

# Malicious Participants in Group Key Exchange: Key Control and Contributiveness in the Shadow of Trust

## [Full version]

Emmanuel Bresson<sup>1</sup> and Mark Manulis<sup>2\*</sup>

<sup>1</sup> DCSSI Crypto Lab, Paris, France, [emmanuel.bresson@polytechnique.org](mailto:emmanuel.bresson@polytechnique.org)

<sup>2</sup> Horst Görtz Institute, Ruhr-University of Bochum, Germany, [mark.manulis@rub.de](mailto:mark.manulis@rub.de)

**Abstract.** Group key exchange protocols allow their participants to compute a secret key which can be used to ensure security and privacy for various multi-party applications. The resulting group key should be computed through cooperation of all protocol participants such that none of them is trusted to have any advantage concerning the protocol's output. This trust relationship states the main difference between group key exchange and group key transport protocols. Obviously, misbehaving participants in group key exchange protocols may try to influence the resulting group key, thereby disrupting this trust relationship, and also causing further security threats. This paper analyzes the currently known security models for group key exchange protocols with respect to this kind of attacks by malicious participants and proposes an extended model to remove the identified limitations. Additionally, it proposes an efficient and provably secure generic solution, a compiler, to guarantee these additional security goals for group keys exchanged in the presence of malicious participants.

**Key words:** group key exchange, malicious participants, key control, contributiveness, security model, compiler

## 1 Introduction

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. In almost 25 years of research on group key establishment protocols two different classes – *group key transport* and *group key exchange* (or equivalently *group key distribution* and *group key agreement*) – became subject of the most scientific papers in this area. The following informal definitions, originally proposed by Menezes *et al.* [42] for two-party protocols and extended here for the group setting, help to understand the fundamental difference between these classes.

**Definition 1 (Group Key Transport).** A group key transport (GKT) protocol *is a group key establishment protocol where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).*

The main characteristic of GKT protocols is that the group key is chosen by a single party and then securely transferred to all group members. This definition leaves open whether the party itself is one of the group members or some trusted third party (TTP). Also the requirement on secure transfer of group keys forebodes the existence of secret communication channels between the party that transfers the group keys and other group members.

**Definition 2 (Group Key Exchange).** A group key exchange (GKE) protocol *is a group key establishment protocol 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.*

---

\* The corresponding author was supported by the European Commission through IST-2002-507932 ECRYPT.

Obviously, in GKE protocols all group members have to interact in order to compute the group key. A minor difference to GKT protocols is that GKE protocols do not require the existence of secure channels between its participants since no secure transfer takes place. A more important difference is that in GKE protocols no party (or a coalition thereof) is allowed to choose the group key on behalf of the whole group. This requirement implies that in GKE protocols participants do not trust each other during the computation of the group key and provides the background for the consideration of malicious participants during the protocol execution whose goal is to influence the resulting value of the group key. Note that the behavior of malicious participants may strongly deviate from the protocol specification. In case that malicious participants succeed and enforce the computation of the same value for the group key in two (or more) different protocol sessions or applications that use the same GKE protocol, new security threats become evident. For example, if the group key is used for encryption then it would be no more possible to distinguish between cipher texts produced in different sessions or applications making interference attacks possible, e.g., encrypted messages generated by honest participants can be replayed.

In the paradigm of *provable security*, security analysis must hold in some formal security model. Early GKE protocols have been analyzed without such formal settings and in a rather heuristic way; some of them have been, therefore, broken later. In 2001, Bresson *et al.* [15] introduced the first (computational) security model for GKE protocols (referred to as the BCPQ model), adapting some ideas by Bellare and Rogaway [6, 8]. Several variations and refinements have been suggested thereafter [13, 34, 33]; we refer to [41] for the survey and analysis of many currently known GKE security models. The BCPQ model assumes honest participants and deal with two major security notions. The first notion is *authenticated key exchange* (AKE) security which requires the indistinguishability of computed group keys from random keys. Although this notion has been stated for GKE protocols we stress that similar threats should be prevented for GKT protocols too. This follows from the general trust assumption that participants of any group key establishment protocol do not deliberately reveal computed group keys to the adversary. The second notion is *mutual authentication* (MA) security originating from [6] and meaning that two parties authenticate bilaterally: it requires that all protocol participants are ensured to actually compute the same key.

A number of papers, e.g., [43, 2, 21, 33], point out that the consideration of dishonest participants (either curious or malicious) is of prime importance in the group setting, because such misbehaving insiders can have catastrophic effects on the protocol security. For example, [21] noticed that some protocols proven secure in the BCPQ-like models are vulnerable to unknown key-share attacks – in which the attacker is *believed* (from some participant’s view) to be a group member. In case of MA-security, malicious participants may try to disrupt honest participants so that they compute different keys but cannot notice this.

Mitchel *et al.* in [43] first mentioned the issue of *key control* by which a misbehaving participant can influence the value of the key. A related notion called *contributiveness* was proposed by Ateniese *et al.* [2] requiring that all protocol participants equally contribute to the computation of the group key. These requirements implicitly state a difference between GKT and GKE protocols in the spirit of Definitions 1 and 2. The main reason is that key control and contributiveness assume that none of the protocol participants is trusted to choose the group key on behalf of other participants. However, the way towards formal definitions of these requirements is not obvious. A weaker security model (as in [12]) would consider protocol participants that have biased pseudo-random generators and a curious adversary obtaining some extra information about the key. In this paper we consider a stronger setting (in spirit of [9]), where malicious participants try to influence honest participants computing some special value as a group key (including the so-called *key replication* attacks [39]).

In addition to usual corruptions where the adversary obtains full control over the parties we also consider *strong corruptions* [46, 49, 13], that is, capabilities of the adversary to reveal internal memory of participants. We also consider strong corruptions in the context of a curious adversary that reveals (but not modifies) ephemeral secrets of honest participants. Currently, security against strong corruptions is considered in a rather restrictive way, as part of the *strong forward secrecy* requirement in the context of AKE-security [13]. In order to talk about security of GKE protocols against strong corruptions in general we expand these considerations for other requirements within our security model.

**Contributions and Organization** This paper provides an extended treatment of security of GKE protocols in the presence of malicious participants and strong corruptions. In other words, we formally define what a “secure group key” means in such scenario. As a starting motivation, in Sections 2 and 2.1 we first discuss why currently known security models for GKE protocols are not mature enough to deal with malicious participants and strong corruptions. Then, in Section 3 we extend the notions of AKE- and MA-security and propose a new definition of contributiveness which models the main difference between GKE and GKT protocols in the sense of Definition 2.

In Section 4 we describe the relationship between our formal definitions of MA-security and contributiveness through some informally stated requirements from the previous literature. To prove the soundness and feasibility of our extensions, in Section 5 we propose a generic solution (compiler) which turns any AKE-secure GKE protocol into an enhanced protocol, which provably satisfies our advanced security requirements under standard cryptographic assumptions.

## 2 Related Work

**General Security Notions for GKE Protocols** AKE-security as defined in [15,13,34] subsumes some informal security goals defined in the literature: *key secrecy* [23] or *implicit key authentication* [42] which ensures that no party except for legitimate participants learns the established group key, *security against impersonation attacks* [17] or a related notion of *entity authentication* [6] requiring that an adversary must not be able to replace an honest participant in the execution of the protocol; *resistance against known-key attacks* [51,16] meaning that an adversary knowing group keys of previous sessions cannot compute subsequent session keys, *key independence* [36] meaning that an adversary knowing a proper subset of group keys must not be able to discover any other group keys. Also it subsumes (*perfect*) *forward secrecy* [30,23,42] which requires that the disclosure of long-lived keys must not compromise the secrecy of the previously established group keys. The latter can be strengthened by the requirement of *strong forward secrecy* [13] in which the adversary in addition to the long-lived keys reveals the internal data of participants such as ephemeral secrets used during the protocol execution.

The currently available formal definition of MA-security in [15] has been designed to cover the informal definitions of *key confirmation* or *explicit key authentication* [42,6, § 12.2].

According to [21], however, these definitions do not consider security against *unknown key-share attacks* [23,10], in which a corrupted participant can make an honest participant believe that the key is shared with one party though in fact it is shared with another party.

**Informal Security Treatment of Key Control and Contributiveness** There have been only few attempts to handle malicious participants in GKE protocols. Misbehavior of protocol participants was first mentioned in [43]: the authors described the issue of *key control*. They suggested to use hash functions in order to *commit* on the values (ephemeral secrets) used in the group key computation (without proving security of this suggestion). However, when considering strong corruptions commitments do not provide sufficient security since the committed secret value can be revealed as part of the participant’s internal state. Independently, Ateniese *et al.* [2] introduced a more general notion of *unpredictability* (which intuitively implies security against key control). Further, they proposed a related notion called *contributory* group key agreement: the property by which each participant equally contributes to the resulting group key and guarantees its freshness. Based on the definition of contributiveness [2] specifies *complete group key authentication* as the property by which any two parties compute the same key only if all other parties have contributed to it. Moreover, they defined *verifiable contributory* GKE protocols where each participant must be assured of every other participant’s contribution. Some subsequent security models have tried to formalize this approach. Though the protocols in [2] have been broken in [44] we stress that the notions are still worth being formalized.

**The KS Model** Katz and Shin [33] proposed security definitions against malicious participants in a BCPQ-like model: briefly speaking any user  $U_i$  may have many instance oracles  $\Pi_i^s$ ,  $s \in \mathbb{N}$ . Each oracle represents  $U_i$  in one of many possible concurrent protocol executions. All participants of the

same protocol execution are considered as *partners*. First, the KS model says that  $\mathcal{A}$  impersonates (uncorrupted)  $U_j$  to (accepting)  $\Pi_i^s$  if  $U_j$  belongs to the (expected) partners of  $\Pi_i^s$ , but in fact no oracle  $\Pi_j^t$  is partnered with  $\Pi_i^s$ . In other words, the instance  $\Pi_i^s$  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  $U_j$ . Then, the authors call a protocol secure against *insider impersonation attacks* if for any party  $U_j$  and any instance  $\Pi_i^s$ , the adversary cannot impersonate  $U_j$  to  $\Pi_i^s$ , under the (stronger) condition that neither  $U_j$  nor  $U_i$  is corrupted at the time  $\Pi_i^s$  accepts.

**The BVS Model** Bohli *et al.* [9] proposed another extension (which we refer to as the BVS model) towards security goals in the presence of malicious participants. The process dealing with key control and contributiveness, at an informal level, runs as follows. In a first stage, the adversary  $\mathcal{A}$  interacts with the users and may corrupt some of them;  $\mathcal{A}$  then specifies an unused instance oracle  $\Pi_i^s$  and a subset  $K$  in the session key space  $\mathcal{K}$ . In the second stage, the adversary tries to make  $\Pi_i^s$  accept a session key  $k \in K$  but is not allowed to corrupt  $U_i$ . The BVS model defines a GKE protocol as being *t-contributory* if the adversary succeeds with only negligible probability, with the total number of corruptions remains (strictly) less than  $t$ . A *n-contributory* protocol between  $n$  participants is called a *key agreement*.

