# Modeling Insider Attacks on Group Key-Exchange Protocols

Jonathan Katz*† Ji Sun Shin*

**Abstract**

Protocols for authenticated key exchange (AKE) allow a group of parties within an insecure network to establish a common session key which can then be used to secure their future communication. It is fair to say that *group* AKE is currently less well understood than the case of two-party AKE; in particular, attacks by *malicious insiders* — a concern specific to the group setting — have so far been considered only in a relatively "ad-hoc" fashion. The aim of this work is to address this by providing a formal, comprehensive model and definition of security for group AKE which automatically encompasses insider attacks. We do so by defining an appropriate ideal functionality for group AKE within the universal composability (UC) framework. As a side benefit, any protocol secure with respect to our definition is secure even when run concurrently with other protocols, and the key generated by any such protocol may be used securely in any subsequent application.

We show that our resulting definition of security is strictly stronger than a previous definition suggested by Bresson, et al. (termed AKE-security), and that our definition implies all previously-suggested notions of security against insider attacks. We also show a simple technique for converting any "AKE-secure" protocol into one secure with respect to our definition.

**Key words:** Group key exchange, Insider attacks, Universal composability.

## 1 Introduction

Protocols for authenticated key exchange (AKE) allow a group of parties within a completely insecure network to establish a common *session key* and furthermore to be assured that they are sharing this key with *each other* (i.e., with their intended partners). The case of 2-party AKE has been extensively investigated [18, 10, 19, 2, 6, 4, 30, 13, 14] both from a definitional standpoint as well as from the standpoint of designing efficient protocols for this task. Less well understood is the *group setting* where a session key is to be established among more than two parties. Formal definitional work in this setting began only recently with the introduction of a formal model by Bresson, et al. [7, 8, 9] (with some additional modifications by Katz and Yung [24]) which was based on earlier work in the two-party setting by Bellare, Pointcheval, and Rogaway [2, 3, 5]. We refer to protocols secure in the model of Bresson, et al. as "AKE-secure".

The above-mentioned model of Bresson, et al. does *not* take into account any notion of security against "insider attacks", and in fact it is easy to see that protocols secure in their model may be completely insecure against attacks by malicious insiders (cf. Claim 1, below). Indeed, developing a formal framework in which to model insider attacks in the group setting was left as an open question

*{jkatz,sunny}@cs.umd.edu. Dept. of Computer Science, University of Maryland.

†Research supported by NSF Trusted Computing Grant #0310751 and NSF CAREER award #0447075.

by, e.g., Katz and Yung [24]. The lack of a comprehensive and formal model which adequately treats insider attacks has led to a number of definitions being introduced in a relatively "ad-hoc" fashion[1] (e.g., [25, 28, 1, 20, 16]) as well as a number of claimed "attacks" on provably-secure protocols (e.g., [29, 34, 33, 17]). Recent work of Cachin and Strobl [11] deals with the case of "crash" (i.e., fail-stop) faults, but does not deal with the more general case of *adversarial* (Byzantine) failures. Note that the possibility of insider faults/attacks in the group setting represents a *qualitative* difference from the two-party setting where insider attacks are not much of a concern.

Of course, when an insider is malicious there is no way to prevent this malicious player from learning the value of the session key computed by a group of which he is a valid member. However, there are still important and potentially-avoidable security concerns to be addressed: for example, a malicious insider should not have the ability to learn the session keys computed by groups of which he is *not* a member, and should not be able to impersonate other honest players or to cause different (honest) members of a group to compute different session keys without detecting that something is amiss. Although one can continue to list various properties of this sort which any "secure" group AKE protocol should satisfy (as done in, e.g., [28, 16]), it is unclear how to determine when any given list of attacks has exhausted all the relevant possibilities! It is for precisely this reason that a comprehensive model is so important.

### 1.1 Our Contributions

As a way to better model insider attacks, we introduce definitions of security for group AKE protocols within the universal composability (UC) framework [12]. In the UC framework, security of a cryptographic task is defined by specifying an appropriate ideal-world functionality; a secure protocol is then defined as a protocol which adequately "mimics" this ideal-world functionality (in a way made formal in [12]; see Appendix A for a brief review). By suitably constructing an ideal functionality for the task of group key exchange, one is assured that any secure protocol will automatically guarantee security against both insider and outsider attacks. The primary advantage of working within the UC framework is that we need only specify what it is we wish a group AKE protocol to *achieve* (via introduction of the appropriate ideal-world functionality), rather than provide a laundry-list of attacks which we wish to *prevent*. As a sanity check, however, we show that any UC-secure group AKE protocol is AKE-secure (i.e., is secure against "outsider" attacks in the sense of Bresson, et al.) and is also secure against the various insider attacks listed in [28, 16]. As mentioned earlier, the definition of Bresson, et al. does not guarantee *any* security against insider attacks; thus, our definition is strictly stronger than AKE-security.

Working within the UC framework yields other advantages as well, as noted by Canetti and Krawczyk in the context of two-party key exchange [14]. In particular, two additional advantages include: (1) protocols proven secure within the UC framework remain secure even when run concurrently with any other set of protocols, and the session keys generated by any UC-secure protocol may be securely used by any application calling the protocol as a sub-routine; also (2) we obtain a definition which guarantees security even in the so-called *strong corruption model* where honest players may be compromised at any point during execution of the protocol. Although some previous definitions of group AKE describe security against such attacks, none of the above-mentioned references shows a group AKE protocol which is secure in the strong corruption model.

As mentioned above, the definition of security developed in this work is strictly stronger than that of AKE-security. We show, however, a simple and efficient *compiler* which transforms any

[1] By "ad-hoc" we do not (necessarily) mean "informal". Rather, we mean that these works do not present a general framework in which to deal with insider attacks, but instead consider a seemingly "ad-hoc" set of such attacks.

AKE-secure protocol into a UC-secure protocol. Our compiler is essentially the one suggested (without proof) by Katz and Yung [24, Section 2.1], and is fundamentally *different* from the one used by Canetti and Krawczyk [14] in the two-party setting.[2] In particular (informally), the compiler suggested by Canetti and Krawczyk [14] authenticates an "ack" message using a message authentication code (MAC) keyed by the session key $sk$ generated by the two parties. In our setting, however, this would *not* result in a protocol secure against insider attacks since a malicious insider knows $sk$ as well! Instead, our compiler requires the parties to *sign* an "ack" message using a long-term key established for this purpose. (For exactly this reason, our compiler is also fundamentally different from the ones suggested by Bellare, et al. [5] and Bresson, et al. [7] to achieve explicit authentication.) However, some additional subtleties arise (see Section 4 for details): in particular, we must ensure that the "ack" message both (1) does not leak information about $sk$ (in a computational sense); yet (2) corresponds to a *unique* possible $sk$. Thus, for example, the "ack" message *cannot* simply be computed as $v = F_{sk}(r)$, where $F$ is a pseudorandom function and $r$ is a random value: in this case (even if $r$ is public), $v$ would not necessarily correspond to a unique $sk$.

### 1.2 Previous Related Work

Clearly, the work most relevant to our own is that of Canetti and Krawczyk [14] who consider *two-party* key exchange within the UC framework. Our work relies extensively on theirs, however our goals are somewhat different in that we set out with the aim of modeling insider attacks (which are not of much concern in the two-party setting) and merely view the UC framework as a convenient way to achieve this goal. (In contrast, Canetti and Krawczyk were most concerned with composability, and were specifically interested in using the UC framework for that reason.)

We have already mentioned the work of Cachin and Strobl [11] which gives a framework in which to analyze security in the presence of crash faults and presents protocols secure with respect to their definition. However, their work does not take into account the more general case of Byzantine faults by those taking part in the protocol. Works which come closest to providing a formal model in which to fully analyze insider attacks in the context of group AKE protocols include [28, 16], which both present lists of security concerns to be addressed when malicious insiders are present. In contrast, we view this work as providing a single, simple, and comprehensive definition of insider attacks; moreover, our definition encompasses all the definitions of those previous papers. A mechanism for protecting against certain insider attacks in given by [28]; however, their approach is tailored for a specific protocol and no proofs of security or formal definitions are given (indeed, depending on how various components of their scheme are instantiated, specific attacks appear possible).

