathbf{K}} = f_{K_{i^*}^s}(v_2)$ where $K_{i^*}^s$ is the intermediate value computed by $\Pi_{i^*}^s$.

Game $\mathbf{G}_0$. This is the real game $\mathsf{Game}_{\mathtt{TDH1}}^{\mathsf{con}-n}(\kappa)$, in which the honest players are replaced by a simulator.

Game $\mathbf{G}_1$. In this game we abort the simulation if the same nonce r_i is used by any honest oracle Π_i^s in two different sessions. Considering r_i being uniform for every honest oracle Π_i^s, and since there are at most N users we have

$$\Pr[\mathsf{Win}_0^{\mathsf{con}}] - \Pr[\mathsf{Win}_1^{\mathsf{con}}] \leq \frac{Nq_s^2}{2^\kappa}. \tag{10.18}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_{i^*} computed by $\Pi_{i^*}^s$ in any session of the attack stage.[4] According to Lemma 5.20 we need to estimate the probability of the opposite event, i.e., that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*}. We consider two cases: $i^* = 1$ and $i^* > 1$. Note that all other oracles except for $\Pi_{i^*}^s$ can be corrupted.

Case $i^* = 1$: In any session of the attack stage honest oracle Π_1^s computes $\rho_1 := f_{\rho_0 \oplus \pi(r_1)}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_0 \oplus \pi(r_1)$ (denoted R_1) in the prepare stage $\mathcal{A}$'s probability to output $\rho_1 = f_{R_1}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}_F^{\mathsf{coll}}(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_1 in the prepare stage and tries to influence Π_1^s computing exactly this value in some session of the attack stage. Note that the honest oracle Π_1^s chooses r_1 uniformly

[4] Note, in $\mathbf{G}_0$ and $\mathbf{G}_1$ the adversary only outputs a value for the resulting group key. In $\mathbf{G}_2$ we consider the additional (in)ability of the adversary to output the value for ρ_{i^*}. Since we are only interested in the probability of the adversarial success under this "condition event" (without changing the game in case that this event occurs; see also Section 5.6.1) the simulator does not need to detect whether $\mathcal{A}$ is able to output the correct value or not. The same considerations are applicable to $\mathbf{G}_3$ w.r.t. $K_{i^*}^s$.

at random for every session in the attack stage. Since π is a permutation the value $\pi(r_1)$ is also uniform and fixed for every attack-ed session. Hence, the adversary must influence in the attack-ed session the oracle $\Pi^s_{i^*}$ to compute $\rho_0 = R_1 \oplus \pi(r_1)$ which is fixed and uniformly distributed for each such session. Note that $\mathcal{A}$ learns the required ρ_0 only after having received r_1, that is during the attack-ed session. Since U_{i^*} is uncorrupted its oracle computes ρ_0 according to the protocol specification, that is $\rho_0 := f_{x_{\langle 0,0\rangle}}(v_0)$. Having excluded PRF collisions and random guesses we consider the required PRF key $x_{\langle 0,0\rangle}$ (which is influenceable by $\mathcal{A}$ as shown above) as a fixed value unknown to $\mathcal{A}$ in any session of the attack stage. The probability that $\mathcal{A}$ recovers it is intuitively bound by $\mathsf{Adv}^{\mathsf{prf}}_F(\kappa)$. This is because any adversary that is able to reveal the PRF key can act as a distinguisher for the pseudo-randomness of f.

Case $i^* > 1$: In any session of the attack stage honest oracle $\Pi^s_{i^*}$ computes $\rho_{i^*} := f_{\rho_{i^*-1}\oplus\pi(r_{i^*})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*-1}\oplus\pi(r_{i^*})$ (denoted R_{i^*}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*} = f_{R_{i^*}}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*} in the prepare stage and tries to influence $\Pi^s_{i^*}$ computing exactly this value in some session of the attack stage. Since r_{i^*} is chosen by honest $\Pi^s_{i^*}$ at random in every attack-ed session and π is a permutation the value $\pi(r_{i^*})$ is uniform and fixed for each attack-ed session. Hence, the adversary must influence in the attack stage the oracle $\Pi^s_{i^*}$ to compute $\rho_{i^*-1} = R_{i^*} \oplus \pi(r_{i^*})$ which is fixed and uniformly distributed for each attack-ed session. Note that $\mathcal{A}$ learns the required ρ_{i^*-1} only after having received r_{i^*}, that is during the attack-ed session. Since U_{i^*} is uncorrupted its oracle computes ρ_{i^*-1} according to the protocol specification, that is $\rho_{i^*-1} := f_{\rho_{i^*-2}\oplus\pi(r_{i^*-1})}(v_0)$. Having excluded PRF collisions and random guesses we consider the PRF key $\rho_{i^*-2} \oplus \pi(r_{i^*-1})$ as a fixed value unknown to the adversary. The probability that $\mathcal{A}$ recovers it is intuitively bound by $\mathsf{Adv}^{\mathsf{prf}}_F(\kappa)$. This is because any adversary that is able to reveal the PRF key can act as a distinguisher for the pseudo-randomness of f.

Since there are at most N users and q_s sessions we have,

$$\Pr[\mathsf{Win}^{\mathsf{con}}_1] - \Pr[\mathsf{Win}^{\mathsf{con}}_2] \leq q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}. \tag{10.19}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output ρ_{i^*} computed by $\Pi^s_{i^*}$ in any session of the attack stage. Note that we do not need to consider the values ρ_l, $l < i^*$ computed by $\Pi^s_{i^*}$ since in order to compute $K^s_{i^*}$ every honest oracle must compute the whole sequence $\rho_0, \ldots, \rho_n$. Thus, it is sufficient to argue that the probability of $\mathcal{A}$ influencing any ρ_l, $l \geq i^*$, computed by $\Pi^s_{i^*}$ in any attack-ed session is negligible.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $K^s_{i^*} := \rho_n$ computed by $\Pi^s_{i^*}$ in any session of the attack stage. Again, the simulator does not need to detect whether this event occurs since both games proceed identical in any case. According to Lemma 5.20 we need to estimate the probability of the opposite event, i.e., that $\mathcal{A}$ being in the prepare stage is able to output $K^s_{i^*}$.

Based on the "hybrid technique" (used for the first time in the proof of Theorem 9.11) we define a sequence of auxiliary games $\mathbf{G}'_{3,l}$, $l = i^*, \ldots, n$. Each of these games is identical to the previous one in the sequence with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output ρ_l computed by $\Pi^s_{i^*}$ in any session of the attack stage. Obviously, $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$ and $\mathbf{G}'_{3,n} = \mathbf{G}_3$. According to Lemma 5.20 we need to estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_l.

Since $\rho_l := f_{\rho_{l-1} \oplus \pi(r_l)}(v_0)$ for all $l > i^*$ and each r_l is not chosen by $\Pi^s_{i^*}$ each two consecutive auxiliary games have the same difference. Hence, it is sufficient to compute this difference between any two consecutive auxiliary games. In the following we compute the difference between $\mathbf{G}'_{3,i^*+1}$ and $\mathbf{G}'_{3,i^*} = \mathbf{G}_2$. We intuitively estimate the probability that $\mathcal{A}$ being in the prepare stage is able to output ρ_{i^*+1} computed by $\Pi^s_{i^*}$ in any session of the attack stage.

We argue by intuition. Since $\Pi^s_{i^*}$ is honest, in the attack stage it computes $\rho_{i^*+1} := f_{\rho_{i^*} \oplus \pi(r_{i^*+1})}(v_0)$. Intuitively, without knowing the PRF key given by the XOR sum $\rho_{i^*} \oplus \pi(r_{i^*+1})$ (denoted R_{i^*+1}) in the prepare stage $\mathcal{A}$'s probability to output $\rho_{i^*+1} = f_{R_{i^*+1}}(v_0)$ in that stage is bound by the probability that either $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + 1/2^\kappa$. Thus, we have to discuss the case where $\mathcal{A}$ chooses R_{i^*+1} in the prepare stage and tries to influence $\Pi^s_{i^*}$ computing exactly this value in some session of the attack stage. Recall that $\mathcal{A}$ is allowed to corrupt any user $U_{l \neq i^*}$, and thus choose each nonce r_l, $l \neq i^*$. Since $\mathcal{A}$ learns ρ_{i^*} only in the attack-ed session (as observed in Game $\mathbf{G}_2$) and having excluded PRF collisions and random guesses the probability that $\mathcal{A}$ is able to influence $\Pi^s_{i^*}$ computing R_{i^*+1} in the attack stage is bound by the probability that in the attack-ed session $\mathcal{A}$ computes the appropriate nonce r_{i^*+1} such that $\pi(r_{i^*+1}) = R_{i^*+1} \oplus \rho_{i^*}$ holds. Since π is one-way this probability is intuitively bound by $\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa)$. Thus, $\mathcal{A}$ is able to output ρ_{i^*+1} while being in the prepare stage with the probability of at most $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + \mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + 1/2^\kappa$. Since there are at most q_s sessions the total probability that $\mathcal{A}$ is able to do this is at most

$$q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}.$$

The above sum upper-bounds the difference between $\mathbf{G}'_{3,i+1}$ and $\mathbf{G}_2$. Since there are at most N auxiliary games (due to $n \leq N$) we obtain

$$\Pr[\mathsf{Win}^{\mathsf{con}}_2] - \Pr[\mathsf{Win}^{\mathsf{con}}_3] \leq Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa) + \frac{Nq_\mathsf{s}}{2^\kappa}. \tag{10.20}$$

As a consequence of the "condition event" in this game, in every subsequent game of the sequence the adversary, while being in the prepare stage, is not able to output $K^s_{i^*}$ computed by $\Pi^s_{i^*}$ in any session of the attack stage.

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ with the "condition event" that $\mathcal{A}$ being in the prepare stage is NOT able to output $\mathbf{K}^s_{i^*}$ computed by $\Pi^s_{i^*}$ in any session of the attack stage. Note that in every attack-ed session, the honest oracle $\Pi^s_{i^*}$ computes $\mathbf{K}^s_{i^*} := f_{K^s_{i^*}}(v_2)$. Intuitively, since in the prepare stage $K^s_{i^*}$ is unknown to $\mathcal{A}$ (as observed in Game $\mathbf{G}_3$), $\mathcal{A}$'s probability to output $\mathbf{K}^s_{i^*}$ in that stage is bound by the probability that $\mathcal{A}$ chooses a different PRF key and succeeds (thus a PRF collision occurs) or succeeds by a random guess, i.e., $\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + 1/2^\kappa$. Hence,

$$\Pr[\mathsf{Win}^{\mathsf{con}}_3] - \Pr[\mathsf{Win}^{\mathsf{con}}_4] \leq q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + \frac{q_\mathsf{s}}{2^\kappa}. \tag{10.21}$$

Obviously the probability of $\mathsf{Win}^{\mathsf{con}}_4$ is 0, meaning that the adversary did not output a correct value $\tilde{\mathbf{K}}$ in the prepare stage.

Considering Equations 10.18 to 10.21 we obtain the desired inequality

$$\mathsf{Succ}^{\mathsf{con}-n}_{\mathtt{TDH1}}(\kappa) \leq \frac{Nq_\mathsf{s}^2 + Nq_\mathsf{s} + 2q_\mathsf{s}}{2^\kappa} + (N+2)q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_\mathsf{s}\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa).$$

□

Remark 10.10. Similar to the compilers C-ACON and C-AMACON, the role of nonces r_i in TDH1 is twofold: they ensure security against replay attacks and are also used to provide contributiveness for the resulting session group key.

10.4 Dynamic TDH1

In this section we extend the static version of the TDH1 protocol from the previous section by additional operations TDH1.Join$^+$ and TDH1.Leave$^+$ which are supposed to handle addition and exclusion of group members, respectively. For the description of this dynamic version of the TDH1 protocol we need the same authentication and key confirmation and derivation functions given in Sections 10.3.1 and 10.3.4, respectively. Additionally, we require some new functions concerning tree management and key exchange.

10.4.1 Additional Tree Management Functions

In addition to the initial tree management function Tree_Init described in Section 10.3.2 the dynamic TDH1 protocol requires functions to handle dynamic group changes. For each change the tree structure and the assignment of users to the corresponding leaf nodes has to be updated.

There is a special oracle $\Pi^s_\gamma \in \mathcal{G}$ which is involved in all dynamic changes. The index γ of this oracle depends on the event to be handled and on the current group $\mathcal{G}$ with the corresponding tree structure T_n. In the following we describe the additional tree management functions called TDH1.Join$^+$ and TDH1.Leave$^+$.