## 2.1 Discussion on the KS and BVS Models

**Missing Key Control and Contributiveness in the KS Model** Katz and Shin proposed a compiler to turn any AKE-secure protocol (in the sense of BCPQ) into a protocol secure in their extended model. In the following we illustrate how malicious participants can predict the resulting value of the group key, that is, the KS model does not provide key control and contributiveness.

The KS compiler uses a pseudo-random function  $f_k$  with  $k \in_R \{0, 1\}^\kappa$  and runs (intuitively) as follows. Each player acknowledges the key  $k_i$  obtained in the input protocol by signing and sending a token  $\text{ack}_i := f_{k_i}(v_0)$  where  $v_0$  is a public value. If all verifications match, players terminate the compiled protocol with key  $K_i := f_{k_i}(v_1)$  where  $v_1 \neq v_0$  is another public value. We argue that the compiler does not ensure unpredictability and contributiveness since  $K_i$  may be predictable as soon as  $k_i$  is, and  $K_i$  is not composed of each participant's contribution if  $k_i$  is not.

**Absence of Strong Corruptions in the BVS Model** A first drawback of the BVS model is that  $\mathcal{A}$  is not adaptive in her choice of  $\Pi_i^s$ , because she is required to commit to it in the first stage (not in the second). The second, main drawback is that strong corruptions are not allowed: therefore, contributiveness does not capture attacks in which  $\mathcal{A}$  tries to influence the session key using the (passive) knowledge of the internal states of the honest oracles (but without knowing their long-term secrets).

## 3 Our Extended Security Model

In the following we propose a security model for GKE protocols that includes extended security definitions concerning MA-security and contributiveness, while taking into account strong corruptions. Similar to [34, 33, 9] our model assumes that the communication channel is fully controlled by the adversary which can simply refuse to deliver protocol messages (even those originated by honest participants). Therefore, our definitions do not deal with the denial-of-service attacks and fault-tolerance issues but rather aim to recognize that the actual protocol execution deviates from the original specification and prevent that an honest participant accepts a “biased” group key.

### 3.1 Protocol Participants, Variables

**Users, Instance Oracles** Similar to [13]  $\mathcal{U}$  is a set of  $N$  users. Each user  $U_i \in \mathcal{U}$  holds a long-lived key  $LL_i$ . In order to handle participation of  $U_i$  in distinct concurrent protocol executions we consider that  $U_i$  has an unlimited number of instances called *oracles*;  $\Pi_i^s$ , with  $s \in \mathbb{N}$ , denotes the  $s$ -th instance oracle of  $U_i$ .

**Internal States** Every  $\Pi_U^s$  maintains an *internal state information*  $\mathbf{state}_U^s$  which is composed of all private, ephemeral information used during the protocol execution. The long-lived key  $LL_U$  is, in nature, excluded from it (moreover the long-lived key is specific to the user, not to the oracle).

**Session Group Key, Session ID, Partner ID** In each session we consider a new group  $\mathcal{G}$  of  $n \in [1, N]$  participating oracles. Each oracle in  $\mathcal{G}$  is called a *group member*. By  $\mathcal{G}_i$  for  $i \in [1, n]$  we denote the index of the user related to the  $i$ -th oracle involved in  $\mathcal{G}$  (this  $i$ -th oracle is denoted  $\Pi(\mathcal{G}, i)$ ). Thus, for every  $i \in [1, n]$  there exists  $\Pi(\mathcal{G}, i) = \Pi_{\mathcal{G}_i}^s \in \mathcal{G}$  for some  $s \in \mathbb{N}$ . Every participating oracle  $\Pi_U^s \in \mathcal{G}$  computes the *session group key*  $k_{\mathcal{G}}^s \in \{0, 1\}^\kappa$ . Every session is identified by a unique *session id*  $\mathbf{sid}_{\mathcal{G}}^s$ . This value is known to all oracles participating in the same session. Similarly, each oracle  $\Pi_U^s \in \mathcal{G}$  gets a value  $\mathbf{pid}_U^s$  that contains the identities of participating users (including  $U$ ), or formally

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

We say that two oracles,  $\Pi_i^{s_i}$  and  $\Pi_j^{s_j}$ , are *partnered* if  $U_i \in \mathbf{pid}_j^{s_j}$ ,  $U_j \in \mathbf{pid}_i^{s_i}$ , and  $\mathbf{sid}_i^{s_i} = \mathbf{sid}_j^{s_j}$ .

**Instance Oracle States** An oracle  $\Pi_U^s$  may be either *used* or *unused*. The oracle is considered as unused if it has never been initialized. Each unused oracle  $\Pi_U^s$  can be initialized with the long-lived key  $LL_U$ . The oracle is initialized as soon as it becomes part of some group  $\mathcal{G}$ . 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  $\Pi_U^s$  learns its partner id  $\mathbf{pid}_U^s$  (and possibly  $\mathbf{sid}_U^s$ ) and turns into a *processing* state where it sends, receives and processes messages according to the description of the protocol. During the whole processing state the internal state information  $\mathbf{state}_U^s$  is maintained by the oracle. The oracle  $\Pi_U^s$  remains in the processing state until it collects enough information to compute the session group key  $k_{\mathcal{G}}^s$ . As soon as  $k_{\mathcal{G}}^s$  is computed  $\Pi_U^s$  *accepts* and *terminates* the protocol execution meaning that it would not send or receive further messages. If the protocol execution fails (due to any adversarial actions) then  $\Pi_U^s$  terminates without having accepted, i.e., the session group key  $k_{\mathcal{G}}^s$  is set to some undefined value.

### 3.2 Definition of a Group Key Exchange Protocol

**Definition 3 (GKE Protocol).** A group key exchange protocol  $P$  consists of the key generation algorithm  $\mathbf{KeyGen}$ , and a protocol  $\mathbf{Setup}$  defined as follows:

$P.\mathbf{KeyGen}(1^\kappa)$ : On input a security parameter  $1^\kappa$  each user in  $\mathcal{U}$  is provided with a long-lived key  $LL_U$ .

$P.\mathbf{Setup}(\mathcal{S})$ : On input a set  $\mathcal{S}$  of  $n$  unused oracles a new group  $\mathcal{G}$  is created and set to be  $\mathcal{S}$ , then a probabilistic interactive protocol is executed between oracles in  $\mathcal{G}$ .

We call  $P.\mathbf{Setup}$  an *operation*. We say that a protocol is *correct* if all oracles in  $\mathcal{G}$  accept with the same session group key  $k$ . We assume it is the case for all protocols in this paper.

### 3.3 Adversarial Model

**Queries to the Instance Oracles** We now consider an adversary  $\mathcal{A}$  which is a Probabilistic Polynomial-Time (PPT) algorithm having complete control over the network.  $\mathcal{A}$  can invoke protocol execution and interact with protocol participants via queries to their oracles.

$\mathbf{Execute}(\mathcal{S})$ : This query models  $\mathcal{A}$  eavesdropping the honest operation execution of  $P.\mathbf{Setup}$ .

$P.\mathbf{Setup}(\mathcal{S})$  is executed and  $\mathcal{A}$  is given the transcript of the execution.

$\mathbf{Send}(\Pi_U^s, m)$ : This query models  $\mathcal{A}$  sending messages to the oracles.  $\mathcal{A}$  receives the response which  $\Pi_U^s$  would have generated after having processed the message  $m$  according to the description of  $P$ . The adversary can ask an oracle  $\Pi_U^s$  to invoke  $P.\mathbf{Setup}$  with the oracles in  $\mathcal{S}$  via the query of the form  $\mathbf{Send}(\Pi_U^s, \mathcal{S})$  which gives  $\mathcal{A}$  the first message that  $\Pi_U^s$  would generate in this case. Thus, using  $\mathbf{Send}$  queries the adversary can actively participate in  $P.\mathbf{Setup}$ .

*RevealKey*( $\Pi_U^s$ ):  $\mathcal{A}$  is given the session group key  $k_U^s$ . This query is answered only if  $\Pi_U^s$  has accepted.

*RevealState*( $\Pi_U^s$ ):  $\mathcal{A}$  is given the internal state information  $\mathbf{state}_U^s$ .

*Corrupt*( $U$ ):  $\mathcal{A}$  is given the long-lived key  $LL_U$ .

*Test*( $\Pi_U^s$ ): This query is used to model the AKE-security of a protocol. It can be asked by  $\mathcal{A}$  as soon as  $\Pi_U^s$  accepts, but only once in  $\mathcal{A}$ 's execution. The query is answered as follows: 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.

We say that  $\Pi_U^s$  is a *malicious participant* if the adversary has previously asked the *Corrupt*( $U$ ) query. In all other cases  $\Pi_U^s$  is *honest*. We say the adversary is *curious* if it asks a *RevealState*( $\Pi_U^s$ ) query for some honest  $\Pi_U^s$ . This is possible since long-lived keys are separated from the ephemeral secrets stored in  $\mathbf{state}_U^s$ .

### 3.4 Security Goals

**AKE-Security with Strong Forward Secrecy** As defined in [13] strong forward secrecy states that AKE-security of previously computed session keys is preserved if the adversary obtains long-lived keys of protocol participants and internal states of their oracles in later protocol sessions.

**Definition 4 (Oracle Freshness).** *In the execution of  $P$  the oracle  $\Pi_U^s$  is fresh if all of the following holds:*

1. no  $U_i \in \text{pid}_U^s$  is asked for a *Corrupt* query prior to a query of the form *Send*( $\Pi_j^{s_j}, m$ ) such that  $U_j \in \text{pid}_U^s$  before  $\Pi_U^s$  and all its partners accept,
2. neither  $\Pi_U^s$  nor its partners are asked for a *RevealState* query before  $\Pi_U^s$  and all its partners accept,
3. neither  $\Pi_U^s$  nor any of its partners is asked for a *RevealKey* query after having accepted.

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

Note that in our model each  $\Pi_U^s$  is bound to one particular protocol execution (session). Thus,  $\Pi_U^s$  remains fresh if *RevealState* and *RevealKey* queries have been asked to other oracles owned by  $U$ , that is to oracles participating in other sessions. Hence, in contrast to [13] (and [34]) our definition allows the adversary to obtain knowledge of internal states from earlier sessions too.

