MODULARITY OF PROTOCOL DESIGN. Since GKE protocols are used as building blocks for high-level applications, it is interesting to design them in a modular way: applications that make use of these protocols may have specific security goals, and thus it is desirable that a specific GKE protocol can provide the corresponding properties. Modular design allows to build such "*à la carte*" protocols. In order to provide these modular constructions in a generic way, so-called "compilers" have been developed, e.g., [32, 33]. They allow designers to enhance security of a protocol in a *black-box* manner, that is, independently of the implementation of the protocol being enhanced. In general, security enhancement means enlarging the class of adversaries the protocol can deal with, or adding new security properties.

CONTRIBUTIONS AND ORGANIZATION. We start with the brief overview of the BCPQ model and its security definitions. In Section 3 we point out a problem in the BCPQ model between its technical core – the notion of *partnering* – and its definition of MA-security. In Section 4 we analyze some well-known variants of the BCPQ model, i.e., [27, 32, 33], from the perspective of general applicability (independence of protocol design), technical construction of partnering, and MA-security whereby focusing on possible attacks carried out by *malicious participants*. By malicious participants we mean legitimate protocol participants who are fully controlled by the adversary. We emphasize that consideration of malicious participants makes sense in the scope of MA-security but not of AKE-security that deals with the secrecy of the group key since malicious participants learn the established group key anyway.

After identifying some drawbacks in the mentioned variants we describe in Section 5 an extended "game-based" security model with revised definitions of AKE- and MA-security under consideration of malicious participants. Our model is based on the more powerful BCPQ refinement from [13] that considers AKE-security in the presence of (partial) internal state corruptions (strong corruptions). We also introduce an additional notion of *backward secrecy* which leads to new corruption models in case of AKE-security.

In Section 7.2 we provide a brief analysis of some known security-enhancing compilers for GKE protocols. In particular, we show that the compiler proposed by Katz and Yung in [33] needs some additional assumptions in order to be considered as a really generic solution. On the other hand, in order to show that our extended security definitions are feasible enough for the construction of practical reductionist security proofs, in Section 7.3 we describe a compiler `C-AMA` (as a slightly modified combination of the compilers from [33] and [32]) that satisfies our *stronger* definitions of AKE- and MA-security for any GKE protocol and prove its security under standard cryptographic assumptions.

2 Overview of the BCPQ Model

The BCPQ model extends the methodology introduced by Bellare and Rogaway [8, 9] to a group setting. Each protocol participant $U_i \in ID$[3], $i = 1, \ldots, n$ is modeled by an unlimited number of instances called *oracles* and denoted $\Pi_i^{s_i}$ (s_i-th instance of U_i) that can be involved in different concurrent executions of $\mathtt{P}$. Each user U_i is assumed to have a long-lived key LL_i (either symmetric or asymmetric). The BCPQ model uses session ids to define the notion of partnering which is the technical construction used in the definition of all security goals. A session id of an oracle $\Pi_i^{s_i}$ is defined as $\mathrm{SID}(\Pi_i^{s_i}) := \{\mathrm{SID}_{ij} \mid U_j \in ID\}$ where SID_{ij} is the concatenation of all flows that $\Pi_i^{s_i}$ exchanges with another oracle $\Pi_j^{s_j}$. According to the BCPQ model two oracles $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are called *directly partnered*, denoted $\Pi_i^{s_i} \leftrightarrow \Pi_j^{s_j}$, if both oracles *accept* (compute the session key) and if $\mathrm{SID}(\Pi_i^{s_i}) \cap \mathrm{SID}(\Pi_j^{s_j}) \neq \emptyset$. Further, oracles $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ are *partnered* if, in the graph $G_{\mathrm{SIDS}} := (V, E)$ with $V := \{\Pi_l^{s_l} \mid U_l \in ID,\ l = 1, \ldots, n\}$ and $E := \{(\Pi_l^{s_l}, \Pi_{l'}^{s_{l'}}) \mid \Pi_l^{s_l} \leftrightarrow \Pi_{l'}^{s_{l'}}\}$, there exists a sequence of oracles $(\Pi_{l_1}^{s_{l_1}}, \Pi_{l_2}^{s_{l_2}}, \ldots, \Pi_{l_k}^{s_{l_k}})$ with $l_k > 1$, $\Pi_i^{s_i} = \Pi_{l_1}^{s_{l_1}}$, $\Pi_j^{s_j} = \Pi_{l_k}^{s_{l_k}}$, and $\Pi_{l-1}^{s_{l-1}} \leftrightarrow \Pi_l^{s_l}$ for all $l = l_2, \ldots, l_k$. This kind of partnering is denoted $\Pi_i^{s_i} \leftrightsquigarrow \Pi_j^{s_j}$. The BCPQ model uses graph G_{SIDS} to construct (in polynomial time $|V|$) the graph of partnering $G_{\mathrm{PIDS}} := (V', E')$ with $V' = V$ and $E' = \{(\Pi_l^{s_l}, \Pi_{l'}^{s_{l'}}) \mid \Pi_l^{s_l} \leftrightsquigarrow \Pi_{l'}^{s_{l'}}\}$, and defines the *partner id* for an oracle $\Pi_i^{s_i}$ as $\mathrm{PIDS}(\Pi_i^{s_i}) = \{\Pi_l^{s_l} \mid \Pi_i^{s_i} \leftrightsquigarrow \Pi_l^{s_l}\ \forall\, l \in \{1, n\} \setminus \{i\}\}$.

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

The BCPQ model considers a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ which while executed is allowed to send messages to the oracles (and invoke the protocol execution) via a Send query, reveal the session key computed by an oracle via a Reveal query, obtain a long-lived key of a user via a Corrupt query (note that the oracle's internal state information is not revealed), and ask a Test query to obtain either a session key or a random number. Using this adversarial setting the BCPQ model specifies two security goals for a GKE protocol: AKE-security and MA-security, both based on the notion of partnering. We emphasize that the above definition of partnering has been further used in the BCPQ variants proposed in [12–14] and these models in turn have been used in security proofs of GKE protocols in [12–15].

For the AKE-security the model requires that, during its execution, adversary $\mathcal{A}$ asks a single Test query to a fresh oracle. An oracle $\Pi_i^{s_i}$ is *fresh* if (1) it has accepted, (2) no oracle has been asked for a Corrupt query before $\Pi_i^{s_i}$ accepts, and (3) neither $\Pi_i^{s_i}$ nor any of its partners have been asked for a Reveal query. A GKE protocol is said to be AKE-secure if $\mathcal{A}$ cannot guess which value it has received in response to its Test query, i.e., the session key or a random number, significantly better than at random. This definition of AKE-security, applied with an appropriate definition of freshness, subsumes the following earlier informal definitions:

- *key secrecy* [24] (a.k.a *implicit key authentication* [39]) which requires that each legitimate protocol participant is assured that no other party except for other legitimate participants learns the established group key;
- *resistance against known-key attacks* [17, 48] meaning that an adversary who knows group keys of previous sessions must not be able to compute subsequent session keys, *key independence* [35] meaning that an adversary who knows a proper subset of group keys must not be able to discover any other group keys;
- *perfect forward secrecy* [24, 30, 39] requiring that the disclosure of long-term keying material must not compromise the secrecy of the established keys from earlier protocol runs.

The definition of MA-security in the BCPQ model captures the fact that it is hard for a computationally bounded adversary $\mathcal{A}$ to impersonate any participant U_i through its oracle $\Pi_i^{s_i}$. For a GKE protocol among n users to be MA-secure, the probability that there exists at least one oracle $\Pi_i^{s_i}$ which accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| \neq n-1$ is required to be negligible. In other words, for such protocols, the authors claim that if each participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$ then no impersonation attacks could have occurred — thus the informal notion of *mutual authentication* [8][4] meaning that each participating oracle is assured of every other oracle's participation in the protocol is satisfied.

Further, we point the reader's attention to the following claims given by the authors of [15]:

> In the definition of partnering, we do not require that the session key computed by partnered oracles be the same since it can easily be proven that the probability that **partnered** oracles come up with different session keys is negligible. [15, Footnote 3]

> We are not concerned with partnered oracles coming up with different session keys, since our definition of partnering implies the oracles have exchanged *exactly* the same flows. [15, Section 7.4]

If these claims hold then the above definition of MA-security additionally captures the following informal security goals earlier specified in the literature:

- *key confirmation* meaning that each protocol participant must be assured that all other protocol participant that have accepted hold identical group keys[5],
- *explicit key authentication* [39], i.e., key confirmation and mutual authentication at the same time.

In Section 3 we explain why the definition of MA-security might not be general enough for GKE protocols. We do not pretend having broken some provably MA-secure scheme. In contrast, we explain why, if every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n-1$, it does not necessarily mean that the considered protocol provides mutual authentication and key confirmation. To do so, we exhibit cases where an impersonation attack may likely result in different group keys accepted by different partnered oracles.

[4] introduced originally for two-party protocols

[5] This is a slightly modified definition from [39] wrt. to the arguments from [43] on impossibility of the assurance of some participant that other participants have actually accepted the group key.

3 Problems with the Definition of MA-Security in the BCPQ Model

PROBLEMS. We provide examples for the following two scenarios:

1. there exists GKE protocols where an active adversary $\mathcal{A}$ can impersonate one of the participants through its oracle but nevertheless every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n - 1$
2. there exists GKE protocols where each participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = n - 1$ but there are at least two partnered oracles that have computed different keys.

Note that these problems become visible only in the group setting with at least three protocol participants: therefore, it does not concern the original notion of mutual authentication by Bellare and Rogaway [7] defined via matching conversations. Before we give examples using a concrete GKE protocol we provide an abstract description of our idea. Figure 1 shows the abstract messages denoted m_i (index i specifies the order in which messages have been sent) that have been exchanged between the oracles (at least three participants are required) during the honest execution of any GKE protocol from [12–15]. A concrete equivalent message of each abstract message m_i can be found in the corresponding *up-* or *downflow* stage of any of these GKE protocols.

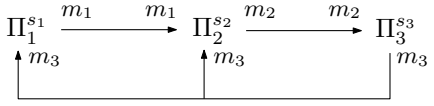

Fig. 1. Honest execution of protocols in [12–15]. By m_i at the beginning of the arrow we mean the original message sent by the oracle, and by m_i at the end of the arrow we mean the corresponding message received by another oracle.