Steiner [31, Section 5.2] (see also [26]) proposes an ideal functionality for group AKE within the framework of Pfitzmann and Waidner [27]. Insider attacks were not the specific focus there, but nevertheless it is claimed that protocols securely realizing the given ideal functionality are also secure against (certain classes of) insider attacks. Arguably, the ideal functionality defined here is simpler and more straightforward than the one given in [31] (although, to be fair, this depends to some extent on one's relative familiarity with [12] vs. [27]). Furthermore, although a specific, $O(n)$-round protocol (where $n$ is the group size) is proven secure in [31], no generic compiler for constructing secure protocols is given.

[2] As our compiler achieves a different goal, it goes without saying that it is also very different from the compilers of Bellare-Canetti-Krawczyk [4] and Katz-Yung [24] which convert unauthenticated protocols to authenticated protocols.

## 2 A Review of Prior Definitions of Security

In this section, we review the notion of "AKE-security" as proposed by Bresson, et al. [7, 8, 9] with some modifications by Katz and Yung [24]. Next, we formally introduce two notions of security against insider attacks. We conclude this section by showing that, as one might expect, AKE-security does not imply security against either of the insider attacks considered here. This motivates the introduction of a new model of security which *does* adequately handle such attacks.

**Participants and initialization.** For simplicity, we assume a fixed, polynomial-size set of $m$ players $\mathcal{U} = \{U_1, \ldots, U_m\}$. Any subset of these players is allowed to run the group AKE protocol at any time (possibly concurrently) in order to share a session key. During some initialization phase which occurs before the protocol is ever run, each player $U$ runs a key-generation algorithm $\mathcal{G}(1^k)$ to generate a long-term public/secret key pair $(PK_U, SK_U)$. Player $U$ keeps the secret key $SK_U$ private and the public key $PK_U$ is assumed to be known by all other participants and the adversary as well. Following most previous work, we assume that all long-term keys are honestly-generated; this is equivalent to assuming that no players are corrupted before the initialization phase concludes. We stress that **this is for simplicity only**, and it is easy to see that our compiler of Section 4 remains secure even if corrupted players choose their long-term keys in an arbitrary manner (possibly depending on the public key of the honest players), and even if corrupted players share different public keys with different sets of honest players (i.e., the PKI is inconsistent).

**Session IDs, partner IDs, and related notions.** Each player $U \in \mathcal{U}$ is allowed to run the protocol multiple times with possibly different groups of participants; following [2], we model this via the use of *instances*, and denote instance $i$ of player $U$ as $\Pi_U^i$. We treat *session IDs* in a different manner than [7, 24], and follow [4, 13, 14] in assuming that unique session IDs are provided by some higher-level protocol when the group key-exchange protocol is first initiated. Thus, all members taking part in a given execution of a protocol will *de facto* have the same session ID. Besides being more in line with the way session IDs are handled in the UC framework, this also simplifies matters in the group setting where each player's view (i.e., transcript) of a single execution of the protocol may be different (this in contrast to the two-party setting, where the transcripts of two players executing a protocol are identical and hence the session ID can be defined as some function of the common transcript). Moreover, since a single player may be concurrently running multiple instances of a group key-exchange protocol, players in practice need a way to distinguish the sessions to which incoming messages belong. Thus, in some sense, pre-defined session IDs are already implicit in the models of [7, 24] anyway.

The session ID of instance $\Pi_U^i$ is denoted $\mathsf{sid}_U^i$. The partner ID for instance $\Pi_U^i$ (denoted $\mathsf{pid}_U^i$) consists of the identities of the players in the group with whom $\Pi_U^i$ intends to establish a session key, including $U$ itself. The value of $\mathsf{pid}_U^i$ is established, along with $\mathsf{sid}_U^i$, when instance $\Pi_U^i$ first initiates the protocol. Session IDs and partner IDs are public information.

We say an instance $\Pi_U^i$ *accepts* when it computes a valid session key $\mathsf{sk}_U^i$. (An instance may also terminate without accepting, and in this case it does not output any session key at all. Whether or not a particular instance has accepted or has instead terminated without acceptance is public information.) If an instance computes a session key $\mathsf{sk}_U^i$, we assume it outputs $(\mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{sk}_U^i)$. Once an instance accepts, it remains in an accepting state. Finally, we say instances $\Pi_U^i$ and $\Pi_{U'}^j$ (with $U \neq U'$) are *partnered* iff (1) they have both accepted; (2) $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j$; and (3) $\mathsf{pid}_U^i = \mathsf{pid}_{U'}^j$.

**Correctness.** We define correctness following [7, 8, 9, 24] by requiring that if the adversary honestly forwards all messages between instances of players in a given set $\mathsf{pid}$, and each such instance holds the same value $\mathsf{sid}$, then these instances all accept and output identical session keys.

**Adversarial model.** Actions of an adversary are modeled using various oracles:

- $\mathsf{Send}(U, i, M)$ sends message $M$ to instance $\Pi_U^i$ and outputs the response by this instance. A query $\mathsf{Send}(U, i, (\mathsf{sid}, \mathsf{pid}))$ prompts instance $\Pi_U^i$ to initiate the protocol using session ID $\mathsf{sid}$ and partner ID $\mathsf{pid}$ (where we require $U \in \mathsf{pid}$).
- $\mathsf{StrongCorrupt}(U)$ outputs the internal state of any instances $\Pi_U^i$ of player $U$ which are currently executing the protocol, *in addition* to the long-term secret key $SK_U$ of player $U$. This models complete corruption of player $U$. (This oracle call does not reveal previously-generated session keys of any instances $\Pi_U^i$ of $U$ who have terminated the protocol. Such session keys can be obtained using the $\mathsf{Reveal}$ oracle.)
- $\mathsf{Reveal}(U, i)$ provides the adversary with the session key $\mathsf{sk}_U^i$ of instance $\Pi_U^i$.
- $\mathsf{Test}(U, i)$ does not correspond to any real-world action, but provides a means of defining security (see below). This query is allowed only when $\Pi_U^i$ has accepted. In response to this oracle call, a random bit $b$ is chosen. If $b = 0$ a random session key is output, while if $b = 1$ the session key $\mathsf{sk}_U^i$ is output. The adversary is allowed to access this oracle only once, at any time during its execution.

We remark that all prior work in the group setting of which we are aware considered only the *weak corruption model* whereby the adversary obtains a corrupted user's long-term secret key but *not* their internal state. In contrast, the definition of AKE security we present here is in the *strong corruption model*.

**AKE security.** We define AKE-security following [7, 24] (other than the fact that we are in the strong corruption model). Say instance $\Pi_U^i$ is *associated with session* $(\mathsf{sid}, \mathsf{pid})$ if $\mathsf{sid}_U^i = \mathsf{sid}$ and $\mathsf{pid}_U^i = \mathsf{pid}$. We say a player $U$ is *corrupted* if the adversary queries $\mathsf{StrongCorrupt}(U_i)$. A session $(\mathsf{sid}, \mathsf{pid})$ is *corrupted* if there exists a $U \in \mathsf{pid}$ who is corrupted while there is an instance $\Pi_{U'}^i$ associated with this session (possibly with $U' = U$) who has not yet terminated. We say an instance $\Pi_U^i$ associated with session $(\mathsf{sid}, \mathsf{pid})$ is *fresh* if (1) the adversary has never queried $\mathsf{Reveal}(U', j)$ for any instance $\Pi_{U'}^j$ associated with $(\mathsf{sid}, \mathsf{pid})$, and (2) the session $(\mathsf{sid}, \mathsf{pid})$ is not corrupted. The adversary *succeeds* (denoted by event $\mathsf{Succ}$) if it queries the $\mathsf{Test}$ oracle regarding a *fresh* instance, and correctly guesses the value of the bit $b$ used by the $\mathsf{Test}$ oracle in answering this query. Define the advantage of adversary $\mathcal{A}$ attacking protocol $\pi$ to be $\mathsf{Adv}_{\mathcal{A},\pi}^{\mathsf{ake}} \stackrel{\text{def}}{=} |\Pr[\mathsf{Succ}] - \frac{1}{2}|$. Protocol $\pi$ is said to be *AKE-secure* if, for any poly-time adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A},\pi}^{\mathsf{ake}}$ is negligible (as a function of the security parameter). We remark that our definition of freshness automatically ensures that AKE-security encompasses forward secrecy.