**Definition 5 (AKE-Security).** *Let  $P$  be a GKE protocol from Definition 3 and  $b$  a uniformly chosen bit. Consider an active adversary  $\mathcal{A}$  participating in game  $\text{Game}_{\text{sfs},P}^{\text{ake}-b}(\kappa)$  defined as follows:*

- after initialization,  $\mathcal{A}$  interacts with instance oracles using queries;
- at some point  $\mathcal{A}$  asks a *Test* query to a **fresh** oracle  $\Pi_U^s$  which has accepted, and 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 trying to guess  $b$ .

The output of  $\mathcal{A}$  is the output of the game. The advantage function (over all adversaries running within time  $\kappa$ ) in winning this game is defined as:

$$\text{Adv}_{\text{sfs},P}^{\text{ake}}(\kappa) := \left| 2 \Pr[\text{Game}_{\text{sfs},P}^{\text{ake}-b}(\kappa) = b] - 1 \right|$$

A GKE protocol  $P$  is AKE-secure with strong forward secrecy (**AGKE-sfs**) if this advantage is negligible.

**MA-Security** Our definition of mutual authentication security differs from the one in [13,15] which does not consider malicious participants and curious adversaries and is vulnerable to unknown key-share attacks.

**Definition 6 (MA-Security).** Let  $P$  be a correct GKE protocol and  $\text{Game}_P^{\text{ma}}(\kappa)$  the interaction between the instance oracles and an active adversary  $\mathcal{A}$  who is allowed to query *Send*, *Execute*, *RevealKey*, *RevealState*, and *Corrupt*. 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  $U_j \in \text{pid}_i^{s_i}$  that is uncorrupted at the time  $\Pi_i^{s_i}$  accepts, such that

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

The maximum probability of this event (over all adversaries running within time  $\kappa$ ) is denoted  $\text{Succ}_P^{\text{ma}}(\kappa)$ . We say that a GKE protocol  $P$  is MA-secure (**MAGKE**) if this probability is a negligible function of  $\kappa$ .

This definition subsumes informal requirements of *unknown key-share resilience* [23,10], *key confirmation* [42], and the original notion of *mutual authentication* [6] in the context of group key exchange protocols as claimed in Section 4.

Note that  $U_i$  and  $U_j$  must be uncorrupted, however,  $\mathcal{A}$  is allowed to reveal internal states of their oracles.

**Contributiveness** In the following we propose a definition which deals with the issues of key control, contributiveness and unpredictability of session group keys in case of strong corruptions; this, again, is important for the security of GKE protocols in the assumed trust relationship. Informally, we consider an active PPT adversary which is allowed to corrupt up to  $n - 1$  group members and reveal internal states of all  $n$  oracles during the execution of  $P$  aiming to achieve that there exists at least one uncorrupted group member whose oracle accepts the session group key chosen by the adversary. In particular, our definition prevents malicious participants from being able to influence honest participants to accept some previously used session group key in a later session (this includes so-called *key replication* attacks [39]). We stress that considering strong corruptions in the following definition is important, since they are also considered in the definitions of AKE- and MA-security. Note also that any GKE protocol satisfying our strong definition implicitly prevents key control and provides contributiveness for the same goal where only weak corruptions are considered (as in [9]).

**Definition 7 (Contributiveness).** Let  $P$  be a correct GKE protocol and  $\mathcal{A}$  an adversary running in two stages, *prepare* and *attack*, that interacts with the instance oracles in the following game  $\text{Game}_P^{\text{con}}(\kappa)$ :

- $\mathcal{A}(\text{prepare})$  is given access to the queries *Send*, *Execute*, *RevealKey*, *RevealState*, and *Corrupt*. At the end of the stage, it outputs  $\tilde{k} \in \{0,1\}^\kappa$ , and some state information  $St$ . After  $\mathcal{A}$  makes its output and all previously asked queries are processed the following sets are built:  $\mathcal{G}_{\text{us}}$  consisting of all honest used oracles,  $\mathcal{G}_{\text{std}}$  consisting of all honest oracles that are in the stand-by state ( $\mathcal{G}_{\text{std}} \subseteq \mathcal{G}_{\text{us}}$ ), and  $\Psi$  consisting of session ids  $\text{sid}_i^{s_i}$  for every  $\Pi_i^{s_i} \in \mathcal{G}_{\text{us}}$ . Then  $\mathcal{A}$  is invoked for the *attack* stage.
- $\mathcal{A}(\text{attack}, St)$  is given access to queries *Send*, *Execute*, *RevealKey*, *RevealState*, and *Corrupt*. At the end of the stage  $\mathcal{A}$  outputs  $(s, U)$ .

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

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

The maximal probability (over all adversaries running within time  $\kappa$ ) in winning the game is defined as

$$\text{Succ}_P^{\text{con}}(\kappa) := \Pr[\mathcal{A} \text{ wins in Game}_P^{\text{con}}(\kappa)]$$

We say that a GKE protocol  $P$  is contributory (CGKE) if this probability is a negligible function of  $\kappa$ .

**Comments** The first requirement ensures that  $\Pi_U^s$  belongs to an uncorrupted user. The condition  $\Pi_U^s \notin \mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$  prevents the case where  $\mathcal{A}$  while being an operation participant outputs  $\tilde{k}$  for the still running operation which is then accepted by  $\Pi_U^s$  that participates in the same operation (this is not an attack since participants do not compute group keys synchronously). Note that  $\mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$  consists of all oracles that at the end of the **prepare** stage have already terminated or remain in the processing state. Similarly, the condition  $\text{id}_U^s \notin \Psi$  prevents that  $\mathcal{A}$  while being in the **attack** stage outputs  $(s, U)$  such that  $\Pi_U^s$  has accepted with  $\tilde{k}$  already in the **prepare** stage; otherwise as soon as  $\Pi_U^s$  computes some  $k_U^s$  in the **prepare** stage  $\mathcal{A}$  can trivially output  $\tilde{k} = k_U^s$ . The second requirement allows  $\mathcal{A}$  to corrupt at most  $n - 1$  (out of totally  $n$ ) participants in the session where  $\Pi_U^s$  accepts with  $\tilde{k}$ .

Note also that  $U$  must be uncorrupted but curious  $\mathcal{A}$  is allowed to reveal the internal state of  $\Pi_U^s$  during the execution of the **attack** stage (this is because our model separates  $LLU$  from  $\text{state}_U^s$ ). Also, due to the adaptiveness and strong corruptions the adversary in this game seems to be strictly stronger<sup>1</sup> than in [9].

The following example highlights this idea. We consider the well-known two-party Diffie-Hellman (DH) key exchange<sup>2</sup> [22], and show that if a malicious participant is able to (passively) reveal internal states of the oracles (strong corruptions) then it has full control over the obtained key. Let  $U_1$  and  $U_2$  have their corresponding oracles  $\Pi_1^{s_1}$  and  $\Pi_2^{s_2}$ . They choose ephemeral secret exponents  $x_1$  and  $x_2$ , then exchange the (authenticated) values  $g^{x_1}$  and  $g^{x_2}$ , respectively, and finally compute the key  $k := g^{x_1 x_2}$ . Now assume  $U_1$  is malicious. She specifies  $\tilde{k}$  as  $g^{\tilde{x}}$  for some chosen  $\tilde{x}$  before the execution of the protocol. Since the communication model is asymmetric (this is also the common case in praxis)  $U_1$  waits to receive  $g^{x_2}$  sent by the honest  $U_2$ , then queries  $\text{RevealState}(\Pi_2^{s_2})$  to obtain  $x_2$  as part of the internal state of  $\Pi_2^{s_2}$ , and finally computes  $x_1 := \tilde{x}/x_2$  and sends  $g^{x_1}$  to  $U_2$ . It is easy to see that  $U_2$  accepts with  $k := (g^{x_1})^{x_2} = g^{\tilde{x}} = \tilde{k}$ . As observed in [40] similar attacks can be found against many currently known group key exchange protocols (that are extensions of the original DH protocol), e.g., [17, 34, 20, 24, 31, 50, 3, 48, 5, 45, 38, 37, 32, 4, 26].

## 4 Unifying Relationship of MA-Security and Contributiveness

In this section we present some claims to illustrate that given definitions of MA-security and contributiveness unify many related informal definitions proposed in the previous literature, particularly in [42, 2, 10]. Note that missing formalism in the informal definitions allows only argumentative proofs.

<sup>1</sup> Definition 7 ensures unpredictability of group keys and is sufficient for preventing (application interference) attacks resulting from the same group key being computed twice. However, (similar to [9]) we do not deal with the unpredictability of *some bits* of the group key. Independent of the question on reasonability of this attack (since its threats are not obvious), it is impossible to achieve such *decisional* contributiveness can for the asymmetric communication and strong corruptions with the same bounds as in Definition 7, i.e., up to  $n - 1$  corruptions and possible  $\text{RevealState}$  queries to all  $n$  oracles. Note that during the protocol execution there are no secrets from the adversarial perspective. Thus, intuitively, *decisional* contributiveness is related to the problem of asynchronous distributed consensus for probabilistic algorithms without trusted parties for which a theoretical bound of at most  $(n - 1)/3$  corrupted parties exists, e.g., [11, 19, 18]. On the other hand, in the weak corruption model (without  $\text{RevealState}$  queries) *decisional* contributiveness can be easily achieved, e.g., via (hash) commitments as in [43, 35], whereas strong corruptions can reveal the committed secrets as part of  $\text{state}_U^s$ .

<sup>2</sup> As observed in [40] similar attacks can be found against many currently known group key exchange protocols.

**Claim 1** *If  $P$  is a MAGKE protocol then it provides key confirmation and mutual authentication (explicit key authentication) in the sense of [42, 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 (implicit) key authentication then there exists at least one honest participant  $U_i \in \mathcal{G}$  whose oracle  $\Pi_i^{s_i}$  has accepted with a session group key  $k_i^{s_i}$  and there exists at least one another honest participant  $U_j \in \text{pid}_i^{s_i}$  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 6 this is a successful attack against the MA-security of  $P$ . This, however, contradicts to the assumption that  $P$  is a MAGKE protocol.

**Claim 2** *If  $P$  is a MAGKE protocol then it is resistant against unknown key-share attacks in the sense of [10, 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  $U_i \notin \text{pid}_j^{s_j}$  must hold (otherwise after having accepted  $U_j$  would believe that the key is shared with  $U_i$ ) whereas  $U_j \in \text{pid}_i^{s_i}$ . This implies  $(\text{pid}_j^{s_j}, \text{sid}_j^{s_j}) \neq (\text{pid}_i^{s_i}, \text{sid}_i^{s_i})$ . On the other hand,  $P$  is by assumption MAGKE. Thus, according to Definition 6 for any  $U_j \in \text{pid}_i^{s_i}$  there must exist a corresponding oracle  $\Pi_j^{s_j}$  such that  $(\text{pid}_j^{s_j}, \text{sid}_j^{s_j}) = (\text{pid}_i^{s_i}, \text{sid}_i^{s_i})$ . This is a contradiction.

