Extended Definitions of AKE- and MA-Security for Group Key Exchange Protocols (Full Version)

Emmanuel Bresson[1] and Mark Manulis[2]

[1] DCSSI Crypto Lab, France
emmanuel.bresson@polytechnique.org
[2] Horst-Görtz Institute, Ruhr-University of Bochum, Germany
mark.manulis@rub.de

Abstract. Group key exchange (GKE) protocols can be used to guarantee confidentiality and group authentication in a variety of group applications. The notion of provable security subsumes the existence of an abstract formalization (security model) that considers the environment of the protocol and identifies its security goals. The first security model for GKE protocols was proposed by Bresson, Chevassut, Pointcheval, and Quisquater in 2001, and has been subsequently applied in many security proofs. Their definitions of AKE- and MA-security became meanwhile standard. In this paper we show that the technical construction of their model opens some problems w.r.t. the definition of MA-security. We revise the original model and provide extended definitions for AKE- and MA-security considering attacks of malicious participants and a new notion of backward secrecy. Finally, we describe a generic solution (compiler) which provides both of these requirements and show feasibility of our model by proving security of this compiler using standard cryptographic assumptions.

Key words: group key exchange, extended security model, malicious participants, compiler for AKE- and MA-security

1 Introduction

MOTIVATION. Security of many privacy-preserving multi-party applications like encrypted group communication for audio/video conferences, chat systems, computer-supported collaborative workflow systems, or secure server replication systems depends on group key exchange (GKE) protocols. Security of those latter is therefore critical. The paradigm of *provable security* is used across the modern literature to prove in a mathematical way, and under reasonable assumptions, that a cryptographic scheme achieves the required security goals. Such proofs are usually constructed using a formal setting that specifies: (1) the computing environment (involved users, their trust relationship, cryptographic parameters, communication...), (2) the adversarial environment and (3) the definitions of some concrete security goals. Security of earlier GKE protocols [3,5,17,29,33,34,39,41–43] has been analyzed heuristically based on informal definitions so that some of them have been broken later, e.g., [37,38]. In particular some fundamental notions, like key secrecy [22], resistance against known-key attacks [16, 44], implicit and explicit key authentication [17, 36], and forward secrecy [28, 36] were not formally defined at that time. In 2001 Bresson, Chevassut, Pointcheval, and Quisquater [15] introduced the first computational (game-based) security model (referred to as the BCPQ model) designed for GKE protocols. They adopted some ideas previously proposed by Bellare and Rogaway [7, 9] in the context of two- and three-party key establishment protocols. The BCPQ model, as well as its refinements [12, 13] and variants [14, 25, 30, 31], have been meanwhile used in many GKE security proofs including [1, 11–15, 24, 25, 30–32] and became *de facto* standard. Therefore, it is immense important that the definitions provided by these models are correct and general enough to be applicable for *any* GKE protocol, regardless of its concrete specification.

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., [30, 31]. 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 between the technical core of the BCPQ model – 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., [25,30,31], 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 propose in Section 5 an extended computational security model with the main goal to revise the definition of MA-security such that it considers attacks of malicious participants. Our extended model is based on the more powerful BCPQ refinement from [13] that considers AKE-security in the presence of (partial) internal state corruptions. We also introduce an additional notion of *backward secrecy* which leads to new corruption models in case of AKE-security.

In order to show that our extended model is feasible, i.e., practical enough to construct reductionist security proofs, in Section 6 we describe a compiler C-AMA that satisfies our stronger definitions of AKE- and MA-security for any GKE protocol and prove its security under standard cryptographic assumptions. Our compiler is actually a combination of the well-known compiler for AKE-security from [31] (we use a slightly modified version) and the compiler proposed in [30] that achieves security against insider attacks in the sense of [30].

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 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 $V := \{\Pi_l^{s_l} | \, U_l \in ID, \; l = 1, \ldots, n\}$ and $E := \{(\Pi_l^{s_l}, \Pi_{l'}^{s_{l'}}) | \, \Pi_l^{s_l} \leftrightarrow \Pi_{l'}^{s_{l'}}\}$, 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 $\mathrm{PIDS}(\Pi_i^{s_i}) = \{\Pi_l^{s_l} \mid \Pi_i^{s_i} \leftrightsquigarrow \Pi_l^{s_l} \; \forall \, l \in \{1, n\} \setminus \{i\}\}$.

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 $\Pi_i^{s_i}$

[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.

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* [22] (a.k.a *implicit key authentication* [36]) 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* [16, 44] meaning that an adversary who knows group keys of previous sessions must not be able to compute subsequent session keys, *key independence* [33] 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* [22,28,36] 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* [36] meaning that each protocol participant must be assured that every other protocol participant actually has possession of the computed group key,
- *explicit key authentication* [36], 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.

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.

[4] introduced originally for two-party protocols

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.

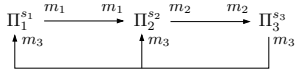

Fig. 1. Honest execution of protocols in [12–15]. By m_i at the beginning of the arrow we mean the original message sent by the oracle, and by m_i at the end of the arrow we mean the corresponding message received by another oracle.

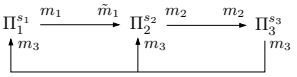

Fig. 2. Protocol execution where $\mathcal{A}$ impersonates U_1

Obviously, Figure 1 shows a correct execution of the protocol since no message is modified. Figure 3 specifies the session ids of the oracles $\Pi_1^{s1}, \ldots, \Pi_3^{s3}$ 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 participating oracle $\Pi_i^{s_i}$ with $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ mutual authentication and key confirmation are not necessarily provided.