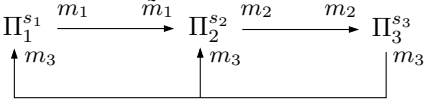

Fig. 2. Protocol execution where $\mathcal{A}$ impersonates U_1

Obviously, Figure 1 shows a correct execution of the protocol since no message is modified. Figure 3 specifies the session ids of the oracles $\Pi_1^{s1}, \ldots, \Pi_3^{s_3}$ during this honest protocol execution using the construction from the BCPQ model.

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

Fig. 3. $\mathrm{SID}(\Pi_i^{s_i})$ in the honest protocol execution

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

Fig. 4. $\mathrm{SID}(\Pi_i^{s_i})$ in the protocol execution with impersonation of U_1

To show the first problem we consider the case where $\mathcal{A}$ impersonates U_1 and modifies message m_1 to $\tilde{m}_1$ (Figure 2) such that $\mathrm{SID}_{21} = \tilde{m}_1$ (Figure 4). We *cannot* generally assume that all oracles accept after this modification but we may assume that there exists protocols for which this is the case (our example later is such a protocol where the oracles nevertheless accept). If so, we show that every participating oracle $\Pi_i^{s_i}$ accepts with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$. To that goal, we need to show that $\Pi_i^{s_i} \leftrightsquigarrow \Pi_j^{s_j}$ (or $\Pi_i^{s_i} \leftrightarrow \Pi_j^{s_j}$) still holds for any two participating $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$. Thus we need to look more precisely on the session ids of the oracles. First note that $\mathrm{SID}_{12} = m_1$. Though $\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_2^{s_2}) = \{m_1, m_3\} \cap \{\tilde{m}_1, m_2|m_3\} = \emptyset$ and thus $\Pi_1^{s_1} \not\leftrightarrow \Pi_2^{s_2}$, we still have $\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_3^{s_3}) = \{m_1, m_3\} \cap \{m_3, m_2|m_3\} = m_3$ and $\mathrm{SID}(\Pi_3^{s_3}) \cap \mathrm{SID}(\Pi_2^{s_2}) = \{m_3, m_2|m_3\} \cap \{\tilde{m}_1, m_2|m_3\} = m_2|m_3$ so that $\Pi_1^{s_1} \leftrightsquigarrow \Pi_2^{s_2}$ (and $\Pi_1^{s_1} \leftrightarrow \Pi_3^{s_3}$ and $\Pi_2^{s_2} \leftrightarrow \Pi_3^{s_3}$ as well). Hence, $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ for every $\Pi_i^{s_i}$: all oracles are still partnered though the impersonation attack. Oracle

$\Pi_2^{s_2}$ having received a different message than the one originally sent by $\Pi_1^{s_1}$, this may result in different group keys computed by $\Pi_1^{s_1}$ and $\Pi_2^{s_2}$.

CONCRETE EXAMPLE. Consider the GKE protocol described in the same paper as the BCPQ model [15] but without the additional confirmation round; this additional round belongs to a concrete protocol design but not to a general security model; it is not required that a protocol uses this additional round to achieves MA-security.

Recall, our goal is to show that despite of the acceptance of each oracle $\Pi_i^{s_i}$ with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ mutual authentication and key confirmation are not necessarily provided.

The protocol proceeds as described in Figure 5; $[m]_{U_i}$ denotes a message m signed by $\Pi_i^{s_i}$, and $V(m) \stackrel{?}{=} 1$ its verification; g is a generator of a cyclic group of prime order p. Upon computing $K = g^{x_1x_2x_3}$ each oracle derives the resulting group key $k := \mathcal{H}(ID, FL_3, K)$ with a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^l$ where l is the security parameter. Now we consider that $\Pi_1^{s_1}$ chooses $x_1 \in \mathbb{Z}_p^\star$ but $\mathcal{A}$ drops the original message $[Fl_1]_{U_1}$

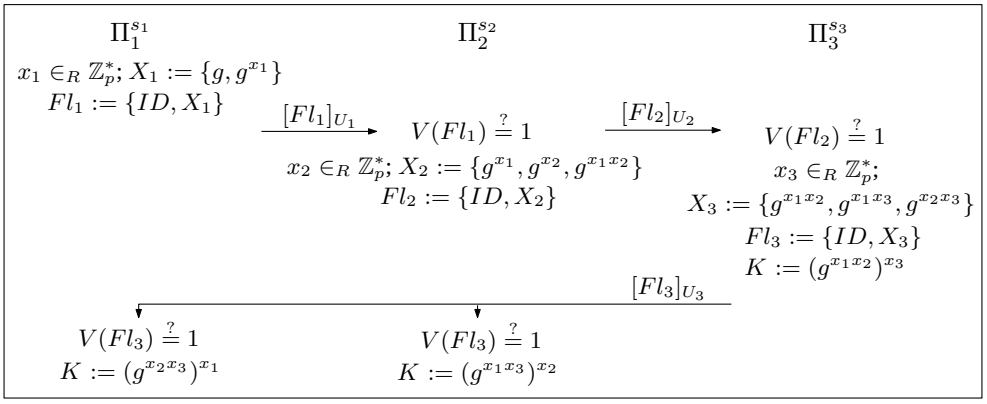

Fig. 5. Execution of the protocol in [15] with three participants

and replays a corresponding message from some previous protocol execution. The replayed message is likely to be $[\widetilde{Fl_1}]_{U_1}$ with $\widetilde{Fl_1} := (ID, \widetilde{X_1})$ and $\widetilde{X_1} := \{g, g^{\widetilde{x_1}}\}$ for some $\widetilde{x_1} \neq x_1$. Obviously, $\Pi_2^{s_2}$ can still verify the replayed message, i.e., $V(\widetilde{Fl_1}) = 1$ holds. It is easy to see that $X_2 = \{g^{\widetilde{x_1}}, g^{x_2}, g^{\widetilde{x_1}x_2}\}$ and $X_3 := \{g^{\widetilde{x_1}x_2}, g^{\widetilde{x_1}x_3}, g^{x_2x_3}\}$ so that $\Pi_1^{s_1}$ computes $K = g^{x_1x_2x_3}$ whereas $\Pi_2^{s_2}$ and $\Pi_3^{s_3}$ compute another value, i.e., $K = g^{\widetilde{x_1}x_2x_3}$. Thus the derived group keys are different. Though (without the confirmation round) all oracles accept since all signature verifications remain correct. Moreover, and similarly to the abstract problem description above, it is easy to check that $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ for every $\Pi_i^{s_i}$, $i \in \{1,2,3\}$.

This illustrates the case where $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ and yet the protocol does not provide mutual authentication and key confirmation. We stress, again, that this does not contradict the MA-security of the proposed protocol when the additional round is executed. However, there may exist other protocols (including our example) for which this statement is not true (if the MA-security is tentatively achieved with other techniques), and thus it is worth studying the generality/applicability of MA-security definition in the BCPQ model.

Furthermore, we stress that the more general definition of MA-security should also consider attacks by malicious protocol participants (the BCPQ model had no intention to consider such scenario). For example, as noted in [22] many BCPQ-like models fail to provide security against *unknown key-share attacks* [10] because they do not consider malicious (corrupted) participants during the protocol execution. It is interesting to notice that, while malicious participants surely break the AKE-security (session key indistinguishability), their actions against MA-security left some open questions: though the work by Katz and Shin [32] provides a partial solution, it is worth noticing that protection (i.e., resilience) has no satisfying solution yet. It is a subject of future work to be able to detect and eliminate the dishonest players in such a way that the remaining, honest ones, can compute a common key.

4 Discussion on Technical Constructions of some BCPQ-Variants and their Definitions of MA-Security

A VARIANT BY KATZ AND YUNG. The modification by Katz and Yung [33] (denoted as the KY model) suggests a different construction of partner ids and session ids. The *partner id* of an oracle $\Pi_i^{s_i}$ consists of the identities of all group members who intend to participate in the protocol and is defined straight after the invocation of the protocol execution. Hence, opposite to the BCPQ model, this set is known before the protocol completes. Further, the *session id* of $\Pi_i^{s_i}$ is simply the concatenation of all messages that $\Pi_i^{s_i}$ has sent or received during the protocol execution. Note that the KY model was proposed in the context of a GKE protocol over an unreliable asynchronous broadcast channel. Obviously, in this case every oracle computes the same session id. However, the KY model fails in protocols where some messages are sent over unicast, e.g., in protocols from [12–15]. Note, however, that any construction of session ids based on the concatenation of exchanged message flows has one significant drawback — it becomes available after the protocol termination only. Since some protocols use uniqueness of session ids as protection against replay and protocol interference attacks it is desirable to have a unique session id prior to the protocol termination. The *partnering* between two oracles holds if they have equal partner ids and equal session ids. Note that the KY model does not provide own definition of MA-security but refers to the one in the BCPQ model. Due to the different construction of partnering the identified problems in the BCPQ model have no consequences here.

Finally, another limitation of the KY model is that it is not intended for considering malicious participants.

A VARIANT BY DUTTA *et al.* The modification by Dutta *et al.* [27] is similar to the KY model regarding the construction of *partner ids*. However, they define a *session id* of an oracle $\Pi_i^{s_i}$ as $\{(U_1, s_1), \ldots, (U_n, s_n)\}$ where each pair (U_j, s_j), $j \in \{1, \ldots, n\}$ corresponds to the oracle $\Pi_j^{s_j}$ of the protocol participant U_j, and say that two oracles are *partnered* if they have equal partner ids and equal session ids. In order to keep session ids unique the authors require the uniqueness of oracles for each new session. To have this notion made sense, [25] suggests to use a counter value as an additional parameter which should be increased for every new oracle of the user. Though this makes unique session ids available prior to the protocol termination, it forces the counter to be saved after each execution and protected against modifications. A more practical approach seems to be using nonces in each new protocol execution (excluding the highly improbable case of collisions). Note also that [27] and [25] had no intentions to consider mutual authentication and key confirmation.

A VARIANT BY KATZ AND SHIN. Katz and Shin [32] proposed a different security model (referred to as the KS model) for GKE protocols, and provide a security analysis in the framework of Universal Composability (UC) [20]. The KS model provides the first formal treatment of GKE protocols security in the presence of malicious participants. The KS model is an extension of the BCPQ and KY models. The partner ids and the partnering relationship between the oracles is similar to the KY model but unique session ids are assumed to be provided by some high-level application mechanism.