**Claim 3** *If  $P$  is a CGKE protocol then its output is unpredictable by any subset of  $n-1$  participants.*

*Proof (informal).* If the output of  $P$  is predictable by a subset  $\tilde{\mathcal{G}} \subset \mathcal{G}$  of  $n-1$  protocol participants then there exists  $\tilde{k}$  which was predicted by  $\tilde{\mathcal{G}}$  and accepted by some oracle  $\Pi_U^s$  of an uncorrupted user  $U \in \mathcal{G} \setminus \tilde{\mathcal{G}}$ . However, this implies that there exists an adversary  $\mathcal{A}$  who corrupts up to  $n-1$  users whose oracles are partnered with  $\Pi_U^s$  and predicts the session group key accepted by  $\Pi_U^s$ . This is a contradiction to the assumption that  $P$  is a CGKE protocol.

**Claim 4** *If  $P$  is a CGKE protocol then  $P$  is contributory in the sense of [2, 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  $\Pi_U^s$  who accepts a session group key without having contributed to its computation, i.e., the session group key accepted by  $\Pi_U^s$  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  $\Pi_U^s$  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 5** *If  $P$  is a CGKE and a MAGKE protocol then  $P$  provides complete group key authentication in the sense of [2, 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 CGKE protocol then according to the previous claim  $P$  is contributory. Hence, none of the honest users accepts the key without having contributed to its computation. Since  $P$  is a MAGKE protocol all honest users accept the same session group key. Hence, all honest users have contributed to the session group key. Therefore, there can be no pair of users who accept the same group key which is not contributed to by all other honest users. Thus,  $P$  provides complete group key authentication.

The notion of verifiable contributiveness [2] 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 6** *If  $P$  is a CGKE and MAGKE protocol then  $P$  is verifiable contributory in the sense of [2, 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 all honest users accept the same session group key. Since  $P$  is also a CGKE protocol and, therefore, contributory the accepted group key is contributed to by each honest user.

## 5 Our Compiler for MA-Security and Contributiveness

In this section we propose a compiler which can be used to turn any AKE-secure GKE protocol into a GKE protocol which is additionally MA-secure and provides contributiveness<sup>3</sup>.

Our compiler denoted **C-MACON** can be seen as an extension of the compiler in [33] that according to our model satisfies the requirement of MA-security<sup>4</sup> but not of contributiveness. If  $P$  is a GKE protocol, by **C-MACON<sub>P</sub>** we denote the compiled protocol.

In the following, we assume that each message sent by  $\Pi_U^s$  can be parsed as  $U|m$  consisting of the sender's identity  $U$  and a message  $m$ . Additionally, an authentication token  $\sigma$ , e.g., a digital signature on  $m$ , can be attached. Our compiler is formally described in Definition 8: it is based on a one-way permutation  $\pi$ , a collision-resistant pseudo-random function ensemble  $F$ , and an existentially unforgeable digital signature  $\Sigma$  (for completeness, we provide more details on these well-known primitives in Appendix D). The description is given from the perspective of one particular operation execution (session). Therefore, by  $\Pi_i^s \in \mathcal{G}$  we consider the  $i$ -th oracle in  $\mathcal{G}$  assuming that there exists an index  $j \in [1, N]$  such that  $U_j$  owns  $\Pi_i^s$ . Similar, by  $sk'_i$  and  $pk'_i$  (resp.,  $sk_i$  and  $pk_i$ ) we denote the private and public keys of  $U_j$  used in the compiled protocol (resp., in the underlying protocol).

**Main Ideas** After computing the session group key  $k$  in the underlying protocol  $P$  participants execute **C-MACON**. In its first communication round they exchange randomly chosen nonces  $r_i$  that are then concatenated into a session id **sid** (this is a classical way to define unique session ids). Then, each participant iteratively computes values  $\rho_1, \dots, \rho_n$  by adequately using the pseudo-random function  $f$ , in such a way that every random nonce (contribution of each participant) is embedded into the computation of  $K := \rho_n$ . The intuition is that malicious participant cannot influence this computation. The second communication round of **C-MACON** is used to ensure key confirmation. For this purpose we apply the same technique as in [33], i.e., every participant computes a key confirmatory token  $\mu_i = f_K(v_1)$  using a public input value  $v_1$ , signs it and sends it to other participants. After verifying signatures each party accepts with the session group key  $\mathbf{K} = f_K(v_2)$  with public input value  $v_2 \neq v_1$ . All intermediate values are then erased.

**Definition 8 (Compiler C-MACON).** *Let  $P$  be a GKE protocol from Definition 3,  $\pi : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$  a permutation,  $F := \{f_k\}_{k \in \{0, 1\}^\kappa}$ ,  $\kappa \in \mathbb{N}$  a function ensemble with domain and range  $\{0, 1\}^\kappa$ , and  $\Sigma := (\text{Gen}, \text{Sign}, \text{Verify})$  a digital signature scheme. A compiler for MA-security and  $n$ -contributiveness, denoted **C-MACON<sub>P</sub>**, 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  $\Sigma.\text{Gen}(1^\kappa)$ . This is in addition to any key pair  $(sk_i, pk_i)$  used in  $P$ .*

**MACON:** *After an oracle  $\Pi_i^s$  computes  $k_i^s$  in the execution of  $P$  it proceeds as follows.*

**Round 1:** *It chooses a random MACON nonce  $r_i \in_R \{0, 1\}^\kappa$  and sends  $U_i|r_i$  to every oracle  $\Pi_j^s$  with  $U_j \in \text{pid}_i^s$ . After  $\Pi_i^s$  receives  $U_j|r_j$  from  $\Pi_j^s$  with  $U_j \in \text{pid}_i^s$  it checks whether  $|r_j| \stackrel{?}{=} \kappa$ . If this verification fails then  $\Pi_i^s$  terminates without accepting;*

<sup>3</sup> For completeness we mention that with the compiler proposed by Katz and Yung in [34] there exists a generic solution which can be used to achieve AKE-security for any GKE protocol which is secure against passive attacks.

<sup>4</sup> The proof of this statement can be directly derived from the proof of MA-security of our compiler (Theorem 2).

**Round 2:** 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  $l \in \{2, \dots, n\}$  where  $v_0$  is a public value. Then, it defines the intermediate key  $K_i^s := \rho_n$  and  $\text{sid}_i^s := r_1 | \dots | r_n$  and computes a MACON token  $\mu_i := f_{K_i^s}(v_1)$  where  $v_1$  is a public value, together with a signature  $\sigma_i := \Sigma.\text{Sign}(sk'_i, \mu_i | \text{sid}_i^s | \text{pid}_i^s)$ . Then, it sends  $U_i | \sigma_i$  to every oracle  $\Pi_j^s$  with  $U_j \in \text{pid}_i^s$  and every other private information from  $\text{state}_i^s$  (including  $k_i^s$  and each  $\rho_l, l \in [1, n]$ ). After  $\Pi_i^s$  receives  $U_j | \sigma_j$  from  $\Pi_j^s$  with  $U_j \in \text{pid}_i^s$  it checks whether  $\Sigma.\text{Verify}(pk'_j, \mu_i | \text{sid}_i^s | \text{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$ . If this verification fails then  $\Pi_i^s$  terminates without accepting; otherwise it accepts with the session group key  $\mathbf{K}_i^s := f_{K_i^s}(v_2)$  where  $v_2 \neq v_1$  is another public value, and erases every other private information from  $\text{state}_i^s$  (including  $K_i^s$ ).

Note that C-MACON can be considered as an add-on protocol that should be executed after the execution of P. Moreover, with the MACON nonces we achieve not only the uniqueness of session ids but also the randomization and contributiveness (via successive evaluations of  $f$ ) for the intermediate value  $K$ , for the key confirmatory MACON tokens (as in [33]) and for the derived resulting session group key  $\mathbf{K}$ .

## 5.1 Complexity of C-MACON

Obviously, C-MACON requires two communication rounds. This is similar to the KS compiler [33] in case that no session ids are predefined and have to be negotiated first. Each participant must generate one digital signature and verify  $n$  signatures where  $n$  is the total number of session participants. This is also similar to the KS compiler. C-MACON achieves contributiveness at an additional cost of  $n$  executions of the one-way permutation  $\pi$  and  $n$  executions of the pseudo-random function  $f$  per participant.<sup>5</sup>

## 5.2 Security Analysis

Let P be a GKE protocol from Definition 3. For this analysis we require  $\Sigma$  to be *existentially unforgeable under chosen message attacks* (EUF-CMA) according to Definition 13,  $\pi$  to be *one-way* as in Definition 9, and  $F$  to be *collision-resistant pseudo-random* according to Definitions 10 and 11.

Recall that we assume ephemeral secret information being independent of the long-lived key; that is,  $\text{state}_U^s$  may contain ephemeral secrets used in P, the session key  $k_U^s$  computed in P, and  $\rho_1, \dots, \rho_n$  together with some (implementation specific) temporary variables used to compute these values. Note that  $\text{state}_U^s$  is erased at the end of the protocol. By contrast, temporary data used by  $\Sigma.\text{Sign}(sk'_U, m)$  usually depends on the long-lived key and thus should be executed under the same protection mechanism as  $sk'_U$ , e.g., in a smart card [13]<sup>6</sup>. Let  $q_s$  be the total number of executed protocol sessions during the attack.

The following theorem (whose proof appears in Appendix A) shows that C-MACON<sub>P</sub> preserves the AKE-security with strong forward secrecy of the underlying protocol P.

**Theorem 1 (AKE-Security of C-MACON<sub>P</sub>).** *For any AGKE-sfs protocol P if  $\Sigma$  is EUF-CMA and F is pseudo-random then C-MACON<sub>P</sub> is also a AGKE-sfs protocol, and*

$$\text{Adv}_{\text{sfs, C-MACON}_P}^{\text{ake}}(\kappa) \leq 2N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s \text{Adv}_{\text{sfs, P}}^{\text{ake}}(\kappa) + 2(N+2)q_s \text{Adv}_F^{\text{prf}}(\kappa).$$

The following theorems (whose proofs appear in Appendix B and C respectively) concern the MA-security and the contributiveness of C-MACON<sub>P</sub> in the presence of malicious participants and strong corruptions.

<sup>5</sup> Note that costs of XOR operations are usually omitted in the complexity analysis if public-key cryptography operations are present. Note also that pseudo-random functions can be realized using techniques of the symmetric cryptography massively reducing the required computational effort.

<sup>6</sup> Smart cards have limited resources. However, in C-MACON each  $\Pi_U^s$  has to generate only one signature.