## 2.1 Modeling Insider Attacks within the Above Framework

Here, we provide definitions of insider attacks within the AKE-security model of the previous section. Although the definitions given here will be superseded by the definitions of the following section, these definitions may be of independent interest as they appear to be the first formal definitions of insider security for group key exchange within the AKE-security framework. We first define a notion of *agreement* and then define security against *insider impersonation attacks*. The first notion was motivated by a suggestion in [24], and is also implicit in the security definitions of [13, 14] (for the two-party case). Our definition of insider impersonation attacks is stronger than the numerous varieties of insider attacks considered in [28, 16] (in particular, a protocol secure against our notion of insider impersonation attacks is also secure with respect to all the notions

considered in [28, 16]) with the exception that we do not consider "attacks" in which an adversary corrupts a player $U$ and then impersonates *other players* to (honest instances of) $U$. The primary reason for this is that when we move to the UC framework a player is either corrupted or not and so such an attack no longer makes sense (namely, there is no longer any such thing as an "honest instance" of a corrupted player $U$). We remark, however, that our compiler of Section 4 can be shown to prevent these types of attacks as well.

Our notion of *agreement* requires that any partnered instances (of uncorrupted players) agree on the session key they output. (Recall that if two instances are partnered, then by definition they have accepted. Agreement does *not* require that either all parties involved in an execution of the protocol accept or else no parties accept; this is impossible to achieve in an asynchronous model in which the adversary controls all communication in the network.)

**Definition 1** An adversary $\mathcal{A}$ *violates agreement* if there exist partnered instances $\Pi_U^i, \Pi_{U'}^j$ such that (1) neither $U$ nor $U'$ are corrupted, but (2) $\mathsf{sk}_U^i \neq \mathsf{sk}_{U'}^j$. We say a protocol *guarantees agreement* if the probability that any poly-time adversary violates agreement is negligible. $\diamondsuit$

Toward our definition of security against insider impersonation, we say an adversary *impersonates $U'$ to $\Pi_U^i$* (where $U'$ is uncorrupted) if (1) $\Pi_U^i$ accepts and (2) $U' \in \mathsf{pid}_U^i$, but (3) there does not exist any instance $j$ of $U'$ with $(\mathsf{sid}_{U'}^j, \mathsf{pid}_{U'}^j) = (\mathsf{sid}_U^i, \mathsf{pid}_U^i)$. Before describing notions of security against impersonation in the context of insider attacks, we first provide for comparison a notion of security against *outsider* impersonation attacks which does *not* take into account insider attacks yet is not implied by AKE-security (see Claim 1 below).

**Definition 2** An adversary $\mathcal{A}$ succeeds in an *outsider impersonation attack* if there exist a party $U'$ and an instance $\Pi_U^i$ such that (1) $\mathcal{A}$ impersonates $U'$ to $\Pi_U^i$ and (2) no players in $\mathsf{pid}_U^i$ are corrupted at the time $\Pi_U^i$ accepts. We say a protocol is *secure against outsider impersonation attacks* if the probability that any poly-time adversary succeeds in the above attack is negligible. $\diamondsuit$

We can extend the above to encompass insider attacks as well.

**Definition 3** An adversary $\mathcal{A}$ succeeds in an *insider impersonation attack* if there exist a party $U'$ and an instance $\Pi_U^i$ such that (1) $\mathcal{A}$ impersonates $U'$ to $\Pi_U^i$ and (2) neither $U$ nor $U'$ is corrupted at the time $\Pi_U^i$ accepts. We say a protocol is *secure against insider impersonation attacks* if the probability that any poly-time adversary succeeds in the above attack is negligible. $\diamondsuit$

Note that security against insider impersonation attacks implies security against outsider impersonation attacks.

As useful shorthand, we will say that a protocol is *secure against insider attacks* if it is AKE-secure, secure against insider impersonation attacks, and guarantees agreement. It is not hard to see that an AKE-secure protocol need not be secure against insider attacks: In fact, an AKE-secure protocol does not even guarantee security against *outsider* impersonation attacks.

**Claim 1** *There exists (under standard cryptographic assumptions) an AKE-secure protocol which is neither secure against* outsider *impersonation attacks nor guarantees agreement.*

**Proof** We describe a "silly" protocol which is AKE-secure but which is not secure against outsider impersonation attacks (we remark that there are more "natural" protocols with the same properties, but it is easiest to prove the claim with the protocol we describe). Basically, any AKE-secure protocol which achieves *implicit* authentication but not *explicit* authentication will suffice. Here is one possibility: starting with any AKE-secure protocol $\Pi$, construct protocol $\Pi'$ as follows:

- Upon receiving message $b|m$, run protocol $\Pi$ on input message $m$.

- When protocol $\Pi$ instructs to send message $m$, send message $0|m$.
- When the protocol has concluded, compute a temporary session key $sk'$ exactly as directed by $\Pi$. If all incoming messages were pre-pended by a "0", set $sk = sk'$ and accept iff directed to by $\Pi$. Otherwise, choose $sk$ at random and accept.

It is easy to see that $\Pi'$ remains AKE-secure. (Informally, if the adversary sends a message pre-pended with a "1" to an instance then that instance generates a session key which is chosen at random independent of everything else; if the adversary always sends messages pre-pended with a "0" to some instance then that instance will essentially just run $\Pi$, which is AKE-secure.) It is also easy to see that $\Pi'$ is not secure against outsider impersonation attacks since an adversary can cause any instance of any player to accept by simply sending to that instance a message pre-pended with a "1".

The above protocol does not guarantee agreement, either. To see this, consider an adversary who acts as a man-in-the-middle in an honest execution of the protocol between two parties, but who flips one of the pre-pended bits from a "0" to a "1". In this case, both players will accept but will compute different session keys with all but negligible probability. ■

# 3 Universally Composable Group Key Exchange Protocols

In this section, we formally define UC-security for group key exchange and then show that any UC-secure protocol is automatically AKE-secure and furthermore is also secure against insider attacks. In the following section, we show an efficient compiler which converts any AKE-secure group key-exchange protocol into a UC-secure group key-exchange protocol.

## 3.1 Group Key Exchange in the UC Framework

For a general overview of the UC framework, we refer the reader to [12, 14]; the latter, in particular, focuses on the UC framework in the context of (two-party) key exchange. A brief recap of the UC model is also provided in Appendix A. Roughly speaking, in the UC framework a cryptographic task is defined by specifying an appropriate ideal-world functionality $\mathcal{F}$; a protocol $\pi$ is then said to securely realize the desired task if the actions of the participants running $\pi$ in the real world (in the presence of a real-world adversary) can be appropriately *simulated* by an ideal-world adversary having access only to the ideal functionality $\mathcal{F}$ (and dummy parties interacting with $\mathcal{F}$). Thus, to formally define a notion of UC-security for group key exchange protocols, we specify an appropriate ideal-world functionality $\mathcal{F}_{\mathcal{GKE}}$ for group key exchange. The functionality is given in Figure 1. We now briefly explain the functionality and describe some choices made in its definition.

**The basic interface of the group key exchange protocols.** A player $U_i$ runs a group key exchange protocol with an input of the form $(\texttt{new-session}, \mathsf{sid}, \mathsf{pid})$ where $\mathsf{pid}$ is the set of identities of players with whom $U_i$ wants to share a session key and $\mathsf{sid}$ is a session ID. The local output of the protocol run by player $U_i$ takes the form $(\mathsf{sid}, \mathsf{pid}, \kappa)$ where $\kappa \in \{0,1\}^k$ is the session key and $k$ is the security parameter.