The protocol proceeds as described in Figure 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 $K = g^{x_1 x_2 x_3}$ 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 $[Fl_1]_{U_1}$ and replays a

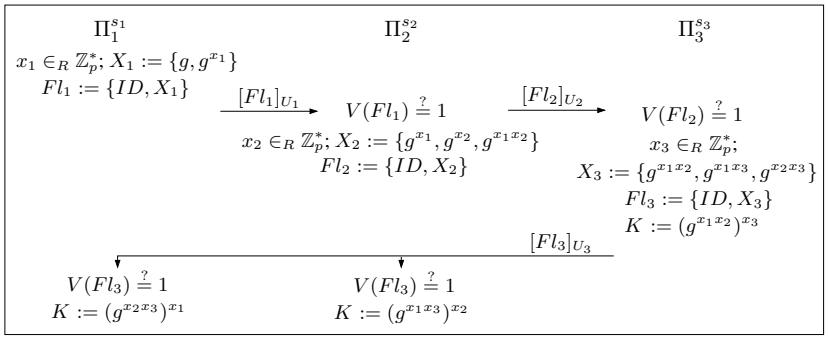

Fig. 5. Execution of the protocol in [15] with three participants

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, similarly to the abstract problem description above, we show that $|\mathrm{PIDS}(\Pi_i^{s_i})| = 2$ for every $\Pi_i^{s_i}$, $i \in \{1, 2, 3\}$, i.e., $\mathrm{SID}(\Pi_1^{s_1}) \cap \mathrm{SID}(\Pi_3^{s_3}) = \{[Fl_1]_{U_1}, [Fl_3]_{U_3}\} \cap \{[Fl_3]_{U_3}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} = [Fl_3]_{U_3}$ and $\mathrm{SID}(\Pi_3^{s_3}) \cap \mathrm{SID}(\Pi_2^{s_2}) = \{[Fl_3]_{U_3}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} \cap \{[\widetilde{Fl_1}]_{U_1}, [Fl_2]_{U_2}|[Fl_3]_{U_3}\} = [Fl_2]_{U_2}|[Fl_3]_{U_3}$.

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 [21] 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 [30] 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 [31] (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 a 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.* [25] 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, [23] 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 [25] and [23] had no intentions to consider mutual authentication and key confirmation.

A VARIANT BY KATZ AND SHIN. Katz and Shin [30] 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) [19]. 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 [18] rather than on a reductionist approach [6, 35].

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 Revised and Extended Computational Security Model for GKE Protocols

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. Each user $U_i \in \mathcal{U}$ may hold a long-lived key LL_i generated by some probabilistic algorithm $\mathtt{GenLL}(1^\kappa)$, and can have an unlimited number of instances Π_i^s, $s \in \mathbb{N}$ called *oracles* which can be involved in distinct *concurrent* executions of the GKE protocol $\mathtt{P}$. For each concurrent execution of $\mathtt{P}$ we consider a *group* $\mathcal{G} \subseteq \mathcal{U}$ of size $n \in [1, N]$. For every participating *group member* $U_i \in \mathcal{G}$ there exists a corresponding instance Π_i^s, so we sometimes use $\mathcal{G}$ to denote the set of oracles of group members involved in the execution of $\mathtt{P}$. Every Π_U^s maintains an *internal state information* $\mathtt{state}_U^s$ which is private and used by the oracle to compute the group key.

SESSION GROUP KEY, SESSION ID, PARTNER ID. During each execution each Π_i^s computes the *session group key* $k_i^s \in \{0,1\}^\kappa$. Every session is identified by a *session id* $\mathtt{sid}$. We stress that unique session ids are either provided by a high-level application (which implicitly assumes that such session ids are available within the environment) or specified in the actual protocol description prior to the protocol execution (for example via nonces). We define the *partner id* of $\Pi_i^{s_i}$ as $\mathtt{pid}_i^{s_i} := \{U_j \,|\, U_j \in \mathcal{G}\}$ and 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 $\mathtt{sid}_i^{s_i} = \mathtt{sid}_j^{s_j}$. Intuitively, oracles of all members who participate in the same session of $\mathtt{P}$ are considered as partners.

INSTANCE ORACLE STATES: STAND-BY, PROCESSING, ACCEPTED, TERMINATED. Once an oracle Π_U^s is initialized it turns into a *stand-by* state where it waits for the protocol execution to be invoked. Upon receiving such invocation Π_U^s learns $\mathtt{pid}_U^s$ (and possibly $\mathtt{sid}_U^s$) and turns into a *processing* state where it communicates and processes messages according to the protocol specification, maintaining $\mathtt{state}_U^s$. The oracle Π_U^s remains in this state until it collects enough information to compute k_U^s. As soon as k_U^s is computed Π_U^s *accepts*. Then processing of messages may continue until the oracle finally *terminates*. If the protocol execution fails then Π_U^s terminates without having accepted, i.e., k_U^s is set to some undefined value. Acceptance can be modeled by a boolean variable which is initially set to false and toggles to true as soon as k_U^s is set. Note that once the oracle is terminated it can no longer be initialized. The oracle is said to be *used* if it has been already initialized.

5.2 Definition of a Group Key Exchange Protocol

Definition 1 (GKE Protocol). *A group key exchange protocol $\mathtt{P}$ is an interactive protocol between the oracles $\Pi_1^{s_1}$, ..., $\Pi_n^{s_n}$ with $U_i \in \mathcal{G}$ until all oracles turn into the terminated state.*

CORRECTNESS. The following definition allows to exclude "useless" GKE protocols.

Definition 2 (Correctness). *A GKE protocol $\mathtt{P}$ is* correct *if for any honest protocol execution between the oracles $\Pi_1^{s_1}$, ..., $\Pi_n^{s_n}$ with $U_i \in \mathcal{G}$, $\mathtt{sid}_1^{s_1} = \ldots = \mathtt{sid}_n^{s_n}$, all oracles accept with the same session group key k.*

All schemes in this paper are assumed to satisfy the correctness requirement.

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:

- Setup($\mathcal{G}$): The GKE protocol P is executed between the unused oracles $\Pi_1^{s_1}$, ..., $\Pi_n^{s_n}$, each related to a corresponding group member $U_i \in \mathcal{G}$. $\mathcal{A}$ is given the transcript of the execution.
- Send(Π_U^s, m): $\mathcal{A}$ receives the response which Π_U^s would have generated after having processed the message m according to the description of P (this can be the empty string if m is incorrect). A new execution of P is invoked via a Send('setup', $\Pi_U^s, \mathcal{G}$) query.
- RevealKey(Π_U^s): $\mathcal{A}$ is given the session group key k_U^s. This query is answered only if Π_U^s has accepted.
- RevealState(Π_U^s): $\mathcal{A}$ is given the internal state information $\texttt{state}_U^s$.[5]
- Corrupt(U): $\mathcal{A}$ is given the long-lived key LL_U.
- Test(Π_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 only ask queries Setup, RevealKey, and Test, while an *active* adversary can additionally execute the query Send. In addition, they might be allowed to ask RevealState and/or Corrupt queries, depending on the considered scenario. These separations are used to define security goals in a modular way.

