

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

[Fast abstract*]

Emmanuel Bresson¹ and Mark Manulis²

¹ DCSSI Crypto Lab Paris

emmanuel@bresson.org

² Horst Görtz Institute, Ruhr-University of Bochum, Germany

mark.manulis@nds.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.

1 Introduction

The establishment of group keys is fundamental for a variety of security mechanisms in group applications. Two different classes of protocols can be identified: (*group*) *key transport* (GKT), in which the key is chosen by a single party and transmitted to the other parties via secure channels, and (*group*) *key exchange* (GKE), in which all parties interact in order to compute the key. In GKE protocols, no secure channels are needed and, more important, no party is allowed to choose the key on behalf of the group: in other words, group members do not trust each other.

In the paradigm of *provable security*, security analysis must hold in some formal security model. The first such model for GKE protocols (referred to as the BCPQ model) was introduced by Bresson *et al.* in [8], based on earlier work by Bellare and Rogaway [3], and with subsequent variants and refinements [7, 13]; we refer to [17] for a survey. These models mainly focused on the following two notions: *authenticated key exchange* (AKE) security which requires the indistinguishability of computed group keys from random keys, and *mutual authentication* (MA) which means that any two parties authenticate bilaterally and actually compute the same key.

A number of papers [20, 1, 11, 13] point out that the consideration of dishonest participants (either curious or malicious) is of prime importance in the group setting, because they can have catastrophic effects on the protocol security.

Mitchel *et al.* in [20] 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.* [1] 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 – namely, that in GKE protocols none of the participants is trusted to choose the group key on behalf of other participants.

In addition to usual corruptions where the adversary obtains full control over the parties we also consider *strong corruptions* [21, 22, 7], that is, capabilities of the adversary to reveal internal memory of 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 [7]. 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.

* The extended version of this abstract appears in the Proceedings of the 4th International Conference on Automatic and Trusted Computing (ATC 2007).

Contributions and Organization In this paper we focus on the security of GKE protocols in the presence of malicious participants and strong corruptions. In Sections 2 and 2.1 we first discuss why currently known security models for GKE protocols are not mature enough to deal with such attacks. Then, in Section 3 we extend the notions of AKE- and MA-security and propose a new definition of contributiveness.

To prove the soundness and feasibility of our extensions, in Section 4 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 [8, 7, 14] subsumes some informal security goals defined in the literature: *key secrecy* or *implicit key authentication* [19] which ensures that no party except for legitimate participants learns the established group key, *security against impersonation attacks* or a related notion of *entity authentication* [2] requiring that an adversary must not be able to replace an honest participant in the execution of the protocol; *resistance against known-key attacks* [10] meaning that an adversary knowing group keys of previous sessions cannot compute subsequent session keys, *key independence* [15] 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* [19] 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* [7] in which the adversary in addition to the long-lived keys reveals the internal data of participants.

The currently available formal definition of MA-security in [8] has been designed to cover the informal definitions of *key confirmation* which combined with *mutual authentication* ensures that all identified protocol participants have actually computed the same group key (this is also known as *explicit key authentication* [19]).

According to [11], however, these definitions do not consider security against *unknown key-share attacks* [5], 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 mentioned first in [20]: the authors described the issue of *key control*. Independently, Ateniese *et al.* [1] 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. Some subsequent security models have tried to formalize this approach.

The KS Model Katz and Shin [13] proposed security definitions against malicious participants in a BCPQ-like model: briefly speaking any user U_i may have many instance oracles Π_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*) Π_i^s if U_j belongs to the (expected) partners of Π_i^s , but in fact no oracle Π_j^t is partnered with Π_i^s . In other words, the instance Π_i^s computes the session key and U_i believes that U_j does so, but in fact an adversary has participated in the protocol on behalf of U_j . Then, the authors call a protocol secure against *insider impersonation attacks* if for any party U_j and any instance Π_i^s , the adversary cannot impersonate U_j to Π_i^s , under the (stronger) condition that neither U_j nor U_i is corrupted at the time Π_i^s accepts.

The BVS Model Bohli *et al.* [4] proposed another extension (referred 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 Π_i^s and a subset K in the session key space \mathcal{K} . In the second stage, the adversary tries to make Π_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. However, in their compiler malicious participants are able to predict the resulting value of the group key, that is, the KS model does not deal with the issues of key control and contributiveness.