**Overview of the functionality.** We summarize the functionality as described in Figure 1, providing some commentary along the way. As expected, the functionality begins with an "initialization" phase in which the functionality waits to be notified by each of the players who are supposed to take part in an execution of the protocol. Once $\mathcal{F}_{\mathcal{GKE}}$ receives a notification from each of the players

**Ideal Functionality $\mathcal{F}_{\mathcal{GKE}}$**

$\mathcal{F}_{\mathcal{GKE}}$ proceeds as follows, running on security parameter $k$, with players $U_1, ..., U_n$, and an ideal adversary $\mathcal{S}$.

**Initialization:** Upon receiving a value $(\texttt{new-session}, \mathsf{sid}, \mathsf{pid})$ from player $U_i$ for the first time (where $\mathsf{pid}$ is a non-empty set of distinct user identities), record $(\mathsf{sid}, \mathsf{pid}, U_i)$ and send this to $\mathcal{S}$. In addition, if there are already $|\mathsf{pid}| - 1$ recorded tuples $(\mathsf{sid}, \mathsf{pid}, U_j)$ for $U_j \in \mathsf{pid} \setminus \{U_i\}$ then store $(\mathsf{sid}, \mathsf{pid}, \texttt{ready})$ and send this to $\mathcal{S}$.

**Key Generation:** Upon receiving a message $(\mathsf{sid}, \mathsf{pid}, \texttt{ok})$ from $\mathcal{S}$ where there is a recorded tuple $(\mathsf{sid}, \mathsf{pid}, \texttt{ready})$, do:

- If all $U \in \mathsf{pid}$ are uncorrupted, choose $\kappa \leftarrow \{0,1\}^k$ and store $(\mathsf{sid}, \mathsf{pid}, \kappa)$.
- If any of the $U \in \mathsf{pid}$ are corrupted, wait for $\mathcal{S}$ to send a message $(\texttt{key}, \kappa)$ and then store $(\mathsf{sid}, \mathsf{pid}, \kappa)$.

**Key Delivery:** If $\mathcal{S}$ sends a message $(\texttt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ where there is a recorded tuple $(\mathsf{sid}, \mathsf{pid}, \kappa)$ and $U_i \in \mathsf{pid}$, then send $(\mathsf{sid}, \mathsf{pid}, \kappa)$ to player $U_i$. (This message is delivered to $U_i$ immediately, as discussed in the text.)

**Player Corruption:** If $\mathcal{S}$ corrupts $U_i \in \mathsf{pid}$ where there is a recorded tuple $(\mathsf{sid}, \mathsf{pid}, \kappa)$ and message $(\mathsf{sid}, \mathsf{pid}, \kappa)$ has *not* yet been sent to $U_i$, then the adversary is given $\kappa$. Otherwise, $\mathcal{S}$ is given nothing.

Figure 1: The group key-exchange functionality $\mathcal{F}_{\mathcal{GKE}}$.

— with identical values of $\mathsf{sid}$ and $\mathsf{pid}$ — the functionality enters a "ready" state and informs the adversary to this effect by sending a $\mathsf{ready}$ message to the adversary.

Let $|\mathsf{pid}| = n$. At this point, the $n$ players expected to take part in the protocol are now all ready to receive a key. However, the functionality does not choose a key until it receives an "$\mathsf{ok}$" message from the adversary. The purpose of the $\mathsf{ready}/\mathsf{ok}$ messages is to allow the adversary the opportunity to corrupt players at some point in time *after* they have expressed interest in generating a shared key, but *before* this key has actually been generated. (In the real world, this corresponds to corrupting a player after it has begun execution of the protocol, but before it has terminated.) This "delay" in the functionality seems necessary in order to properly model corruptions that may occur at any time during execution of the protocol, and indeed — although omitted there — also seems necessary for the proof of security in [14].

Once the adversary sends the $\mathsf{ok}$ message, the functionality chooses a key. At this point, if none of the players in $\mathsf{pid}$ is corrupted, the session key is chosen uniformly at random from $\{0,1\}^k$. If at least one of the players in $\mathsf{pid}$ *is* corrupted, the adversary is allowed to choose the value of the session key, as in [14]. Finally, this key is delivered to the players according to a scheduling determined by the adversary. In particular, a key is delivered to a player only when delivery is requested by the adversary. Once the adversary requests that the functionality deliver a key to a player, we make the convention that the key is delivered to this player *immediately*. This convention follows the recent revisions of the UC framework (see [12, footnote 11]), and is *different* (and, in our mind, more natural) than the definitional choice made in [14].[3]

[3]Seemingly, another way to achieve the same effect would be to have the functionality output its messages to each of the parties immediately, and then have delivery of these messages be under the adversary's control. In order to properly model session state corruption (see below) as well as forward secrecy, however, we will require the

**Multiple sessions and session state corruption.** As discussed in [14], although key-exchange protocols are generally viewed as handling multiple sessions, it suffices (in the UC framework) to consider protocols and ideal functionalities handling only a *single* session. *Universal composition with joint state* (see [14, 15]) can then be used to obtain the so-called "multi-session" extension which handles multiple executions of the protocol. An important point is that for our purposes there is not even any efficiency loss in doing so, since the multi-session extension of an authenticated key-exchange protocol is the same as the underlying single-session protocol except that a "multi-session authentication module" is used. The latter are easy to construct using, e.g., any digital signature scheme as long as the unique (sub-)session ID of any given session is concatenated to any messages that are signed [15].

Focusing on single-session protocols simplifies the definitions and the analysis. However, as discussed in [14], doing so necessitates a slight change in the UC framework itself. In particular, when considering the multi-session extension of $\mathcal{F}_{\mathcal{GKE}}$ one needs to augment the basic UC model with a notion of *session-state corruption* which is intended to capture the same sort of attacks modeled by the Reveal oracle in the definition of AKE-security. (In the "standard" UC framework there is no notion of obtaining the state of a player without fully corrupting the player, nor is it possible for the adversary to obtain the state of a player for only a subset of that player's executions.) Such session state corruption is not explicit in Figure 1 since that figure only presents the single-session version of the functionality.

**Perfect forward secrecy.** Perfect forward secrecy is the notion that corruption of a player should not reveal previous session keys generated by that player. We have already noted above (and in footnote 3) that the presence of the "key delivery" phase of $\mathcal{F}_{\mathcal{GKE}}$, and the convention by which messages from $\mathcal{F}_{\mathcal{GKE}}$ are delivered immediately, are intended to ensure that $\mathcal{F}_{\mathcal{GKE}}$ "knows" when messages have been delivered, and we have claimed that such knowledge is helpful for an accurate modeling of forward secrecy. We now explain why this is so.

Notice that if a player is corrupted *after* having output the session key (in some execution of the protocol), then forward secrecy requires that the adversary *not* be able to learn the value of the session key computed by the player in that execution. Since the functionality is now "aware" of when a player has output the session key (since the player outputs the session key immediately after receiving the message from $\mathcal{F}_{\mathcal{GKE}}$, and the player receives this messages immediately after $\mathcal{F}_{\mathcal{GKE}}$ sends it), the functionality can give the appropriate information to the adversary when a corruption occurs. In particular, if the adversary corrupts a player *after* the session key has been sent to that player ("corresponding" to a corruption in the real world *after* the player has output a session key), the adversary is given nothing.

### 3.2 Relation to Previous Definitions

We say a group key exchange protocol is *UC-secure* if it securely realizes the (multi-session extension of) ideal functionality $\mathcal{F}_{\mathcal{GKE}}$. In other words, for any adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can determine whether it is interacting with $\mathcal{A}$ and players running the protocol or interacting with $\mathcal{S}$ in the ideal world and the (multi-session extension of) ideal functionality $\mathcal{F}_{\mathcal{GKE}}$. The following claims serve as useful "sanity checks" for our definition:

**Claim 2** *Any UC-secure group key-exchange protocol is AKE-secure.*

functionality to "know" when a message has been delivered to a party. This is achieved by having delivery occur immediately upon the adversary's request.

The proof, which appears in Appendix B.1, is quite straightforward and is very similar to the proof of the analogous result in the two party setting [14].