Among other things, the KS model defines *security against insider attacks* as a combination of two requirements: *agreement* and *security against insider impersonation attacks*:

- the adversary $\mathcal{A}$ is said to *violate agreement* if there exist two partnered oracles Π_i^s and Π_j^t such that neither U_i nor U_j is corrupted but Π_i^s and Π_j^t have accepted with different session keys. Intuitively, this considers key confirmation in case that all other participants are malicious (corrupted);
- the adversary $\mathcal{A}$ is said to *impersonate U_j to (accepting) Π_i^s* if U_j is uncorrupted and belongs to the (expected) partner id 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; a protocol is said to be *secure against insider impersonation attacks* if for any party U_j and any instance Π_i^s, $\mathcal{A}$ 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.

Obviously, the last requirement assumes the existence of at least two uncorrupted participants, but allows the adversary to corrupt other participants: an active adversary can thus generate fresh messages on behalf of other

participants. Intuitively, security against insider impersonation attacks considers mutual authentication and unknown key-share resilience in the presence of malicious participants. Note that the KS model does not describe what relationship do their formal definitions have with the well-known informal definitions. Their security analysis holds in the framework of UC-security, based on the simulatability approach [19] rather than on a reductionist approach [6, 37].

Further, in addition to their model, Katz and Shin proposed a compiler to turn any GKE protocol which is secure in the BCPQ model into a protocol which is secure in their UC-based model, and provided simulatability-based security proofs for this case. However, they left open the question whether their definitions of agreement and security against insider impersonation attacks are practical enough for the construction of reductionist security proofs (even though UC-security is considered to be stronger).

5 The Revised "Game-based" Security Model

In the following we propose a new variant of the BCPQ model while considering malicious participants. We provide an alternative definition (which we call MA-security to keep consistency with all previous models) that can be used to replace definitions of agreement and security against insider impersonation attacks of the KS model. One advantage is that, for the same requirements, our model needs only one definition (and consequently one reductionist proof) whereas in the KS model two definitions are needed. Furthermore, we prove that our definition really unifies the informal notions of key confirmation, mutual authentication and unknown key-share resilience in the presence of malicious participants while the KS model does not explicitly prove this.

Our variant is based on the refined BCPQ model as presented in [13], a refinement that considers *strong corruptions*, i.e., attacks against internal states. We also extend the original definition of AKE-security (in a modular way) considering a new requirement which we call *backward secrecy*.

5.1 Protocol Participants, Variables

USERS, INSTANCE ORACLES. We consider $\mathcal{U}$ as a set of N users in the universe. 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*; Π_i^s, with $s \in \mathbb{N}$, denotes the s-th instance oracle of U_i.

INTERNAL STATES. Every Π_U^s maintains an *internal state information* $\mathtt{state}_U^s$ which is composed of all private, ephemeral information used during the protocol execution. 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* $k_U^s \in \{0,1\}^\kappa$. Every session is identified by a unique *session id* $\mathtt{sid}_U^s$. This value is known to all oracles participating in the same session. Similarly, each oracle $\Pi_U^s \in \mathcal{G}$ gets a value $\mathtt{pid}_U^s$ that contains the identities of participating users (including U), or formally

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

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

INSTANCE ORACLE STATES: STAND-BY, PROCESSING, ACCEPTED, TERMINATED. An oracle Π_U^s may be either *used* or *unused*. The oracle is considered as unused if it has never been initialized. Each unused oracle Π_U^s 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 Π_U^s learns its partner id $\mathtt{pid}_U^s$ (and

possibly $\mathtt{sid}_U^s$) and turns into a *processing* state where it sends, receives and processes messages according to the description of the protocol. During the whole processing state the internal state information $\mathtt{state}_U^s$ is maintained by the oracle. The oracle Π_U^s remains in the processing state until it collects enough information to compute the session group key k_U^s. After Π_U^s computes k_U^s it *accepts* and *terminates* the execution of the protocol operation (possibly after some additional auxiliary steps), meaning that it would not send or receive further messages. If the protocol execution fails (due to any adversarial actions) then Π_U^s terminates without having accepted, i.e., the session group key k_U^s is set to some undefined value.

5.2 Definition of a Group Key Exchange Protocol

Definition 1 (GKE Protocol). *A group key exchange protocol* $\mathtt{P}$ *consists of a key generation algorithm* $\mathtt{KeyGen}$*, and a protocol* $\mathtt{Setup}$ *defined as follows:*

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

5.3 Adversarial Model

QUERIES TO THE INSTANCE ORACLES. The adversary $\mathcal{A}$ is represented by a PPT machine and is assumed to have complete control over all communication in the network. It may interact with group members by making the following oracle queries:

- $\mathsf{Setup}(\mathcal{S})$: This query models $\mathcal{A}$ eavesdropping the honest operation execution of $\mathtt{P}.\mathtt{Setup}$. $\mathtt{P}.\mathtt{Setup}(\mathcal{S})$ is executed and $\mathcal{A}$ is given the transcript of the execution.
- $\mathsf{Send}(\Pi_U^s, m)$: This query models $\mathcal{A}$ sending messages to the oracles. $\mathcal{A}$ receives the response which Π_U^s would have generated after having processed the message m according to the description of $\mathtt{P}$. The adversary can ask an oracle Π_U^s to invoke $\mathtt{P}.\mathtt{Setup}$ with the oracles in $\mathcal{S}$ via the query of the form $\mathsf{Send}(\Pi_U^s, \mathcal{S})$ which gives $\mathcal{A}$ the first message that Π_U^s would generate in this case. Thus, using Send queries the adversary can actively participate in $\mathtt{P}.\mathtt{Setup}$.
- $\mathsf{RevealKey}(\Pi_U^s)$: $\mathcal{A}$ is given the session group key k_U^s. This query is answered only if Π_U^s has accepted.
- $\mathsf{RevealState}(\Pi_U^s)$: $\mathcal{A}$ is given the internal state information $\mathtt{state}_U^s$.[6]
- $\mathsf{Corrupt}(U)$: $\mathcal{A}$ is given the long-lived key LL_U.
- $\mathsf{Test}(\Pi_U^s)$: This query will be used to model the AKE-security of a GKE protocol. It can be asked by $\mathcal{A}$ at any time, to any oracle having accepted, but only once during the entire attack. 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.

A *passive* adversary can eavesdrop the execution of the protocol operations via Setup queries, reveal session group keys, internal states and corrupt participants via $\mathsf{RevealKey}$, $\mathsf{RevealState}$, and $\mathsf{Corrupt}$ queries, respectively, and is also allowed to ask Test queries. Additionally, it is given access to the Send queries, however, with the restriction that it is not allowed to inject, replay, or modify messages. Thus, a passive adversary can truly forward, drop, and delay messages, or deliver them out of order. Although Setup queries can be simulated using Send with the appropriate invocation messages, the presence of these queries still allows a separate treatment of the (weaker) passive adversaries who are restricted to the eavesdropping of the protocol execution in the sense of [33]. Note that in our model the passive adversary is stronger than in [33] and is comparable to the one from [21].

[6] This kind of the adversarial query has previously been mentioned by Canetti and Krawczyk in their model for two-party protocols [21].

FORWARD SECRECY. The notion of forward secrecy allows to distinguish between damages (in previously completed sessions) that result from actions of the adversary in the current session. Similar to [13] we distinguish between *weak-forward secrecy* (wfs) where the adversary is additionally allowed to ask Corrupt queries, and *strong-forward secrecy* (sfs) where it is additionally allowed to ask Corrupt and RevealState queries.

BACKWARD SECRECY. The notion of backward secrecy is symmetric to that of forward secrecy in the sense that it considers damages to the AKE-security of future sessions after actions of the adversary in past/current sessions. The notion might seem useless at first glance (such actions can make secrecy just impossible), however, there might exist intermediate actions, such as corruptions of internal states, that do not compromise future session keys (or at least not all of them). We distinguish between *weak-backward secrecy* (wbs) where the adversary is allowed to ask RevealState queries, and *strong-backward secrecy* (sbs) where the adversary is additionally allowed to ask Corrupt queries[7].

Note that in our definition of AKE-security that models key secrecy, the adversary is a non-participating party and not a malicious participant, even in case that it reveals long-lived keys prior to the protocol execution. In order to consider only non-participating adversaries we introduce the following notion of α-fresh sessions.

ORACLE FRESHNESS, CORRUPTION MODELS, ADVERSARIAL SETTINGS. The notion of freshness for an oracle Π_U^s is needed to distinguish between various definitions of security with respect to different flavors of backward or forward secrecy. Each flavor $\alpha \in \{\emptyset, \texttt{wbs}, \texttt{wfs}, \texttt{sbs}, \texttt{sfs}\}$ leads to a different definition of freshness.

Definition 2 (α**-Freshness).** *Let $\alpha \in \{\emptyset, \texttt{wfs}, \texttt{wbs}, \texttt{sfs}, \texttt{sbs}\}$. The oracle $\Pi_U^s \in \mathcal{G}$ is*

$\emptyset$-fresh *if: neither Π_U^s nor any of its partners is asked for a RevealKey query after having accepted;*

wbs-fresh *if: (1) neither Π_U^s nor any of its partners is asked for a RevealState query after $\mathcal{G}$ is created, and (2) neither Π_U^s nor any of its partners is asked for a RevealKey query after having accepted;*

wfs-fresh *if: (1) no $U_i \in \mathtt{pid}_U^s$ is asked for a Corrupt query prior to a query of the form Send$(\Pi_j^{s_j}, m)$ such that $U_j \in \mathtt{pid}_U^s$ before Π_U^s and all its partners accept, and (2) neither Π_U^s nor any of its partners is asked for a RevealKey query after having accepted;*

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

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

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

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

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

The notion of α-fresh sessions becomes important in security proofs in order to distinguish between "honest" and "corrupted" sessions. Intuitively, the above definitions are to be used as follows. In each of the secrecy cases

[7] In case of backward secrecy Corrupt queries are more damageable than RevealState queries because the long-lived keys are usually used for authentication and their knowledge allows the adversary to impersonate users in subsequent sessions and learn the session group key.

considered, either the adversary is not allowed to ask some "bad" queries, more precisely such queries are allowed but immediately make the oracle unfresh. The scenario aims to delimit the "bad" queries. To properly manage the adversarial capabilities for each scenario of freshness, we distinguish between the following corruption models.