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[6].

Note that in our definition of AKE-security (Definition 6) 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 $\alpha \in \{\texttt{wbs}, \texttt{wfs}, \texttt{sbs}, \texttt{sfs}\}$ leads to a different definition of freshness.

Definition 3 (α-Freshness). *In the execution of P the oracle Π_U^s is*

- *wbs-fresh if: (1) no RevealState queries have been made by $\mathcal{A}$ in the current or any further execution, and (2) after Π_U^s has accepted neither it nor any of its partners has been asked for a RevealKey query until the next execution of P;*

[5] This kind of the adversarial query has previously been mentioned by Canetti and Krawczyk in their model for two-party protocols [20].

[6] 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.

– $\texttt{wfs}$*-fresh if: (1) no* Corrupt *queries have been made by* $\mathcal{A}$ *in the current or any previous execution, and (2) after* Π_U^s *has accepted neither it nor any of its partners has been asked for a* RevealKey *query until the next execution of* $\texttt{P}$*;*
– $\texttt{sbs}$*-fresh if: (1) neither* Corrupt *nor* RevealState *queries have been made by* $\mathcal{A}$ *in the current or any further execution, and (2) after* Π_U^s *has accepted neither it nor any of its partners has been asked for a* RevealKey *query until the next execution of* $\texttt{P}$*;*
– $\texttt{sfs}$*-fresh if: (1) neither* Corrupt *nor* RevealState *queries have been made by* $\mathcal{A}$ *in the current or any previous execution, and (2) after* Π_U^s *has accepted neither it nor any of its partners has been asked for a* RevealKey *query until the next execution of* $\texttt{P}$*.*

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

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

– $\texttt{wcm}$ (weak corruption model): *A passive adversary* $\mathcal{A}_{\texttt{wcm}}$ *is given access to the queries* Setup *and* RevealKey*. If active, it can also ask* Send *queries.*
– $\texttt{wcm-bs}$ (weak corruption model for backward secrecy): *A passive adversary* $\mathcal{A}_{\texttt{wcm-fs}}$ *is given access to the queries* Setup*,* RevealKey *and* RevealState*. If active, it can also ask* Send *queries.*
– $\texttt{wcm-fs}$ (weak corruption model for forward secrecy): *A passive adversary* $\mathcal{A}_{\texttt{wcm-fs}}$ *is given access to the queries* Setup*,* RevealKey *and* Corrupt*. If active, it can also ask* Send *queries.*
– $\texttt{scm}$ (strong corruption model): *A passive adversary* $\mathcal{A}_{\texttt{scm}}$ *is given access to the queries* Setup*,* RevealKey*,* RevealState *and* Corrupt*. If active, it can also ask* Send *queries.*

A concrete proof 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 possible *adversarial settings* $(\alpha, \beta) \in \{(\emptyset, \texttt{wcm}), (\texttt{wfs}, \texttt{wcm-fs}), (\texttt{wbs}, \texttt{wcm-bs}), (\texttt{sbs}, \texttt{scm}), (\texttt{sfs}, \texttt{scm})\}$, where $\emptyset$ denotes the freshness type for GKE protocols that do not provide any form of forward or backward secrecy.

Remark 1. In practice long-lived keys are mostly used to achieve authentication rather than 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, 30] 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. Our security definitions state requirements for the session group key accepted by an oracle.

AKE-SECURITY. We now give a formal definition of AKE-security (indistinguishability of session group keys).

Definition 5 ($\mathbf{Game}^{\mathsf{ake}-b}_{\mathcal{A}_\beta,\mathtt{P}}(\kappa)$). *Let $\mathtt{P}$ be a correct GKE protocol and b a uniformly chosen bit. Consider an adversarial setting (α,β) sampled from $\{(\emptyset, \mathtt{wcm}), (\mathtt{wbs}, \mathtt{wcm\text{-}bs}), (\mathtt{wfs}, \mathtt{wcm\text{-}fs}), (\mathtt{sbs}, \mathtt{scm}), (\mathtt{sfs}, \mathtt{scm})\}$ and an (active) adversary $\mathcal{A}_\beta$. We define game $\mathtt{Game}^{\mathsf{ake}-b}_{\mathcal{A}_\beta,\mathtt{P}}(\kappa)$ as follows:*

- *after initialization $\mathcal{A}_\beta$ interacts with instance oracles using queries;*
- *if $\mathcal{A}_\beta$ asks a* Test *query to an α-fresh oracle Π^s_U which has accepted, it receives either $\mathtt{key}_1 := k^s_U$ (if $b = 1$) or $\mathtt{key}_0 \in_R \{0,1\}^\kappa$ (if $b = 0$);*
- *$\mathcal{A}_\beta$ continues interacting with instance oracles;*
- *when $\mathcal{A}_\beta$ terminates, it outputs a bit trying to guess which case it was dealing with.*

The output of $\mathcal{A}_\beta$ is the output of the game. The advantage function of $\mathcal{A}_\beta$ in winning the game is defined as

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

Based on this definition we specify the requirement of AKE-security of a GKE protocol $\mathtt{P}$ as follows.

Definition 6 (KE-Security, AKE-Security). *$\mathtt{P}$ is a* AKE-secure protocol with α-secrecy $(\mathtt{AGKE}\text{-}\alpha)$ *if for any* active *PPT $\mathcal{A}_\beta$ the advantage $\mathtt{Adv}^{\mathsf{ake}}_{\mathcal{A}_\beta,\mathtt{P}}(\kappa)$ is negligible. Note, if $\alpha = \emptyset$, we say that $\mathtt{P}$ is a* AKE-secure protocol. *When restricting to* passive *adversaries, we say that $\mathtt{P}$ is* KE-secure with α-secrecy $(\mathtt{GKE}\text{-}\alpha)$.