**Claim 3** *Any UC-secure group key-exchange protocol is secure against insider attacks.*

The intuition behind the proof of this claim, which appears in Appendix B.2, is rather straightforward given the definition of the ideal functionality in Figure 1. The main idea is that if a protocol is not secure against insider attacks, then there exists an adversary $\mathcal{A}$ which violates security against insider attacks when attacking this protocol. We use $\mathcal{A}$ along with an appropriate environment $\mathcal{Z}$ to distinguish interactions of $\mathcal{A}$ in the real world from interactions of *any* adversary $\mathcal{S}$ in the ideal world, thus proving that the protocol is not UC-secure.

# 4 Compiling AKE-Secure Protocols into UC-Secure Protocols

We have already shown (cf. Claims 1–3) that UC-security is strictly stronger than AKE-security. We show here, however, that any AKE-secure protocol $\pi$ can be *compiled* to give a UC-secure protocol $\pi'$. Our compiler is essentially the one suggested (without proof) in [24, Section 2.1] but is fundamentally *different* — although similar in spirit — from the compiler analyzed in [14] (as well as those of [5, 7]). Specifically, as pointed out in the Introduction, the compiler of [14] authenticates an "ack" message using a MAC keyed with a session key known to all parties participating in the protocol. Such an approach would simply not work in our setting, since a malicious insider would know the value of this session key and hence be able to impersonate the "ack" message of other (honest) players. Instead, our compiler uses a long-term signature scheme to *sign* an "ack" message of a similar sort. We have mentioned already in the Introduction, though, that certain technicalities arise. We set the stage for dealing with these in the following section.

## 4.1 Technical Preliminaries

We assume the existence of a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ which is existentially unforgeable against adaptive chosen-message attack. We also use a pseudorandom function family $F$ with the additional guarantee of what we term *collision-resistance*: Informally, this means that there exists a value $v_0$ such that no efficient adversary can find two different keys $s, s'$ such that $F_s(v_0) = F_{s'}(v_0)$. Formally:

**Definition 4** Let $\mathcal{F} = \{F^k\}$ with $F^k = \{F_s\}_{s\in\{0,1\}^k}$ be a pseudorandom function family (PRF). We say $\mathcal{F}$ is a *collision-resistant PRF* if there is an efficient procedure $\mathsf{Sample}$ such that the following is negligible in $k$ for all poly-time adversaries $\mathcal{A}$:

$$\Pr[v_0 \leftarrow \mathsf{Sample}(1^k); s, s' \leftarrow \mathcal{A}(1^k, v_0) : s, s' \in \{0,1\}^k \wedge s \neq s' \wedge F_s(v_0) = F_{s'}(v_0)].$$

Informally, the definition requires that for all $k$ large enough there exists an (efficiently computable) $v_0$ such that the function defined by $g(x) \stackrel{\text{def}}{=} F_x(v_0)$ is collision-resistant. $\diamondsuit$

It is easy to construct a collision-resistant PRF in the random oracle model: if $H$ is a random oracle, simply set $F_s(x) \stackrel{\text{def}}{=} H(s|x)$. It is also possible to construct[4] a collision-resistant PRF in the standard model based on any one-way permutation:

[4]We have subsequently noticed that the same result was previously shown by Fischlin [21]; in fact, a more efficient construction is also given there. We include the proof of Lemma 4 for self-containment.

**AKE→UC compiler**

Let $F$ be a collision-resistant PRF, and assume that $v_0$ is output by $\mathsf{Sample}(1^k)$ and publicly-known. Let $v_1 \neq v_0$ also be publicly-known.[a]

**Initialization Phase:** During the initialization phase of $\pi'$, long-term keys are initialized (in addition to any already present for $\pi$): each player $U_i$ runs $\mathsf{Gen}(1^k)$ to generate verification/signing keys $(PK_i, SK_i)$.

**The Protocol:** Players run protocol $\pi$. If $U_i$ would terminate without accepting in $\pi$, then it terminates without accepting in $\pi'$. Otherwise, if $U_i$ would accept (in protocol $\pi$) with output $(\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{sk}_i)$, this player performs the following additional steps:

1. $U_i$ computes $\mathsf{ack}_i = F_{\mathsf{sk}_i}(v_0)$ and $\mathsf{sk}'_i = F_{\mathsf{sk}_i}(v_1)$. Next, $U_i$ erases all its local state except for $\mathsf{ack}_i, \mathsf{sk}'_i$, $\mathsf{pid}_i$, and $\mathsf{sid}_i$. Then, $U_i$ computes a signature $\sigma_i \leftarrow \mathsf{Sign}_{SK_i}(\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{ack}_i)$ and sends the message $(U_i, \mathsf{ack}_i, \sigma_i)$ to all players in $\mathsf{pid}_i$.
2. Upon receipt of $|\mathsf{pid}_i| - 1$ messages $(U_j, \mathsf{ack}_j, \sigma_j)$ from all other players $U_j \in \mathsf{pid}_i \setminus \{U_i\}$, player $U_i$ checks that $\mathsf{Vrfy}_{PK_j}((\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{ack}_i), \sigma_j) = 1$ for all $j$. Assuming all verifications succeed, $U_i$ accepts, erases its internal state, and outputs $(\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{sk}'_i)$. If any of the verifications do not succeed, $U_i$ terminates without accepting (and with no output).

[a] As pointed out in the text, for the specific collision-resistant PRFs discussed in Section 4.1 no public information is needed.

Figure 2: The compiler is applied to AKE-secure protocol $\pi$ to yield UC-secure protocol $\pi'$.

**Lemma 4** *Assuming the existence of a one-way permutation, there exists a collision-resistant PRF.*

**Proof** We show that the Goldreich-Goldwasser-Micali [22] construction of a PRF from a one-way permutation $f$ actually gives a collision-resistant PRF. Recall the GGM construction: given one-way permutation $f : \{0,1\}^k \to \{0,1\}^k$ with hard-core predicate $h : \{0,1\}^k \to \{0,1\}$, first construct a length-doubling pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$ via:

$$G(s) = f^k(s)\, h(f^{k-1}(s)) \cdots h(s).$$

Let $G_0(s)$ denote the first $k$ bits of $G(s)$, and let $G_1(s)$ denote the last $k$ bits of $G(s)$. For a binary string $x = x_1 \cdots x_\ell$, define

$$F_s(x) = G_{x_\ell}(\cdots(G_{x_2}(G_{x_1}(s)))\cdots).$$

It is shown in [22] that the function family $\mathcal{F} = \{F^k\}$ with $F^k = \{F_s\}_{s\in\{0,1\}^k}$ is pseudorandom.

Now, note that $F_s(0^\ell) = f^{\ell\cdot k}(s)$. Since $f$ is a permutation, this means that the function $g(x) = F_x(0^\ell)$ is a permutation, and hence collision-resistant. (In fact, we achieve something even stronger than required by Definition 4.1: first, the $\mathsf{Sample}$ algorithm here is deterministic; second, collision-resistance holds information theoretically.) ■

## 4.2 The Compiler

Our compiler is presented in Figure 2, and we briefly describe it here. Given any protocol $\pi$, we construct protocol $\pi'$ as follows: first, we assume that values $v_0$ and $v_1 \neq v_0$ are publicly known[5]

[5] We remark that for both constructions of collision-resistant PRFs given in the previous section (i.e., based on random oracles or one-way permutations), this public information is not needed.

(where, informally, $v_0$ is a value for which Definition 4 is satisfied). During the initialization phase of $\pi'$, each player $U_i$ establishes long-term verification/signing keys $(PK_i, SK_i)$, in addition to any keys needed by $\pi$. The compiled protocol then runs $\pi$ until the point when $U_i$ is ready to accept (in $\pi$) with key $\mathsf{sk}_i$. (If $U_i$ would terminate without accepting in $\pi$, then $U_i$ terminates without accepting in $\pi'$.) Then, player $U_i$ computes $\mathsf{ack}_i = F_{\mathsf{sk}_i}(v_0)$ and $\mathsf{sk}'_i = F_{\mathsf{sk}_i}(v_1)$, and erases the rest of its state. It signs $\mathsf{ack}_i$ (along with $\mathsf{sid}_i, \mathsf{pid}_i$) and sends a copy of this signature to all other players. $U_i$ then waits to receive a message of this form from all players in $\mathsf{pid}_i \setminus \{U_i\}$. If any of the signatures it receives do not verify as expected, then $U_i$ terminates without accepting. Otherwise, it accepts with $\mathsf{sk}'_i$ as its session key.