- Tree_Join$^+(T_n, \mathcal{G}, \mathcal{J})$. Let $n := |\mathcal{G}|$ and $n_J := |\mathcal{J}|$. This function prepends n_J leaf nodes to the root of T_n resulting in the updated tree T_{n+n_J}. Then, the function assigns each $\Pi^{s'}_j \in \mathcal{J}$, $j = 1, \ldots, n_J$, to the position $\langle n + j, 1\rangle$ in T_{n+n_J} and changes its index to $n + j$. Finally, $\Pi^s_\gamma := \Pi^s_n$. The function returns $(T_{n+n_J}, \mathcal{G} := \{\Pi^s_1, \ldots, \Pi^s_{n+n_J}\}, \Pi^s_\gamma)$. (see Figure 10.3 for an example)
- Tree_Leave$^+(T_n, \mathcal{G}, \mathcal{L})$. Let $n := |\mathcal{G}|$ and $n_L := |\mathcal{L}|$. This function removes $\langle l_j, v_j\rangle$ for each $\Pi^s_j \in \mathcal{L}$ from T_n resulting in the updated tree T_{n-n_L}. For each $\Pi^s_i \in \mathcal{G} \setminus \mathcal{L}$ the index is changed to $i - \#L_i$ where $\#L_i$ is the number of lower indexed oracles removed from T_n. If indices of all oracles have been changed then $\Pi^s_\gamma = \Pi^s_1$, else Π^s_γ is the highest indexed oracle whose index remained unchanged. The function returns $(T_{n-n_L}, \mathcal{G} := \{\Pi^s_1, \ldots, \Pi^s_{n-n_L}\}$, $\Pi^s_\gamma)$. (see Figure 10.4 for an example)

Note that one of the outputs of functions Tree_Join$^+$ and Tree_Leave$^+$ is a modified group $\mathcal{G} := \mathcal{G} \bigcup \mathcal{J}$ and $\mathcal{G} := \mathcal{G} \setminus \mathcal{L}$, respectively.

10.4.2 Additional Key Exchange Functions

In addition to the key exchange functions described in Section 10.3.3 the dynamic version of TDH1 requires the following functions:

- TDH1_Rand(X). The function returns $X' := \{g^{x^2_{\langle l,v\rangle}} \mid \forall x_{\langle l,v\rangle} \in X\}$.
- TDH1_Up$^\star(l, x_{\langle l,0\rangle}, Y)$. The function returns

$$X := \{x_{\langle j-1,0\rangle} := (y_{\langle j,1\rangle})^{x_{\langle j,0\rangle}} \mid y_{\langle j,1\rangle} \in Y, \forall j = l, \ldots, 1\}.$$

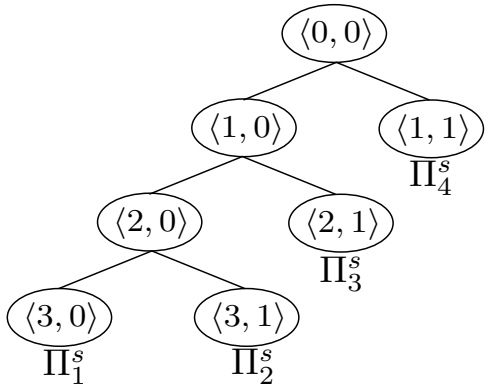

Fig. 10.3. Example: `Tree_Join`$^+$. This tree is returned by `Tree_Join`$^+$ on input T_3 and $\mathcal{G}$ as in Figure 10.1, and $\mathcal{J} = \{\Pi_1^{s'}\}$. Note that $\Pi_1^{s'}$ has been changed to Π_4^s, $\mathcal{G} = \{\Pi_1^s, \ldots, \Pi_4^s\}$ and Π_γ^s is Π_3^s.

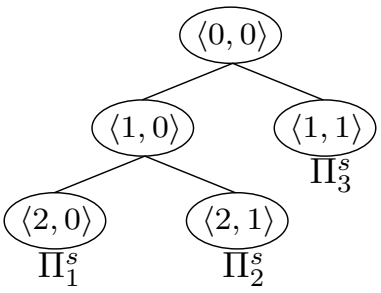

Fig. 10.4. Example: `Tree_Leave`$^+$. This tree is returned by `Tree_Leave`$^+$ on input T_4 and $\mathcal{G}$ as in Figure 10.3, and $\mathcal{L} = \{\Pi_3^s\}$. Note that Π_4^s is changed to Π_3^s since $\#L_1 = \#L_2 = 0$, and $\#L_4 = 1$. Π_γ^s is Π_2^s.

10.4.3 Protocol Execution

In order to process dynamic group changes more efficiently compared to the initialization procedure participants of the dynamic `TDH1` protocol need to save some additional private state information. For this purpose we need to extend the static setup operation of `TDH1` with additional computation steps, which are also required in both dynamic operations `TDH1.Join`$^+$ and `TDH1.Leave`$^+$.

Operation `TDH1.Setup` (Dynamic Case)

Figure 10.5 shows an example of the dynamic `TDH1.Setup` operation with three participating oracles Π_1^s, Π_2^s, and Π_3^s. The oracles proceed exactly as described for the static version of `TDH1.Setup` in Figure 10.2 until each of them computes $\rho_0 := f_{x_{\langle 0,0\rangle \lfloor \kappa}}(v_0)$, $\rho_1 := f_{\rho_0 \oplus \pi(r_1)}(v_0)$, $\rho_2 := f_{\rho_1 \oplus \pi(r_2)}(v_0)$, the intermediate value $K = \rho_3 := f_{\rho_2 \oplus \pi(r_2)}(v_0)$, and the resulting session group key $\mathbf{K} := f_K(v_2)$. Additionally, each Π_i^s computes X_i', i.e., $X_1' := \{g^{x^2_{\langle 2,0\rangle}}, g^{x^2_{\langle 1,0\rangle}}, g^{x^2_{\langle 0,0\rangle}}\}$, $X_2' := \{g^{x^2_{\langle 2,1\rangle}}, g^{x^2_{\langle 1,0\rangle}}, g^{x^2_{\langle 0,0\rangle}}\}$, and $X_3' := \{g^{x^2_{\langle 1,1\rangle}}, g^{x^2_{\langle 0,0\rangle}}\}$. Each Π_i^s saves $(\mathcal{G}, T_3, X_i)$ and erases all other state information. Let each updated $g^{x^2_{\langle l,v\rangle}}$ be denoted as $x'_{\langle l,v\rangle}$. Note that Π_1^s and Π_2^s still share $x'_{\langle 1,0\rangle}$ and $x'_{\langle 0,0\rangle}$, and all three oracles still share $x'_{\langle 0,0\rangle}$.

In the following we give the formal description of the dynamic `TDH1.Setup` operation.

Round 1. Each oracle $\Pi_i^s \in \mathcal{G}$ computes $T_n :=$ `Tree_Init`$(\mathcal{G})$, $(x_{\langle l_i,v_i\rangle}, y_{\langle l_i,v_i\rangle}) :=$ `TDH1_Pick`(1^κ), $r_i :=$ `Auth_Nonce`(1^κ), $\sigma_i :=$ `Auth_Sig`$(U_i, 0|y_{\langle l_i,v_i\rangle}|r_i|\mathtt{pid}_i^s)$, and broadcasts $U_i|0|y_{\langle l_i,v_i\rangle}|r_i|\sigma_i$.

Round 2. When $\Pi_i^s \in \mathcal{G}$ receives $U_j|0|y_{\langle l_j,v_j\rangle}|r_j|\sigma_j$ from each $\Pi_{j\neq i}^s \in \mathcal{G}$ it verifies `Auth_Ver`$(U_j, 0|y_{\langle l_j,v_j\rangle}|r_j|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ and checks whether $|r_j| \stackrel{?}{=} \kappa$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi_i^s \in \mathcal{G}$ defines $\mathtt{sid}_i^s := r_1| \ldots |r_n$ and $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_i - 1, \ldots, 1\}$. Then, Π_1^s computes $X_1 :=$ `TDH1_Up`$(n-1, 0, x_{\langle n-1,0\rangle}, y_{\langle n-1,1\rangle}, Y_1)$, $\hat{Y} :=$ `TDH1_Exp`$^\star(n-2, X_1)$, $\sigma_1 :=$ `Auth_Sig`$(U_1, 1|\hat{Y}|\mathtt{sid}_1^s|\mathtt{pid}_1^s)$, and broadcasts $U_1|1|\hat{Y}|\sigma_1$.

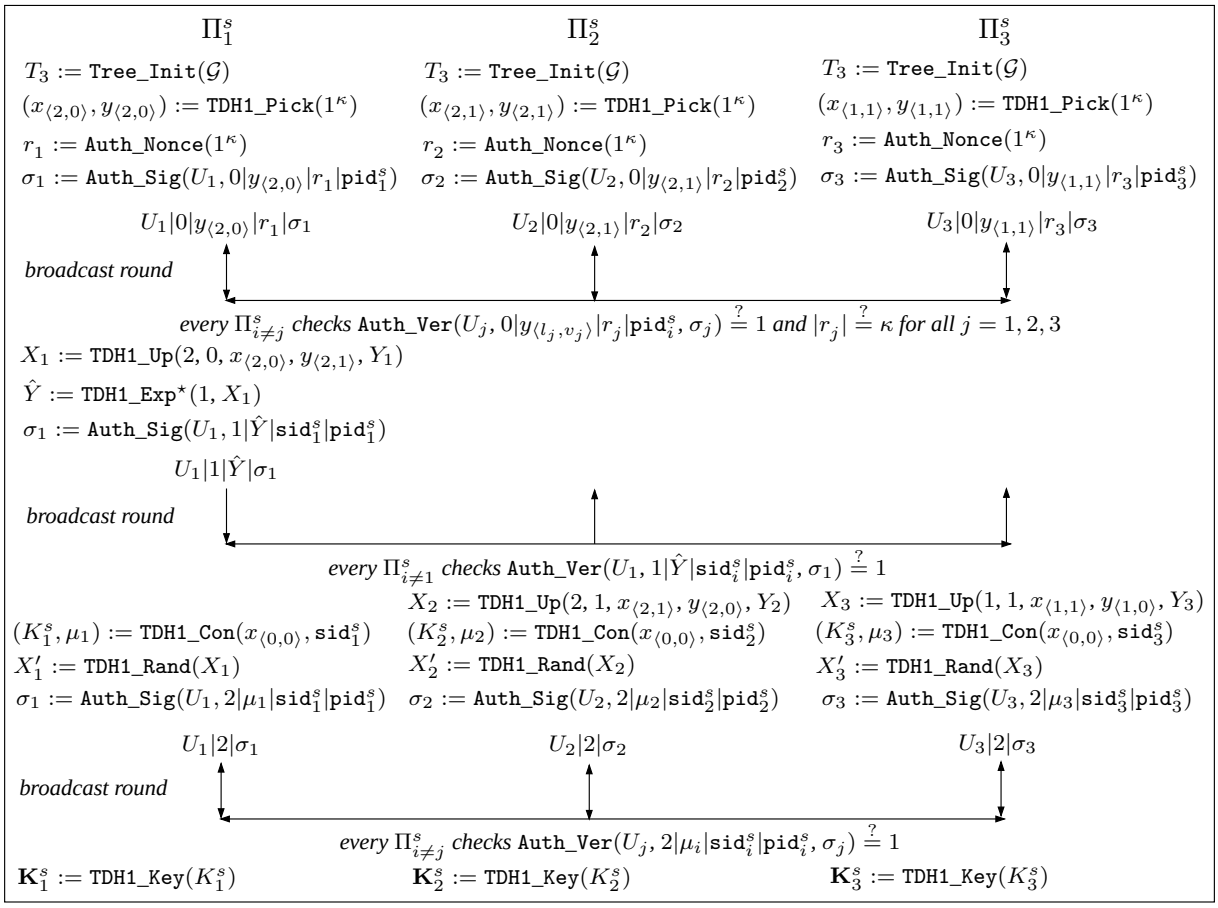

Fig. 10.5. Example: Operation TDH1.Setup (Dynamic Case) with $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s\}$. Public values: $\mathtt{sid}_i^s = r_1|r_2|r_3$, $Y_1 = Y_2 = \{y_{\langle 1,1\rangle}\}$, $Y_3 = \emptyset$, $\hat{Y} = \{y_{\langle 1,0\rangle}\}$, where $y_{\langle 1,1\rangle} = g^{x_{\langle 1,1\rangle}}$, $y_{\langle 1,0\rangle} = g^{x_{\langle 1,0\rangle}}$. Secret values: $X_1 = \{x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_2 = \{x_{\langle 2,1\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_3 = \{x_{\langle 1,1\rangle}, x_{\langle 0,0\rangle}\}$, where $x_{\langle 0,0\rangle} = g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$, $x_{\langle 1,0\rangle} = g^{x_{\langle 2,0\rangle} x_{\langle 2,1\rangle}}$ and $x_{\langle 2,0\rangle}$, $x_{\langle 2,1\rangle}$, $x_{\langle 1,1\rangle} \in_R \mathbb{G}$.

Round 3. When $\Pi_{i\neq 1}^s \in \mathcal{G}$ receives $U_1|1|\hat{Y}|\sigma_1$ it verifies $\mathtt{Auth\_Ver}(U_1, 1|\hat{Y}|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_1) \stackrel{?}{=} 1$. If this verification fails then the protocol outputs a failure. Else $\Pi_{i\neq 1}^s \in \mathcal{G}$ obtains $X_i := \mathtt{TDH1\_Up}(l_i, v_i, x_{\langle l_i, v_i\rangle}, y_{\langle l_i, 1-v_i\rangle}, Y_i)$. Each $\Pi_i^s \in \mathcal{G}$ computes $(K_i^s, \mu_i) := \mathtt{TDH1\_Con}(x_{\langle 0,0\rangle}, \mathtt{sid}_i^s)$ with $x_{\langle 0,0\rangle} \in X_i$, $X_i' := \mathtt{TDH1\_Rand}(X_i)$, and erases every private information from $\mathtt{state}_i^s$ (including all $x_{\langle l,v\rangle}$, and $\rho_0, \ldots, \rho_n$) except for K_i^s and X_i'. Further, it computes $\sigma_i := \mathtt{Auth\_Sig}(U_i, 2|\mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$, and broadcasts $U_i|2|\sigma_i$.