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.

MA-SECURITY. In the following we propose a new definition of MA-security that considers an adversary who is allowed to corrupt and act on behalf of participants during the protocol execution.

Definition 7 ($\mathbf{Game}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{P}}(\kappa)$, MA-Security). *Let $\mathtt{P}$ be a correct GKE protocol and $\mathtt{Game}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{P}}(\kappa)$ the interaction between instance oracles and an active adversary $\mathcal{A}_{ma}$ that is allowed to query* Send*,* Setup*,* RevealKey*,* RevealState*, and* Corrupt*. We say that $\mathcal{A}_{ma}$* wins *if at some point during the interaction there exist:*

- *an uncorrupted user U_i whose instance oracle $\Pi^{s_i}_i$ has accepted with $k^{s_i}_i$,*
- *another user U_j with $U_j \in \mathtt{pid}^{s_i}_i$ that is uncorrupted at the time $\Pi^{s_i}_i$ accepts,*

such that:

- *either there is no instance oracle $\Pi^{s_j}_j$ with $(\mathtt{pid}^{s_j}_j, \mathtt{sid}^{s_j}_j) = (\mathtt{pid}^{s_i}_i, \mathtt{sid}^{s_i}_i)$,*
- *or there exists an instance oracle $\Pi^{s_j}_j$ with $(\mathtt{pid}^{s_j}_j, \mathtt{sid}^{s_j}_j) = (\mathtt{pid}^{s_i}_i, \mathtt{sid}^{s_i}_i)$ that has accepted with $k^{s_j}_j \neq k^{s_i}_i$.*

The probability of this event is denoted $\mathsf{Succ}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{P}}(\kappa)$. We say that $\mathtt{P}$ is a MA-secure *GKE protocol $(\mathtt{MAGKE})$ if this probability is negligible for any PPT adversary $\mathcal{A}_{ma}$.*

Note we do not deal with α-fresh sessions since malicious participants learn established keys implicitly. Also in $\mathtt{Game}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{P}}(\kappa)$ the adversarial Test query is useless.

In the following we present some claims to illustrate the relationship between our definition 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 (note that [30] does not provide such claims for its definitions of *agreement* and *security against insider impersonation attacks*).

The informal definition of *key confirmation* [36] means that each protocol participant must be assured that every other protocol participant actually has possession of the computed group key. 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 [36]. The related requirement called *unknown key-share resilience* surfaced in [22] 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 $\mathtt{P}$ is a $\mathtt{MAGKE}$ protocol then it provides key confirmation and mutual authentication (explicit key authentication) in the sense of [36].

Proof. If $\mathtt{P}$ 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 7 this is a successful attack against the MA-security of $\mathtt{P}$. This, however, contradicts to the assumption that $\mathtt{P}$ is a $\mathtt{MAGKE}$ protocol. □

Claim. If $\mathtt{P}$ is a $\mathtt{MAGKE}$ protocol then it is resistant against unknown key-share attacks in the sense of [10].

Proof. If $\mathtt{P}$ is not resistant against unknown key-share attack then there exist at least two honest participants U_i and U_j whose oracles $\Pi_i^{s_i}$ resp. $\Pi_j^{s_j}$ participating in the same protocol execution hold different partner ids $\mathtt{pid}_i^{s_i} \neq \mathtt{pid}_j^{s_j}$ such that w.l.o.g. $U_j \in \mathtt{pid}_i^{s_i}$ and $U_i \notin \mathtt{pid}_j^{s_j}$. According to Definition 7 in any $\mathtt{MAGKE}$ protocol for every honest participant $U_j \in \mathtt{pid}_i^{s_i}$ there must exist a corresponding oracle $\Pi_j^{s_j}$ with $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$. Note this implies $\mathtt{pid}_j^{s_j} = \mathtt{pid}_i^{s_i}$. Therefore, since $\mathtt{P}$ is a $\mathtt{MAGKE}$ protocol the probability that $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ hold different partner ids is negligible. □

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

6.1 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 8 (Security-Enhancing GKE Protocol Compiler $\mathtt{C}$). *A* security-enhancing GKE protocol 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 not provided by* $\mathtt{P}$.

6.2 Overview on Compilers for GKE Protocols

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 [31] the following compiler which provides AKE-security. It uses a digital signature scheme Σ (see Appendix A for details).

Definition 9 (Compiler for AKE-Security by Katz and Yung [31]). *Let* $\mathtt{P}$ *be a GKE protocol, and* $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ *a digital signature scheme. A* compiler for AKE-security*, denoted* $\mathtt{C\text{-}A}$*, consists of an an initialization algorithm and 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* $\Pi_1^{s_1}$*, ...,* $\Pi_n^{s_n}$ *invoked prior to any operation execution of* $\mathtt{P}$*. Each* $\Pi_i^{s_i}$ *chooses a random nonce* $r_i \in_R \{0,1\}^\kappa$ *and sends* $U_i|0|r_i$ *to every partnered oracle* $\Pi_j^{s_j}$*. After* $\Pi_i^{s_i}$ *receives* $U_j|0|r_j$ *from all partnered oracles it computes* $\mathtt{nonces} := U_1|r_1| \ldots |U_n|r_n$*. Then, members of* $\mathcal{G}$ *execute* $\mathtt{P}$ *with the following changes:*

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

Katz and Yung proved security of this compiler in the KY model that does not consider strong corruptions. From the perspective of our model their security proofs consider the adversarial settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$. Further, the compiler in [31] 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 $\Pi_i^{s_i}$ 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 proof. We argue that it is possible to omit sequence numbers in the compiler due to the following reasons. First, it is reasonable to assume that any GKE protocol simply fails to process a message which is unexpected according to its natural specification, i.e., we consider verification of sequence numbers as job of the underlying protocol `P`[7]. Note also that sequence numbering in [31] restarts for each new execution of the protocol. Thus, sequence numbers do not provide any additional security advantages for the protocol (e.g., they do not protect against replay attacks). Second, 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. Thus, compiler has to be configured each time a different GKE protocol is used. 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.