**Theorem 2 (MA-Security of C-MACON<sub>P</sub>).** *For any GKE protocol P if  $\Sigma$  is EUF-CMA and F is collision-resistant then C-MACON<sub>P</sub> is MAGKE, and*

$$\text{Succ}_{\text{C-MACON}_P}^{\text{ma}}(\kappa) \leq N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s \text{Succ}_F^{\text{coll}}(\kappa).$$

**Theorem 3 (Contributiveness of C-MACON<sub>P</sub>).** *For any GKE protocol P if  $\pi$  is one-way and F is collision-resistant pseudo-random then C-MACON<sub>P</sub> is CGKE, and*

$$\text{Succ}_{\text{C-MACON}_P}^{\text{con}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N + 2)q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + Nq_s \text{Succ}_{\pi}^{\text{ow}}(\kappa).$$

*Remark 1.* Note that the contributiveness of C-MACON<sub>P</sub> depends neither on AKE-security of P nor on the security of the digital signature scheme  $\Sigma$ . Hence our compiler can also be used for unauthenticated GKE protocols by omitting digital signatures of exchanged messages. However, in this case it would guarantee only contributiveness but not MA-security in the presence of malicious participants. The latter can be only guaranteed using digital signatures (as also noticed in [33] for their definition of security against insider attacks). Note also that C-MACON<sub>P</sub> provides contributiveness in some even stronger sense than required in Definition 7 (for details we refer to Remark 3 in Appendix C).

## 6 Conclusion

In this paper we have addressed the main difference in the trust relationship between participants of group key exchange (GKE) and those of group key transport (GKT) protocols, namely, the question of key control and contributiveness. This has been done from the perspective of malicious participants and powerful adversaries who are able to reveal the internal memory of honest participants. The proposed security model based on the extension of the well-known notion of AKE-security with strong forward secrecy from [13] towards additional requirements of MA-security and contributiveness seems to be stronger than the previous models for group key exchange protocols that address similar issues. The described compiler C-MACON satisfies these additional security requirements and extends the list of currently known compilers for GKE protocols, i.e., the compiler for AKE-security by Katz and Yung [34] and the compiler for security against “insider attacks” by Katz and Shin [33] (that according to our model provides MA-security but not contributiveness). Finally, group key exchange protocols that satisfy our stronger interpretation of key control and contributiveness also provide resilience in the following (weaker) cases: (i) where participants do not have intentions to control the value of the group key, e.g., do not know that their source of randomness is biased (as in [12]), and (ii) where the adversary is given access only to the weak corruptions (as in [9]).

## References

1. 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.
2. 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.
3. 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.
4. R. Barua, R. Dutta, and P. Sarker. 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.
5. 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.

6. 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.
7. 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.
8. 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.
9. 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/>.
10. C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003. ISBN:3-540-43107-1.
11. 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.
12. 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. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.
23. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.
24. 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.
25. 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.
26. 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.
27. 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.
28. O. Goldreich. *Foundations of Cryptography - Basic Tools*, volume 1. Cambridge University Press, 2001. ISBN:0-521-79172-3.

29. 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.
30. C. G. Günther. An Identity-Based Key-Exchange Protocol. In *Advances in Cryptology – EURO-CRYPT’89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer, 1990.
31. I. Ingemarsson, D. T. Tang, and C. K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–719, 1982.
32. 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.
33. 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.
34. 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.
35. 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.
36. 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.
37. Y. Kim, A. Perrig, and G. Tsudik. Group Key Agreement Efficient in Communication. *IEEE Transactions on Computers*, 53(7):905–921, July 2004.
38. 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.
39. H. Krawczyk. HMAC: A High-Performance Secure Diffie-Hellman Protocol. *CRYPTO’05*, 546–566.
40. M. Manulis. Security-Focused Survey on Group Key Exchange Protocols. Technical Report 2006/03, Horst-Görtz Institute, Network and Data Security Group, November 2006. also available at <http://eprint.iacr.org/2006/395>.
41. M. Manulis. Survey on Security Requirements and Models for Group Key Exchange. Technical Report 2006/02, Horst-Görtz Institute, Network and Data Security Group, November 2006. also available at <http://eprint.iacr.org/2006/388>.
42. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996. ISBN:0-8493-8523-7.
43. C. J. Mitchell, M. Ward, and P. Wilson. Key Control in Key Agreement Protocols. *Electronic Letters*, 34(10):980–981, 1998.
44. O. Pereira and J.-J. Quisquater. Some Attacks upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 11(4):555–580, 2003.
45. 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.
46. 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/>.
47. 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>.
48. 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.
49. M. Steiner. *Secure Group Key Agreement*. PhD thesis, Saarland University, March 2002.
50. 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.
51. Y. Yacobi and Z. Shmueli. On Key Distribution Systems. In *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 344–355. Springer, August 1990.

## A Proof of Theorem 1

In our proofs we use a well-known proving technique called *sequence of games* [47] (see also Appendix C.1) which allows to reduce complexity of “reductionist” security proofs for complex cryptographic protocols, and became meanwhile standard for security proofs of group key exchange protocols, e.g., [13, 14, 35, 26, 25, 1].

**Theorem 1 (AKE-Security)** For any AGKE-sfs protocol  $P$  if  $\Sigma$  is EUF-CMA and  $F$  is pseudo-random then  $\text{C-MACON}_P$  is also a AGKE-sfs protocol, and

$$\text{Adv}_{\text{sfs}, \text{C-MACON}_P}^{\text{ake}}(\kappa) \leq 2N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s \text{Adv}_{\text{sfs}, P}^{\text{ake}}(\kappa) + 2(N+2)q_s \text{Adv}_F^{\text{prf}}(\kappa).$$

*Proof.* We define a sequence of games  $\mathbf{G}_i$ ,  $i = 0, \dots, 7$  and corresponding events  $\text{Win}_i^{\text{ake}}$  as the events that the output bit  $b'$  of  $\mathbf{G}_i$  is identical to the randomly chosen bit  $b$ . The queries made by the adversary  $\mathcal{A}$  are answered by a simulator  $\Delta$ . 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  $\text{Win}_i^{\text{ake}}$  and  $\text{Win}_{i+1}^{\text{ake}}$ .

**Game  $\mathbf{G}_0$ .** This game is the real game  $\text{Game}_{\text{sfs}, \text{C-MACON}_P}^{\text{ake-b}}(\kappa)$  played between a simulator  $\Delta$  and  $\mathcal{A}$ . We assume that the *Test* query is asked to a fresh oracle  $\Pi_i^s$ . 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  $\Pi_i^s$ .

**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  $\sigma$  is a valid signature on  $m$  that has not been previously output by an honest oracle  $\Pi_i^s$  before querying *Corrupt*( $U_i$ ). In other words the simulation fails if  $\mathcal{A}$  outputs a successful forgery; we denote this event as *Forge*.

In order to estimate  $\Pr[\text{Forge}]$  we show that using  $\mathcal{A}$  we can construct an EUF-CMA forger  $\mathcal{F}$  against the signature scheme  $\Sigma$  as follows.  $\mathcal{F}$  is given a public key  $pk$  and has access to the corresponding signing oracle. During the initialization of  $\text{C-MACON}_P$ ,  $\mathcal{F}$  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  $\Sigma.\text{Gen}(1^\kappa)$ .  $\mathcal{F}$  generates also all key pairs  $(sk_i, pk_i)$  with  $U_i \in \mathcal{U}$  if any are needed for the original execution of  $P$ . The forger simulates all queries of  $\mathcal{A}$  in a natural way by executing  $\text{C-MACON}_P$  by itself, and by obtaining the necessary signatures with respect to  $pk_{i^*}'$  from its signing oracle. This is a perfect simulation for  $\mathcal{A}$ : by assumption, no *Corrupt*( $U_{i^*}$ ) occurs—forger  $\mathcal{F}$  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  $\Pr[\text{Forge}]/N$ . This implies

$$|\Pr[\text{Win}_1^{\text{ake}}] - \Pr[\text{Win}_0^{\text{ake}}]| \leq \Pr[\text{Forge}] \leq N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa). \quad (1)$$

**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  $\Pi_i^s$  in two different sessions. If  $q_s$  is the total number of protocol sessions, the probability that a randomly chosen MACON nonce  $r_i$  appears twice is  $q_s^2/2^\kappa$  for one uncorrupted user. Since there are at most  $N$  users we obtain

$$|\Pr[\text{Win}_2^{\text{ake}}] - \Pr[\text{Win}_1^{\text{ake}}]| \leq \frac{Nq_s^2}{2^\kappa} \quad (2)$$

Note that this game excludes replay attacks in the MACON protocol because  $\text{sid}_i^s$  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:  $\Delta$  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. Let  $\mathbf{Q}$  be the event that this guess is correct. Obviously,  $\Pr[\mathbf{Q}] = 1/q_s$ . Thus, we get

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

This implies

$$\Pr[\text{Win}_2^{\text{ake}}] = q_s \left( \Pr[\text{Win}_3^{\text{ake}}] - \frac{1}{2} \right) + \frac{1}{2}. \quad (3)$$

**Game G<sub>4</sub>.** In this game we consider the simulator  $\Delta$  as an active adversary against the AKE-security of the underlying protocol  $\mathsf{P}$  that participates in  $\text{Game}_{\text{sf},\mathsf{P}}^{\text{ake}-1}(\kappa)$ , i.e., the *Test* query of  $\Delta$  to an accepted fresh oracle  $\Pi_i^s$  is answered with the real session group key  $k_i^s$  computed in  $\mathsf{P}$ . In the following we show how  $\Delta$  answers the queries of  $\mathcal{A}$ . Note that  $\Delta$  and  $\mathcal{A}$  operate in the same adversarial model, i.e., are both active, have access to the same queries, and must ask own *Test* query to a fresh oracle. In fact, we never require  $\Delta$  to execute operations of  $\mathsf{P}$  itself but to forward each related query of  $\mathcal{A}$  as its own query and respond accordingly.  $\Delta$  itself performs only those additional computations that are necessary for the additional rounds of  $\mathsf{C-MACON}$ .

The simulator  $\Delta$  which is initialized with the public keys  $\{pk'_i\}_{U_i \in \mathcal{U}}$  (if any are given in the original protocol  $\mathsf{P}$ ) generates all key pairs  $(sk'_i, pk'_i)$  honestly using  $\Sigma.\text{Gen}(1^\kappa)$ , and provides the active adversary  $\mathcal{A}$  with the set of the public keys  $\{pk'_i, pk_i\}_{U_i \in \mathcal{U}}$ . Then,  $\Delta$  runs  $\mathcal{A}$  as a subroutine and answers its queries.