Definition 3 (Corruption Model β). *A PPT adversary $\mathcal{A}$ is an adversary that acts with respect to the corruption model β according to the following definition:*

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

A concrete proof of AKE-security needs to specify capabilities of the adversary depending on the intended freshness type. Combining definitions for freshness and corruption we obtain a set of reasonable *adversarial settings* $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{wbs}, \texttt{wcm-bs}), (\texttt{sbs}, \texttt{scm}), (\texttt{sfs}, \texttt{scm})\}$. Note that other imaginable settings are not reasonable from the perspective of the attacks. Nevetheless, separation in five types of freshness and four types of corruption models allows to talk about the corruption model in a general way, that is not necessarily in the context of AKE-security (but also wrt. other security requirements).

Remark 1. In practice long-lived keys are mostly used to achieve authentication rather than for the actual group key computation. It is thus intuitively clear that if an adversary is able to corrupt a group member (obtaining its long-lived key) then it can impersonate that member in subsequent sessions. Therefore, achieving AKE-security in the $(\texttt{sbs}, \texttt{scm})$ sense would require the long-lived keys to be fresh for each new execution, a contradiction with the long-lived key terminology. To the contrary, the adversarial setting $(\texttt{wbs}, \texttt{wcm-bs})$ appears of great interest since it concerns only oracle internal state information and is independent of any long-term secrets. Moreover we argue that $(\texttt{wbs}, \texttt{wcm-bs})$ is important since in previous models [13, 32] a persistent internal state is used in both past and future sessions, and thus, while forward secrecy looks at (state) corruptions in later sessions, backward secrecy must legitimately look at state corruptions in previous sessions.

5.4 Security Goals

In this section we describe security goals for a GKE protocol. We give a formal definition of (Authenticated)KeyExchange-security (indistinguishability of session group keys), and a new definition of MA-security that considers malicious participants and internal state corruptions of honest participants.

Definition 4 ((A)KE-Security). *Let $\mathtt{P}$ be a correct GKE protocol and b a uniformly chosen bit. Consider a reasonable adversarial setting (α, β) and an (active) adversary $\mathcal{A}$. We define game $\mathsf{Game}^{\mathsf{(a)ke}-b}_{\alpha,\beta,\mathtt{P}}(\mathcal{A}, \kappa)$ as follows:*

- *after initialization $\mathcal{A}$ interacts with instance oracles using queries;*
- *if $\mathcal{A}$ asks a* Test *query to an α-fresh oracle Π^s_U which has accepted, it receives either $key_1 := k^s_U$ (if $b = 1$) or $key_0 \in_R \{0,1\}^\kappa$ (if $b = 0$);*
- *$\mathcal{A}$ continues interacting with instance oracles;*
- *when $\mathcal{A}$ terminates, it outputs a bit trying to guess which case it was dealing with.*

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

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

We say that $\mathtt{P}$ is an (A)KE-secure protocol with α-secrecy, *denoted* $(\mathtt{A})\mathtt{GKE}$-α, *if the advantage $\mathsf{Adv}^{\mathsf{(a)ke}}_{\alpha,\beta,\mathtt{P}}(\kappa)$ is negligible. If $\alpha = \emptyset$, we just say that $\mathtt{P}$ is* (A)KE-secure.

We emphasize that at this step we do not consider malicious participants (users), but only (partially) corrupted oracles for dealing with forward- and backward-secrecy.

The following definition of MA-security considers an adversary who is allowed to corrupt and act on behalf of participants during the protocol execution.

Definition 5 (MA-Security). *Let* $\mathtt{P}$ *be a correct GKE protocol and* $\mathsf{Game}_{\mathtt{P}}^{\mathsf{ma}}(\mathcal{A}, \kappa)$ *the interaction between the instance oracles and an active adversary* $\mathcal{A}$ *which can query* Send, Setup, 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* $U_j \in \mathtt{pid}_i^{s_i}$ *that is uncorrupted at the time* $\Pi_i^{s_i}$ *accepts, such that*

1. *there exists* ***no*** *instance oracle* $\Pi_j^{s_j}$ *with* $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$, ***or***
2. *there exists* ***an*** *instance oracle* $\Pi_j^{s_j}$ *with* $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$ *that 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* $\mathsf{Succ}_{\mathtt{P}}^{\mathsf{ma}}(\kappa)$. *We say that a GKE protocol* $\mathtt{P}$ *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. Hence, our MA-security definition seems to be stronger than definitions of security against insider attacks in the KS model.

Note also that we do not deal with α-fresh sessions since malicious participants learn established keys implicitly. Also in $\mathsf{Game}_{\mathtt{P}}^{\mathsf{ma}}(\mathcal{A}, \kappa)$ the adversarial Test query is useless. In the next section we show how this definition subsumes informal requirements of unknown key-share resilience, key confirmation, and mutual authentication (note that this is missing for the definitions in [32]).

6 Claims Concerning MA-Security

In the following we present some claims to illustrate the relationship between our definitions of MA-security and the related mostly important informal notions concerning key confirmation, mutual authentication and unknown key-share resilience, since this relationship may be difficult to see at first sight. The informal definition of *key confirmation* [39] refined wrt. to the arguments in [43] means that each protocol participant must be assured that all other protocol participant that have accepted hold identical group keys. The notion of *mutual authentication* introduced in [8] for two-party protocols when considered for group key exchange protocols means that each identified protocol participant is known to actually possess the established group key. Note that this requirement is similar to *explicit key authentication* [39]. The related requirement called *unknown key-share resilience* surfaced in [24] means that an active adversary must not be able to make one protocol participant believe that the key is shared with one party when it is in fact shared with another party. Note that the adversary may be a malicious participant and does not need necessarily to learn the established key [10].

The missing formalism of the original informal definitions allows only argumentative proofs for our claims. We also stress that none of the previously proposed models provides such claims for their definitions.

Claim. If P is a MAGKE protocol then it provides key confirmation and mutual authentication (explicit key authentication), 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. If P does not provide key confirmation and mutual 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 $U_j \in \mathtt{pid}_i^{s_i}$ whose oracle $\Pi_j^{s_j}$ has accepted with a different session group key $k_j^{s_j} \neq k_i^{s_i}$. According to Definition 5 this is a successful attack against the MA-security of P. This, however, contradicts to the assumption that P is a MAGKE protocol. □

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

7 Compiler for AKE-Security and MA-Security under Standard Assumptions

7.1 Security-Enhancing Compilers and their Goals

Imagine, there exists a *black-box* implementation of a GKE protocol which should be used by some group application. Assume this GKE implementation provides strong security properties but not all of them are in fact needed for the application. This means that some communication or computation resources are uselessly spent, resulting in a less efficient high-level application.

Assume, on the other hand, that the given GKE implementation does not satisfy all security requirements desired for the particular group application. Instead of designing and implementing a new GKE protocol in an *ad-hoc* fashion, it is desirable to have a generic technique which can be applied to the given *black-box* implementation in order to enhance its security.

We are thus concerned with the following question. What is a good strategy for the implementation of GKE protocols? Of course this depends on the relationship between the protocol and the application using it. Should the protocol be designed for a very specific application (and not likely to be re-used), it might be better to consider all stated requirements in the implementation and optimize the protocol accordingly. However, what to do if the GKE implementation should be flexible and easily modifiable, in order to be reused by various applications without any significant additional effort? Obviously, a good strategy (though not always optimal) is to implement a GKE protocol in a modular way: one starts with the basic implementation that satisfies the most common set of security requirements, then continues with the implementation of optional modules that can be added to provide extended security requirements. The main goal of security-enhancing GKE protocol compilers is to enable secure construction of GKE protocols in such a modular way.

Definition 6 (Security-Enhancing GKE Protocol Compiler $\mathtt{C}$). *A* security-enhancing GKE compiler $\mathtt{C}$ *is a procedure which takes as input a GKE protocol* $\mathtt{P}$ *and outputs a compiled GKE protocol* $\mathtt{C}_{\mathtt{P}}$ *with additional security properties possibly missing in* $\mathtt{P}$.

7.2 Discussion on Existing Compilers for GKE Protocols

In the following we provide some analysis on currently known security-enhancing compilers for GKE protocols.

Each compiler description holds in 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 Π_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).

Compiler for AKE-Security The requirement on KE-security, i.e., key indistinguishability with respect to passive adversaries states the basic security requirement for any GKE protocol. To the contrary, the requirement of AKE-security may be optional. For example, if a network or a high-level application provides authentication implicitly then it is sufficient to use a KE-secure protocol. Therefore, it is reasonable to specify AKE-security as an additional property and design a compiler which adds AKE-security to any KE-secure protocol. Katz and Yung proposed in [33] the following compiler which provides AKE-security. It uses a EUF-CMA digital signature scheme Σ (see Appendix A for details).

Definition 7 (Compiler for AKE-Security by Katz and Yung [33]). *Let $\mathtt{P}$ be a GKE protocol and $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ a digital signature scheme. A* compiler for AKE-security *consists of an initialization algorithm and a modified protocol execution defined as follows:*

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

`The protocol`**:** *This is an interactive protocol between the partnered oracles Π_1^s, ..., Π_n^s invoked prior to any operation execution of $\mathtt{P}$. Each Π_i^s chooses a random nonce $r_i \in_R \{0,1\}^\kappa$ and sends $U_i|0|r_i$ to every partnered oracle Π_j^s. After Π_i^s receives $U_j|0|r_j$ from all partnered oracles it computes $\mathtt{sid}_i^s := U_1|r_1| \ldots |U_n|r_n$. Then, members of $\mathcal{G}$ execute $\mathtt{P}$ with the following changes:*

- *If Π_i^s is supposed to send a message $U_i|t|m$ then it computes additionally $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, t|m|\mathtt{sid}_i^s)$ and outputs a modified message $U_i|t|m|\sigma_i$.*
- *If Π_i^s receives $U_j|t|m|\sigma_j$ it checks whether (1) $U_j \in \mathtt{pid}_i^s$, (2) t is the next expected sequence number, and (3) $\Sigma.\mathtt{Verify}(pk'_j, t|m|\mathtt{sid}_i^s, \sigma_j) \stackrel{?}{=} 1$. If any of these verifications fail then Π_i^s terminates without accepting; otherwise it proceeds according to the specification of $\mathtt{P}$ upon receiving $U_j|t|m$.*
- *After Π_i^s computes the session group key k_i^s in the execution of $\mathtt{P}$ it accepts with this key.*

Missing Generality of Katz-Yung Compiler Katz and Yung proved security of this compiler assuming an unreliable asynchronous broadcast channel and a passive adversary, which is only an *eavesdropper*. We show that in this case their compiler is not really generic: there exist GKE protocols that are secure against eavesdroppers but become insecure against active adversaries (even after the execution of the above compiler).