Key Computation. When $\Pi_i^s \in \mathcal{G}$ receives $U_j|2|\sigma_j$ from each $\Pi_{j\neq i}^s \in \mathcal{G}$ it verifies $\mathtt{Auth\_Ver}(U_j, 2|\mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi_i^s \in \mathcal{G}$ computes $\mathbf{K}_i^s := \mathtt{TDH1\_Key}(K_i^s)$, erases every private information from $\mathtt{state}_i^s$ (including K_i^s) except for X_i', and accepts with $\mathbf{K}_i^s$.

Once again, the main difference to the static version of TDH1.Setup is that each oracle has to save a tuple $(\mathcal{G}, T_{n+n_J}, X_i')$ where each secret value $x_{\langle l_i,0\rangle} \in X_i'$ is known to each oracle Π_j^s at position $\langle l_j, v_j\rangle$ with $l_j > l_i$. These values are essential for the efficient update of the session group key in case that dynamic group changes occur.

Operation TDH1.Join$^+$

Figure 10.6 shows an example of the TDH1.Join$^+$ operation. The initial group $\mathcal{G}$ consists of

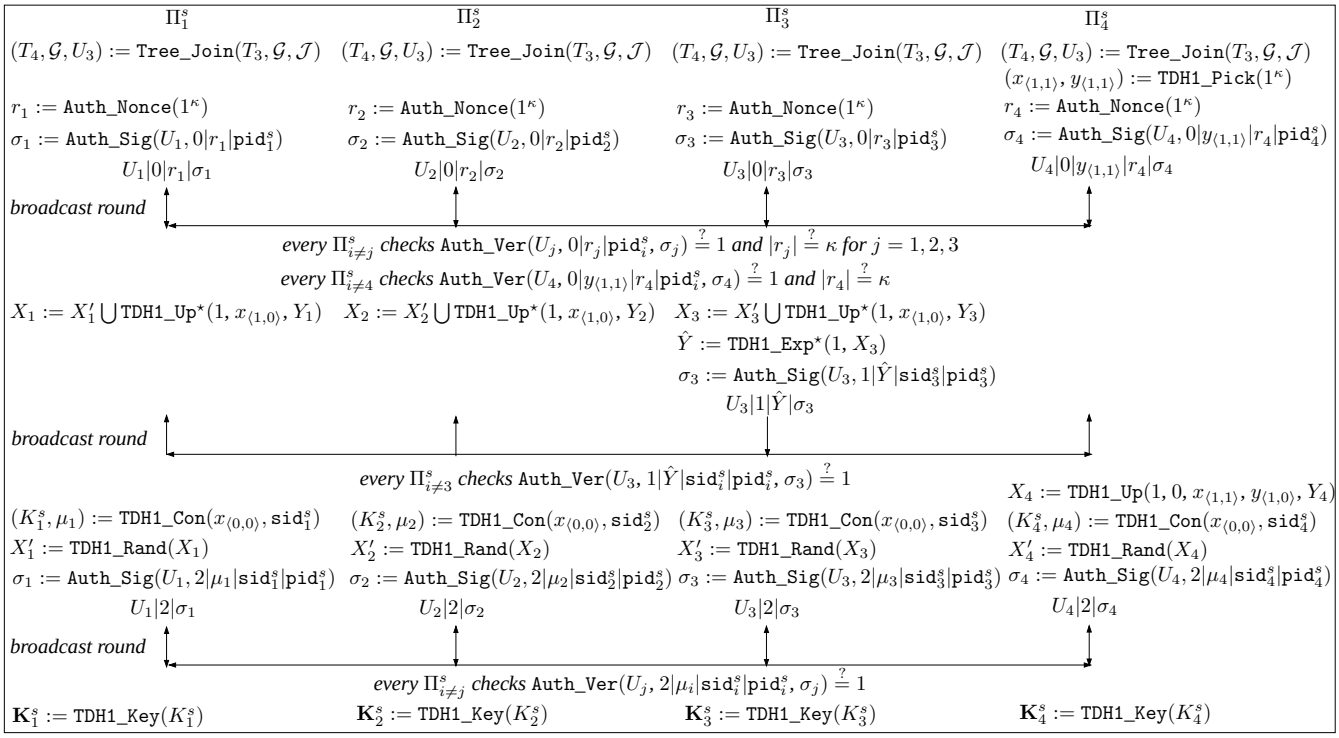

Fig. 10.6. Example: Operation $\texttt{TDH1.Join}^+$ with $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s\}$, T_3 from Figure 10.1 and $\mathcal{J} = \{\Pi_1^{s'}\}$. Note that after the update of T_3 the group is changed to $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s, \Pi_4^s\}$. Public values: $\texttt{sid}_i^s = r_1|r_2|r_3|r_4$, $Y_1 = Y_2 = Y_3 = \{y_{\langle 1,1\rangle}\}$, $Y_4 = \emptyset$, $\hat{Y} = \{y_{\langle 1,0\rangle}\}$, where $y_{\langle 1,1\rangle} = g^{x_{\langle 1,1\rangle}}$, $y_{\langle 1,0\rangle} = g^{x_{\langle 1,0\rangle}}$. Secret values: $X_1 = \{x_{\langle 3,0\rangle}, x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_2 = \{x_{\langle 3,1\rangle}, x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_3 = \{x_{\langle 2,1\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_4 = \{x_{\langle 1,1\rangle}, x_{\langle 0,0\rangle}\}$, where $x_{\langle 0,0\rangle} = g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$, $x_{\langle 1,0\rangle} = g^{x_{\langle 2,0\rangle} x_{\langle 2,1\rangle}}$, $x_{\langle 2,0\rangle} = g^{x_{\langle 3,0\rangle} x_{\langle 3,1\rangle}}$, and $x_{\langle 3,0\rangle}, x_{\langle 3,1\rangle}, x_{\langle 2,1\rangle}, x_{\langle 1,1\rangle} \in_R \mathbb{G}$.

the three oracles Π_1^s, Π_2^s, and Π_3^s. The joining group $\mathcal{J}$ consists of $\Pi_1^{s'}$ (note that each index denotes the order of the oracle in the group due to some sorting function and that there are permanent identities beside the index, thus $\Pi_1^{s'}$ can belong to any $U_i \in \mathcal{U}$). All oracles compute the updated tree structure T_4 (Figure 10.3). The joining oracle $\Pi_1^{s'}$ is assigned to the leaf node $\langle 1,1\rangle$ and its index is changed, i.e., $\Pi_1^{s'}$ is from now on denoted as Π_4^s, whereas labels of leaf nodes of Π_1^s, Π_2^s, Π_3^s are changed to $\langle 3,0\rangle$, $\langle 3,1\rangle$, $\langle 2,1\rangle$, respectively. As noted in the description of $\texttt{TDH1.Setup}$ and w.r.t. the changed labels Π_1^s and Π_2^s still share $x'_{\langle 2,0\rangle}$ and $x'_{\langle 1,0\rangle}$, and all three initial oracles still share $x'_{\langle 1,0\rangle}$. In the following we can omit $'$ in the notations since each $x'_{\langle l,v\rangle}$ can be seen as a replacement of the erased $x_{\langle l,v\rangle}$. The joined oracle Π_4^s chooses own exponent $x_{\langle 1,1\rangle}$ and broadcasts $y_{\langle 1,1\rangle} := g^{x_{\langle 1,1\rangle}}$ together with own randomly chosen nonce r_4, whereas initial oracles Π_1^s, Π_2^s, and Π_3^s choose and broadcast their fresh nonces only. Upon receiving $y_{\langle 1,1\rangle}$: oracle Π_1^s builds X_1 composed of the already known $x_{\langle 3,0\rangle}, x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}$ (part of X_1' from the previous operation), and newly computed $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$; oracle Π_2^s builds X_2 composed of the already known $x_{\langle 3,1\rangle}, x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}$ (part of X_2' from the previous operation), and newly computed $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$; oracle Π_3^s builds X_3 composed of the already known $x_{\langle 2,1\rangle}, x_{\langle 1,0\rangle}$ (part of X_3' from the previous operation), and newly computed $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$. Then, Π_3^s builds $\hat{Y}$ consisting of $y_{\langle 1,0\rangle} := g^{x_{\langle 1,0\rangle}}$ and broadcasts it so that Π_4^s can build X_4 composed of the previously chosen $x_{\langle 1,0\rangle}$ and newly computed $x_{\langle 0,0\rangle} := (y_{\langle 1,0\rangle})^{x_{\langle 1,1\rangle}}$. Then, each Π_i^s, $i \in [1,4]$ computes the intermediate value K and the key confirmation token μ, computes X_i' and erases all private state information except for X_i'. That is every oracle saves $(\mathcal{G}, T_4, X_i')$,

where $\mathcal{G}$ consists of Π_1^s, Π_2^s, Π_3^s, and Π_4^s. Finally, oracles exchange and verify signatures on the computed confirmation tokens and accept with the session group key $\mathbf{K}$.

In the following we give the formal description of $\texttt{TDH1.Join}^+$.

Round 1. Each $\Pi_j^{s'} \in \mathcal{J}$ computes $T_n := \texttt{Tree\_Init}(\mathcal{G})$. Then, each $\Pi_i^s \in \mathcal{G}$ and $\Pi_j^{s'} \in \mathcal{J}$ computes $(T_{n+n_J}, \mathcal{G}, \Pi_\gamma^s) := \texttt{Tree\_Join}^+(T_n, \mathcal{G}, \mathcal{J})$. Then,

- each $\Pi_{i\leq\gamma}^s \in \mathcal{G}$ computes $r_i := \texttt{Auth\_Nonce}(1^\kappa)$, $\sigma_i := \texttt{Auth\_Sig}(U_i, 0|r_i|\texttt{pid}_i^s)$ and broadcasts $U_i|0|r_i|\sigma_i$
- each $\Pi_{i>\gamma}^s \in \mathcal{G}$ computes $r_i := \texttt{Auth\_Nonce}(1^\kappa)$, $(x_{\langle l_i,v_i\rangle}, y_{\langle l_i,v_i\rangle}) := \texttt{TDH1\_Pick}(1^\kappa)$, $\sigma_i := \texttt{Auth\_Sig}(U_i, 0|y_{\langle l_i,v_i\rangle}|r_i|\texttt{pid}_i^s)$ and broadcasts $U_i|0|y_{\langle l_i,v_i\rangle}|r_i|\sigma_i$.

Round 2. When $\Pi_{i\neq j}^s \in \mathcal{G}$ receives $U_j|0|r_j|\sigma_j$ from each $\Pi_{j\leq\gamma}^s \in \mathcal{G}$ resp. $U_j|0|y_{\langle l_j,v_j\rangle}|r_j|\sigma_j$ from each $\Pi_{j>\gamma}^s \in \mathcal{G}$ it verifies $\texttt{Auth\_Ver}(U_j, 0|r_j|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ resp. $\texttt{Auth\_Ver}(U_j, 0|y_{\langle l_j,v_j\rangle}|r_j|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ and checks whether $|r_j| \stackrel{?}{=} \kappa$. If any of these verifications fails then the protocol outputs a failure. Then, each $\Pi_i^s \in \mathcal{G}$ defines $\texttt{sid}_i^s := r_1| \ldots |r_{n+n_J}$, and

- each $\Pi_{i\leq\gamma}^s \in \mathcal{G}$ defines $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_\gamma - 1, \ldots, 1\}$
- each $\Pi_{i>\gamma}^s \in \mathcal{G}$ defines $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_i - 1, \ldots, 1\}$.

Further, each $\Pi_{i\leq\gamma}^s \in \mathcal{G}$ computes $X_i := X_i' \bigcup \texttt{TDH1\_Up}^\star(n_J, x_{\langle n_J,0\rangle}, Y_i)$ with $x_{\langle n_J,0\rangle} \in X_i'$. Additionally, Π_γ^s computes $\hat{Y} := \texttt{TDH1\_Exp}^\star(n_J, X_\gamma)$, $\sigma_\gamma := \texttt{Auth\_Sig}(U_\gamma, 1|\hat{Y}|\texttt{sid}_\gamma^s|\texttt{pid}_\gamma^s)$, and broadcasts $U_\gamma|1|\hat{Y}|\sigma_\gamma$.