*Execute queries:* If  $\mathcal{A}$  invokes a protocol session via a *Execute*( $\mathcal{S}$ ) query then  $\Delta$  forwards this query as its own and obtains the transcript  $T$  of the  $\mathsf{P.Setup}(\mathcal{S})$  execution. The goal of  $\Delta$  is to extend  $T$  to a transcript  $T'$  for the corresponding execution of  $\mathsf{C-MACON}_\mathsf{P.Setup}(\mathcal{S})$ . Therefore,  $\Delta$  chooses random nonces  $r_i$  for each  $\Pi_i^s$  in  $\mathcal{G}$  which is composed of the ordered oracles in  $\mathcal{S}$  and computes  $\text{sid} := r_1 | \dots | r_n$ . In order to build  $T'$  the simulator, first, appends  $\{U_i | r_i\}_{1 \leq i \leq n}$  to  $T$ . Next, if the invoked session is the  $q_s^*$ -th session then  $\Delta$  asks own *Test* query to any oracle activated via the *Execute* query and obtains real  $k$ . Otherwise, if the session is not the  $q_s^*$ -th session then  $\Delta$  asks own *RevealKey* query to any of the mentioned oracles and obtains real  $k$ . Hence, in any case  $\Delta$  knows real  $k$  which it then uses to compute the sequence  $\rho_1, \dots, \rho_n$  (note that  $K = \rho_n$ ) and the MACON token  $\mu := f_K(v_1)$ . Then,  $\Delta$  computes  $\sigma_i := \Sigma.\text{Sign}(sk'_i, \mu | \text{sid} | \text{pid})$  and appends messages of the form  $\{U_i | \sigma_i\}_{1 \leq i \leq n}$  to  $T'$ . Then,  $\Delta$  computes  $\mathbf{K} := f_K(v_2)$  and gives  $T'$  to  $\mathcal{A}$ .

*Send queries:* All *Send* queries of  $\mathcal{A}$  related to the execution of the underlying protocol  $\mathsf{P}$  are forwarded by  $\Delta$  as its own queries and replied with whatever  $\Delta$  obtains. Note that the execution of the MACON protocol starts after  $\Pi_i^s$  computed  $k_i^s$  in  $\mathsf{P}$ . By *Send<sub>F</sub>* to an oracle  $\Pi_i^s$  we define the *final Send* query of  $\mathcal{A}$  concerning the execution of  $\mathsf{P.Setup}$ , which results in  $\Pi_i^s$  having computed  $k_i^s$ . This means that all further valid *Send* queries to  $\Pi_i^s$  would be related to the additional communication rounds of the MACON protocol. Having received *Send<sub>F</sub>* for an oracle  $\Pi_i^s$  the simulator forwards it as its own query implying the computation of  $k_i^s$  by  $\Pi_i^s$ . Then,  $\Delta$  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,  $\Delta$  chooses  $r_i \in_R \{0, 1\}^\kappa$  and responds with  $U_i | r_i$ .

By *Send<sub>F+</sub>* we denote the *Send* query which includes messages of the form  $U_j | r_j$  for each  $U_j \neq i$  who holds an oracle in the current group  $\mathcal{G}$ . This query has the form  $\text{Send}(\Pi_i^s, (U_1 | r_1) | \dots | (U_n | r_n))$  where  $n$  is the number of operation participants and  $(U_i | r_i)$  is not part of the query message. Note that  $\Pi_i^s$  must have computed  $k_i^s$  prior to the *Send<sub>F+</sub>* query; otherwise the query is unexpected. Upon receiving a (valid) *Send<sub>F+</sub>* query for  $\Pi_i^s$  the simulator computes the sequence  $\rho_1, \dots, \rho_n$ , defines  $K_i^s := \rho_n$ , computes the MACON token  $\mu_i := f_{K_i^s}(v_1)$ , the signature  $\sigma_i := \Sigma.\text{Sign}(sk'_i, \mu_i | \text{sid}_i^s | \text{pid}_i^s)$  and responds with  $U_i | \sigma_i$  to  $\mathcal{A}$ .

The last valid *Send* query to  $\Pi_i^s$  in the execution of  $\mathsf{C-MACON}_\mathsf{P}$  has the form  $\text{Send}(\Pi_i^s, (U_1 | \sigma_1) | \dots | (U_n | \sigma_n))$  whereby  $(U_i | \sigma_i)$  is not part of the query message. Upon receiving this query  $\Delta$  verifies all included signatures. If all signatures are valid  $\Delta$  computes  $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ .

*Corrupt queries:* When  $\mathcal{A}$  asks a *Corrupt*( $U_i$ ) query  $\Delta$  forwards own *Corrupt*( $U_i$ ) query to obtain  $sk_i$  and replies with  $(sk_i, sk'_i)$ . Note that  $\Delta$  and  $\mathcal{A}$  have identical restrictions concerning the *Corrupt* queries.

*RevealState queries:* When  $\mathcal{A}$  asks a *RevealState*( $\Pi_i^s$ ) query  $\Delta$  forwards own *RevealState*( $\Pi_i^s$ ) query to obtain  $\text{state}_i^s$ . If  $\Pi_i^s$  is waiting for the last *Send* query of the form  $\text{Send}(\Pi_i^s, (U_1 | \sigma_1) | \dots | (U_n | \sigma_n))$  then  $\Delta$  inserts  $K_i^s$  (which is not treated as erased yet) into  $\text{state}_i^s$  and returns it to  $\mathcal{A}$ ; otherwise it simply forwards  $\text{state}_i^s$  to  $\mathcal{A}$ . Note that  $\Delta$  and  $\mathcal{A}$  have identical restrictions concerning the *RevealState* queries.

*RevealKey queries:* When  $\mathcal{A}$  asks a *RevealKey*( $\Pi_i^s$ ) query  $\Delta$  checks that  $\Pi_i^s$  has accepted; otherwise an empty string is returned. Then,  $\Delta$  returns the session group key  $\mathbf{K}_i^s$ . Note that  $\Delta$  is always able to do this since it executes the last steps of the MACON protocol itself, i.e., the computation of  $\rho_1, \dots, \rho_n$ ,  $\mu_i$ , and  $\mathbf{K}_i^s$  for every (honest) oracle  $\Pi_i^s$ .

*Test query:* Note that in this game we are dealing with the *Test* query asked to an oracle  $\Pi_i^s$  that has participated in the  $q_s^*$ -th session. The simulator  $\Delta$  already knows  $\mathbf{K}_i^s$  since this value is computed by  $\Delta$  for every (honest) oracle  $\Pi_i^s$  in the simulation. Thus,  $\Delta$  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  $b = 0$ .

We claim that this is a perfect simulation for  $\mathcal{A}$ . Since  $\Delta$  uses the real  $k_i^s$  to derive  $\mathbf{K}_i^s$  we can consider this game as a “bridging step” so that

$$\Pr[\text{Win}_4^{\text{ake}}] = \Pr[\text{Win}_3^{\text{ake}}]. \quad (4)$$

**Game  $\mathbf{G}_5$ .** In this game we consider the simulator  $\Delta$  as an active adversary against the AKE-security of the underlying protocol  $\mathsf{P}$  that participates in  $\text{Game}_{\text{sfs}, \mathsf{P}}^{\text{ake}-0}(\kappa)$ , i.e., the *Test* query of  $\Delta$  to an accepted fresh oracle  $\Pi_i^s$  is answered with a random bit string instead of the real key  $k_i^s$ .  $\Delta$  answers all queries of  $\mathcal{A}$  exactly as described in Game  $\mathbf{G}_4$ . Note that the only difference is that in this game  $\rho_1$  (which is in turn used to compute  $\rho_2, \dots, \rho_n, K_i^s$ , and  $\mathbf{K}_i^s$ ) is computed using a random string a not the real intermediate key  $k_i^s$  as in the previous game. By a “hybrid argument” we obtain

$$|\Pr[\text{Win}_5^{\text{ake}}] - \Pr[\text{Win}_4^{\text{ake}}]| \leq \text{Adv}_{\text{sfs}, \mathsf{P}}^{\text{ake}}(\kappa). \quad (5)$$

**Game  $\mathbf{G}_6$ .** This game is identical to Game  $\mathbf{G}_5$  except that in the  $q_s^*$ -th session each  $\rho_i$ ,  $i = 1, \dots, n$  is replaced by a random value sampled from  $\{0, 1\}^\kappa$ . Notice, this implies that  $K_i^s$  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  $\mathbf{G}'_{6,l}$ ,  $l = 1, \dots, n+1$  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  $\rho_i$ ,  $i \leq l$ , 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  $\rho_i$ ,  $i = 1, \dots, n-1$  by a random bit string implies uniform distribution of the PRF key  $\rho_i \oplus \pi(r_{i+1})$  used in the computation of  $\rho_{i+1}$ , and that  $k_i^s$  used to compute  $\rho_1$  is already uniform according to Game  $\mathbf{G}_5$ .

Since  $n \leq N$  we get

$$|\Pr[\text{Win}_6^{\text{ake}}] - \Pr[\text{Win}_5^{\text{ake}}]| \leq N \text{Adv}_F^{\text{prf}}(\kappa). \quad (6)$$

**Game  $\mathbf{G}_7$ .** This game is identical to Game  $\mathbf{G}_6$  except that in the  $q_s^*$ -th session  $\mathbf{K}_i^s$  and the MACON token  $\mu_i$  are replaced by random values sampled from  $\{0, 1\}^\kappa$  (note that same values must be chosen for all participants of the  $q_s^*$ -th session). Recall that  $K_i^s$  used to compute  $\mathbf{K}_i^s$  and  $\mu_i$  is uniform according to Game  $\mathbf{G}_6$ . Notice that this implies that  $\mathbf{K}_i^s$  is uniformly distributed in this game. Obviously,

$$|\Pr[\text{Win}_7^{\text{ake}}] - \Pr[\text{Win}_6^{\text{ake}}]| \leq 2 \text{Adv}_F^{\text{prf}}(\kappa). \quad (7)$$

Since  $\mathbf{K}_i^s$  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.,

$$\Pr[\text{Win}_7^{\text{ake}}] = \frac{1}{2} \quad (8)$$

Considering Equations 1 to 8 we get

$$\begin{aligned} \Pr[\text{Game}_{\text{sfs}, \mathsf{C-MACON}_p}^{\text{ake}-b}(\kappa) = b] &= \Pr[\text{Win}_0^{\text{ake}}] \\ &= N \text{Succ}_\Sigma^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s \left( \Pr[\text{Win}_3^{\text{ake}}] - \frac{1}{2} \right) + \frac{1}{2} \\ &\leq N \text{Succ}_\Sigma^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s \text{Adv}_{\text{sfs}, \mathsf{P}}^{\text{ake}}(\kappa) + (N+2)q_s \text{Adv}_F^{\text{prf}}(\kappa) + \frac{1}{2}. \end{aligned}$$

This results in the desired inequality

$$\text{Adv}_{\text{sfs}, \mathsf{C-MACON}_p}^{\text{ake}}(\kappa) \leq 2N \text{Succ}_\Sigma^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s \text{Adv}_{\text{sfs}, \mathsf{P}}^{\text{ake}}(\kappa) + 2(N+2)q_s \text{Adv}_F^{\text{prf}}(\kappa).$$