We consider the following (pathologic, but illustrative) protocol between Π_1^s and Π_1^s. First Π_1^s chooses his exponent $x_1 \in_R \mathbb{Z}_q^\star$ and sends $X_1 := g^{x_1}$ to Π_2^s. If Π_2^s receives X_1 within some specified time period δ then Π_2^s replies with $X_2 := g^{x_2}$ for some randomly chosen $x_2 \in_R \mathbb{Z}_q^\star$ and accepts with the Diffie-Hellman session key $g^{x_1 x_2}$. Similar if Π_1^s receives X_2 within time δ then it accepts with $g^{x_1 x_2}$ too. However, if an oracle does not receive data in time, it accepts with g. If the passive adversary is just an eavesdropper, messages are delivered on time, and the protocol is "passively" secure. But an active adversary can drop messages so that both participants accept with g. Obviously, it is insufficient to restrict passive adversaries to be just eavesdroppers. Passive attacks should also model the unreliability of the communication, like we do.

Further, the compiler in [33] assumes that each sent message is of the form $U_i|t|m$ where t is a sequence number which starts with 0 and is incremented each time an oracle Π_i^s sends a new message. Before any received message is processed by the original protocol `P` the compiler checks whether this message is expected or not with respect to the next expected sequence number. Note that Katz and Yung introduced sequence numbers in order to simplify the description of their proofs. In our setting we can easily omit sequence numbers since any protocol which is KE-secure in our setting resists attacks based on modification of the delivery order of the protocol messages. Note also that sequence numbers do not provide any additional security advantages for the protocol (e.g., they do not protect against replay attacks). Another reason is that in order to check whether a message is expected or not the compiler must explicitly know the total number of messages required in the protocol execution. Thus, these numbers should be additionally given as input to the compiler. This additional effort must be applied for each GKE protocol to be used with the compiler. This can be considered as an additional inconvenience, especially if the protocol's implementation is available as a "black-box" that allows access only to the established group key.

Compilers for MA-Security The first compiler for key confirmation and mutual authentication was proposed by Bresson *et al.* [15]. However, their definition of MA is (*de facto*) the old one, and the proof is conducted in the Random Oracle Model [8].

Katz and Shin [32] showed how to turn an AKE-secure GKE protocol into a UC-secure GKE protocol that provides security against insider attacks (see Section 4).

It requires a EUF-CMA digital signature scheme Σ, and a collision-resistant pseudo-random function f (see Appendix A for details).

Definition 8 (Compiler for Security against Insider Attacks by Katz and Shin [32]). *Let $\mathtt{P}$ be a GKE protocol, $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ a digital signature scheme, $F := \{f_k\}_{k \in \{0,1\}^\kappa}$ a function ensemble with range*

$\{0,1\}^\lambda$, $\lambda \in \mathbb{N}$ *and domain* $\{0,1\}^\kappa$*, and* $\mathtt{sid}_i^s$ *is a unique session id. A* compiler for security against insider attacks *consists of an initialization algorithm and a protocol defined as follows:*

`Initialization`**:** *each* $U_i \in \mathcal{U}$ *generates his own additional private/public key pair* (sk'_i, pk'_i) *using* $\Sigma.\mathtt{Gen}(1^{\kappa'})$.
`The protocol`**:** *After an oracle* Π_i^s *accepts with* $(k_i^s, \mathtt{pid}_i^s, \mathtt{sid}_i^s)$ *in* `P`*:*

- *it computes* $\mu_i := f_{k_i^s}(v_0)$ *where* v_0 *is a constant public value;*
- *it computes* $K_i^s := f_{k_i^s}(v_1)$ *where* $v_1 \neq v_0$ *is another constant public value;*
- *it erases its local state information except for* μ_i*,* K_i^s*,* $\mathtt{pid}_i^s$*, and* $\mathtt{sid}_i^s$*;*
- *it computes a signature* $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ *and sends* $U_i|\sigma_i$ *to its partnered oracle* Π_j^s*.*

After $\Pi_i^{s_i}$ *receives* $U_j|\sigma_j$ *from all its partnered oracle* Π_j^s*:*

- *it checks whether* $\Sigma.\mathtt{Verify}(pk'_j, \mu_i|\mathtt{sid}_i^s|\mathtt{pid}_i^s, \sigma_j) \stackrel{?}{=} 1$*;*
- *if all checks pass, it accepts with the session key* K_i^s*.*

Working in the UC framework, Katz and Shin assume that session ids are unique and specified by some high-level application (see [4] for setting up session ids in the UC framework). In this case the above compiler can be proven to satisfy our MA-security requirement. On the other hand, if such unique session ids are not available then it is obvious that some additional communication rounds are needed to set up such session ids. Intuitively, leaving out session ids would allows replay attacks against the compiled protocol as illustrated in the following.

Imagine, the adversary $\mathcal{A}$ corrupts $n-2$ participants (except for U_i and U_j) in some previous session and behaves honestly in another session. Thus, he learns the key $\bar{k}_i^t$ computed in that session and the message $U_i|\bar{\sigma}_i$ sent by Π_i^t during the compiler round of that session. Remind, $\bar{\sigma}_i$ is computed on $\mathtt{pid}_i^t$ and $\bar{\mu}_i = f_{\bar{k}_i^t}(v_0)$. After the compiled protocol is executed $\mathcal{A}$ invokes a new session with the same protocol participants. Thus, users U_i and U_j participate via fresh oracles Π_i^s and Π_j^s, respectively. Note also that we have $\mathtt{pid}_i^s = \mathtt{pid}_i^t$. Let us assume that $\mathcal{A}$ can influence Π_j^s in some way such that it computes $k_j^s = \bar{k}_i^t$ and Π_i^s computes a different key $k_i^s \neq \bar{k}_i^t$. Then $\mathcal{A}$ intercepts (and drops) the original message $U_i|\sigma_i$ and replays $U_i|\bar{\sigma}_i$ to Π_j^s. Because $\mu_j = f_{k_j^s}(v_0) = f_{\bar{k}_i^t}(v_0) = \bar{\mu}_i$, oracle Π_j^s verifies $\bar{\sigma}_i$ successfully but $k_j^s \neq k_i^s$ (which results in $K_j^s \neq K_i^s$). Thus, uncorrupted oracles Π_i^s and Π_j^s accept with different session keys.

7.3 Compiler `C-AMA`

In the following we describe a compiler (denoted `C-AMA`) which provides both AKE- and MA-security for any GKE protocol `P` which satisfies the basic requirement of (unauthenticated) KE-security wrt. to our security model, i.e., where the passive adversary is given access to the query Send but is restricted not to modify, replay, or introduce protocol messages. `C-AMA` combines KY and KS compilers whereby slight modifications are applied (as mentioned in the previous chapter).

`C-AMA` uses digital signatures and collision-resistant pseudo-random functions (see Appendix A for details) and can be proven secure in the standard model. `C-AMA` uses nonces to achieve uniqueness of protocol sessions and security of concurrent executions without relying on session ids given by high-level applications.

Definition 9 (Compiler for AKE- and MA-Security `C-AMA`). *Let* `P` *be a GKE protocol,* $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ *a digital signature scheme,* $F := \{f_k\}_{k\in\{0,1\}^\kappa}$ *a function ensemble with range* $\{0,1\}^\lambda$*,* $\lambda \in \mathbb{N}$ *and domain* $\{0,1\}^\kappa$*. A* compiler for AKE-security and MA-security*, denoted* `C-AMA`*, consists of an algorithm* `INIT` *and a protocol* `AMA` *defined as follows:*

`INIT`**:** *each* $U_i \in \mathcal{U}$ *generates own private/public key pair* (sk'_i, pk'_i) *using* $\Sigma.\mathtt{Gen}(1^{\kappa'})$.
`AMA`**:** *prior to the execution of* `P`*:*

- *Each* Π_i^s *chooses a* AMA nonce $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|r_i$ *to its partners.*
- *After* Π_i^s *receives all* $U_j|r_j$*, it computes* $\mathtt{sid}_i^s := r_1|\ldots|r_n$*.*

Then it invokes the execution of `P` *and proceeds as follows:*

- *If* Π_i^s *in* `P` *outputs a message* $U_i|m$ *then in* $\mathtt{C\text{-}AMA}_P$ *it computes additionally* $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, m|\mathtt{sid}_i^s|\mathtt{pid}_i^s)$ *and outputs a modified message* $U_i|m|\sigma_i$*.*

- *On receiving $U_j|m|\sigma_j$ from a partner Π_j^s it checks whether $\Sigma.\mathtt{Verify}(pk'_j, m|sid_i^s|pid_i^s, \sigma_j) \stackrel{?}{=} 1$. If this fails then Π_i^s terminates; otherwise it proceeds as in $\mathtt{P}$ upon receiving $U_j|m$.*
- *After an oracle Π_i^s computes k_i^s in $\mathtt{P}$ it computes an* AMA token *$\mu_i := f_{k_i^s}(v_0)$ where v_0 is a constant public value, a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, \mu_i|sid_i^s|pid_i^s)$ and sends $U_i|\sigma_i$ to every Π_j^s with $U_j \in pid_i^s$.*
- *On receiving $U_j|\sigma_j$ from its partner, Π_i^s checks if $\Sigma.\mathtt{Verify}(pk'_j, \mu_i|sid_i^s|pid_i^s, \sigma_j) \stackrel{?}{=} 1$.*
- *If all checks pass, Π_i^s computes $K_i^s := f_{k_i^s}(v_1)$ where $v_1 \neq v_0$ is another constant public value, erases all private state information from $state_i^s$ (including k_i^s) and accepts with K_i^s.*

PERFORMANCE ANALYSIS. C-AMA requires two further rounds: one to exchange random nonces and another one to exchange signatures on AMA tokens. As for the computation costs, every participant generates one signature for every message sent during P and one additional signature on the computed AMA token. Furthermore, every participant must verify one signature for every incoming message in P and $n-1$ signatures during the additional confirmation round. The computation of the AMA token μ and of the session key K can be seen as negligible.