We remark that our compiler actually fulfills two purposes. First, it ensures that the resulting protocol $\pi'$ satisfies the "ACK-property" as defined in [14]. Informally (see [14] for further details), a protocol satisfying the ACK-property has the feature that once an (uncorrupted) player $U_i$ outputs a session key $\mathsf{sk}_i$, the internal state of all *other* players in $\mathsf{pid}_i$ can be simulated given $\mathsf{sk}_i$ and the public information. The ACK-property was shown in [14] to be essential for proving UC security of key-exchange protocols. In addition to this, the use of a signature scheme (rather than a message authentication code as in [14]) ensures security against insider attacks. Use of a signature scheme is necessary, as a malicious insider knows the session key computed by the uncorrupted parties taking part in the protocol. Finally, we stress that the compiler requires the use of a *collision-resistant* PRF; the proof of security (below) breaks down if a "regular" PRF (without the collision-resistance property) is used instead.

We claim the following regarding the above compiler:

**Theorem 5** *If $\pi$ is an AKE-secure protocol, then applying the compiler of Figure 2 to $\pi$ results in a UC-secure protocol $\pi'$.*

**Proof** (Sketch) For simplicity, we show that $\pi'$ realizes $\mathcal{F}_{\mathcal{GKE}}$; however, it is not hard to adapt the proof below to show that $\pi'$ realizes the multi-session extension of $\mathcal{F}_{\mathcal{GKE}}$. (Alternately, universal composition with joint state [12, 14, 15] could be used to show that the multi-session extension of $\pi'$ securely realizes the multi-session extension of $\mathcal{F}_{\mathcal{GKE}}$.)

Let $\mathcal{A}$ be a real-life adversary. We describe in full an ideal-process adversary $\mathcal{S}$ such that no poly-time environment $\mathcal{Z}$ can tell whether it interacts with $\mathcal{A}$ and players running $\pi'$ in the real world, or with $\mathcal{S}$ and (dummy) players communicating with $\mathcal{F}_{\mathcal{GKE}}$ in the ideal world. $\mathcal{S}$ proceeds as follows (when we say $\mathcal{S}$ "aborts" we mean it sends a special abort signal to $\mathcal{Z}$ and halts):

1. Messages from $\mathcal{Z}$ to $\mathcal{S}$ are forwarded to $\mathcal{A}$, and messages from $\mathcal{A}$ to $\mathcal{S}$ are forwarded to $\mathcal{Z}$.

2. $\mathcal{S}$ generates public/private keys on behalf of all players, and gives the resulting public keys to $\mathcal{A}$. These include both the keys for $\pi$ as well as the keys required by the compiler itself.

3. When $\mathcal{S}$ receives a message $(\mathsf{sid}, \mathsf{pid}, U_i)$ from $\mathcal{F}_{\mathcal{GKE}}$ for an uncorrupted player $U_i$, it begins simulating for $\mathcal{A}$ a copy of protocol $\pi'$ being run by $U_i$ with session ID $\mathsf{sid}$ and partner ID $\mathsf{pid}$. Any messages sent by $\mathcal{A}$ to $U_i$ are processed by this simulated copy of $\pi'$, and any messages output by the simulated copy of $\pi'$ are given to $\mathcal{A}$.

4. If at any point in time a simulated copy of $\pi'$ being run on behalf of an uncorrupted player $U_i$ outputs a session key $\mathsf{sk}'$, adversary $\mathcal{S}$ checks to see whether any of the players in $\mathsf{pid}$ have been corrupted.

   (a) If no players in $\mathsf{pid}$ are corrupted, then:

i. If $\mathcal{S}$ has not yet sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ checks that it has received message $(\mathsf{sid}, \mathsf{pid}, \mathtt{ready})$ from $\mathcal{F}_{\mathcal{GKE}}$. If not, $\mathcal{S}$ aborts. Otherwise, it sends $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$, followed by $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$.

ii. If $\mathcal{S}$ has already sent the message $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ sends the message $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$.

(b) Otherwise, say $C \subseteq \mathsf{pid} \setminus U_i$ are the corrupted players. Then:

i. If $\mathcal{S}$ has not yet sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ first sends $(\mathtt{new\text{-}session}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$ on behalf of any of the players in $C$ who have not done so already. If $\mathcal{S}$ has not received message $(\mathsf{sid}, \mathsf{pid}, \mathtt{ready})$ from $\mathcal{F}_{\mathcal{GKE}}$ by this point, it aborts. Otherwise, it sends $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$, $(\mathtt{key}, \mathsf{sk}')$, and $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$.

ii. If $\mathcal{S}$ has already sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$ and no players in $\mathsf{pid}$ were corrupted at that point in time, then $\mathcal{S}$ sends $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$.

iii. Otherwise, $\mathcal{S}$ has already sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ and $(\mathtt{key}, \mathsf{sk}'')$ to $\mathcal{F}_{\mathcal{GKE}}$ (i.e., a player in $\mathsf{pid}$ was corrupted at the time the "ok" message was sent). If $\mathsf{sk}'' \neq \mathsf{sk}'$ then $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ sends $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$.

5. When $\mathcal{A}$ corrupts a player $U_i$, $\mathcal{S}$ corrupts that player in the ideal world. $\mathcal{S}$ also gives $\mathcal{A}$ all the secret keys of player $U_i$. Finally, $\mathcal{S}$ provides $\mathcal{A}$ with the current internal state of $U_i$ as follows:

(a) If $\mathcal{S}$ has not yet sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ simply gives $\mathcal{A}$ the current internal state of the simulated copy of $\pi'$ being run on behalf of $U_i$.

(b) If $\mathcal{S}$ sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ to $\mathcal{F}_{\mathcal{GKE}}$ but has not yet sent $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ obtains $(\mathsf{sid}, \mathsf{pid}, \kappa)$ from $\mathcal{F}_{\mathcal{GKE}}$ when it corrupts $U_i$. If the simulated copy of $\pi'$ being run on behalf of $U_i$ does not include a value $\mathsf{ack}_i$, then $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ hands to $\mathcal{A}$ the internal state $(\mathsf{ack}_i, \kappa, \mathsf{sid}, \mathsf{pid})$.

(c) If $\mathcal{S}$ sent $(\mathsf{sid}, \mathsf{pid}, \mathtt{ok})$ and $(\mathtt{deliver}, U_i, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$, then $\mathcal{S}$ returns an empty internal state to $\mathcal{A}$.

The above constitutes a complete description of $\mathcal{S}$. However, we will only sketch the argument that no poly-time $\mathcal{Z}$ can distinguish its interactions with $\mathcal{S}$ (in the ideal world) from its interactions with $\mathcal{A}$ (in the real world). We begin by stating a claim which is central to the proof. In the remainder of the proof, we let $U_i$ refer interchangeably to the simulated copy of $\pi'$ being run by $\mathcal{S}$ on behalf of $U_i$; in contrast, when we wish to refer to a dummy player in the ideal world we will write $U_i^0$.