Absence of Strong Corruptions in the BVS Model A first drawback of the BVS model is that \mathcal{A} is required to commit to Π_i^s 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.

3 Our Extended Security Model

In the following we present a security model for GKE protocols that includes extended security definitions concerning MA-security and contributiveness, while taking into account strong corruptions. Similar to [13, 4] 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.

3.1 Protocol Participants, Variables

Users, Instance Oracles, Long-Lived Keys Let \mathcal{U} be a set of N users. Each $U_i \in \mathcal{U}$ holds a long-lived key LL_i . Moreover each user U_i has several instances called *oracles*, denoted Π_i^s for $s \in \mathbb{N}$, participating distinct concurrent executions. And each protocol execution considers a set of oracles \mathcal{G} , called a *group*, the elements of which are called *group members*.

Internal States Every Π_U^s maintains an *internal state information state* 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). An oracle Π_U^s is *unused* until initialization (by which it is given the long-lived key LL_U). It then becomes a group member, associated to a particular session, and turns into the *stand-by* state where it waits for an invocation to execute the protocol. When the protocol starts, the oracle learns its partner id pid_U^s (and possibly sid_U^s) and turns into a *processing* state where it sends, receives and processes messages. During that stage, the internal state information state_U^s is maintained. After having computed k_U^s the oracle Π_U^s *accepts* and *terminates* the execution of the protocol operation (possibly after some additional auxiliary steps) meaning that it would not send or receive further messages. If the protocol fails, Π_U^s terminates without accepting, and k_U^s is set to an **undefined** value.

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* $k_U^s \in \{0, 1\}^\kappa$. Every session is identified by a unique *session id* sid_U^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 pid_U^s that contains the identities of participating users (including U), or formally

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

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

Session Group Key, Session ID, Partner ID Every session is identified by a unique, publicly-known *session id* sid_U^s . In each session each oracle Π_U^s gets a value pid_U^s that contains the identities of participating users (including U) and computes the *session group key* $k_U^s \in \{0, 1\}^\kappa$, where κ is a security parameter. Formally we write

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

and say that $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are *partnered* if $U_i \in \text{pid}_j^{s_j}$, $U_j \in \text{pid}_i^{s_i}$, and $\text{sid}_i^{s_i} = \text{sid}_j^{s_j}$ (similar to [13]).

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

A protocol is said correct if, when no adversary is present, all participants compute the same key. Note that our definition is independent of the communication channel, e.g., (asymmetric) broadcast, multicast, or unicast.

3.2 Adversarial Model

Now we consider an adversary \mathcal{A} which is a Probabilistic Polynomial-Time (PPT) algorithm having complete control over the network. \mathcal{A} can interact with protocol participants via queries to their oracles. Note that our security model (similar to [7, 13, 4]) does not deal with the issues of denial-of-service and fault-tolerance; our security definitions aim to prevent honest participants from accepting the group key biased by malicious participants.

Queries to the Instance Oracles

- $\text{Execute}(\mathcal{S})$: \mathcal{A} eavesdrops an honest execution of $P.\text{Setup}$ between a chosen set of oracles and is given the resulting transcript of $P.\text{Setup}(\mathcal{S})$;
- $\text{Send}(\Pi_U^s, m)$: \mathcal{A} sends message m to oracle Π_U^s and receives the response Π_U^s would have generated after having (honestly) processed message m . The response may be empty if m is incorrect. The adversary can have Π_U^s invoking $P.\text{Setup}$ with the oracles in \mathcal{S} via a query of the form $\text{Send}(\text{start}', \Pi_U^s, \mathcal{S})$: \mathcal{A} gets the first message that Π_U^s would generate in this case;
- $\text{RevealKey}(\Pi_U^s)$: \mathcal{A} is given the session group key k_U^s , provided Π_U^s has accepted;
- $\text{RevealState}(\Pi_U^s)$: \mathcal{A} is given the internal state information state_U^s ;
- $\text{Corrupt}(U)$: \mathcal{A} is given the long-lived key LL_U ;
- $\text{Test}(\Pi_U^s)$: \mathcal{A} tests the semantic security of k_U^s . Formally, a bit b is privately flipped and \mathcal{A} is given k_U^s if $b = 1$ and a random string if $b = 0$.