SECURITY ANALYSIS. Our first theorem shows that C-AMA adds AKE-security to any KE-secure GKE protocol. Following Remark 1, we do not consider the adversarial setting (sbs, scm). Well-known definitions of EUF-CMA-security for Σ as well as pseudo-randomness and collision-resistance for F can be found in Appendix A.

Theorem 1 (AKE-Security of $\mathtt{C\text{-}AMA_P}$). *Let $(\alpha, \beta) \in \{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sfs}, \mathtt{scm})\}$ be an adversarial setting, let $\mathtt{P}$ be a GKE-α protocol, and $\mathcal{A}$ an active adversary in the corruption model β launching at most q_s sessions of $\mathtt{C\text{-}AMA}_\mathtt{P}$. If Σ is EUF-CMA and F is pseudo-random then $\mathtt{C\text{-}AMA}_\mathtt{P}$ is AGKE-α, and*

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa) \leq 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^{\kappa-1}} + 2q_\mathsf{s}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 4q_\mathsf{s}\mathsf{Adv}^{\mathsf{prf}}_{F}(\kappa).$$

Proof. In our proofs we use a well-known proving technique called *sequence of games* [44] 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., [1, 13, 14, 26, 27, 34].

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

Game $\mathbf{G}_0$: This game is the real game $\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\mathtt{C\text{-}AMA}_\mathtt{P}}(\mathcal{A}, \kappa)$ played between a simulator $\mathcal{S}$ and an active adversary $\mathcal{A}$. Assume that the Test query is asked to an α-fresh oracle Π_i^s. Keep in mind that on the test query the adversary receives either a random string or a session group key K_i^s.

Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_0$ with the only exception that the simulation fails and bit b' is set at random if $\mathcal{A}$ asks a Send query on some $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $\mathsf{Corrupt}(U_i)$. In other words the simulation fails if $\mathcal{A}$ outputs a successful forgery. In order to estimate $\Pr[\mathsf{Forge}]$ we show that using $\mathcal{A}$ we can construct a EUF-CMA forger $\mathcal{F}$ against the signature scheme Σ as follows. $\mathcal{F}$ is given a public key pk and has access to the corresponding signing oracle. During the initialization of $\mathtt{C\text{-}A_P}$, $\mathcal{F}$ chooses uniformly at random a user $U_{i^*} \in \mathcal{U}$ and defines $pk'_{i^*} := pk$. All other key pairs, i.e., (sk'_i, pk'_i) for every $U_{i \neq i^*} \in \mathcal{U}$ are generated honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$. $\mathcal{F}$ generates also all key pairs (sk_i, pk_i) with $U_i \in \mathcal{U}$ if any are needed for the original execution of $\mathtt{P}$. The forger simulates all queries of $\mathcal{A}$ in a natural way by executing $\mathtt{C\text{-}A_P}$, and by obtaining the necessary signatures with respect to pk'_{i^*} from its signing oracle. This is a perfect simulation for $\mathcal{A}$ since by assumption no $\mathsf{Corrupt}(U_{i^*})$ may occur (otherwise $\mathcal{F}$ would not be able to answer it). Assuming Forge occurs, $\mathcal{A}$ outputs a new valid message/signature pair with respect to some pk'_i; since i^* was randomly chosen and the simulation is perfect, $\Pr[i = i^*] = 1/N$. In that case $\mathcal{F}$ outputs this pair as its forgery. Its success probability is given by $\Pr[\mathsf{Forge}]/N$. This implies

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

Game $\mathbf{G}_2$: This game is identical to $\mathbf{G}_1$ except that the simulation fails and bit b' is set at random if an AMA nonce r_i is used by any uncorrupted user's oracle Π_i^s in two different sessions. If q_s is the total number of protocol

sessions, the probability that a randomly chosen A nonce r_i appears twice is $q_{\mathsf{s}}^2/2^\kappa$ for one particular user. Since there are at most N users we obtain

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

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

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

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

This implies

$$\Pr[\mathsf{Win}_2^{\mathsf{ake}}] = q_{\mathsf{s}}\left(\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \frac{1}{2}\right) + \frac{1}{2}. \tag{3}$$

Game $\mathbf{G}_4$: In this game we consider the simulator $\mathcal{S}$ as a passive adversary against the KE-security of P that participates in $\mathsf{Game}^{\mathsf{ke}-1}_{\alpha,\beta,\mathtt{P}}(\mathcal{S}, \kappa)$, i.e., the Test query of $\mathcal{S}$ to an accepted α-fresh oracle Π_i^s in P is answered with the real session group key k_i^s. In the following we show how $\mathcal{S}$ answers the queries of $\mathcal{A}$.

We focus on the construction w.r.t. the adversarial settings $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$ and describe a simpler construction for the settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ at the end of the proof. Note that in case of $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$ and $(\mathtt{sfs}, \mathtt{scm})$ the active adversary $\mathcal{A}$ is given access to the Corrupt query and can actively participate in the protocol execution via Send queries in any session which is not α-fresh. Thus, if the guess of $\mathcal{S}$ is correct then no active participation of $\mathcal{A}$ in the q_{s}^*-th session is possible. Also none of the oracles participating in the q_{s}^*-th session can be asked for a RevealState query, and after these oracles have accepted none of them can be asked for a RevealKey query either. With these observations in mind we construct $\mathcal{S}$ as follows.

$\mathcal{S}$ corrupts every user $U_i \in \mathcal{U}$ to obtain the long-lived key pair (sk_i, pk_i) used in the original protocol P (if any such keys are defined). Then, $\mathcal{S}$ generates all key pairs (sk_i', pk_i') honestly using $\Sigma.\mathtt{Gen}(1^\kappa)$, and provides the active adversary $\mathcal{A}$ with the set of the public keys $\{pk_i', pk_i\}_{U_i \in \mathcal{U}}$. $\mathcal{S}$ initializes the list *TList* and runs $\mathcal{A}$ as a subroutine.

The idea of the reduction is that in all sessions except for the q_{s}^*-th session $\mathcal{S}$ executes the operation of $\mathtt{C\text{-}AMA}_{\mathtt{P}}$ itself, whereas in the q_{s}^*-th session $\mathcal{S}$ asks own Setup query to obtain a transcript T for the operation execution of P which it extends to a transcript T' for the simulated operation execution of $\mathtt{C\text{-}A}_{\mathtt{P}}$. An entry $(\mathtt{sid}, \perp)$ is saved in *TList* for every session processed directly by $\mathcal{S}$ whereas $(\mathtt{sid}, T')$ is saved for the q_{s}^*-th session. In both cases sid specifies the unique session id used in that session. We need also to consider that $\mathcal{A}$ can invoke the q_{s}^*-th session either via a Setup or an appropriate Send query. The queries of $\mathcal{A}$ are answered by $\mathcal{S}$ as follows.

Setup queries: If $\mathcal{A}$ invokes a protocol session via a $\mathsf{Setup}(\mathcal{S})$ query and the invoked session is not the q_{s}^*-th session then $\mathcal{S}$ executes $\mathtt{C\text{-}AMA}_{\mathtt{P}}$ itself and saves $(\mathtt{sid}, \perp)$ in *TList* where sid is the unique session id built by $\mathcal{A}$ for that session. $\mathcal{S}$ can simulate the operation execution efficiently because it knows the long-lived keys of all users. If the invoked session is the q_{s}^*-th session then $\mathcal{S}$ forwards the received $\mathsf{Setup}(\mathcal{S})$ query as its own query and obtains a transcript T for the execution of $\mathtt{P.Setup}$ between the oracles Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{S}$. Then $\mathcal{S}$ extends T to a transcript T' for the corresponding execution of $\mathtt{C\text{-}AMA}_{\mathtt{P}}.\mathtt{Setup}$ by adding the initial messages of the form $\{U_i|r_i\}_{1\leq i\leq n}$. $\mathcal{S}$ also builds the corresponding q_{s}^*-th session id $\mathtt{sid} := r_1|\ldots|r_n$. Furthermore, for each successive message $U_i|m$ in T the simulator computes a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', m|\mathtt{sid}|\mathtt{pid})$ and appends the modified message $U_i|m|\sigma_i$ to T'. Then $\mathcal{S}$ asks own

Test query to any oracle activated via the Setup query in the q_{s}^*-th session and obtains (real) k which it then uses to compute the AMA token $\mu := f_k(v_0)$. Then, $\mathcal{S}$ computes $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', \mu|\mathtt{sid}|\mathtt{pid})$ and appends messages of the form $\{U_i|\sigma_i\}_{1\le i\le n}$ to T'. $\mathcal{S}$ saves $(\mathtt{sid}, T')$ in *TList* and gives T' to $\mathcal{A}$.

Send queries: By Send_0 we define the query of the form $\mathsf{Send}(\Pi_i^s, \mathcal{S})$ which invokes a new operation execution for Π_i^s. Consequently the second Send query to the same oracle Π_i^s should include messages of the form $U_j|r_j$ for each $U_{j\neq i}$ who holds an oracle in $\mathcal{S}$. We denote such second query as $\mathsf{Send}_1(\Pi_i^s, (U_1|r_1)|\ldots|(U_n|r_n))$ whereby each U_j that is part of $(U_j|r_j)$ is different from U_i. Note that Π_i^s must have received a Send_0 query before in order to be able to answer the Send_1 query.

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

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

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

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

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

RevealState queries: If $\mathcal{A}$ asks a query of the form $\mathsf{RevealState}(\Pi_i^s)$ then $\mathcal{S}$ finds an entry $(\mathtt{sid}_i^s, T^*)$ in *TList*. If $T^* = \perp$ it means that $\mathcal{S}$ executes the protocol itself and is, therefore, able to answer this query directly. If $T^* = T'$ then $\mathcal{S}$ checks whether Π_i^s has already accepted. In this case $\mathcal{S}$ asks its own RevealState query to obtain $\mathtt{state}_i^s$ and replies accordingly. Note that if Π_i^s has not yet accepted in $\mathtt{C\text{-}AMA}_{\mathtt{P}}.\mathtt{Setup}$ then an empty string is returned.