**Claim 6** *Except with negligible probability, whenever an uncorrupted player $U_i$ outputs $(\mathsf{sid}, \mathsf{pid}, \mathsf{sk}')$ and holds state*[6] *$(\mathsf{ack}, \mathsf{sk}', \mathsf{pid}, \mathsf{sid})$, then every uncorrupted player $U_j \in \mathsf{pid}$ has ended its execution of $\pi$ and either holds state $(\mathsf{ack}, \mathsf{sk}', \mathsf{pid}, \mathsf{sid})$ (if $U_j$ has not yet completed its execution of $\pi'$) or has already output $(\mathsf{sid}, \mathsf{pid}, \mathsf{sk}')$ (if $U_j$ has completed its execution of $\pi'$).*

**Proof (of claim)** Say uncorrupted player $U_i$ accepts and holds state $(\mathsf{ack}_i, \mathsf{sk}_i', \mathsf{pid}_i, \mathsf{sid}_i)$. Then it must be the case that $U_i$ has received valid signatures on $(\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{ack}_i)$ from all other players in $\mathsf{pid}_i$. Considering any uncorrupted player $U_j \in \mathsf{pid}_i$, unless the adversary $\mathcal{A}$ has forged a signature with respect to the public key of $U_j$ (which occurs with only negligible probability by security of

[6] Note that this represents the state held by the player immediately *before* it outputs $\mathsf{sk}'$; after it outputs the key, this state is erased as directed by the compiler.

the signature scheme), this means that a simulated copy of $\pi'$ being run by $U_j$ has generated a signature on $(\mathsf{sid}_i, \mathsf{pid}_i, \mathsf{ack}_i)$ and so, in particular, $U_j$ has ended its execution of $\pi$. The only thing remaining to show is that the value $\mathsf{sk}'_j$ held by $U_j$ is identical to the value $\mathsf{sk}'_i$ held by $U_i$. Since, by construction of the compiler, $F_{\mathsf{sk}_i}(v_0) = \mathsf{ack}_i = F_{\mathsf{sk}_j}(v_0)$ and $F$ is a collision-resistant PRF, we have $\mathsf{sk}_i = \mathsf{sk}_j$ except with negligible probability. Assuming this to be the case, we then have

$$\mathsf{sk}'_j = F_{\mathsf{sk}_j}(v_1) = F_{\mathsf{sk}_i}(v_1) = \mathsf{sk}'_i,$$

as desired. □

**Corollary 7** *Except with negligible probability, any uncorrupted players $U_i, U_j$ who output a session key will in fact output the* same *session key.*

We now summarize the differences, from the point of view of $\mathcal{Z}$, between an interaction with $\mathcal{A}$ and an interaction with $\mathcal{S}$:

• Steps 1, 2, and 3 of $\mathcal{S}$ do not introduce any differences from the point of view of $\mathcal{Z}$.

• Step 4(a)(i) introduces two differences. First, $\mathcal{S}$ may abort. Second, the key output by "dummy" player $U_i^0$ (as observed by $\mathcal{Z}$) is chosen uniformly at random by $\mathcal{F}_{\mathcal{GKE}}$, not as $\mathsf{sk}'$.

We first claim that the probability that $\mathcal{S}$ aborts at this step is negligible. If $\mathcal{S}$ aborts at this step, it means that $\mathcal{F}_{\mathcal{GKE}}$ has not yet sent $(\mathsf{sid}, \mathsf{pid}, \texttt{ready})$ to $\mathcal{S}$ or, equivalently, there exists some player $U_j^0 \in \mathsf{pid} \setminus U_i^0$ from whom $\mathcal{F}_{\mathcal{GKE}}$ has not yet received $(\texttt{new-session}, \mathsf{sid}, \mathsf{pid})$. But Claim 6 shows that, except with negligible probability, all uncorrupted players in $\mathsf{pid}$ have completed their execution of $\pi'$ and thus, in particular, have sent $(\texttt{new-session}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$ (note that $\mathcal{S}$ does not begin running the simulated copy of $\pi'$ for a player $U_j$ until $\mathcal{S}$ receives $(\mathsf{sid}, \mathsf{pid}, U_j)$ from $\mathcal{F}_{\mathcal{GKE}}$, and $\mathcal{F}_{\mathcal{GKE}}$ does not send this until player $U_j^0$ sends the appropriate "$\texttt{new-session}$" message to $\mathcal{F}_{\mathcal{GKE}}$). Since, in this step, all players in $\mathsf{pid}$ are uncorrupted, it follows that $\mathcal{S}$ aborts with only negligible probability.

We claim also that it is computationally indistinguishable (from the point of view of $\mathcal{Z}$) whether "dummy" player $U_i^0$ outputs a random session key (as it does in the ideal world) or $U_i$ outputs the session key $\mathsf{sk}'$ (as would occur in the real world). We may consider two cases:

1. If $\mathcal{A}$ never corrupts any players in the remainder of its execution, then this claim follows readily from Corollary 7, AKE-security of $\pi$, and the pseudorandomness of $F$. (Corollary 7 is needed to argue that any uncorrupted players who output a session key in the simulations being provided by $\mathcal{S}$ will output *identical* session keys, exactly as observed by $\mathcal{Z}$ for the outputs of the dummy players.)

2. If $\mathcal{A}$ later corrupts some players in $\mathsf{pid}$, then due to the way corruptions are handled by $\mathcal{S}$ this will not introduce any noticeable difference from the point of view of $\mathcal{Z}$ (again relying on Claim 6, AKE-security of $\pi$, and the pseudorandomness of $F$). In particular, if a player $U' \in \mathsf{pid}$ is corrupted before it outputs a key, then $\mathcal{S}$ obtains the key $\kappa$ from $\mathcal{F}_{\mathcal{GKE}}$ and "patches" the internal state of $U'$ appropriately (cf. step 5(b)). If $U' \in \mathsf{pid}$ is corrupted after it outputs a key, there is nothing to simulate (cf. step 5(c)).

• Step 4(a)(ii) introduces the following difference: the key output by "dummy" player $U_i^0$ (as observed by $\mathcal{Z}$) is chosen uniformly at random by $\mathcal{F}_{\mathcal{GKE}}$, not as $\mathsf{sk}'$. That this is inconsequential follows a similar line of reasoning as in the case of step 4(a)(i).

• In step 4(b)(i), $\mathcal{S}$ may abort. However, this only occurs if there is some uncorrupted player in $\mathsf{pid}$ who has not yet sent $(\texttt{new-session}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\mathcal{GKE}}$. As argued in the case of step 4(a)(i), however, it follows from Claim 6 that this occurs with only negligible probability.

We remark also that the key $\mathsf{sk}'$ that $\mathcal{S}$ sends to $\mathcal{F}_{\mathcal{GKE}}$ matches exactly the key that (uncorrupted) player $U_i$ outputs. So the simulation is perfect in that respect. Furthermore, Corollary 7 indicates that, except with negligible probability, if the simulated copy of $\pi'$ being run on behalf of any other honest player later outputs a session key, that key will be $\mathsf{sk}'$.

• In step 4(b)(ii), $U_i$ has output a session key $\mathsf{sk}'$. Furthermore, in this step, we know that at the time the first uncorrupted player (say, $U_j$) accepted, all players in $\mathsf{pid}$ were uncorrupted. But then Corollary 7 shows that, with all but negligible probability, the key $\mathsf{sk}'$ output here is identical to the key previously output by $U^*$. Similarly, the session keys output by the "dummy" parties $U_j^0$ and $U_i^0$ (as observed by $\mathcal{Z}$) will be identical.

• In step 4(b)(iii), we know there was an uncorrupted player who previously output session key $\mathsf{sk}''$. Corollary 7 indicates that every uncorrupted player who outputs a session key will output the same session key $\mathsf{sk}''$. So, $\mathcal{S}$ will not abort (except with negligible probability), and the simulation in this step will be perfect.

• Steps 5(a) and 5(c) do not introduce any differences from the point of view of $\mathcal{Z}$ (note, in particular, that since $\mathcal{S}$ has not sent the "ok" message to $\mathcal{F}_{\mathcal{GKE}}$, no session key has yet been chosen by $\mathcal{F}_{\mathcal{GKE}}$ in step 5(a)).

• In step 5(b), note that if $\mathcal{S}$ has sent an "ok" message to $\mathcal{F}_{\mathcal{GKE}}$ then there must be some other player $U_j$ — different from the player $U_i$ being corrupted in this step — who was uncorrupted at the time it accepted (this is because $\mathcal{S}$ only sends the "ok" message when this occurs). From Claim 6, this means that with all but negligible probability $U_i$ indeed has a value $\mathsf{ack}_i$ as part of its internal state (and so $\mathcal{S}$ will not abort in this step). It then follows from the pseudorandomness of $F$ that including the value $\kappa$ (that was output already by "dummy" player $U_j^0$) in the internal state is computationally indistinguishable from using the actual session key computed by $U_i$.