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

3.3 Security Goals

AKE-Security with Strong Forward Secrecy As defined in [7] 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 2 (Oracle Freshness). In the execution of P the oracle Π_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 $\text{Send}(\Pi_j^{s_j}, m)$ such that $U_j \in \text{pid}_U^s$ before Π_U^s and all its partners accept,
2. neither Π_U^s nor its partners are asked for a RevealState query before Π_U^s and all its partners accept,
3. neither Π_U^s nor any of its partners is asked for a RevealKey query after having accepted.

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

Note that in our model each Π_U^s is bound to one particular protocol execution (session). Thus, Π_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 [7] (and [14]) our definition allows the adversary to obtain knowledge of internal states from earlier sessions too.

Definition 3 (AKE-Security). Let P be a correct GKE protocol 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 Π_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 κ) 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 [7,8] which does not consider malicious participants and curious adversaries and is vulnerable to unknown key-share attacks.

Definition 4 (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 κ) 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 κ .

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 unifies the requirements of key control, contributiveness and unpredictability of group keys. Informally, we consider an active adversary \mathcal{A} which can corrupt participants and be *curious* during the protocol session in such a way that there exists at least one honest oracle that 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 (*key-replication attacks* [16]).

Definition 5 (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}_{us} consisting of all honest used oracles, \mathcal{G}_{std} consisting of all honest oracles that are in the stand-by state ($\mathcal{G}_{\text{std}} \subseteq \mathcal{G}_{\text{us}}$), and Ψ 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 the queries Send, Execute, RevealKey, RevealState, and Corrupt. At the end of the stage \mathcal{A} outputs (s, U) .

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

1. Π_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 Π_U^s .

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

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

We say that a GKE protocol P is contributory (**CGKE**) if this probability is a negligible function of κ .

Comments The first requirement ensures that Π_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 Π_U^s that participates in the same operation (this is not an attack since participants do not compute group keys synchronously). Note that $\mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$ consists of all oracles that at the end of the **prepare** stage have already terminated or remain in the processing state. Similarly, the condition $\text{sid}_U^s \notin \Psi$ prevents that \mathcal{A} while being in the **attack** stage outputs (s, U) such that Π_U^s has accepted with \tilde{k} already in the **prepare** stage; otherwise as soon as Π_U^s computes some 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 Π_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 Π_U^s during the execution of the **attack** stage (this is because our model separates LL_U from state_U^s). Also, due to the adaptiveness and strong corruptions the adversary in this game seems to be strictly stronger than in [4].

The following example highlights this idea. We consider the well-known two-party Diffie-Hellman (DH) key exchange³ [12], 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 $k := (g^{x_1})^{x_2} = g^{\tilde{x}} = \tilde{k}$.

4 Our Compiler for MA-Security and Contributiveness

Our compiler denoted **C-MACON** can be seen as an extension of the compiler in [13] that according to our model satisfies the requirement of MA-security⁴ but not of contributiveness. If P is a GKE protocol, by **C-MACON_P** we denote the compiled protocol.

In the following, we assume that each message sent by Π_U^s can be parsed as $U|m$ consisting of the sender's identity U and a message m . Additionally, an authentication token σ , e.g., a digital signature on m , can be attached. Our compiler is formally described in Definition 6: it is based on a *one-way* permutation π , a *collision-resistant pseudo-random* function ensemble F , and an *existentially unforgeable* digital signature scheme Σ .

Definition 6 (Compiler C-MACON). *Let P be a GKE protocol from Definition 1, $\pi : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ 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_P**, consists of the algorithm **INIT** and a two-round protocol **MACON** defined as follows:*

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

MACON: *After an oracle Π_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 Π_j^s with $U_j \in \text{pid}_i^s$. After Π_i^s receives $U_j|r_j$ from Π_j^s with $U_j \in \text{pid}_i^s$ it checks whether $|r_j| \stackrel{?}{=} \kappa$. If this verification fails then Π_i^s terminates without accepting;*

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 Π_j^s with $U_j \in \text{pid}_i^s$ and erases every other private information from state_i^s (including k_i^s and each ρ_l , $l \in [1, n]$) except for K_i^s .*

After Π_i^s receives $U_j|\sigma_j$ from Π_j^s with $U_j \in \text{pid}_i^s$ it checks whether $\Sigma.\text{Verify}(pk_j', \mu_j | \text{sid}_i^s | \text{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If this verification fails then Π_i^s terminates without accepting; otherwise it accepts with

³ As observed in [18] similar attacks can be found against many currently known group key exchange protocols.