Note that the negligibility of  $\text{Adv}_{\text{sfs}, \mathsf{C-MACON}_p}^{\text{ake}}(\kappa)$  follows from the negligibility of the arguments at the right hand side of the inequality based on the assumptions that  $\Sigma$  is existentially unforgeable under chosen message attacks (EUF-CMA),  $F$  is pseudo-random and  $\mathsf{P}$  is AGKE-sfs.  $\square$

## B Proof of Theorem 2

**Theorem 2 (MA-Security)** *For any GKE protocol  $P$  if  $\Sigma$  is EUF-CMA and  $F$  is collision-resistant then  $\text{C-MACON}_P$  is MAGKE, and*

$$\text{Succ}_{\text{C-MACON}_P}^{\text{ma}}(\kappa) \leq N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s \text{Succ}_F^{\text{coll}}(\kappa).$$

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

**Game  $\mathbf{G}_0$ .** This game is the real game  $\text{Game}_{\text{C-MACON}_P}^{\text{ma}}(\kappa)$  played between  $\Delta$  and  $\mathcal{A}$ . Note that the goal of  $\mathcal{A}$  is to achieve that there exists an uncorrupted user  $U_i$  whose corresponding oracle  $\Pi_i^s$  accepts with  $\mathbf{K}_i^s$  and another user  $U_j \in \text{pid}_i^s$  that is uncorrupted at the time  $\Pi_i^s$  accepts and either does not have a corresponding oracle  $\Pi_j^s$  with  $(\text{pid}_j^s, \text{sid}_j^s) = (\text{pid}_i^s, \text{sid}_i^s)$  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  $U_i|\sigma$  such that  $\sigma$  is a valid signature that has not been previously output by an oracle  $\Pi_i^s$  before querying  $\text{Corrupt}(U_i)$ , i.e., the simulation fails if  $\mathcal{A}$  outputs a successful forgery (event *Forge* occurs). Similar to Game  $\mathbf{G}_1$  from the proof of Theorem 1 (Equation 1) we have

$$|\Pr[\text{Win}_1^{\text{ma}}] - \Pr[\text{Win}_0^{\text{ma}}]| \leq N \text{Succ}_{\mathcal{F}, \Sigma}^{\text{euf-cma}}(\kappa). \quad (9)$$

**Game  $\mathbf{G}_2$ .** This game is identical to Game  $\mathbf{G}_1$  except that the simulation aborts if a MACON nonce  $r_i$  is used by any uncorrupted user  $U_i$  in two different sessions. Similar to Game  $\mathbf{G}_2$  from the proof of Theorem 1 (Equation 2) we obtain

$$|\Pr[\text{Win}_2^{\text{ma}}] - \Pr[\text{Win}_1^{\text{ma}}]| \leq \frac{Nq_s^2}{2^\kappa}. \quad (10)$$

This game implies that  $\text{sid}_i^s$  computed by any uncorrupted user's oracle  $\Pi_i^s$  is unique for each new session. Note that  $\text{sid}_i^s$  is used to generate the signature  $\sigma_i$  in the MACON protocol of the compiler.

Hence, this prevents attacks where  $\Pi_i^s$  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  $\Pi_i^s$  does not accept unless it successfully verifies  $\sigma_j$  for all  $U_j \in \text{pid}_i^s$  in the MACON protocol of C-MACON. Having excluded forgeries and replay attacks we follow that for every user  $U_j \in \text{pid}_i^s$  that is uncorrupted at the time  $\Pi_i^s$  accepts there exists a corresponding instance oracle  $\Pi_j^s$  with  $(\text{pid}_j^s, \text{sid}_j^s) = (\text{pid}_i^s, \text{sid}_i^s)$ . Thus, according to Definition 6  $\mathcal{A}$  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,  $\Pi_i^s$  and  $\Pi_j^s$  have accepted with  $\mathbf{K}_i^s = f_{K_i^s}(v_2)$  resp.  $\mathbf{K}_j^s = f_{K_j^s}(v_2)$  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  $\Pi_i^s$  and  $\Pi_j^s$  have been delivered without any modification. In particular, oracle  $\Pi_i^s$  received the signature  $\sigma_j$  computed on  $\mu_j = f_{K_j^s}(v_1)$  and  $\Pi_j^s$  received the signature  $\sigma_i$  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

$$\begin{aligned} \Pr[\mathbf{K}_i^s \neq \mathbf{K}_j^s \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] = \\ \Pr[f_{K_i^s}(v_2) \neq f_{K_j^s}(v_2) \wedge f_{K_i^s}(v_1) = f_{K_j^s}(v_1)] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa). \end{aligned}$$

Hence,

$$|\Pr[\text{Win}_2^{\text{ma}}] - \Pr[\text{Win}_1^{\text{ma}}]| \leq q_s \text{Succ}_F^{\text{coll}}(\kappa). \quad (11)$$

Considering Equations 9 and 11 we get the desired inequality

$$\begin{aligned} \text{Succ}_{\text{C-MACON}_P}^{\text{ma}}(\kappa) &= \Pr[\text{Win}_0^{\text{ma}}] \\ &\leq N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s \text{Succ}_F^{\text{coll}}(\kappa). \end{aligned}$$

Note that the negligibility of  $\text{Succ}_{\mathcal{G}\text{-MACCONP}}^{\text{ma}}(\kappa)$  follows from the negligibility of the arguments at the right hand side of the inequality based on the assumption that  $F$  is collision-resistant and  $\Sigma$  is existentially unforgeable under chosen message attacks (EUF-CMA).  $\square$

## C Proof of Theorem 3

### C.1 Transitions based on “Condition Events” in the “Sequence of Games” Technique

In the following proof we apply a new transition technique within the framework of “sequence of games” [47]. In particular we extend this framework with an additional kind of transitions between games: we call them *transitions based on “condition events”*. Recall that the classical framework considers the construction of a sequence of games  $\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_n$  starting with the original game between the adversarial algorithm  $\mathcal{A}$  and its environment (typically a simulator). In the security proof, one is interested in a particular event, whose occurrence in each game  $\mathbf{G}_i$  is represented by event  $\text{Win}_i$ . The goal of the proof is to relate probabilities of successive events  $\text{Win}_0, \dots, \text{Win}_n$ . In the survey provided in [47], the technique consists of the following three transition types: transitions based on indistinguishability, transitions based on “failure events”, and transitions based on “bridging steps”. These types have been identified based on a large number of available security proofs.

Transitions based on indistinguishability work as follows, assuming two computationally indistinguishable distributions  $D_0$  and  $D_1$ . One constructs games  $\mathbf{G}_{i+1}$  from  $\mathbf{G}_i$  such that if their difference is detected by the adversary then it is possible to distinguish  $D_0$  from  $D_1$  with advantage at least  $|\Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}]|$ . The assumption on  $D_0$  and  $D_1$  ensures that this advantage is negligible.

Transitions based on “failure events” are used to construct  $\mathbf{G}_{i+1}$  from  $\mathbf{G}_i$  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 a classical lemma ensures that  $|\Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}]| \leq \Pr[F]$ .

Transitions based on “bridging steps” are transitions where  $\Pr[\text{Win}_i] = \Pr[\text{Win}_{i+1}]$ . Such transitions can be built by restating the way of computation of certain entities while keeping their resulting values equivalent. They may be useful to prepare transitions of other types.

In the following we describe an additional type of transitions – transitions based on “condition events”. The background is given by the following lemma.

**Lemma 1.** *Let  $A, B, C$  be events defined in some probability distribution, and suppose that  $\Pr[B] = \Pr[A|C]$ . Then*

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

*Proof.* We compute

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

$\square$

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  $\text{Win}_{i+1}$  as the event that  $\text{Win}_i$  occurs given  $C$ . Then according to Lemma 1  $\Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}] \leq \Pr[\neg C]$ . 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  $\neg C$ . 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  $\neg C$ . 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 usually, e.g., in the proofs of [13] and in our proofs of Theorems 1 and 2 (Games  $\mathbf{G}_1$  and  $\mathbf{G}_2$ ), the simulator must be able to detect this event in order to change the process of  $\mathbf{G}_{i+1}$ .

**Theorem 3 (Contributiveness)** For any GKE protocol  $P$  if  $\pi$  is one-way and  $F$  is collision-resistant pseudo-random then  $\text{C-MACON}_P$  is CGKE, and

$$\text{Succ}_{\text{C-MACON}_P}^{\text{con}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N+2)q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + Nq_s \text{Succ}_\pi^{\text{ow}}(\kappa).$$

*Remark 2.* 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  $\mathbf{G}_2$  to  $\mathbf{G}_4$  could be found. The reason is that from the perspective of the adversary all values are known, especially all PRF keys used to compute  $\rho_l$ ,  $l = 1, \dots, n$ . 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. Note that in [9] 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.* In the following we consider an adversary  $\mathcal{A}$  from Definition 7 who wins in  $\text{Game}_{\text{C-MACON}_P}^{\text{con}}(\kappa)$  (which event we denote  $\text{Win}^{\text{con}}$ ). Assume that  $\mathcal{A}$  wins in  $\text{Game}_{\text{C-MACON}_P}^{\text{con}}(\kappa)$  (which event we denote  $\text{Win}^{\text{con}}$ ). 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  $\text{Game}_{\text{C-MACON}_P}^{\text{con}}(\kappa)$ , 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 uncorrupted user’s oracle  $\Pi_i^s$  in two different sessions. Similar to Game  $\mathbf{G}_2$  from the proof of Theorem 1 (Equation 2) we have

$$|\Pr[\text{Win}_0^{\text{con}}] - \Pr[\text{Win}_1^{\text{con}}]| \leq \frac{Nq_s^2}{2^\kappa}. \quad (12)$$

**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  $\rho_{i^*}$  computed by  $\Pi_{i^*}^s$  in any session of the **attack** stage.<sup>7</sup> According to Lemma 1 we need to estimate the probability of the opposite event, i.e., that  $\mathcal{A}$  being in the **prepare** stage is able to output  $\rho_{i^*}$ . We consider two cases:  $i^* = 1$  and  $i^* > 1$ . Note that all other oracles except for  $\Pi_{i^*}^s$  can be corrupted. An important observation for our argumentation in this game is that the random nonce  $r_{i^*}$  is chosen by  $\Pi_{i^*}^s$  after the computation of  $k_{i^*}^s$  in  $P$ . In other words, when  $\Pi_{i^*}^s$  chooses  $r_{i^*}$  the key  $k_{i^*}^s$  is already defined and cannot be influenced (changed) any more, thus as soon as the compiler round starts  $k_{i^*}^s$  can be considered as some fixed value.