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

Test query: Note that in this game we are dealing with the Test query asked to an oracle Π_i^s that has participated in the q_{s}^*-th session. The simulator $\mathcal{S}$ already knows k_i^s since it has already asked own Test query to build the transcript T' straight after the invocation of the q_{s}^*-th session. Thus, $\mathcal{S}$ computes the resulting session group key $K_i^s := f_{k_i^s}(v_1)$ as specified in $\mathtt{C\text{-}AMA}_{\mathtt{P}}$, chooses a random bit $b \in_R \{0,1\}$ and returns K_i^s if $b = 1$ or a random string sampled from $\{0,1\}^\kappa$ if $b = 0$.

This provides a perfect simulation for $\mathcal{A}$. Since $\mathcal{S}$ uses the real k_i^s to derive K_i^s we can consider this game as a "bridging step" so that

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \Pr[\mathsf{Win}_3^{\mathsf{ake}}]. \tag{4}$$

When dealing with the adversarial settings $(\emptyset, \texttt{wcm})$ and $(\texttt{wbs}, \texttt{wcm-bs})$ the above simulation can be simplified since no Corrupt queries need to be considered.

More precisely, when $\mathcal{A}$ asks for a $\mathsf{Setup}(\mathcal{S})$ query $\mathcal{S}$ first forwards it to obtain a transcript T of the execution of $\mathtt{P}.\mathtt{Setup}$. Next, $\mathcal{S}$ chooses random nonces $r_i \in_R \{0,1\}^\kappa$ for every oracle Π_i^s in $\mathcal{G}$ which is composed of the ordered oracles in $\mathcal{G}$, computes $\mathtt{sid}$ and specifies $\{U_i|r_i\}_{1\le i\le n}$ as the initial messages in T'. Then, for each successive message $U_i|m$ in T the passive adversary $\mathcal{A}'$ computes a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk_i', m|\mathtt{sid}|\mathtt{pid})$ and appends the modified message $U_i|m|\sigma_i$ to T'. Then, $\mathcal{S}$ asks own Test query to obtain (real) k which it then uses to compute the AMA token μ and required digital signatures as described above. Finally, appends corresponding messages to T', saves $(\mathtt{sid}, T')$ in *TList* and gives T' to $\mathcal{A}$.

Since in this case we are not dealing with Corrupt queries the simulator can answer all Send queries from the predefined transcripts. All other queries, i.e., RevealState (only in the setting $(\texttt{wbs}, \texttt{wcm-bs})$), RevealKey, and Test asked by $\mathcal{A}$ are answered by $\mathcal{S}$ similar to the settings $(\texttt{wfs}, \texttt{wcm-fs})$ and $(\texttt{sfs}, \texttt{scm})$.

Game $\mathbf{G}_5$: In this game we consider the simulator $\mathcal{S}$ as a passive adversary against the KE-security of $\mathtt{P}$ that participates in $\mathsf{Game}^{\mathsf{ke}-0}_{\alpha,\beta,\mathtt{P}}(\mathcal{S},\kappa)$, i.e., the Test query of $\mathcal{S}$ to an accepted α-fresh oracle Π_i^s in $\mathtt{P}$ is answered with a random bit string instead of the real key k_i^s. $\mathcal{S}$ answers all queries of $\mathcal{A}$ exactly as described in $\mathbf{G}_4$. By a "hybrid argument" we obtain

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_5] - \Pr[\mathsf{Win}^{\mathsf{ake}}_4]| \le \mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa). \tag{5}$$

Game $\mathbf{G}_6$: This game is identical to $\mathbf{G}_5$ except that in the q_{s}^*-th session K and the AMA token μ are replaced by random values sampled from $\{0,1\}^\kappa$. Recall that k used to compute K and μ is uniform according to $\mathbf{G}_5$. Hence,

$$|\Pr[\mathsf{Win}^{\mathsf{ake}}_6] - \Pr[\mathsf{Win}^{\mathsf{ake}}_5]| \le 2\mathsf{Adv}^{\mathsf{prf}}_F(\kappa). \tag{6}$$

Obviously, in this game $\mathcal{A}$ gains no advantage from the obtained information and cannot, therefore, guess b better than by a random choice, i.e.,

$$\Pr[\mathsf{Win}^{\mathsf{ake}}_6] = \frac{1}{2} \tag{7}$$

Considering Equations 1 to 7 we get:

$$\begin{aligned}
\Pr[\mathsf{Game}^{\mathsf{ake}-b}_{\alpha,\beta,\texttt{C-AMA}_\mathtt{P}}(\kappa) = b] &= \Pr[\mathsf{Win}^{\mathsf{ake}}_0] \\
&\le N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + \Pr[\mathsf{Win}^{\mathsf{ake}}_2] \\
&= N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\left(\Pr[\mathsf{Win}^{\mathsf{ake}}_3] - \frac{1}{2}\right) + \frac{1}{2} \\
&\le N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + \\
&\quad 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa) + \frac{1}{2}.
\end{aligned}$$

This results in the desired inequality

$$\mathsf{Adv}^{\mathsf{ake}}_{\alpha,\beta,\texttt{C-AMA}_\mathtt{P}}(\kappa) \le 2N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa-1}} + 2q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{ke}}_{\alpha,\beta,\mathtt{P}}(\kappa) + 4q_{\mathsf{s}}\mathsf{Adv}^{\mathsf{prf}}_F(\kappa).$$

□

Our next theorem shows that C-AMA also provides MA-security for any GKE protocol P. Note that our definition of MA-security allows malicious participants. Therefore, opposed to the BCPQ compiler, we are not concerned about the AKE-security of P.

Theorem 2 (MA-Security of $\texttt{C-AMA}_\mathtt{P}$). *Let P be a GKE protocol and $\mathcal{A}$ an active adversary launching at most q_{s} sessions of $\texttt{C-AMA}_\mathtt{P}$. If Σ is EUF-CMA and F is collision-resistant then $\texttt{C-AMA}_\mathtt{P}$ is* MAGKE*, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\texttt{C-AMA}_\mathtt{P}}(\kappa) \le N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^\kappa} + q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_F(\kappa).$$

Proof. We define a *sequence of games* $\mathbf{G}_i$, $i = 0, \ldots, 2$ and corresponding events $\mathsf{Win}_i^{\mathsf{ma}}$ meaning that $\mathcal{A}$ wins in $\mathbf{G}_i$. The queries made by $\mathcal{A}$ are answered by a simulator $\mathcal{S}$.
Game $\mathbf{G}_0$: This game is the real game $\mathsf{Game}^{\mathsf{ma}}_{\mathtt{C\text{-}AMA}_{\mathtt{P}}}(\mathcal{A}, \kappa)$ played between $\mathcal{S}$ and $\mathcal{A}$. Note that the goal of $\mathcal{A}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle Π_i^s accepts with K_i^s and another user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts and either does not have a corresponding oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$ or has such an oracle but this oracle accepts with $K_j^s \neq K_i^s$.
Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_0$ with the only exception that the simulation fails if $\mathcal{A}$ asks a Send query on a message $U_i|m|\sigma$ (or $U_i|\sigma$) such that σ is a valid signature that has not been previously output by an oracle Π_i^s before querying $\mathsf{Corrupt}(U_i)$, i.e., the simulation fails if $\mathcal{A}$ outputs a successful forgery. According to Equation 1 we obtain,

$$|\Pr[\mathsf{Win}_1^{\mathsf{ma}}] - \Pr[\mathsf{Win}_0^{\mathsf{ma}}]| \leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) \tag{8}$$

Game $\mathbf{G}_2$: This game is identical to $\mathbf{G}_1$ except that the simulator aborts if an AMA nonce r_i is used by any uncorrupted user U_i in two different sessions. Similar to Equation 2 we get

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} \tag{9}$$

Note that this prevents attacks where Π_i^s during any session of the AMA protocol receives a replayed message of the form $U_j|m|\bar{\sigma}_j$ or $U_j|\bar{\sigma}_j$ where U_j is uncorrupted and $\bar{\sigma}_j$ is a signature computed by its oracle in some previous session. Note that Π_i^s does not accept unless it successfully verifies all required σ_j for all $U_j \in \mathtt{pid}_i^s$ in the AMA protocol of C-AMA. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}_i^s$ that is uncorrupted at the time Π_i^s accepts there exists a corresponding instance oracle Π_j^s with $(\mathtt{pid}_j^s, \mathtt{sid}_j^s) = (\mathtt{pid}_i^s, \mathtt{sid}_i^s)$. Thus, according to Definition 5 $\mathcal{A}$ wins in this game only if any of these oracles has accepted with $K_j^s \neq K_i^s$.

Assume that $\mathcal{A}$ wins in this game. Then there exist two uncorrupted oracles Π_i^s and Π_j^s that have accepted with $K_i^s = f_{k_i^s}(v_1)$ resp. $K_j^s = f_{k_j^s}(v_1)$ where k_i^s resp. k_j^s are corresponding keys computed during the execution of P such that $K_i^s \neq K_j^s$. Having eliminated forgeries and replay attacks between the oracles of any two uncorrupted users we follow that messages exchanged between Π_i^s and Π_j^s have been delivered without any modification. In particular, oracle Π_i^s received the signature σ_j computed on $\mu_j = f_{k_j^s}(v_0)$ and Π_j^s received the signature σ_i computed on $\mu_i = f_{k_i^s}(v_0)$. Since both oracles have accepted we have $\mu_i = \mu_j$; otherwise oracles cannot have accepted because signature verification would fail. The probability that $\mathcal{A}$ wins in this game is given by

$$\Pr[K_i^s \neq K_j^s \wedge f_{k_i^s}(v_0) = f_{k_j^s}(v_0)] = \Pr[f_{k_i^s}(v_1) \neq f_{k_j^s}(v_1) \wedge f_{k_i^s}(v_0) = f_{k_j^s}(v_0)] \leq q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa).$$

Thus

$$\Pr[\mathsf{Win}_2^{\mathsf{ma}}] \leq q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa). \tag{10}$$

Considering Equations 8 to 10 we get the desired inequality

$$\begin{aligned}\mathsf{Succ}^{\mathsf{ma}}_{\mathtt{C\text{-}AMA}_{\mathtt{P}}}(\kappa) &= \Pr[\mathsf{Win}_0^{\mathsf{ma}}] \\ &\leq N\mathsf{Succ}^{\mathtt{euf-cma}}_{\Sigma}(\kappa) + \frac{Nq_{\mathsf{s}}^2}{2^{\kappa}} + q_{\mathsf{s}}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa).\end{aligned}$$

□

8 Conclusion

In this paper we found some problems with the definition of MA-security in the foundational BCPQ model. We proposed a revised definition which considers malicious participants and unifies many of the well-known informal notions. Additionally, we extended the (strong) forward secrecy in AKE-security by the symmetrically opposed notion of (strong) backward secrecy. Further we described the provably secure generic compiler C-AMA that adds AKE- and MA-security to any GKE protocol which is passively secure (wrt. to an adversary which can change the delivery order of messages, and delay or drop them).