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

the session group key $\mathbf{K}_i^s := f_{K_i^s}(v_2)$ where $v_2 \neq v_1$ is another public value, after having erased every other private information from 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 [13]) and for the derived resulting session group key \mathbf{K} .

4.1 Security Analysis

We provide the corresponding definitions of security of used cryptographic primitives and proofs of all theorems in the full version of this paper [9].

Theorem 1 (AKE-Security of C-MACON_P). *For any AGKE-sfs protocol P if Σ is EUF-CMA and F is pseudo-random then C-MACON_P is also a AGKE-sfs protocol, and*

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

Theorem 2 (MA-Security of C-MACON_P). *For any GKE protocol P if Σ is EUF-CMA and F is collision-resistant then C-MACON_P is MAGKE, and*

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

Theorem 3 (Contributiveness of C-MACON_P). *For any GKE protocol P if π is one-way and F is collision-resistant pseudo-random then C-MACON_P is CGKE, and*

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

5 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 [7] 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 [14] and the compiler for security against “insider attacks” by Katz and Shin [13] (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 [6]), and (ii) where the adversary is given access only to the weak corruptions (as in [4]).

References

1. Ateniese, G., Steiner, M., Tsudik, G.: Authenticated Group Key Agreement and Friends. *ACM CCS*, (1998) 17–26
2. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. *CRYPTO*, (1993) 232–249
3. Bellare, M., Rogaway, P.: Provably Secure Session Key Distribution: The Three Party Case. *STOC*, (1995) 57–66
4. Bohli, J.-M., Vasco, M. I. G., Steinwandt, R.: Secure Group Key Establishment Revisited. To appear in *International Journal of Information Security*. <http://eprint.iacr.org/2005/395>
5. Boyd, C., Mathuria, A.: *Protocols for Authentication and Key Establishment*. Springer (2003)
6. Bresson, E., Catalano, D.: Constant Round Authenticated Group Key Agreement via Distributed Computation. *PKC*, (2004) 115–129

7. Bresson, E., Chevassut, O., Pointcheval, D.: Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. *EUROCRYPT*, (2002) 321–336
8. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.J.: Provably Authenticated Group Diffie-Hellman Key Exchange. *ACM CCS*, (2001) 255–264
9. Bresson, E., Manulis, M.: Full version of this paper. *Available from the authors' homepages.*
10. Burmester, M.: On the Risk of Opening Distributed Keys. *CRYPTO*, (1994) 308–317
11. Choo, K.K.R., Boyd, C., Hitchcock, Y.: Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. *ASIACRYPT*, (2005) 585–604
12. Diffie, W., Hellman, M.E.: New Directions in Cryptography. *IEEE IT*, (1976) 22(6):644–654
13. Katz, J., Shin, J.S.: Modeling Insider Attacks on Group Key-Exchange Protocols. *ACM CCS*, (2005) 180–189
14. Katz, J., Yung, M.: Scalable Protocols for Authenticated Group Key Exchange. *CRYPTO*, (2003) 110–125
15. Kim, Y., Perrig, A., Tsudik, G.: Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. *ACM CCS*, (2000) 235–244
16. Krawczyk, H.: HMQR: A High-Performance Secure Diffie-Hellman Protocol. *CRYPTO*, (2005) 546–566
17. Manulis, M.: Survey on Security Requirements and Models for Group Key Exchange. Technical Report. <http://eprint.iacr.org/2006/388>
18. Manulis, M.: Security-Focused Survey on Group Key Exchange Protocols. Technical Report. <http://eprint.iacr.org/2006/395>
19. Menezes, A., van Oorschot, P.C., Vanstone, S.: *Handbook of Applied Cryptography*. CRC Press, October (1996)
20. Mitchell, C.J., Ward, M., Wilson, P.: Key Control in Key Agreement Protocols. *El. Letters*, (1998) 34(10):980–981
21. Shoup, V.: On Formal Models for Secure Key Exchange (Version 4). Technical Report. <http://shoup.net/>
22. Steiner, M.: *Secure Group Key Agreement*. PhD thesis, (2002)