The first compiler for key confirmation and mutual authentication in GKE protocols was proposed by Bresson *et al.* [15] based on a cryptographic hash function. However, their proof was performed with respect to the definition

[7] By doing so we can consider, in the proof of such compiler, a passive adversary who is allowed to delete or delay messages, or deliver them out of order but not actively inject new messages. This would make our passive adversary stronger and closer to the "authenticated-links model adversary" described by Canetti and Krawczyk [20]. However, we do not consider this stronger passive adversary within our model in order to keep the model more general. Note that the reliability and the order integrity for delivered messages may not necessarily be part of a concrete GKE protocol but also of some underlying network protocol.

of MA-security in the BCPQ model that does not consider security against malicious participants and they also used non-standard assumptions of the Random Oracle Model [8].

The following construction by Katz and Shin [30] has been designed for secure GKE protocols in the framework of Universal Composability (UC) [19] and can be used to turn any AKE-secure GKE protocol into a UC-secure GKE protocol that provides security against insider attacks (see Section 4). It requires a digital signature scheme Σ, and a collision-resistant pseudo-random function f (see Appendix A for details).

Definition 10 (Compiler for Security against Insider Attacks by Katz and Shin [30]). *Let* $\mathtt{P}$ *be a GKE protocol,* $\Sigma := (\mathtt{Gen}, \mathtt{Sign}, \mathtt{Verify})$ *a digital signature scheme,* $F := \left\{f_k\right\}_{k\in\{0,1\}^\kappa}$ *a function ensemble with range* $\{0,1\}^\lambda$*,* $\lambda \in \mathbb{N}$ *and domain* $\{0,1\}^\kappa$*, and* $\mathtt{sid}_i^{s_i}$ *is a unique session id. A* compiler for security against insider attacks *consists of an initialization algorithm and a protocol defined as follows:*

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

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

We would like to draw the reader's attention to the session IDs $\mathtt{sid}s$. Working in the UC framework, Katz and Shin assume that these session ids are unique *and* specified by some high-level application. In this case the above compiler can be proven to satisfy our MA-security requirement. We stress that existence of such unique session ids is of great importance so that some additional communication rounds are required to set up such session ids if they are not provided by the high-level application (see [4] for setting up session ids in the UC framework). Intuitively, leaving out session ids would allows replay attacks against the compiled protocol as illustrated in the following.

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

6.3 Compiler `C-AMA`

Our compiler denoted `C-AMA` and defined in the following can be used to provide both AKE- and MA-security for any GKE protocol `P` which satisfies the basic requirement of (unauthenticated) KE-security. In fact it can be seen as a sequential combination of the KY and KS compilers[8] discussed in the previous section. `C-AMA` is based on digital signatures and collision-resistant pseudo-random functions (see Appendix A for details) and can be proven secure in

[8] We omit sequence numbers used in the KY compiler due to the arguments in the previous section.

the standard model. Note also, we use nonces to achieve uniqueness of protocol sessions and security of concurrent executions, in particular we do not rely on session ids given by a high-level application.[9]

Definition 11 (Compiler for AKE-Security and MA-Security C-AMA). *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 $\{0,1\}^\kappa$. A* compiler for AKE-security and MA-security*, denoted* C-AMA*, consists of an algorithm* INIT *and a protocol* AMA *defined as follows:*

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

AMA: *This is an interactive protocol between the partnered oracles $\Pi_1^{s_1}$, ..., $\Pi_n^{s_n}$ invoked prior to the execution of $\mathtt{P}$. Each $\Pi_i^{s_i}$ chooses a random* AMA nonce *$r_i \in_R \{0,1\}^\kappa$ and sends $U_i|r_i$ to every partnered oracle $\Pi_j^{s_j}$ with $U_j \in \mathtt{pid}_i^{s_i}$. After $\Pi_i^{s_i}$ receives $U_j|r_j$ from all partnered oracles $\Pi_j^{s_j}$ with $U_j \in \mathtt{pid}_i^{s_i}$ it computes $\mathtt{sid}_i^{s_i} := r_1|\ldots|r_n$. Then it invokes the execution of $\mathtt{P}$ and proceeds as follows:*

- *If $\Pi_i^{s_i}$ in $\mathtt{P}$ outputs a message $U_i|m$ then in $\mathtt{C\text{-}AMA}_\mathtt{P}$ it computes additionally $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, m|\mathtt{sid}_i^{s_i})$ and outputs a modified message $U_i|m|\sigma_i$.*
- *If $\Pi_i^{s_i}$ receives a message $U_j|m|\sigma_j$ from its partnered oracle $\Pi_j^{s_j}$ it checks whether $\Sigma.\mathtt{Verify}(pk'_j, m|\mathtt{sid}_i^{s_i}, \sigma_j) \stackrel{?}{=} 1$. If this verification fails then $\Pi_i^{s_i}$ turns into the stand-by state without accepting; otherwise it proceeds according to the specification of $\mathtt{P}$ upon receiving $U_j|m$.*
- *After an oracle $\Pi_i^{s_i}$ computes $k_i^{s_i}$ in the execution of $\mathtt{P}$ it computes an* AMA token *$\mu_i := f_{k_i^{s_i}}(v_0)$ where v_0 is a constant public value, a signature $\sigma_i := \Sigma.\mathtt{Sign}(sk'_i, \mu_i|\mathtt{sid}_i^{s_i}|\mathtt{pid}_i^{s_i})$ and sends $U_i|\sigma_i$ to every partnered oracle $\Pi_j^{s_j}$ with $U_j \in \mathtt{pid}_i^{s_i}$.*
- *After $\Pi_i^{s_i}$ receives $U_j|\sigma_j$ from its partnered oracle $\Pi_j^{s_j}$ it checks whether $\Sigma.\mathtt{Verify}(pk'_j, \mu_i|\mathtt{nonces}, \sigma_j) \stackrel{?}{=} 1$. If this verification fails then $\Pi_i^{s_i}$ turns into the stand-by state without accepting; otherwise after having received and verified these messages from all other partnered oracles it accepts with the session group key $K_i^{s_i} := f_{k_i^{s_i}}(v_1)$ where $v_1 \neq v_0$ is another constant public value, and erases its private state information including $k_i^{s_i}$.*

SECURITY ANALYSIS. Theorem 1 shows that C-AMA adds AKE-security to any KE-secure GKE protocol. While we do not consider the adversarial setting $(\mathtt{sbs}, \mathtt{scm})$ for the reasons mentioned in Remark 1, we do consider $(\mathtt{wbs}, \mathtt{wcm\text{-}bs})$ and $(\mathtt{sfs}, \mathtt{scm})$. By contrast, the original proof in [31] considers only the weaker settings $(\emptyset, \mathtt{wcm})$ and $(\mathtt{wfs}, \mathtt{wcm\text{-}fs})$.