Round 3. When $\Pi_i^s \in \mathcal{G}$ receives $U_\gamma|1|\hat{Y}|\sigma_\gamma$ it verifies $\texttt{Auth\_Ver}(U_\gamma, 1|\hat{Y}|\texttt{sid}_i^s|\texttt{pid}_i^s, \sigma_\gamma) \stackrel{?}{=} 1$. If this verification fails then the protocol outputs a failure, else $\Pi_{i>\gamma}^s \in \mathcal{G}$ obtains $X_i := \texttt{TDH1\_Up}(l_i, v_i, x_{\langle l_i,v_i\rangle}, y_{\langle l_i,1-v_i\rangle}, Y_i)$. Further, each $\Pi_i^s \in \mathcal{G}$ computes $(K_i^s, \mu_i) := \texttt{TDH1\_Con}(x_{\langle 0,0\rangle}, \texttt{sid}_i^s)$ with $x_{\langle 0,0\rangle} \in X_i$, $X_i' := \texttt{TDH1\_Rand}(X_i)$, and erases every private information from $\texttt{state}_i^s$ (including all $x_{\langle l,v\rangle}$, and $\rho_0, \ldots, \rho_n$) except for K_i^s and X_i'. Further, it computes $\sigma_i := \texttt{Auth\_Sig}(U_i, 2|\mu_i|\texttt{sid}_i^s|\texttt{pid}_i^s)$, and broadcasts $U_i|2|\sigma_i$.

Key Computation. When $\Pi_i^s \in \mathcal{G}$ receives $U_j|2|\sigma_j$ from each $\Pi_{j\neq i}^s \in \mathcal{G}$ it verifies $\texttt{Auth\_Ver}(U_j, 2|\mu_i|\texttt{sid}_i^s|\texttt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi_i^s \in \mathcal{G}$ computes $\mathbf{K}_i^s := \texttt{TDH1\_Key}(K_i^s)$, erases every private information from $\texttt{state}_i^s$ (including K_i^s) except for X_i', and accepts with $\mathbf{K}_i^s$.

Note that at the end of the $\texttt{TDH1.Join}^+$ operation every participating oracle Π_i^s saves $(\mathcal{G}, T_{n+n_J}, X_i')$.

Operation $\texttt{TDH1.Leave}^+$

Figure 10.7 shows an example of the $\texttt{TDH1.Leave}^+$ operation. The initial group $\mathcal{G}$ consists of four oracles Π_1^s, Π_2^s, Π_3^s, and Π_4^s. The leaving group $\mathcal{L}$ consists of Π_3^s. All remaining oracles compute the updated tree structure T_3 (Figure 10.4) by removing the leaf node $\langle 2,1\rangle$ (leaf node of Π_3^s). The index of the remaining oracle Π_4^s is changed to 3, i.e., Π_4^s is from now on denoted as Π_3^s. Also labels of leaf nodes of Π_1^s resp. Π_2^s, are changed to $\langle 2,0\rangle$ resp. $\langle 2,1\rangle$. Note that from the previous operation Π_1^s and Π_2^s still share $x'_{\langle 1,0\rangle}$ and $x'_{\langle 0,0\rangle}$, and all three oracles still share $x'_{\langle 0,0\rangle}$ (in the following we omit $'$ in the notations since each $x'_{\langle l,v\rangle}$ can be seen as a replacement of the erased $x_{\langle l,v\rangle}$). Each Π_i^s takes own exponent $x_{\langle l_i,v_i\rangle}$ (known from the previous operation as part of X_i'), computes $y_{\langle l_i,v_i\rangle} := g^{x_{\langle l_i,v_i\rangle}}$ and broadcasts it together with a fresh nonce r_i. Upon

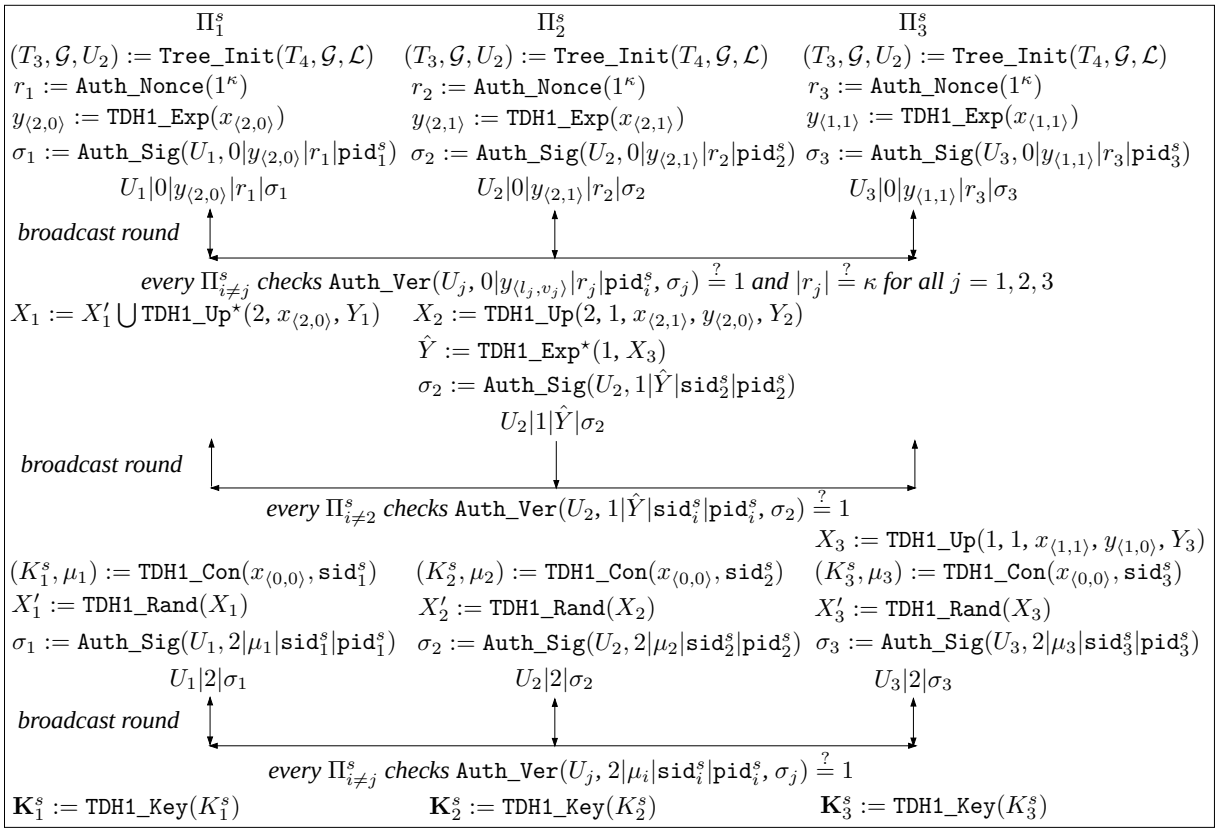

Fig. 10.7. Example: Operation $\texttt{TDH1.Leave}^+$ with $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s, \Pi_4^s\}$, T_4 from Figure 10.3 and $\mathcal{L} = \{\Pi_3^s\}$. Note that after the update of T_4 the group is changed to $\mathcal{G} = \{\Pi_1^s, \Pi_2^s, \Pi_3^s\}$ (former Π_4^s is now Π_3^s). Public values: $\texttt{sid}_i^s = r_1|r_2|r_3$, $Y_1 = Y_2 = \{y_{\langle 1,1\rangle}\}$, $Y_3 = \emptyset$, $\hat{Y} = \{y_{\langle 1,0\rangle}\}$, where $y_{\langle 1,1\rangle} = g^{x_{\langle 1,1\rangle}}$, $y_{\langle 1,0\rangle} = g^{x_{\langle 1,0\rangle}}$. Secret values: $X_1 = \{x_{\langle 2,0\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_2 = \{x_{\langle 2,1\rangle}, x_{\langle 1,0\rangle}, x_{\langle 0,0\rangle}\}$, $X_3 = \{x_{\langle 1,1\rangle}, x_{\langle 0,0\rangle}\}$, where $x_{\langle 0,0\rangle} = g^{x_{\langle 1,0\rangle} x_{\langle 1,1\rangle}}$, $x_{\langle 1,0\rangle} = g^{x_{\langle 2,0\rangle} x_{\langle 2,1\rangle}}$ and $x_{\langle 2,0\rangle}$, $x_{\langle 2,1\rangle}$, $x_{\langle 1,1\rangle} \in_R \mathbb{G}$.

receiving these messages: Π_1^s builds X_1 composed of the already known $x_{\langle 2,0\rangle}$ (part of X_1' from the previous operation), and newly computed $x_{\langle 1,0\rangle} := (y_{\langle 2,0\rangle})^{x_{\langle 2,1\rangle}}$ and $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$; Π_2^s builds X_2 composed of the already known $x_{\langle 2,1\rangle}$ (part of X_2' from the previous operation), and newly computed $x_{\langle 1,0\rangle} := (y_{\langle 2,1\rangle})^{x_{\langle 2,0\rangle}}$ and $x_{\langle 0,0\rangle} := (y_{\langle 1,1\rangle})^{x_{\langle 1,0\rangle}}$. Oracle Π_2^s computes and broadcasts $\hat{Y}$ consisting of $y_{\langle 1,0\rangle} := g^{x_{\langle 1,0\rangle}}$ so that Π_3^s (former Π_4^s) can build X_3 composed of the already known $x_{\langle 1,1\rangle}$ (part of X_4' from the previous operation) and newly computed $x_{\langle 0,0\rangle} := (y_{\langle 1,0\rangle})^{x_{\langle 1,1\rangle}}$. Finally, each Π_i^s, $i \in [1,3]$ computes the intermediate value K and the confirmation token μ, as well as the updated set X_i', and erases all state information except for X_i'. Every oracle saves $(\mathcal{G}, T_3, X_i')$ where $\mathcal{G}$ consists of the remaining participants, i.e., Π_1^s, Π_2^s, Π_3^s (former Π_4^s). Finally, oracles exchange and verify signatures on the computed key confirmation tokens and accept with the updated session group key $\mathbf{K}$.

In the following we give the formal description of $\texttt{TDH1.Leave}^+$.

Round 1. Each oracle $\Pi_i^s \in \mathcal{G} \setminus \mathcal{L}$ computes $(T_{n-n_L}, \mathcal{G}, \Pi_\gamma^s) := \texttt{Tree\_Leave}^+(T_n, \mathcal{G}, \mathcal{L})$ and $r_i := \texttt{Auth\_Nonce}(1^\kappa)$. Then,

- each $\Pi_{i<\gamma-1}^s \in \mathcal{G}$ computes $\sigma_i := \texttt{Auth\_Sig}(U_i, 0|r_i|\texttt{pid}_i^s)$ and broadcasts $U_i|0|r_i|\sigma_i$
- $\Pi_{\gamma-1}^s$ computes $y_{\langle l_\gamma, 1-v_\gamma\rangle} := \texttt{TDH1\_Exp}(x_{\langle l_\gamma, 1-v_\gamma\rangle})$ with $x_{\langle l_\gamma, 1-v_\gamma\rangle} \in X_{\gamma-1}'$, $\sigma_{\gamma-1} := \texttt{Auth\_Sig}(U_{\gamma-1}, 0|y_{\langle l_\gamma, 1-v_\gamma\rangle}|r_{\gamma-1}|\texttt{pid}_{\gamma-1}^s)$ and broadcasts $U_{\gamma-1}|0|y_{\langle l_\gamma, 1-v_\gamma\rangle}|r_{\gamma-1}|\sigma_{\gamma-1}$

– each $\Pi^s_{i\geq\gamma}\in\mathcal{G}$ computes $y_{\langle l_i,v_i\rangle} := \texttt{TDH1\_Exp}(x_{\langle l_i,v_i\rangle})$ with $x_{\langle l_i,v_i\rangle}\in X'_i$, $\sigma_i := \texttt{Auth\_Sig}(U_i, 0|y_{\langle l_i,v_i\rangle}|r_i|\texttt{pid}^s_i)$ and broadcasts $U_i|0|y_{\langle l_i,v_i\rangle}|r_i|\sigma_i$.