Case  $i^* = 1$ : In any session of the **attack** stage honest oracle  $\Pi_1^s$  computes  $\rho_1 := f_{k_1^s \oplus \pi(r_1)}(v_0)$ . Intuitively, without knowing the PRF key given by the XOR sum  $k_1^s \oplus \pi(r_1)$  (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.,  $\text{Succ}_F^{\text{coll}}(\kappa) + 1/2^\kappa$ . Thus, we have to discuss the case where  $\mathcal{A}$  chooses  $R_1$  in the **prepare** stage and tries to influence  $\Pi_1^s$  computing exactly this value in some session of the **attack** stage. Note that as mentioned above  $k_1^s$  is some fixed value at the time point when  $\Pi_1^s$  chooses  $r_1$  uniformly at random. Since  $\pi$  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 an honest  $\Pi_{i^*}^s$  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

<sup>7</sup> 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  $\rho_{i^*}$ . Since we are only interested in the probability of the adversarial success under this “condition event” 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.  $\mathbf{K}_{i^*}^s$ .

by a random guess, i.e.,  $\text{Succ}_F^{\text{coll}}(\kappa) + 1/2^\kappa$ . 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  $\pi$  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  $\rho_{i^*-1}$  only after having received  $r_{i^*}$ , that is during the attack-ed session. Since  $U_{i^*}$  is uncorrupted its oracle computes  $\rho_{i^*-1}$  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  $\text{Adv}_F^{\text{prf}}(\kappa)$ . 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

$$\Pr[\text{Win}_1^{\text{con}}] - \Pr[\text{Win}_2^{\text{con}}] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + \frac{q_s}{2^\kappa}. \quad (13)$$

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  $\rho_{i^*}$  computed by  $\Pi_{i^*}^s$  in any session of the attack stage. Note that we do not need to consider the values  $\rho_l$ ,  $l < i^*$  computed by  $\Pi_{i^*}^s$  since in order to compute  $K_{i^*}^s$  every honest oracle must compute the whole sequence  $\rho_1, \dots, \rho_n$ . Thus, it is sufficient to argue that the probability of  $\mathcal{A}$  influencing any  $\rho_l$ ,  $l \geq i^*$ , 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 1 we need to estimate the probability of the opposite event that  $\mathcal{A}$  in the prepare stage is able to output the mentioned value for  $K_{i^*}^s$ .

Based on the ‘‘hybrid technique’’ we define a sequence of auxiliary games  $\mathbf{G}'_{3,l}$ ,  $l = i^*, \dots, n$ . 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  $\rho_l$  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 1 we need to estimate the probability that  $\mathcal{A}$  being in the prepare stage is able to output  $\rho_l$ .

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  $\rho_{i^*+1}$  computed by  $\Pi_{i^*}^s$  in any session of the attack stage.

We argue by intuition. Since  $\Pi_{i^*}^s$  is honest, it computes  $\rho_{i^*+1} := f_{\rho_{i^*} \oplus \pi(r_{i^*+1})}(v_0)$  in the attack stage. 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.,  $\text{Succ}_F^{\text{coll}}(\kappa) + 1/2^\kappa$ . 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  $\rho_{i^*}$  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  $\Pi_{i^*}^s$  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  $\pi$  is one-way this probability is intuitively bound by  $\text{Succ}_\pi^{\text{ow}}(\kappa)$ . Thus,  $\mathcal{A}$  is able to output  $\rho_{i^*+1}$  while being in the prepare stage with the probability of at most  $\text{Succ}_F^{\text{coll}}(\kappa) + \text{Succ}_\pi^{\text{ow}}(\kappa) + 1/2^\kappa$ . Since there are at most  $q_s$  sessions the total probability that  $\mathcal{A}$  is able to do this is at most

$$q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Succ}_\pi^{\text{ow}}(\kappa) + \frac{q_s}{2^\kappa}.$$

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

$$\Pr[\text{Win}_2^{\text{con}}] - \Pr[\text{Win}_3^{\text{con}}] \leq Nq_s \text{Succ}_F^{\text{coll}}(\kappa) + Nq_s \text{Succ}_\pi^{\text{ow}}(\kappa) + \frac{Nq_s}{2^\kappa}. \quad (14)$$

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.,  $\text{Succ}_F^{\text{coll}}(\kappa) + 1/2^\kappa$ . Hence,

$$\Pr[\text{Win}_3^{\text{con}}] - \Pr[\text{Win}_4^{\text{con}}] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa) + \frac{q_s}{2^\kappa}. \quad (15)$$

Obviously, the probability of  $\text{Win}_4^{\text{con}}$  is 0, meaning that the adversary did not output a correct value of  $\tilde{\mathbf{K}}$  in the **prepare** stage.

Considering Equations 12 to 15 we obtain the desired inequality

$$\begin{aligned} \text{Succ}_{\mathbf{C-MACON}_P}^{\text{con}}(\kappa) &= \Pr[\text{Win}_0^{\text{con}}] \\ &\leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N+2)q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + Nq_s \text{Succ}_\pi^{\text{ow}}(\kappa). \end{aligned}$$

Note that the negligibility of  $\text{Succ}_{\mathbf{C-MACON}_P}^{\text{con}}(\kappa)$  follows from the negligibility of the arguments at the right hand side of the inequality based on the assumption that  $F$  is collision-resistant pseudo-random and  $\pi$  is one-way.  $\square$

*Remark 3.* In the proof of the Theorem 3 we consider that  $\mathcal{A}$  may even predict (control) the output of the original protocol  $\mathbf{P}$ , i.e. the key  $k$ . Even though, our compiler still provides contributiveness for the resulting group key  $\mathbf{K}$ . Hence,  $\mathbf{C-MACON}_P$  considered as a complete GKE protocol provides contributiveness in some even stronger sense than required in Definition 7, i.e.  $\mathcal{A}$  may even be allowed to output  $\tilde{\mathbf{K}}$  before the honest participant  $U$  (who is supposed to accept with  $\tilde{\mathbf{K}}$  in  $\text{Game}_{\mathbf{C-MACON}_P}^{\text{con}}(\kappa)$ ) starts with the  $\mathbf{MACON}$  protocol of the compiler, and not necessarily before the execution of the new  $\mathbf{C-MACON}_P$  session.

## D Cryptographic Tools Used by the Compiler $\mathbf{C-MACON}_P$

We first recall the notion of a one-way permutation.

**Definition 9 (One-Way Permutation).** A function  $\pi : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$  is called a one-way permutation if the following three conditions hold:

- there exists an efficient algorithm that on input  $x$  outputs  $\pi(x)$ ;
- $\pi$  is a permutation;
- for every PPT algorithm  $\mathcal{A}$ , the following probability is negligible (in  $\kappa$ ):

$$\text{Succ}_\pi^{\text{ow}}(\kappa) := \Pr[\pi(\mathcal{A}(1^\kappa, \pi(x))) = \pi(x) \mid x \in_R \{0,1\}^\kappa]$$

In the following we briefly describe the notion of pseudo-random functions which we make use of in our compiler  $\mathbf{C-MACON}_P$ . 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 indistinguishable from that of a truly random function with same domain and range. In our compiler we use the primitive of (efficiently computable) generalized pseudo-random function ensembles, defined in the following (see also [28, Definition 3.6.9]).

**Definition 10 ((Efficiently Computable) Pseudo-Random Function Ensemble  $F$ ).** An ensemble of finite functions  $F := \{ \{ f_k : \{0, 1\}^{p(\kappa)} \rightarrow \{0, 1\}^{p(\kappa)} \}_{k \in \{0, 1\}^\kappa} \}_{\kappa \in \mathbb{N}}$  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  $k \in_R \{0, 1\}^\kappa$  and a function  $\tilde{f}$  in the set of all functions with domain and range  $\{0, 1\}^{p(\kappa)}$ . Consider a PPT adversary  $\mathcal{A}$  asking queries of the form  $\text{Tag}(x)$  and participating in one of the following two games:
  - $\text{Game}_F^{\text{prf}-1}(\kappa)$  where a query  $\text{Tag}(x)$  is answered with  $f_k(x)$ ,
  - $\text{Game}_F^{\text{prf}-0}(\kappa)$  where a query  $\text{Tag}(x)$  is answered with  $\tilde{f}(x)$ .

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 (over all adversaries running within time  $\kappa$ ) in winning the game is defined as

$$\text{Adv}_F^{\text{prf}}(\kappa) := |2 \Pr[\text{Game}_{\mathcal{A}, F}^{\text{prf}-b}(\kappa) = b] - 1|.$$

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

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

In other words in the above definition the goal of the adversary  $\mathcal{A}$  is to distinguish whether replies on its  $\text{Tag}$  queries are generated by a pseudo-random function  $f$  or by a truly random function  $\tilde{f}$  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  $\text{Tag}$  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.

*Remark 4.* As noted in [28] there are some significant differences between using PRFs and the Random Oracle Model (ROM) [7]. 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.

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 [33]. The same property has previously been defined in [27] and denoted there as *fixed-value-key-binding* property of a pseudo-random function ensemble. We also refer to [33] for a possible construction based on one-way permutations and for the proof of Lemma 2).

**Definition 11 (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  $\text{Sample}$  such that for all PPT adversaries  $\mathcal{A}$  the following advantage is a negligible function in  $\kappa$ :

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

**Lemma 2** ([33]). *If one-way permutations exist then there exist collision-resistant pseudo-random functions.*

Further, our compiler C-MACON applies digital signatures which must be existentially unforgeable under chosen message attacks [29].

**Definition 12 (Digital Signature Scheme).** *A signature scheme  $\Sigma := (\text{Gen}, \text{Sign}, \text{Verify})$  consists of the following algorithms:*

**Gen:** *A probabilistic algorithm that on input a security parameter  $1^\kappa$  outputs a secret key  $sk$  and a public key  $pk$ .*

**Sign:** *A (possibly probabilistic) algorithm that on input a secret key  $sk$  and a message  $m \in \{0, 1\}^*$  outputs a signature  $\sigma$ .*

**Verify:** *A deterministic algorithm that on input a public key  $pk$ , a message  $m \in \{0, 1\}^*$  and a candidate signature  $\sigma$  outputs 1 or 0, meaning that the signature is valid or not.*

**Definition 13 (EUF-CMA Security).** *A digital signature scheme  $\Sigma := (\text{Gen}, \text{Sign}, \text{Verify})$  from Definition 12 is said to be existentially unforgeable under chosen message attacks (EUF-CMA) if for any PPT algorithm (forger)  $\mathcal{F}$ , given a public key  $pk$  and access to the signing oracle  $\text{Sign}(sk, \cdot)$ , the probability that  $\mathcal{F}$  outputs a pair  $(m, \sigma)$  such that  $\text{Verify}(pk, m, \sigma) = 1$  but  $m$  was never part of a query  $\text{Sign}(sk, m)$  is negligible. By  $\text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa)$  we denote the probability that  $\mathcal{F}$  outputs a successful forgery.*