The standard definition of EUF-CMA-security for Σ and definitions of collision-resistance and pseudo-randomness for F can be found in Appendix A. Further, by q_s we denote the total number of executed protocol sessions during the duration of the attack.

Theorem 1 (AKE-Security of $\mathtt{C\text{-}AMA}_\mathtt{P}$). *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 $\mathcal{A}_\beta$ an active adversary launching at most q_s sessions of $\mathtt{P}$. Then if Σ is EUF-CMA and F is pseudo-random, $\mathtt{C\text{-}AMA}_\mathtt{P}$ is AGKE-α and*

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

Proof. In our proofs we use a well-known proving technique called *sequence of games* [40] 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, 24, 25, 32].

[9] However, if such unique session ids are provided then similar to the KS compiler we can omit the very first communication round where participants exchange their nonces.

We define a *sequence of games* $\mathbf{G}_i$, $i = 0, \ldots, 4$ 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 $\mathtt{Game}^{\mathsf{ake}-b}_{\mathcal{A}_\beta,\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa)$. The queries made by $\mathcal{A}_\beta$ are answered by a simulator $\mathcal{S}$.

Game $\mathbf{G}_0$: This game is the real game $\mathtt{Game}^{\mathsf{ake}-b}_{\mathcal{A}_\beta,\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa)$ played between $\mathcal{S}$ and $\mathcal{A}_\beta$. Remind, the (unique) Test query can be asked only during an α-fresh session, and is answered either with random string or with the actual session key $K_i^{s_i}$.

Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_0$ with the only exception that the simulator fails and sets b' at random if $\mathcal{A}_\beta$ 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 $\Pi_i^{s_i}$ *before* querying $\mathsf{Corrupt}(U_i)$. In other words the simulation fails if $\mathcal{A}_\beta$ outputs a successful forgery; such event is denoted Forge. Hence,

$$|\Pr[\mathsf{Win}_1^{\mathsf{ake}}] - \Pr[\mathsf{Win}_0^{\mathsf{ake}}]| \leq \Pr[\mathsf{Forge}]. \tag{1}$$

In order to estimate $\Pr[\mathsf{Forge}]$ we show that using $\mathcal{A}_\beta$ 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 $\mathtt{C\text{-}AMA}_\mathtt{P}$, $\mathcal{F}$ 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 $\mathcal{A}_\beta$ in a natural way by executing $\mathtt{C\text{-}AMA}_\mathtt{P}$ by itself, and by obtaining the necessary signatures with respect to pk'_{i^*} from its signing oracle. This is a perfect simulation for $\mathcal{A}_\beta$ since by assumption no $\mathsf{Corrupt}(U_{i^*})$ may occur (otherwise $\mathcal{F}$ would not be able to answer it). Assuming Forge occurs, $\mathcal{A}_\beta$ outputs a new valid message/signature pair with respect to some pk'_i; since i^* was randomly chosen and the simulation is perfect, $\Pr[i = i^*] = 1/N$. In that case $\mathcal{F}$ outputs this pair as its forgery. Its success probability is given by $\Pr[\mathsf{Forge}]/N$. This implies

$$\Pr[\mathsf{Forge}] \leq N\mathsf{Succ}^{\mathtt{euf\text{-}cma}}_{\mathcal{F},\Sigma}(\kappa). \tag{2}$$

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

$$|\Pr[\mathsf{Win}_2^{\mathsf{ake}}] - \Pr[\mathsf{Win}_1^{\mathsf{ake}}]| \leq \Pr[\mathsf{RepAMA}] \leq \frac{Nq_\mathsf{s}^2}{2^\kappa} \tag{3}$$

Note that in this game the same value for $\mathtt{sid}_i^{s_i}$ does not repeat in any two different sessions where at least one user remains uncorrupted. This excludes possible replay attacks in $\mathtt{C\text{-}AMA}_\mathtt{P}$ because the value $\mathtt{sid}_i^{s_i}$ is used to generate signatures of all sent messages.

Game $\mathbf{G}_3$: This game is identical to $\mathbf{G}_2$ except that a random value sampled from $\{0,1\}^\kappa$ is used instead of k in the $\mathtt{AMA}$ protocol of $\mathtt{C\text{-}AMA}$ in all α-fresh sessions. Having excluded forgeries and replay attacks on $\mathtt{C\text{-}AMA}_\mathtt{P}$ we can use the advantage of a passive adversary $\mathcal{A}^*_\beta$ against the original protocol $\mathtt{P}$ to upper-bound

$$|\Pr[\mathsf{Win}_3^{\mathsf{ake}}] - \Pr[\mathsf{Win}_2^{\mathsf{ake}}]| \leq q_\mathsf{s}\mathrm{Adv}^{\mathsf{ke}}_{\mathcal{A}^*_\beta,\mathtt{P}}(\kappa). \tag{4}$$

Note that since k is erased at the end of each $\mathtt{C\text{-}AMA}_\mathtt{P}$ execution the adversary $\mathcal{A}^*_\beta$ in the corresponding corruption model β cannot learn the real value of k used in any α-fresh session since the adversarial setting (α, β) disallows $\mathsf{RevealKey}$, $\mathsf{RevealState}$ or $\mathsf{Corrupt}$ queries in α-fresh sessions.