Round 2. When $\Pi^s_{i\neq j}\in\mathcal{G}$ receives $U_j|0|r_j|\sigma_j$ from each $\Pi^s_{j<\gamma-1}\in\mathcal{G}$ resp. $U_{\gamma-1}|0|y_{\langle l_\gamma,1-v_\gamma\rangle}|r_{\gamma-1}|\sigma_{\gamma-1}$ from $\Pi^s_{\gamma-1}\in\mathcal{G}$ resp. $U_j|0|y_{\langle l_j,v_j\rangle}|r_j|\sigma_j$ from each $\Pi^s_{j\geq\gamma}\in\mathcal{G}$ it verifies $\texttt{Auth\_Ver}(U_j, 0|r_j|\texttt{pid}^s_i, \sigma_j) \stackrel{?}{=} 1$ resp. $\texttt{Auth\_Ver}(U_{\gamma-1}, 0|y_{\langle l_\gamma,1-v_\gamma\rangle}|r_{\gamma-1}|\texttt{pid}^s_i, \sigma_{\gamma-1}) \stackrel{?}{=} 1$ resp. $\texttt{Auth\_Ver}(U_j, 0|y_{\langle l_j,v_j\rangle}|r_j|\texttt{pid}^s_i, \sigma_j) \stackrel{?}{=} 1$, and checks whether $|r_j| \stackrel{?}{=} \kappa$. If any of these verifications fails then the protocol outputs a failure. Then, each $\Pi^s_i\in\mathcal{G}$ defines $\texttt{sid}^s_i := r_1|\ldots|r_{n-n_L}$, and
– each $\Pi^s_{i<\gamma}\in\mathcal{G}$ defines $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_\gamma,\ldots,1\}$
– each $\Pi^s_{i\geq\gamma}\in\mathcal{G}$ defines $Y_i := \{y_{\langle l_j,1\rangle} \mid \forall l_j = l_i-1,\ldots,1\}$.
Further,
– each $\Pi^s_{i<\gamma}\in\mathcal{G}$ computes $X_i := X'_i \bigcup \texttt{TDH1\_Up}^\star(l_\gamma, x_{\langle l_\gamma,0\rangle}, Y_i)$ with $x_{\langle l_\gamma,0\rangle}\in X'_i$.
– Π^s_γ computes $X_\gamma := \texttt{TDH1\_Up}(l_\gamma, v_\gamma, x_{\langle l_\gamma,v_\gamma\rangle}, y_{\langle l_\gamma,1-v_\gamma\rangle}, Y_\gamma)$ with $x_{\langle l_\gamma,v_\gamma\rangle}\in X'_\gamma$.
Additionally, Π^s_γ computes $\hat{Y} := \texttt{TDH1\_Exp}^\star(l_\gamma-1, X_\gamma)$, $\sigma_\gamma := \texttt{Auth\_Sig}(U_\gamma, 1|\hat{Y}|\texttt{sid}^s_\gamma|\texttt{pid}^s_\gamma)$, and broadcasts $U_\gamma|1|\hat{Y}|\sigma_\gamma$.

Round 3. When $\Pi^s_{i\neq\gamma}\in\mathcal{G}$ receives $U_\gamma|1|\hat{Y}|\sigma_\gamma$ it verifies $\texttt{Auth\_Ver}(U_\gamma, 1|\hat{Y}|\texttt{sid}^s_i|\texttt{pid}^s_i, \sigma_\gamma) \stackrel{?}{=} 1$. If this verification fails then the protocol outputs a failure. Else each $\Pi^s_{i>\gamma}\in\mathcal{G}$ obtains $X_i := \texttt{TDH1\_Up}(l_i, v_i, x_{\langle l_i,v_i\rangle}, y_{\langle l_i,1-v_i\rangle}, Y_i)$. Further, each $U_i\in\mathcal{G}$ computes $(K^s_i,\mu_i) := \texttt{TDH1\_Con}(x_{\langle 0,0\rangle}, \texttt{sid}^s_i)$ with $x_{\langle 0,0\rangle}\in X_i$, $X'_i := \texttt{TDH1\_Rand}(X_i)$, and erases every private information from $\texttt{state}^s_i$ (including all $x_{\langle l,v\rangle}$, and $\rho_0,\ldots,\rho_n$) except for K^s_i and X'_i. Further, it computes $\sigma_i := \texttt{Auth\_Sig}(U_i, 2|\mu_i|\texttt{sid}^s_i|\texttt{pid}^s_i)$, and broadcasts $U_i|2|\sigma_i$.

Key Computation. When $\Pi^s_i\in\mathcal{G}$ receives $U_j|2|\sigma_j$ from each $\Pi^s_{j\neq i}\in\mathcal{G}$ it verifies $\texttt{Auth\_Ver}(U_j, 2|\mu_i|\texttt{sid}^s_i|\texttt{pid}^s_i, \sigma_j) \stackrel{?}{=} 1$. If any of these verifications fails then the protocol outputs a failure. Else each $\Pi^s_i\in\mathcal{G}$ computes $\mathbf{K}^s_i := \texttt{TDH1\_Key}(K^s_i)$, erases every private information from $\texttt{state}^s_i$ (including K^s_i) except for X'_i, and accepts with $\mathbf{K}^s_i$.

Note that at the end of the $\texttt{TDH1.Leave}^+$ operation every participating oracle Π^s_i saves $(\mathcal{G}, T_{n-n_L}, X'_i)$.

10.4.4 Security Analysis of Dynamic TDH1

Similar to the discussion in Sections 9.2.1 and 10.4.4 we assume that at any time during the execution of dynamic TDH1 the internal state information $\texttt{state}^s_U$ is independent of LL_U.

The following theorem shows that the dynamic TDH1 protocol is AKE-secure and provides forward secrecy in the strong-corruption model. Note that the protocol does not provide backward secrecy because knowledge of X'_i would allow the adversary to compute the session group key established in any subsequent session. Therefore, backward secrecy (in the weak-corruption model) is only achieved by the static TDH1 protocol where each private exponent $x_{\langle l_i,v_i\rangle}$, $\langle l_i,v_i\rangle\in LN_{T_n}$ is freshly generated for each new session. Since in dynamic TDH1 oracles save additional information, in particular the set X'_i, in order to process dynamic changes more efficiently the security analysis must ensure that this information does not provide the adversary against AKE-security with some non-negligible advantage. Intuitively, the computation process of the values in X'_i becomes an additional important factor.

Theorem 10.11 (AKE-Security of Dynamic TDH1). *If Σ is EUF-CMA, F is pseudo-random, and the DDH and SEDDH assumptions hold in $\mathbb{G}$ then TDH1 is AGKE-sfs, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{sfs},\mathtt{scm},\mathtt{TDH1}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + (2N-2)q_s\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + (4N-2)q_s\mathsf{Adv}^{\mathtt{SEDDH}}_{\mathbb{G}}(\kappa) + (2N+6)q_s\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. We define a sequence of games $\mathbf{G}_i$, $i = 0, \ldots, 8$ and corresponding events $\mathsf{Win}^{\mathsf{ake}}_i$ as the events that the output bit b' of $\mathbf{G}_i$ is identical to the randomly chosen bit b in the game $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sfs},\mathtt{scm},\mathtt{TDH1}}(\kappa)$.

Game $\mathbf{G}_0$. This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sfs},\mathtt{scm},\mathtt{TDH1}}(\kappa)$ played between a simulator Δ and an active adversary $\mathcal{A}$. Assume that the $Test$ query is asked to an α-fresh oracle Π^s_i. Keep in mind that on the test query the adversary receives either a random string or a session group key $\mathbf{K}^s_i$.

Game $\mathbf{G}_1$. This game is identical to Game $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a $Send$ query on some $U_i|\mathtt{seqn}|m|\sigma$ (or $U_i|\mathtt{seqn}|\sigma$) with the sequence number $\mathtt{seqn} \in \{0,1,2\}$ such that σ is a valid signature on m (or on μ_i) that has not been previously output by an oracle Π^s_i before querying $Corrupt(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to the proof of Theorem 10.6 and Equation 10.6 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_1] - \Pr[\mathsf{Win}^{\mathsf{ake}}_0]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa). \tag{10.22}$$

Game $\mathbf{G}_2$. This game is identical to Game $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if a nonce r_i is used by any uncorrupted user's oracle Π^s_i in two different sessions. According to the proof of Theorem 10.6 and Equation 10.7 we get

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_2] - \Pr[\mathsf{Win}^{\mathsf{ake}}_1]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}}. \tag{10.23}$$

Note that this game excludes replay attacks in operations of TDH1.

Game $\mathbf{G}_3$. This game is identical to Game $\mathbf{G}_2$ except that the following rule is added: Δ chooses $q^*_{\mathsf{s}} \in [1, q_{\mathsf{s}}]$ as a guess for the number of sessions (operations of TDH1) invoked before $\mathcal{A}$ asks the query $Test$. If this query does not occur in the q^*_{s}-th session then the simulation fails and bit b' is set at random. Similar to Equation 10.8 we get

$$\Pr[\mathsf{Win}^{\mathsf{ake}}_2] = q_{\mathsf{s}}\left(\Pr[\mathsf{Win}^{\mathsf{ake}}_3] - \frac{1}{2}\right) + \frac{1}{2}. \tag{10.24}$$

Game $\mathbf{G}_4$. This game is identical to Game $\mathbf{G}_3$ except that in the q^*_{s}-th session we add the following rules. Let n be a number of participants in the q^*_{s}-th session. Let l^* be $n-2$ if the operation invoked for the q^*_{s}-th session is TDH1.Setup and $l_\gamma - 1$ otherwise. For each node $\langle l, 0\rangle \in T_n \setminus LN_{T_n}$ with $l = l^*$ down to 0 the simulator chooses a random value $r_{\langle l,0\rangle} \in_R \mathbb{G}$ and computes $x_{\langle l,0\rangle} := g^{r_{\langle l,0\rangle}}$. The idea for the simulation is to replace all secret values in the tree T_n, whose corresponding public values are computed by the function TDH1_Exp* and broadcasted in the q^*_{s}-th session, with truly random values. Note that since no $Corrupt$, $RevealState$, and $RevealKey$ queries occur in sfs-fresh sessions (according to Remark 8.10), and thus in the

q_{s}^*-th session, all the above mentioned secret values $x_{\langle l,0\rangle}$ that are erased at the end of each operation execution of TDH1 remain unknown to $\mathcal{A}$.

In order to estimate the difference $|\Pr[\mathsf{Win}_4^{\mathsf{ake}}] - \Pr[\mathsf{Win}_3^{\mathsf{ake}}]|$ we apply the "hybrid technique". We consider a sequence of auxiliary games $\mathbf{G}_4^l$ for all $l = l^* + 1$ down to 0. Each $\mathbf{G}_4^l$ is identical to Game $\mathbf{G}_3$ except that for every node $\langle j,0\rangle \in T_n \setminus LN_{T_n}$ with $j \geq l$ we have $x_{\langle j,0\rangle} := g^{r_{\langle j,0\rangle}}$ where $r_{\langle j,0\rangle} \in_R \mathbb{G}$ (note that in Game $\mathbf{G}_3$ we have $x_{\langle j,0\rangle} = g^{x_{\langle j+1,0\rangle} x_{\langle j+1,1\rangle}}$).

Let $\mathsf{Win}_{4,l}^{\mathsf{ake}}$ denote the event that $\mathsf{Win}_3^{\mathsf{ake}}$ (or equivalently $\mathsf{Win}_4^{\mathsf{ake}}$) occurs in $\mathbf{G}_4^l$. The only difference between the two neighboring auxiliary games $\mathbf{G}_4^l$ and $\mathbf{G}_4^{l-1}$ is how Δ constructs the value $x_{\langle l-1,0\rangle}$: in $\mathbf{G}_4^l$ the simulator constructs $x_{\langle l-1,0\rangle} = g^{x_{\langle l,0\rangle} x_{\langle l,1\rangle}}$, whereas in $\mathbf{G}_4^{l-1}$ the simulator chooses $r_{\langle l-1,0\rangle} \in_R \mathbb{G}$ and constructs $x_{\langle l-1,0\rangle} = g^{r_{\langle l-1,0\rangle}}$. Obviously, in $\mathbf{G}_4^l$ for $\langle l-1,0\rangle \in T_n \setminus LN_{T_n}$ we have a tuple $(g, g^{x_{\langle l,0\rangle}}, g^{x_{\langle l,1\rangle}}, g^{x_{\langle l,0\rangle} x_{\langle l,1\rangle}})$ taken from the real DDH distribution, i.e., $\mathtt{DDH}^\star$. To the contrary, in $\mathbf{G}_4^{l-1}$ for $\langle l-1,0\rangle \in T_n \setminus LN_{T_n}$ we have a tuple $(g, g^{x_{\langle l,0\rangle}}, g^{x_{\langle l,1\rangle}}, g^{r_{\langle l-1,0\rangle}})$ taken from the random DDH distribution, i.e., $\mathtt{DDH}^{\$}$. Thus, we get

$$|\Pr[\mathsf{Win}_{4,l-1}^{\mathsf{ake}}] - \Pr[\mathsf{Win}_{4,l}^{\mathsf{ake}}]| \leq \mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa)$$

It is easy to see that $\mathbf{G}_4^{l^*+1} = \mathbf{G}_3$ and $\mathbf{G}_4^0 = \mathbf{G}_4$. Since $l^* \leq n-2 \leq N-2$ we get

$$|\Pr[\mathsf{Win}_4^{\mathsf{ake}}] - \Pr[\mathsf{Win}_3^{\mathsf{ake}}]| \leq (N-1)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa). \tag{10.25}$$

Note also that in this game $x_{\langle 0,0\rangle} = g^{r_{\langle 0,0\rangle}}$ is a uniformly distributed value in $\mathbb{G}$ (or equivalently in $\mathbb{Z}_q$), and that consequently each $x'_{\langle l,v\rangle}$ is computed as $g^{r^2_{\langle l,v\rangle}}$.