We refer to our parallel work in [16, 38] for further development, in particular, in the light of the *contributory* nature of GKE protocols and the appropriate generic solution for this security goal.

References

1. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-Based Group Key Exchange in a Constant Number of Rounds. In *Proc. of the 9th Intl. Workshop on Theory and Practice in Public Key Cryptography (PKC'06)*, volume 3958 of *LNCS*, pages 427–442. Springer, April 2006.
2. J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In *Advances in Cryptology – EUROCRYPT'02*, volume 2332 of *LNCS*, pages 83–107. Springer, 2002.
3. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proc. of the 5th ACM Conf. on Computer and Communications Security (CCS'98)*, pages 17–26. ACM Press, 1998.
4. B. Barak, Y. Lindell, and T. Rabin. Protocol Initialization for the Framework of Universal Composability. Cryptology ePrint Archive, Report 2004/006, 2004. `http://eprint.iacr.org/2004/006.pdf`.
5. K. Becker and U. Wille. Communication Complexity of Group Key Distribution. In *Proc. of the 5th ACM Conf. on Computer and Communications Security (CCS'98)*, pages 1–6. ACM Press, 1998.
6. M. Bellare. Practice-Oriented Provable-Security. In *Proc. of the First Intl. Workshop on Information Security (ISW'97)*, volume 1396 of *LNCS*, pages 221–231. Springer, 1998.
7. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology–CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.
8. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of the 1st ACM Conf. on Computer and Communications Security (CCS'93)*, pages 62–73. ACM Press, 1993.
9. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proc. of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM Press, 1995.
10. C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003. ISBN:3-540-43107-1.
11. C. Boyd and J. M. Nieto. Round-Optimal Contributory Conference Key Agreement. In *Proc. of PKC'03*, volume 2567 of *LNCS*, pages 161–174. Springer, 2003.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case. In *Advances in Cryptology – ASIACRYPT'01*, volume 2248 of *LNCS*, pages 290–390. Springer, 2001.
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 *LNCS*, pages 321–336. Springer, 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 *LNCS*, 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 *Proc. of the 8th ACM Conf. on Computer and Communications Security (CCS'01)*, pages 255–264. ACM Press, 2001.
16. E. Bresson and M. Manulis. Malicious Participants in Group Key Exchange: Key Control and Contributiveness in the Shadow of Trust. In *Proc. of the 4th Intl. Conf. on Autonomic and Trusted Computing (ATC'07)*, LNCS vol. 4610, pp. 395–409. Springer, July, 2007.
17. M. Burmester. On the Risk of Opening Distributed Keys. In *Advances in Cryptology – CRYPTO'94*, volume 839 of *LNCS*, pages 308–317. Springer, August 1994.
18. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *LNCS*, pages 275–286. Springer, May 1994.
19. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
20. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proc. of 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE CS, 2001.
21. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - EUROCRYPT'01*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.
22. 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 *LNCS*, pages 585–604. Springer, 2005.
23. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.
24. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.
25. R. Dutta and R. Barua. Constant Round Dynamic Group Key Agreement. In *Information Security: 8th Intl. Conf. (ISC'05)*, volume 3650 of *LNCS*, pages 74–88. Springer, August 2005.
26. R. Dutta and R. Barua. Dynamic Group Key Agreement in Tree-Based Setting. In *Proc. of the 10th Australasian Conf. on Information Security and Privacy (ACISP'05)*, volume 3574 of *LNCS*, pages 101–112. Springer, 2005.

27. R. Dutta, R. Barua, and P. Sarkar. Provably Secure Authenticated Tree Based Group Key Agreement. In *Proc. of the 6th Intl. Conf. on Information and Communications Security (ICICS'04)*, volume 3269 of *LNCS*, pages 92–104. Springer, 2004.
28. M. Fischlin. Pseudorandom Function Tribe Ensembles Based on One-Way Permutations: Improvements and Applications. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 432–445. Springer, 1999.
29. O. Goldreich. *Foundations of Cryptography - Basic Tools*, volume 1. Cambridge University Press, 2001. ISBN:0-521-79172-3.
30. C. G. Günther. An Identity-Based Key-Exchange Protocol. In *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *LNCS*, 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. J. Katz and J. S. Shin. Modeling Insider Attacks on Group Key-Exchange Protocols. In *Proc. of the 12th ACM Conf. on Computer and Communications Security (CCS'05)*, pages 180–189. ACM Press, 2005.
33. J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *Advances in Cryptology - CRYPTO'03*, volume 2729 of *LNCS*, pages 110–125. Springer, 2003.
34. 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 *LNCS*, pages 245–259, 2004.
35. Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *Proc. of the 7th ACM Conf. on Computer and Communications Security (CCS'00)*, pages 235–244. ACM Press, 2000.
36. Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proc. of IFIP TC11 16th Annual Working Conf. on Information Security (IFIP/Sec'01)*, volume 193 of *IFIP Conf. Proc.*, pages 229–244. Kluwer, 2001.
37. N. Koblitz and A. Menezes. Another Look at "Provable Security". *Journal of Cryptology*, 2006. Online Issue, 30. November 2005. Also available at `http://eprint.iacr.org/2005/152.pdf`.
38. M. Manulis. *Provably Secure Group Key Exchange*. PhD thesis, Ruhr University Bochum, June 2007.
39. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.
40. O. Pereira and J.-J. Quisquater. A Security Analysis of the CLIQUES Protocols Suites. In *Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 73–81. IEEE Computer Society Press, June 2001.
41. O. Pereira and J.-J. Quisquater. Some Attacks upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 11(4):555–580, 2003.
42. A. Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In *Proc. of the Intl. Workshop on Cryptographic Techniques and Electronic Commerce 1999*, pages 192–202. City University of Hong Kong Press, 1999.
43. 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/`.
44. 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`.
45. D. G. Steer, L. Strawczynski, W. Diffie, and M. J. Wiener. A Secure Audio Teleconf. System. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *LNCS*, pages 520–528. Springer, 1990.
46. M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proc. of the 3rd ACM Conf. on Computer and Communications Security (CCS'96)*, pages 31–37. ACM Press, 1996.
47. M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proc. of the 18th Intl. Conf. on Distributed Computing Systems (ICDCS'98)*, pages 380–387. IEEE Computer Society Press, 1998.
48. Y. Yacobi and Z. Shmuely. On Key Distribution Systems. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *LNCS*, pages 344–355. Springer, August 1990.

A Cryptographic Tools Used by the Compiler C-AMA

Definition 10 (Digital Signature Scheme). *A signature scheme* $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ *consists of the following algorithms:*

`Gen`: *A probabilistic algorithm that on input a security parameter* 1^κ, $\kappa \in \mathbb{N}$ *outputs a secret key* sk *and a public key* pk.

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

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

Definition 11 (EUF-CMA Security[8]). *A digital signature scheme* $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ *is said to be* existentially unforgeable under chosen message attacks *(EUF-CMA) if for any PPT algorithm (forger)* $\mathcal{F}$ *that receives a public key* pk *and can access to a signing oracle* $\mathtt{Sign}(sk, \cdot)$*, the probability that* $\mathcal{F}$ *outputs a pair* (m, σ) *such that* $\mathtt{Verify}(pk, m, \sigma) = 1$ *but* m *was never part of a query* $\mathtt{Sign}(sk, m)$ *is negligible. By* $\mathsf{Succ}^{\mathtt{euf-cma}}_{\mathcal{F},\Sigma}(\kappa)$ *we denote the probability that* $\mathcal{F}$ *outputs a successful forgery.*

In the following we briefly describe the notion of pseudo-random functions. Informally, a pseudo-random function (PRF) is specified by a 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. The following definition is taken from [29, Definition 3.6.9].

Definition 12 (Efficiently Computable Generalized Pseudo-Random Function Ensemble F**).** *An ensemble of finite functions* $F := \left\{ \left\{ f_k : \{0,1\}^{p(\kappa)} \rightarrow \{0,1\}^{p(\kappa)} \right\}_{k \in \{0,1\}^\kappa} \right\}_{\kappa \in \mathbb{N}}$ *where* $p : \mathbb{N} \rightarrow \mathbb{N}$ *is upper-bounded by a polynomial, is called an* (efficiently computable) pseudo-random function ensemble *if the following two conditions hold:*

1. Efficient computation: *There exists a polynomial-time algorithm that on input* k *and* $x \in \{0,1\}^{p(\kappa)}$ *returns* $f_k(x)$.
2. Pseudo-randomness: *Choose uniformly* $k \in_R \{0,1\}^*$ *and a function* $\tilde{f}$ *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* Tag(x) *and participating in one of the following two games:*
   - $\mathsf{Game}^{\mathsf{prf}-1}_F(\mathcal{A}, \kappa)$ *where a query* Tag(x) *is answered with* $f_k(x)$*,*
   - $\mathsf{Game}^{\mathsf{prf}-0}_F(\mathcal{A}, \kappa)$ *where a query* 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 of* $\mathcal{A}$ *in winning the game is defined as*

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

   We say that F *is pseudo-random if* $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{A},F}(\kappa)$ *is negligible.*

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

Remark 2. As noted in [29] there are some significant differences between using PRFs and the Random Oracle Model (ROM) [8]. 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 [32]. The same property has previously been defined in [28] and denoted there as *fixed-value-key-binding* property of a pseudo-random function ensemble. We also refer to [32] for a possible construction based on one-way permutations and for the proof of Lemma 1).

[8] There exists a stronger security requirement called *strong EUF-CMA* [2]. It allows $\mathcal{F}$ to produce a forgery (m, σ) for a message m that was already queried to the signing oracle, provided that σ was not returned by the signing oracle. However, in our compiler we do not need this stronger property.

Definition 13 (Collision-Resistance of F). *Let F be a pseudo-random function ensemble. We say that F is* collision-resistant *if there is an efficient procedure* `Sample` *such that the following success probability (over all PPT adversaries $\mathcal{A}$) is a negligible function in κ:*

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

Lemma 1. *If one-way permutations exist then there exist collision-resistant pseudo-random functions.*