Game $\mathbf{G}_4$: This game is identical to $\mathbf{G}_3$ except that the session group key K and AMA token μ are replaced with random values in all α-fresh sessions (the same random value must be used for each AMA token μ_i observed by the adversary). We are allowed to do this since in this game k, used for the computation of K and μ, is replaced by a random value (as a result of the previous game). We get

$$|\Pr[\mathsf{Win}_4^{\mathsf{ake}}] - \Pr[\mathsf{Win}_3^{\mathsf{ake}}]| \leq 2q_\mathsf{s}\mathrm{Adv}^{\mathsf{prf}}_{\mathcal{A},F}(\kappa). \tag{5}$$

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

$$\Pr[\mathsf{Win}_4^{\mathsf{ake}}] = \frac{1}{2} \tag{6}$$

Considering Equations 2 to 6 we get:

$$\begin{aligned}\Pr[\mathtt{Game}^{\mathsf{ake}-b}_{\mathcal{A}_\beta,\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa) = b] &= \Pr[\mathsf{Win}_0^{\mathsf{ake}}] \\ &\leq N\mathsf{Succ}^{\mathtt{euf\text{-}cma}}_{\mathcal{F},\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^\kappa} + q_\mathsf{s}\mathrm{Adv}^{\mathsf{ke}}_{\mathcal{A}^*_\beta,\mathtt{P}}(\kappa) + 2q_\mathsf{s}\mathrm{Adv}^{\mathsf{prf}}_{\mathcal{A},F}(\kappa) + \frac{1}{2}.\end{aligned}$$

This results in the desired inequality (we omit notations for $\mathcal{F}$, $\mathcal{A}^*_\beta$ and $\mathcal{A}$ at the right side of the inequality)

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

□

Theorem 2 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 [15], we are not concerned about the AKE-security of P in this case.

Theorem 2 (MA-Security of $\mathtt{C\text{-}AMA}_\mathtt{P}$). *For any GKE protocol P, if Σ is EUF-CMA and F is collision-resistant, then $\mathtt{C\text{-}AMA}_P$ is MAGKE, and*

$$\mathsf{Succ}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa) \leq N\mathsf{Succ}^{\mathtt{euf\text{-}cma}}_{\Sigma}(\kappa) + \frac{Nq_\mathsf{s}^2}{2^\kappa} + q_\mathsf{s}\mathsf{Succ}^{\mathsf{coll}}_{F}(\kappa).$$

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

Game $\mathbf{G}_0$: This game is the real game $\mathtt{Game}^{\mathsf{ma}}_{\mathcal{A}_{ma},\mathtt{C\text{-}AMA}_\mathtt{P}}(\kappa)$ played between $\mathcal{S}$ and $\mathcal{A}_{ma}$. Remind, the goal of $\mathcal{A}_{ma}$ is to achieve that there exists an uncorrupted user U_i whose corresponding oracle $\Pi_i^{s_i}$ accepts with $K_i^{s_i}$ and another user $U_j \in \mathtt{pid}_i^{s_i}$ that is uncorrupted at the time $\Pi_i^{s_i}$ accepts and either does not have a corresponding 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 has such an oracle but this oracle accepts with $K_j^{s_j} \neq K_i^{s_i}$.

Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_0$ with the only exception that the simulation aborts if $\mathcal{A}_{ma}$ 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 $\Pi_i^{s_i}$ before querying $\mathsf{Corrupt}(U_i)$, i.e., the simulation fails if $\mathcal{A}_{ma}$ outputs a successful forgery. According to Equation 2 we obtain,

$$|\Pr[\mathsf{Win}_1^{\mathsf{ma}}] - \Pr[\mathsf{Win}_0^{\mathsf{ma}}]| \leq N\mathsf{Succ}^{\mathtt{euf\text{-}cma}}_{\Sigma}(\kappa) \tag{7}$$

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 3 we get

$$|\Pr[\mathsf{Win}_2^{\mathsf{ma}}] - \Pr[\mathsf{Win}_1^{\mathsf{ma}}]| \leq \frac{Nq_\mathsf{s}^2}{2^\kappa} \tag{8}$$

Note that this prevents attacks where $\Pi_i^{s_i}$ 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 $\Pi_i^{s_i}$ does not accept unless it successfully verifies all required σ_j for all $U_j \in \mathtt{pid}_i^{s_i}$ in the AMA protocol of C-AMA. Having excluded forgeries and replay attacks we follow that for every user $U_j \in \mathtt{pid}_i^{s_i}$ that is uncorrupted at the time $\Pi_i^{s_i}$ accepts there exists a corresponding instance oracle $\Pi_j^{s_j}$ with $(\mathtt{pid}_j^{s_j}, \mathtt{sid}_j^{s_j}) = (\mathtt{pid}_i^{s_i}, \mathtt{sid}_i^{s_i})$. Thus, according to Definition 7 $\mathcal{A}_{ma}$ wins in this game only if any of these oracles has accepted with $K_j^{s_j} \neq K_i^{s_i}$.

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

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

Thus,

$$\Pr[\mathsf{Win}_2^{\mathsf{ma}}] \leq q_{\mathsf{s}}\mathsf{Succ}_F^{\mathsf{coll}}(\kappa). \tag{9}$$

Considering Equations 7 to 9 we get the desired inequality

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

□

7 Conclusion

In this paper we have analyzed several computational (game-based) security models for group key exchange protocols from the perspective of their technical construction and informal definitions of key confirmation, mutual authentication, and unknown-key share resilience. We were able to identify some problems with the definition of MA-security in the BCPQ model which is foundational for many subsequently proposed models. We thus proposed an extended model with a revised definition of MA-security considering attacks of malicious protocol participants and showed unifying relationship between our formal definition and the previously proposed informal notions.

In order to prove soundness and feasibility of our definitions we have described a generic solution, in form of the compiler C-AMA (as a combination of KY and KS compilers), that adds AKE- and MA-security to "passively" secure GKE protocols, and we proved its security under standard assumptions. The provided proofs also imply that (i) the KY compiler [31] satisfies even stronger definitions of AKE-security (w.r.t. the various adversarial settings concerning forward- and backward-secrecy defined in our model), and (ii) the KS compiler [30] satisfies our revised definition of MA-security.