Game $\mathbf{G}_5$. This game is identical to Game $\mathbf{G}_4$ except that in the q_{s}^*-th session a random bit string is sampled from $\{0,1\}^\kappa$ and used instead of the truncated value $x_{\langle 0,0\rangle\lfloor\kappa}$ inside the function TDH1_Con. Since in Game $\mathbf{G}_4$ the entire value $x_{\langle 0,0\rangle}$ is already random in $\mathbb{G}$ (and according to Corollary 10.2 and Remark 10.3 also uniform in $\mathbb{Z}_q$) both games are identical. Thus, replacing $x_{\langle 0,0\rangle}$ with a randomly sampled bit string can be seen as a "bridging step" (using the terminology of [168]) so that

$$\Pr[\mathsf{Win}_5^{\mathsf{ake}}] = \Pr[\mathsf{Win}_4^{\mathsf{ake}}]. \tag{10.26}$$

Game $\mathbf{G}_6$. This game is identical to Game $\mathbf{G}_5$ except that in the q_{s}^*-th session the simulator computes each $x'_{\langle l,v\rangle}$ (part of X_i) as $g^{r'_{\langle l,v\rangle}}$ with some randomly chosen $r'_{\langle l,v\rangle} \in_R \mathbb{G}$. Observe that in this game for every $\langle l,v\rangle \in T_n$ there is a tuple $(g, g^{x_{\langle l,v\rangle}}, g^{r'_{\langle l,v\rangle}})$ taken from the random SEDDH distribution (i.e., $\mathtt{SEDDH}^{\$}$).[5] Note that $x_{\langle l,v\rangle} \in_R \mathbb{G}$ for each $\langle l,v\rangle \in T_n$ as described in Game $\mathbf{G}_4$.

On the other hand, in Game $\mathbf{G}_5$ each $x'_{\langle l,v\rangle}$ is computed as $g^{x^2_{\langle l,v\rangle}}$. Thus, in Game $\mathbf{G}_5$ for every $\langle l,v\rangle \in T_n$ we have a tuple $(g, g^{x_{\langle l,v\rangle}}, g^{x^2_{\langle l,v\rangle}})$ taken from the real SEDDH distribution (i.e., $\mathtt{SEDDH}^\star$).

According to Definition 5.14, and since $|T_n| = 2n-1$ and $n \leq N$ we have

$$|\Pr[\mathsf{Win}_6^{\mathsf{ake}}] - \Pr[\mathsf{Win}_5^{\mathsf{ake}}]| \leq (2N-1)\mathsf{Adv}_{\mathbb{G}}^{\mathtt{SEDDH}}(\kappa). \tag{10.27}$$

Game $\mathbf{G}_7$. This game is identical to Game $\mathbf{G}_6$ except that in the q_{s}^*-th session each ρ_i, $i = 0,\ldots,n$ is replaced by a random value sampled from $\{0,1\}^\kappa$. Notice, this implies that K is uniformly distributed in this session.

In order to estimate the difference to the previous game we apply the "hybrid technique" and define auxiliary games $\mathbf{G}'_{7,l}$, $l = 0,\ldots,n+1$ such that $\mathbf{G}'_{7,0} = \mathbf{G}_6$ and $\mathbf{G}'_{7,n+1} = \mathbf{G}_7$. That is, in

[5] Note that $g^{x_{\langle l,v\rangle}}$ may become public during the q_{s}^*-th session and that $g^{r'_{\langle l,v\rangle}}$ can be obtained by $\mathcal{A}$ during the (q_{s}^*+1)-th session of TDH1 via a $RevealState$ query.

the q_{s}^*-th session in each $\mathbf{G}'_{7,l}$ the intermediate values ρ_i, $i \leq l$, are computed as specified in the protocol (as output of the pseudo-random function f) whereas in $\mathbf{G}'_{7,l+1}$ these values are chosen at random from $\{0,1\}^\kappa$. Note that each replacement of ρ_i, $i = 0, \ldots, n-1$ by a random bit string implies uniform distribution of the PRF key $\rho_i \oplus \pi(r_{i+1})$ used in the computation of ρ_{i+1}, and that $x_{\langle 0,0 \rangle \lfloor \kappa}$ used to compute ρ_0 is already uniform according to Game $\mathbf{G}_5$. Since $n \leq N$ we get

$$|\Pr[\mathsf{Win}_7^{\mathsf{ake}}] - \Pr[\mathsf{Win}_6^{\mathsf{ake}}]| \leq (N+1)\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{10.28}$$

Game $\mathbf{G}_8$. This game is identical to Game $\mathbf{G}_7$ except that in the q_{s}^*-th session the confirmation token μ and the session group key $\mathbf{K}$ are replaced by two different random values sampled from $\{0,1\}^\kappa$. This can be done since the intermediate value K is uniformly distributed according to Game $\mathbf{G}_7$. Obviously,

$$|\Pr[\mathsf{Win}_8^{\mathsf{ake}}] - \Pr[\mathsf{Win}_7^{\mathsf{ake}}]| \leq 2\mathsf{Adv}_F^{\mathsf{prf}}(\kappa). \tag{10.29}$$

Since $\mathbf{K}$ is uniformly distributed in the q_{s}^*-th session $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}_8^{\mathsf{ake}}] = \frac{1}{2}. \tag{10.30}$$

Considering Equations 10.22 to 10.30 we get:

$$\begin{aligned}
\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\mathtt{sfs},\mathtt{scm},\mathtt{TDH1}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}_2^{\mathsf{ake}}] \\
&= N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2} \\
&\leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + (N-1)q_{\mathsf{s}}\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) + \\
&\quad (2N-1)q_{\mathsf{s}}\mathsf{Adv}_{\mathbb{G}}^{\mathtt{SEDDH}}(\kappa) + (N+3)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{ake}}_{\mathtt{sfs},\mathtt{scm},\mathtt{TDH1}}(\kappa) \leq 2N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + (2N-2)q_{\mathsf{s}}\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) + \\
(4N-2)q_{\mathsf{s}}\mathsf{Adv}_{\mathbb{G}}^{\mathtt{SEDDH}}(\kappa) + (2N+6)q_{\mathsf{s}}\mathsf{Adv}_F^{\mathsf{prf}}(\kappa).
\end{aligned}$$

□

Every dynamic operation of the TDH1 protocol utilizes the same key confirmation and derivation mechanism as its static counterpart. Therefore, the following two theorems concerning MA-security and n-contributiveness of the dynamic TDH1 protocol hold with the proofs of Theorems 10.7 and 10.8, respectively. Note that random nonces are freshly generated for every dynamic operation of TDH1.

Theorem 10.12 (MA-Security of Dynamic TDH1). *If Σ is EUF-CMA and F is collision-resistant then dynamic* TDH1 *is* MAGKE*, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{TDH1}}(\kappa) \leq N\mathsf{Succ}_{\Sigma}^{\mathtt{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa),$$

where q_{s} is the total number of executed protocol sessions.

Proof. See proof of Theorem 10.7. □

Theorem 10.13 (n-Contributiveness of Dynamic TDH1). *If F is collision-resistant pseudo-random and π is one-way then dynamic* TDH1 *is a n-*CGKE *protocol, and*

$$\mathsf{Succ}^{\mathsf{con}-n}_{\mathtt{TDH1}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + q_s}{2^\kappa} + (N+1)q_s\mathsf{Succ}^{\mathsf{coll}}_F(\kappa) + q_s\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + Nq_s\mathsf{Succ}^{\mathsf{ow}}_\pi(\kappa),$$

where q_s is the total number of executed protocol sessions.

Proof. See proof of Theorem 10.8. □

10.5 Summary

Considering results of our analysis of currently known provably secure group key exchange protocols in Section 7.4 the dynamic version of our TDH1 protocol is the first constant-round dynamic group key exchange protocol provably secure under standard cryptographic assumptions with respect to strong corruptions and attacks of malicious participants against key control and contributiveness. Note that each operation of TDH1 requires three communication rounds. The TDH1 protocol provides additionally key confirmation and mutual authentication in the presence of malicious participants. It is worth being mentioned that TDH1 is the first provably secure protocol based on the linear binary tree structure that plays a significant role in the communication efficiency of the protocol. Additionally, we note that shown equivalence of the TDDH and DDH assumptions is of independent interest, i.e., the TDDH assumption can be used in other cryptographic constructions whose security is supposed to be shown under standard assumptions.

Conclusions and Further Research Directions

In this chapter we draw some general conclusions and identify some directions for further research on "provable security" of group key exchange protocols.

The proposed model in Chapter 8 is one of the strongest currently available (game-based) computational security models for group key exchange protocols designed for reductionist security proofs with extended security definitions concerning attacks of malicious participants and strong corruptions. Security-enhancing compilers proposed in Chapter 9 can be used to achieve this higher degree of security for group key exchange protocols in a general way. The proposed protocol TDH1 in Chapter 10, especially its dynamic version, is the first provably secure group key exchange which satisfies these strong security requirements under standard cryptographic assumptions. The TDDH assumption from Definition 10.4, whose polynomial equivalence to the DDH assumption has been proven in Theorem 10.5, is of independent interest and can be used in other cryptographic schemes whose security is supposed to be proven in the standard model. Finally, our extension of the popular "game of sequence" technique from [168] by the additional type of transitions based on "condition events" introduced in Section 5.6.1 can also be used in reductionist security proofs of other cryptographic constructions.

In the following we specify a number of interesting research topics concerning provable security of group key exchange protocols resulting from the observations put in light along the lines of this thesis.

As noted in Chapter 8 our security model (similar to the other state-of-the-art models), and consequently our proposed constructions do not deal with denial-of-service attacks and fault-tolerance. Our security definitions aim to recognize attacks occurring during the protocol execution and prevent honest protocol participants from accepting "biased" session group keys. Nevertheless, there are no requirements stating that despite of these attacks honest protocol participants must still be able to complete the protocol execution and compute identical session group keys. In fact, in all our constructions honest participants abort the computation process if some unexpected failures occur. Note also that the adversary has complete control over the communication channel. Therefore, if the adversary refuses to deliver messages according to the protocol specification or if some protocol participants crash then the computation process is blocked. These considerations, in turn, give an interesting open problem on the specification of security definitions and design of concrete group key exchange protocols considering fault-tolerance and denial-of-service attacks.

An interesting pioneer work in this area was done by Cachin and Strobl [55], who proposed the first (simulatability-based) security model and a concrete static asynchronous group key exchange protocol while considering the issues of fault-tolerance. In order to compute the session group key participants perform operations that can be seen as the abstraction of the Burmester-Desmedt protocol [50, 51] with an additional stage where any *asynchronous distributed consensus protocol*, e.g., [53, 65], must be executed prior to acceptance. The protocol ensures that participants complete the execution and accept with identical session group keys even if at most t of them crash, whereby t is a bound specific to the consensus protocol (note that if no trusted

parties are allowed then $(n-1)/3$ is the optimal bound for tolerated crashes [39, 53, 65]). Additionally, [55] describes a fault-tolerant protocol version which provides strong forward secrecy. The authors show that for this case the optimal upper bound for the number of the revealed internal states in later sessions is strictly less than $n - 2t$. This paper opens several directions for further research. First, [55] does not deal with the attacks of malicious protocol participants aiming to control the value of the session group key. Therefore, it is interesting to investigate the consequences of these attacks and their prevention in the context of fault-tolerant group key exchange protocols. Another interesting open question is the identification of malicious participants during the protocol execution. This can be also considered as a research goal in the context of group key exchange protocols without fault-tolerance. Second, the security model and the protocol proposed in [55] deal with the static case so that possible extensions towards the more challenging dynamic case represent another direction for further research. Fourth, the security model specified in [55] uses the simulatability approach (cf. Section 3.3.2); thus, designing an adequate game-based security model can also be considered as an interesting task. Finally, the ideas from [55] concerning the utilization of the asynchronous distributed consensus protocols can be possibly deepened in order to design a generic solution (compiler) for fault-tolerance in group key exchange protocols, and even to combine it with the compilers proposed in this thesis.