References

1. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-Based Group Key Exchange in a Constant Number of Rounds. In *Proceedings of the 9th International Workshop on Theory and Practice in Public Key Cryptography (PKC'06)*, volume 3958 of *Lecture Notes in Computer Science*, pages 427–442. Springer, April 2006.
2. J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. In *Advances in Cryptology – EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.
3. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proceedings of the 5th ACM conference on Computer and Communications Security (CCS'98)*, pages 17–26. ACM Press, 1998.
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 *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS'98)*, pages 1–6. ACM Press, 1998.

6. M. Bellare. Practice-Oriented Provable-Security. In *Proceedings of the First International Workshop on Information Security (ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1998.
7. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology–CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
8. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS'93)*, pages 62–73. ACM Press, 1993.
9. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proceedings 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 *Lecture Notes in Computer Science*, 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 *Lecture Notes in Computer Science*, pages 290–390. Springer, December 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 *Lecture Notes in Computer Science*, pages 321–336. Springer, Mai 2002.
14. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In *Advances in Cryptology – ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 497–514. Springer, December 2002.
15. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS'01)*, pages 255–264. ACM Press, 2001.
16. M. Burmester. On the Risk of Opening Distributed Keys. In *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 308–317. Springer, August 1994.
17. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer, May 1994.
18. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
19. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE CS, 2001.
20. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
21. K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In *Advances in Cryptology – ASIACRYPT'05*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, 2005.
22. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.
23. R. Dutta and R. Barua. Constant Round Dynamic Group Key Agreement. In *Information Security: 8th International Conference (ISC'05)*, volume 3650 of *Lecture Notes in Computer Science*, pages 74–88. Springer, August 2005.
24. R. Dutta and R. Barua. Dynamic Group Key Agreement in Tree-Based Setting. In *Proceedings of the 10th Australasian Conference on Information Security and Privacy (ACISP'05)*, volume 3574 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2005.
25. R. Dutta, R. Barua, and P. Sarkar. Provably Secure Authenticated Tree Based Group Key Agreement. In *Proceedings of the 6th International Conference on Information and Communications Security (ICICS'04)*, volume 3269 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2004.
26. M. Fischlin. Pseudorandom Function Tribe Ensembles Based on One-Way Permutations: Improvements and Applications. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 1999.
27. O. Goldreich. *Foundations of Cryptography - Basic Tools*, volume 1. Cambridge University Press, 2001. ISBN:0-521-79172-3.
28. C. G. Günther. An Identity-Based Key-Exchange Protocol. In *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer, 1990.
29. I. Ingemarsson, D. T. Tang, and C. K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–719, 1982.
30. J. Katz and J. S. Shin. Modeling Insider Attacks on Group Key-Exchange Protocols. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 180–189. ACM Press, 2005.
31. J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *Advances in Cryptology - CRYPTO'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2003.
32. H.-J. Kim, S.-M. Lee, and D. H. Lee. Constant-Round Authenticated Group Key Exchange for Dynamic Groups. In *Advances in Cryptology – ASIACRYPT'04*, volume 3329 of *Lecture Notes in Computer Science*, pages 245–259, 2004.
33. Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*, pages 235–244. ACM Press, 2000.
34. Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proceedings of IFIP TC11 Sixteenth Annual Working Conference on Information Security (IFIP/Sec'01)*, volume 193 of *IFIP Conference Proceedings*, pages 229–244. Kluwer, 2001.
35. 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.
36. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996. ISBN:0-8493-8523-7.

37. O. Pereira and J.-J. Quisquater. A Security Analysis of the CLIQUES Protocols Suites. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 73–81. IEEE Computer Society Press, June 2001.
38. O. Pereira and J.-J. Quisquater. Some Attacks upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 11(4):555–580, 2003.
39. A. Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In *Proceedings of the International Workshop on Cryptographic Techniques and Electronic Commerce 1999*, pages 192–202. City University of Hong Kong Press, 1999.
40. 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.
41. D. G. Steer, L. Strawczynski, W. Diffie, and M. J. Wiener. A Secure Audio Teleconference System. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. Springer, 1990.
42. M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS'96)*, pages 31–37. ACM Press, 1996.
43. M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387. IEEE Computer Society Press, 1998.
44. Y. Yacobi and Z. Shmuely. On Key Distribution Systems. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 344–355. Springer, August 1990.

A Cryptographic Primitives Used by the Compiler C-AMA

Definition 12 (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 13 (EUF-CMA Security[10]). *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* $\mathsf{Succ}^{\mathtt{euf\text{-}cma}}_{\mathcal{F},\Sigma}(\kappa)$ *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 [27, Definition 3.6.9].

Definition 14 (Efficiently Computable Generalized Pseudo-Random Function Ensemble F). *An ensemble of finite functions* $F := \left\{ \left\{ f_k : \{0,1\}^{p(\kappa)} \to \{0,1\}^{p(\kappa)} \right\}_{k \in \{0,1\}^\kappa} \right\}_{\kappa \in \mathbb{N}}$ *where* $p : \mathbb{N} \to \mathbb{N}$ *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* $\mathsf{Tag}(x)$ *and participating in one of the following two games:*
   - $\mathtt{Game}^{\mathsf{prf}-1}_{\mathcal{A},F}(\kappa)$ *where a query* $\mathsf{Tag}(x)$ *is answered with* $f_k(x)$,

[10] 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.

– $\mathtt{Game}_{\mathcal{A},F}^{\mathsf{prf}-0}(\kappa)$ *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*

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

We say that F *is pseudo-random if* $\mathtt{Adv}_{\mathcal{A},F}^{\mathsf{prf}}(\kappa)$ *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 [27] 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 [30]. The same property has previously been defined in [26] and denoted there as *fixed-value-key-binding* property of a pseudo-random function ensemble. We also refer to [30] for a possible construction based on one-way permutations and for the proof of Lemma 1).

Definition 15 (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* $\mathtt{Sample}$ *such that for all PPT adversaries* $\mathcal{A}$ *the following advantage is a negligible function in* κ*:*

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

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