References

[1] OpenNAP: Open Source Napster Server. World Wide Web. `http://opennap.sourceforge.net/`. 24

[2] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *ACM Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997. 44

[3] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Jorunal of Cryptology*, 15(2):103–127, 2002. 44

[4] M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *Topics in Cryptology – CT-RSA'01*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158, April 2001. 99

[5] M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-Based Group Key Exchange in a Constant Number of Rounds. In *Proceedings of the 9th International Workshop on Theory and Practice in Public Key Cryptography (PKC'06)*, volume 3958 of *Lecture Notes in Computer Science*, pages 427–442. Springer, April 2006. 13, 42, 44, 100, 101, 103, 109, 110

[6] R. Ahlswede and I. Csiszár. Common Randomness in Information Theory and Cryptography - i: Secret Sharing. *IEEE Transactions on Information Theory*, 39(4):1121–1132, 1993. 40

[7] S. S. Al-Riyami and K. G. Paterson. Tripartite Authenticated Key Agreement Protocols from Pairings. In *Proceedings of the IMA Conference on Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 332–359. Springer, 2003. 88

[8] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the Performance of Group Key Agreement Protocols. *ACM Transactions on Information and System Security*, 7(3):457–488, 2004. 87

[9] J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In *Advances in Cryptology – EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002. 59

[10] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computer Surveys*, 36(4):335–371, 2004. 24

[11] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005. 44

[12] N. Asokan and P. Ginzboorg. Key-Agreement in Ad-hoc Networks. *Computer Communications*, 23(17):1627–1637, 2000. 96

[13] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proceedings of the 5th ACM conference on Computer and Communications Security (CCS'98)*, pages 17–26. ACM Press, 1998. 13, 37, 65, 66, 93, 94, 124

[14] G. Ateniese, M. Steiner, and G. Tsudik. New Multi-Party Authentication Services and Key Agreement Protocols. *IEEE Journal of Selected Areas in Communications*, 18(4):628–639, 2000. 94

[15] M. Backes, B. Pfitzmann, and M. Waidner. A Composable Cryptographic Library with Nested Operations. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 220–230. ACM Press, 2003. 45

[16] R. Barua, R. Dutta, and P. Sarkar. Extending Joux's Protocol to Multi Party Key Agreement. In *Progress in Cryptology – INDOCRYPT'03*, volume 2904 of *Lecture Notes in Computer Science*, pages 205–217. Springer, December 2003. 91, 99, 107

[17] K. Becker and U. Wille. Communication Complexity of Group Key Distribution. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS'98)*, pages 1–6. ACM Press, 1998. 13, 95, 96

[18] M. Bellare. Practice-Oriented Provable-Security. In *Proceedings of the First International Workshop on Information Security (ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1998. 41

[19] M. Bellare, A. Boldyreva, and A. Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2004. 44

[20] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC'98)*, pages 419–428. ACM Press, 1998. 43, 69

[21] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology–EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, May 2000. 70, 112

[22] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology–CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993. 65, 67, 76, 120

[23] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS'93)*, pages 62–73. ACM Press, 1993. 36, 43, 55, 66, 71

[24] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM Press, 1995. 68

[25] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. A. Smolin. Experimental Quantum Cryptography. *Journal of Cryptology*, 5(1):3–28, 1992. 40

[26] C. H. Bennett, G. Brassard, C. Crépeau, and U. Maurer. Generalized privacy amplification. *IEEE Transaction on Information Theory*, 41(6):1915–1923, 1995. 40

[27] J. Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. Cryptology ePrint Archive, Report 2005/210, 2005. `http://eprint.iacr.org/2005/210.pdf`. 44, 71

[28] I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography*. Cambridge University Press, April 2005. ISBN:0-521-60415-X. 88, 91, 93, 98

[29] S. Blake-Wilson, D. Johnson, and A. Menezes. Key Agreement Protocols and Their Security Analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer, December 1997. 65

[30] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceeding of 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society, 2001. 44

[31] B. Blanchet and D. Pointcheval. Automated Security Proofs with Sequences of Games. In *Advances in Cryptology – CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006. 42, 45

[32] J.-M. Bohli, M. I. G. Vasco, and R. Steinwandt. Secure Group Key Establishment Revisited. Cryptology ePrint Archive, Report 2005/395, 2005. `http://eprint.iacr.org/`. 48, 83, 85, 100, 102, 111, 144

[33] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC'03)*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003. 107

[34] D. Boneh. The Decision Diffie-Hellman Problem. In *ANTS-III: Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 48–63. Springer, 1998. 104

[35] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In *Advances in Cryptology–EUROCRYPT'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004. Available at `http://www.cs.stanford.edu/~xb/eurocrypt04b/`. 91

[36] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003. 88, 91

[37] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Advances in Cryptololgy – ASIACRYPT'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, December 2001. 58, 107

[38] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003. ISBN:3-540-43107-1. 65, 66, 87, 123

[39] G. Bracha. An Asynchronous [(n-1)/3]-resilient Consensus Protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing (PODC'84)*, pages 154–162. ACM Press, 1984. 210

[40] E. Bresson and D. Catalano. Constant Round Authenticated Group Key Agreement via Distributed Computation. In *Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography (PKC'04)*, volume 2947 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2004. 13, 103, 104, 109, 110

[41] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case. In *Advances in Cryptology – ASIACRYPT'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 290–390. Springer, December 2001. 13, 43, 76, 78, 85, 105, 106, 109, 110

[42] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. In *Advances in Cryptology – EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer, Mai 2002. 13, 37, 42, 48, 76, 79, 102, 105, 106, 109, 111, 112, 120, 128

[43] E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In *Advances in Cryptology – ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 497–514. Springer, December 2002. 13, 42, 76, 80, 81, 84, 85, 103, 106, 107, 109, 110

[44] E. Bresson, O. Chevassut, and D. Pointcheval. The Group Diffie-Hellman Key Problems. In *Proceedings of the Workshop on Selected Areas in Cryptography (SAC'02)*, volume 2595 of *Lecture Notes in Computer Science*, pages 325–338. Springer, August 2002. 104, 109

[45] E. Bresson, O. Chevassut, and D. Pointcheval. A Security Solution for IEEE 802.11's Ad-hoc Mode: Password-Authentication and Group Diffie-Hellman Key Exchange. *International Journal on Wireless and Mobile Computing*, 2(1):4–13, 2007. 13, 106, 109, 110

[46] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS'01)*, pages 255–264. ACM Press, 2001. 13, 42, 43, 48, 74, 75, 76, 77, 78, 85, 104, 105, 106, 109, 110, 120

[47] B. Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In *Proceedings of the First International Workshop on Networked Group Communication (NGC'99)*, volume 1736 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 1999. 35

[48] M. Burmester. On the Risk of Opening Distributed Keys. In *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 308–317. Springer, August 1994. 36, 64, 65

[49] M. Burmester and Y. Desmedt. Towards Practical Proven Secure Authenticated Key Distribution. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS'93)*, pages 228–231. ACM Press, 1993. 36, 65

[50] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer, May 1994. 13, 34, 35, 36, 64, 65, 88, 90, 99, 209

[51] M. Burmester and Y. Desmedt. A Secure and Scalable Group Key Exchange System. *Information Processing Letters*, 94(3):137–143, 2005. 90, 99, 209

[52] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical Report 39, DEC Systems Research Center, 1989. `http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-039.html`. 44

[53] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, pages 123–132. ACM Press, 2000. 209, 210

[54] C. Cachin and U. Maurer. Unconditional Security Against Memory-Bounded Adversaries. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 292–306. Springer-Verlag, 1997. 40

[55] C. Cachin and R. Strobl. Asynchronous Group Key Exchange with Failures. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 357–366. ACM Press, 2004. 209, 210

[56] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000. 42

[57] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE CS, 2001. 43, 45, 82

[58] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of IEEE INFOCOM '99*, pages 708–716. IEEE Computer Society, 1999. 28, 29, 30

[59] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM*, 51(4):557–594, 2004. 44

[60] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally Composable Password-Based Key Exchange. In *Advances in Cryptology – EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, 2005. 43

[61] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In *3rd Theory of Cryptography Conference (TCC'06)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006. 45

[62] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001. 43, 71, 116

[63] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *Advances in Cryptology - CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2002. 43

[64] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology – EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002. 43

[65] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC'93)*, pages 42–51. ACM Press, 1993. 209, 210

[66] D. Chaum, J.-H. Evertse, and J. van de Graaf. An Improved Proof for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In *Advances in Cryptology - EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 1987. 91

[67] Z. Chen. Security Analysis on Nalla-Reddy's ID-Based Tripartite Authenticated Key Agreement Protocols. Cryptology ePrint Archive, Report 2003/103, 2003. `http://eprint.iacr.org/`. 88

[68] Z. Cheng, L. Vasiu, and R. Comley. Pairing-Based One-Round Tripartite Key Agreement Protocols. Cryptology ePrint Archive, Report 2004/079, 2004. `http://eprint.iacr.org/`. 88

[69] O. Chevassut, P.-A. Fouque, P. Gaudry, and D. Pointcheval. Key Derivation and Randomness Extraction. Cryptology ePrint Archive, Report 2005/061, 2005. Available at `http://eprint.iacr.org/2005/061.pdf`. 179

[70] O. Chevassut, P.-A. Fouque, P. Gaudry, and D. Pointcheval. The Twist-AUgmented Technique for Key Exchange. In *Public Key Cryptography - PKC'06*, volume 3958 of *Lecture Notes in Computer Science*, pages 410–426. Springer, 2006. 179

[71] K. Y. Choi, J. Y. Hwang, and D. H. Lee. Efficient ID-based Group Key Agreement with Bilinear Maps. In *Public Key Cryptography - PKC'04*, volume 2947 of *Lecture Notes in Computer Science*, pages 130–144. Springer, March 2004. 91

[72] K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Errors in Computational Complexity Proofs for Protocols. In *Advances in Cryptology – ASIACRYPT'05*, volume 3788 of *Lecture Notes in Computer Science*, pages 624–643. Springer, 2005. 69, 78

[73] K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In *Advances in Cryptology – ASIACRYPT'05*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, 2005. 69, 72, 78

[74] R. Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty. In *Proceedings of the 18th ACM Symposium on Theory of Computing (STOC'86)*, pages 364–369. ACM Press, 1986. 123

[75] V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005. 42

[76] G. D. Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. How to Forget a Secret. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 500–509. Springer, 1999. 112, 185

[77] I. Csiszár and J. Körner. Broadcast Channels with Confidential Messages. *IEEE Transactions on Information Theory*, 24(3):339–348, 1978. 40, 41

[78] Y. G. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1989. 31

[79] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976. 13, 35, 64, 87, 88, 129

[80] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992. 36, 64, 65, 66, 120

[81] Y. Dodis and P. Puniya. On the Relation Between the Ideal Cipher and the Random Oracle Models. In *Third Theory of Cryptography Conference (TCC'06)*, volume 3876 of *Lecture Notes in Computer Science*, pages 184–206. Springer Verlag, 2006. 44

[82] D. Dolev and A. C.-C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983. 44

[83] R. Dutta and R. Barua. Constant Round Dynamic Group Key Agreement. In *Information Security: 8th International Conference (ISC'05)*, volume 3650 of *Lecture Notes in Computer Science*, pages 74–88. Springer, August 2005. 13, 81, 84, 102, 109, 110

[84] R. Dutta and R. Barua. Dynamic Group Key Agreement in Tree-Based Setting. In *Proceedings of the 10th Australasian Conference on Information Security and Privacy (ACISP'05)*, volume 3574 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2005. 42, 81, 84, 108, 109, 110

[85] R. Dutta and R. Barua. Overview of Key Agreement Protocols. Cryptology ePrint Archive, Report 2005/289, 2005. `http://eprint.iacr.org/2005/289/`. 87

[86] R. Dutta and R. Barua. Password-Based Encrypted Group Key Agreement. *International Journal of Network Security*, 3(1):23–34, July 2006. Available at `http://isrc.nchu.edu.tw/ijns/`. 13, 81, 84, 103

[87] R. Dutta, R. Barua, and P. Sarkar. Provably Secure Authenticated Tree Based Group Key Agreement. In *Proceedings of the 6th International Conference on Information and Communications Security (ICICS'04)*, volume 3269 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2004. 13, 42, 81, 84, 107, 108, 109, 110

[88] A. K. Ekert. Quantum Cryptography Based on Bell's Theorem. *Physical Review Letters*, 67(6):661–663, 1991. 40

[89] F. Fabrega, J. Herzog, and J. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *IEEE Symposium on Security and Privacy 1998*, pages 160–171. IEEE Press, 1998. 44

[90] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley, 2003. ISBN:0-471-22894-X. 53

[91] M. Fischlin. Pseudorandom Function Tribe Ensembles Based on One-Way Permutations: Improvements and Applications. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 1999. 55

[92] G. Frey and H. Rueck. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 62:865–874, 1994. 91

[93] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane. Codes which Detect Deception. *The Bell Systems Technical Journal*, 53(3):405–424, 1974. 40

[94] O. Goldreich. *Foundations of Cryptography - Basic Tools*, volume 1. Cambridge University Press, 2001. ISBN:0-521-79172-3. 53, 54, 55, 61

[95] O. Goldreich. *Foundations of Cryptography - Volume II Basic Applications*, volume 2. Cambridge University Press, 2004. ISBN:0-521-83084-2. 53, 105

[96] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986. 55

[97] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM Press, 1987. 42

[98] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 64

[99] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988. 59

[100] S. Greenberg. An Annotated Bibliography of Computer Supported Cooperative Work. *ACM SIGCHI Bulletin*, 23(3):29–62, 1991. 25

[101] S. Greenberg. Computer-Supported Cooperative Work and Groupware: An Introduction to the Special Issues. *International Journal of Man-Machine Studies*, 34(2):133–141, 1991. 25

[102] C. G. Günther. An Identity-Based Key-Exchange Protocol. In *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer, 1990. 36, 66

[103] S. Hirose and S. Yoshida. An Authenticated Diffie-Hellman Key Agreement Protocol Secure Against Active Attacks. In *Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography (PKC'98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 135–148. Springer, 1998. 34

[104] Y. Hitchcock, C. Boyd, and J. M. G. Nieto. Tripartite Key Exchange in the Canetti-Krawczyk Proof Model. In *Progress in Cryptology – INDOCRYPT'94*, volume 3348 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2004. 88

[105] D. Hofheinz, J. Müller-Quade, and R. Steinwandt. Initiator-Resilient Universally Composable Key Exchange. In *8th European Symposium on Research in Computer Secu-*

rity (ESORICS'03), volume 2808 of *Lecture Notes in Computer Science*, pages 61–84. Springer, 2003. 43

[106] T. Holenstein. *Strengthening Key Agreement using Hard-Core Sets*. PhD thesis, ETH Zurich, 2006. Reprint as vol. 7 of *ETH Series in Information Security and Cryptography*, ISBN 3-86626-088-2, Hartung-Gorre Verlag, Konstanz, 2006. 41

[107] I. Ingemarsson, D. T. Tang, and C. K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–719, 1982. 13, 89, 91

[108] P. Janson and G. Tsudik. Secure and Minimal Protocols for Authenticated Key Distribution. *Computer Communications*, 18(9):645–653, September 1993. 66

[109] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Algorithmic Number Theory, IV-th Symposium (ANTS IV)*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, July 2000. 13, 88, 91

[110] A. Joux and K. Nguyen. Separating Decision DiffieŰHellman from Computational DiffieŰHellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, September 2003. 91

[111] J. Katz and J. S. Shin. Modeling Insider Attacks on Group Key-Exchange Protocols. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 180–189. ACM Press, 2005. 37, 42, 43, 48, 55, 65, 82, 85, 111, 136, 137

[112] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *Advances in Cryptology - CRYPTO'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2003. 13, 37, 42, 48, 80, 82, 83, 85, 99, 100, 101, 102, 103, 109, 110, 111, 116, 129, 130, 132

[113] R. A. Kemmerer, C. Meadows, and J. K. Millen. Three Systems for Cryptographic Protocol Analysis. *Jorunal of Cryptology*, 7(2):79–130, 1994. 44

[114] H.-J. Kim, S.-M. Lee, and D. H. Lee. Constant-Round Authenticated Group Key Exchange for Dynamic Groups. In *Advances in Cryptology – ASIACRYPT'04*, volume 3329 of *Lecture Notes in Computer Science*, pages 245–259, 2004. 13, 42, 81, 84, 101, 109, 110, 112, 123, 185

[115] Y. Kim, D. Mazzocchi, and G. Tsudik. Admission Control in Peer Groups. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA'03)*, pages 131–139. IEEE Computer Society, 2003. 29, 30, 31

[116] Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*, pages 235–244. ACM Press, 2000. 64, 97, 98

[117] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proceedings of IFIP TC11 Sixteenth Annual Working Conference on Information Security (IFIP/Sec'01)*, volume 193 of *IFIP Conference Proceedings*, pages 229–244. Kluwer, 2001. 64, 95, 97, 98

[118] Y. Kim, A. Perrig, and G. Tsudik. Group Key Agreement Efficient in Communication. *IEEE Transactions on Computers*, 53(7):905–921, July 2004. 13, 97, 98, 179

[119] Y. Kim, A. Perrig, and G. Tsudik. Tree-Based Group Key Agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, February 2004. 36, 64, 97, 98, 99, 179

[120] N. Koblitz and A. Menezes. Another Look at "Provable Security". *Journal of Cryptology*. Online Issue, 30. November 2005. Also available at `http://eprint.iacr.org/2005/152.pdf`. 44

[121] H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In *Advances in Cryptology – CRYPTO'05*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005. 88

[122] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003. 88

[123] S. Lee, Y. Kim, K. Kim, and D.-H. Ryu. An Efficient Tree-Based Group Key Agreement Using Bilinear Map. In *Proceedings of the First International Conference on Applied Cryptography and Network Security (ACNS'03)*, volume 2846 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2003. 13, 98, 99

[124] L. Liao and M. Manulis. Tree-Based Group Key Agreement Framework for Mobile Ad-Hoc Networks. In *Proceedings of 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, volume 2, pages 5–9. IEEE Computer Society, 2006. 98

[125] L. Liao and M. Manulis. Tree-Based Group Key Agreement Framework for Mobile Ad-Hoc Networks. *Future Generation Computer Systems (FGCS)*, 23(6):787–803, July 2007. 98

[126] G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998. 44

[127] M. Manulis. Contributory Group Key Agreement Protocols, Revisited for Mobile Ad-Hoc Groups. In *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'05)*, pages 811–818. IEEE Computer Society, November 2005. 91, 93, 98

[128] U. Maurer. Secret Key Agreement by Public Discussion. *IEEE Transaction on Information Theory*, 39(3):733–742, 1993. 40, 41

[129] U. Maurer. Information-Theoretically Secure Secret-Key Agreement by NOT Authenticated Public Discussion. In *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 209–225. Springer-Verlag, 1997. 40, 41

[130] U. Maurer and S. Wolf. Towards Characterizing when Information-Theoretic Key Agreement is Possible. In *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 196–209. Springer-Verlag, 1996. 40

[131] U. Maurer and S. Wolf. The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999. 57

[132] U. Maurer and S. Wolf. Secret Key Agreement Over a Non-Authenticated Channel — Parts i, ii, iii. *IEEE Transactions on Information Theory*, 49(4):822–851, 2003. 41

[133] A. J. Mayer and M. Yung. Secure Protocol Transformation via "Expansion": From Two-Party to Groups. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS'99)*, pages 83–92. ACM Press, 1999. 34

[134] C. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology – ASIACRYPT'94*, volume 917 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1994. 44

[135] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996. ISBN:0-8493-8523-7. 26, 27, 28, 33, 36, 41, 53, 56, 64, 65, 66, 120, 123

[136] A. Menezes, S. Vanstone, and T. Okamoto. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the 23rd annual ACM Symposium on Theory of Computing (STOC'91)*, pages 80–89. ACM Press, 1991. 91

[137] V. S. Miller. Short Programs for functions on Curves. Manuskript, available at `http://crypto.stanford.edu/miller/miller.pdf`, May 1986. 91

[138] C. J. Mitchell, M. Ward, and P. Wilson. Key Control in Key Agreement Protocols. *Electronic Letters*, 34(10):980–981, 1998. 66, 123

[139] D. Nalla. ID-based Tripartite Key Agreement with Signatures. Cryptology ePrint Archive, Report 2003/144, 2003. `http://eprint.iacr.org/2003/144.pdf`. 88

[140] D. Nalla and K.C.Reddy. ID-based Tripartite Authenticated Key Agreement Protocols from Pairings. Cryptology ePrint Archive, Report 2003/004, 2003. `http://eprint.iacr.org/`. 88

[141] D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Advances in Cryptology – CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001. 35

[142] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *Advances in Cryptology – CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer Verlag, 2002. 44

[143] K. Ohta, S. Micali, and L. Reyzin. Accountable-Subgroup Multisignatures: Extended Abstract. In *Proceedings of ACM Conference on Computer and Communications Security (CCS'01)*, pages 245–254. ACM Press, 2001. 31

[144] O. Pereira and J.-J. Quisquater. A Security Analysis of the CLIQUES Protocols Suites. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 73–81. IEEE Computer Society Press, June 2001. 39, 94

[145] O. Pereira and J.-J. Quisquater. An Attack against Barua *et al.* Authenticated Group Key Agreement Protocol. Technical Report CG-2003-3, UCL Crypto Group, October 2003. 99

[146] O. Pereira and J.-J. Quisquater. Some Attacks upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 11(4):555–580, 2003. 39, 94

[147] A. Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In *Proceedings of the International Workshop on Cryptographic Techniques and Electronic Commerce 1999*, pages 192–202. City University of Hong Kong Press, 1999. 13, 97

[148] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *ACM Conference on Computer and Communications Security (CCS'00)*, pages 245–254. ACM Press, 2000. 42, 45

[149] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, 2004. 24

[150] S. Rafaeli and D. Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Computer Surveys*, 35(3):309–329, 2003. 87

[151] R. Renner, N. Gisin, and B. Kraus. An Information-Theoretic Security Proof for QKD Protocols. *Physical Review A*, 72(012332), 2005. `http://arxiv.org/abs/quant-ph/0502064`. 41

[152] R. Renner and S. Wolf. New Bounds in Secret-Key Agreement: The Gap Between Formation and Secrecy Extraction. In *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 562–577. Springer-Verlag, 2003. 41

[153] M. Roseman and S. Greenberg. GROUPKIT: A Groupware Toolkit for Building Real-Time Conferencing Applications. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'92)*, pages 43–50. ACM Press, 1992. 25

[154] A.-R. Sadeghi and M. Steiner. Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 244–261. Springer, 2001. full version available at `http://www.semper.org/sirene/lit/abstrA1.html`. 58

[155] V. L. Sauter. *Decision Support Systems: An Applied Managerial Approach*. John Wiley & Sons, January 1997. 25

[156] N. Saxena, G. Tsudik, and J. H. Yi. Access Control in Ad Hoc Groups. In *Proceedings of the International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P'04)*, pages 2–7. IEEE Computer Society, 2004. 31

[157] N. Saxena, G. Tsudik, and J. H. Yi. Efficient Node Admission for Short-lived Mobile Ad Hoc Groups. In *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP'05)*, pages 269–278. IEEE Computer Society, 2005. 31

[158] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1995. ISBN:0-471-11709-9. 53

[159] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989. 106

[160] J. Schwenk and Deutsche Telekom AG. Deutsches Patent DE19847941. 98

[161] J. Schwenk, T. Martin, and R. Schaffelhofer. Tree-based Key Agreement for Multicast. In *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues*, volume 192 of *IFIP Conference Proceedings*. Kluwer, 2001. 98

[162] C. E. Shannon. Communication Theory of Secrecy Systems. *The Bell Systems Technical Journal*, 28(4):656–715, 1949. 40, 43, 44, 71

[163] A. T. Sherman and D. A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, 2003. 35

[164] K. Shim. Cryptanalysis of Al-Riyami-Paterson's Authenticated Three Party Key Agreement Protocols. Cryptology ePrint Archive, Report 2003/122, 2003. `http://eprint.iacr.org/`. 88

[165] K. Shim. Cryptanalysis of ID-based Tripartite Authenticated Key Agreement Protocols. Cryptology ePrint Archive, Report 2003/115, 2003. `http://eprint.iacr.org/`. 88

[166] K. Shim. Efficient One-Round Tripartite Authenticated Key Agreement Protocol from the Weil Pairing. *Electronics Letters*, 39(2):208–209, January 2003. 88

[167] V. Shoup. On Formal Models for Secure Key Exchange (Version 4). Technical Report RZ 3120, IBM Research, November 1999. Also available at `http://shoup.net/`. 37, 42, 43, 73, 112

[168] V. Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/2004/332.pdf`. 42, 59, 60, 206, 209

[169] G. J. Simmons. A Survey of Information Authentication. *Contemporary Cryptology, The Science of Information Integrity*, pages 379–419, 1992. 40

[170] D. X. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of 12th IEEE Computer Security Foundations Workshop (CSFW'99),*, pages 192–202. IEEE Computer Society, 1999. 44

[171] D. G. Steer, L. Strawczynski, W. Diffie, and M. J. Wiener. A Secure Audio Teleconference System. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. Springer, 1990. 13, 35, 64, 89, 95, 179

[172] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Information Systems*, 5(2):147–167, 1987. 25

[173] M. Steiner. *Secure Group Key Agreement*. PhD thesis, Saarland University, March 2002. 37, 66

[174] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS'96)*, pages 31–37. ACM Press, 1996. 13, 92, 93, 96, 104

[175] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387. IEEE Computer Society Press, 1998. 64, 93, 94, 95

[176] M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, 2000. 93

[177] H.-M. Sun and B.-T. Hsieh. Security Analysis of Shim's Authenticated Key Agreement Protocols from Pairings. Cryptology ePrint Archive, Report 2003/113, 2003. `http://eprint.iacr.org/2003/113`. 88

[178] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002. 25

[179] J. Vesperman. *Essential CVS*. O'Reilly Media, 2003. 24

[180] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey Framework: Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, September 1999. 35

[181] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures. Internet RFC/STD/FYI/BCP Archives, June 1999. RFC 2627. Available at `http://www.faqs.org/rfcs/rfc2627.html`. 34, 35

[182] M. N. Wegman and J. L. Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*, 22(8):265–279, 1981. 40

[183] M. Wessner and H.-R. Pfister. Group Formation in Computer-Supported Collaborative Learning. In *Proceedings of ACM 2001 International Conference on Supporting Group Work (GROUP'01)*, pages 24–31. ACM Press, 2001. 25

[184] S. Wolf. Strong Security Against Active Attacks in Information-Theoretic Secret-Key Agreement. In *Advances in Cryptology — ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 405–419. Springer-Verlag, 1998. 41

[185] S. Wolf. *Information-Theoretically and Computationally Secure Key Agreement in Cryptography*. PhD thesis, ETH Zürich, 1999. 41, 58

[186] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'98)*, pages 68–79. ACM Press, 1998. 34, 35

[187] D. R. Woolley. Web Conferencing Guide. World Wide Web. http://thinkofit.com/webconf/. 24

[188] A. D. Wyner. The Wiretap Channel. *The Bell Systems Technical Journal*, 54(8):1355–1387, 1975. 40, 41

[189] Y. Yacobi and Z. Shmuely. On Key Distribution Systems. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 344–355. Springer, August 1990. 36, 64

[190] F. Zhang and X. Chen. Attack on Two ID-based Authenticated Group Key Agreement Schemes. Cryptology ePrint Archive, Report 2003/259, 2003. Available at http://eprint.iacr.org/2003/259/. 91

[191] F. Zhang, S. Liu, and K. Kim. ID-Based One Round Authenticated Tripartite Key Agreement Protocol with Pairings. Cryptology ePrint Archive, Report 2002/122, 2002. http://eprint.iacr.org/2002/122. 88, 99