This completes our sketch of the proof. ■

# 5 Conclusion

This paper provides a formal and comprehensive way of modeling insider attacks in group key-exchange protocols. We show that the definition introduced in this work is strictly stronger than that of AKE-security, and then show a simple and efficient compiler which transforms any AKE-secure protocol into one secure with respect to our definition. We hope the framework introduced here will provide a basis for future work analyzing the security of existing group key-exchange protocols, and will also serve as a tool toward developing more efficient protocols secure against insider attacks.

# References


[1] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik. Exploring Robustness in Group Key Agreement. *ICDCS* 2001.

[2] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Crypto* '93.

[3] M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. *STOC* '95.

[4] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. *STOC* '98.

[5] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Eurocrypt* 2000.

[6] S. Blake-Wilson, D. Johnson, and A. Menezes. Key Exchange Protocols and Their Security Analysis. *Proc. 6th IMA Intl. Conf. on Cryptography and Coding*, 1997.

[7] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. *ACM CCCS* 2001.

[8] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange — The Dynamic Case. *Asiacrypt* 2001.

[9] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. *Eurocrypt* 2002.

[10] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *Crypto* '91.

[11] C. Cachin and R. Strobl. Asynchronous Group Key Exchange With Failures. *PODC* 2004.

[12] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Manuscript dated Jan. 28, 2005, available at http://eprint.iacr.org/2000/067. A preliminary version appeared in *FOCS* 2001.

[13] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. *Eurocrypt* 2001.

[14] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Eurocrypt* 2002. Full version available at http://eprint.iacr.org/2002/059.

[15] R. Canetti and T. Rabin. Universal Composition with Joint State. *Crypto* 2003.

[16] Z. Cheng, L. Vasiu, and R. Comley. Pairing-Based One-Round Tripartite Key Agreement Protocols. Available at http://eprint.iacr.org/2004/079.

[17] H.-Y. Chien. Comments: Insider Attack on Cheng et al.'s Pairing-Based Tripartite Key Agreement Protocols. Available at http://eprint.iacr.org/2005/013.

[18] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Trans. Info. Theory* 22(6): 644–654 (1976).

[19] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *DCC* 2(2): 107–125 (1992).

[20] X. Du, Y. Wang, J. Ge, and Y. Wang. An Improved ID-Based Authenticated Group Key Agreement Scheme. Available at http://eprint.iacr.org/2003/260.

[21] M. Fischlin. Pseudorandom Function Tribe Ensembles Based on One-Way Permutations: Improvements and Applications. *Eurocrypt* '99.

[22] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *J. ACM* 33(4): 792–807 (1986).

[23] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. *ANTS* 2000.

[24] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. *Crypto* 2003. Full version available at http://eprint.iacr.org/2003/171.

[25] G. Lowe. A Hierarchy of Authentication Specifications. *Computer Security Foundations Workshop* '97.

[26] B. Pfitzmann, M. Steiner, and M. Waidner. A Formal Model for Multi-Party Group Key Agreement. Technical Report RZ-3383 (#93419), IBM Research.

[27] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and Its Application to Secure Message transmission. *IEEE Security and Privacy*, 2001.

[28] S. Saeednia and R. Safavi-Naini. Efficient Identity-Based Conference Key-Distribution Protocols. *ACISP* '98.

[29] K. Shim. Cryptanalysis of Al-Riyami-Paterson's Authenticated Three Party Key Agreement Protocols. Available at http://eprint.iacr.org/2003/122.

[30] V. Shoup. On Formal Models for Secure Key Exchange. Available at http://eprint.iacr.org/1999/012.

[31] M. Steiner. Secure Group Key Agreement. PhD Thesis, Universitat des Saarlandes, 2002. Available at http://www.semper.org/sirene/publ/Stei_02.thesis-final.pdf.

[32] H.-M. Sun and B.-T. Hsieh. Security Analysis of Shim's Authenticated Key Agreement Protocols from Pairings. Available at http://eprint.iacr.org/2003/113.

[33] Q. Tang and C.J. Mitchell. Rethinking the Security of Some Authenticated Group Key Agreement Schemes. Available at http://eprint.iacr.org/2004/348.

[34] F. Zhang and X. Chen. Attack on an ID-based Authenticated Group Key Agreement Scheme from *PKC* 2004. *Info. Proc. Lett.* 91(4): 191–192 (2004). Also available at http://eprint.iacr.org/2003/259.

# A Brief Review of the UC Framework

We provide a brief review of the universally composable security framework [12]. The framework allows for defining the security properties of cryptographic tasks so that security is maintained under general composition with an unbounded number of instances of arbitrary protocols running concurrently. In the UC framework, the security requirements of a given task are captured by specifying an ideal functionality run by a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs. Informally, then, a protocol securely carries out a given task if running the protocol in the presence of a real-world adversary amounts to "emulating" the desired ideal functionality.

The notion of emulation in the UC framework is considerably stronger than that considered in previous models. As usual, the real-world model includes the parties running the protocol and an adversary $\mathcal{A}$ who controls their communication and potentially corrupts parties, while the ideal-world includes a simulator $\mathcal{S}$ who interacts with an ideal functionality $\mathcal{F}$ and dummy players who simply send input to/receive output from $\mathcal{F}$. In the UC framework, there is also an additional entity called the *environment* $\mathcal{Z}$. This environment generates the inputs to all parties, observes all their outputs, and interacts with the adversary in an arbitrary way throughout the computation. A protocol $\pi$ is said to *securely realize* an ideal functionality $\mathcal{F}$ if for any real-world adversary $\mathcal{A}$ that interacts with $\mathcal{Z}$ and real players running $\pi$, there exists an ideal-world simulator $\mathcal{S}$ that interacts with $\mathcal{Z}$, the ideal functionality $\mathcal{F}$, and the "dummy" players communicating with $\mathcal{F}$, such that *no* poly-time environment $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ (in the real world) or $\mathcal{S}$ (in the ideal world). $\mathcal{Z}$ thus serves as an "interactive distinguisher" between a real-world execution of the protocol $\pi$ and an ideal execution of functionality $\mathcal{F}$. A key point is that $\mathcal{Z}$ cannot be re-wound by $\mathcal{S}$; in other words, $\mathcal{S}$ must provide a so-called "straight-line" simulation.

The following *universal composition theorem* is proven in [12]. Consider a protocol $\pi$ that operates in the $\mathcal{F}$-hybrid model, where parties can communicate as usual and in addition have ideal access to an unbounded number of *copies* of the functionality $\mathcal{F}$. Let $\rho$ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let $\pi^\rho$ be identical to $\pi$ with the exception that the interaction with *each copy* of $\mathcal{F}$ is replaced with an interaction with a *separate instance* of $\rho$. Then, $\pi$ and $\pi^\rho$ have essentially the same input/output behavior. In particular, if $\pi$ securely realizes some functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model then $\pi^\rho$ securely realizes $\mathcal{G}$ in the standard model (i.e., without access to any functionality).

# B Proofs of Claims

## B.1 Proof of Claim 2

We show that any UC-secure protocol is also AKE-secure. The basic idea is very similar to the proof in the two party setting [14] except that we give a direct proof without introducing an intermediate notion of security (cf. SK-security in [14]). Let $\pi$ be a UC-secure group key exchange protocol, and let $\tilde{\pi}$ be the multi-session extension of $\pi$ [12, 14, 15] which UC-securely realizes $\tilde{\mathcal{F}}_{\mathcal{GKE}}$, the multi-session extension of $\mathcal{F}_{\mathcal{GKE}}$. Assume to the contrary that $\tilde{\pi}$ is not AKE-secure. Then there exists an adversary $\tilde{\mathcal{A}}$ breaking the AKE-security of $\tilde{\pi}$ with non-negligible probability. We use $\tilde{\mathcal{A}}$ to construct an environment machine $\mathcal{Z}$ and a real-life adversary $\mathcal{A}$ so that for *any* ideal adversary $\mathcal{S}$, $\mathcal{Z}$ can distinguish whether it interacts with $\mathcal{A}$ and players running $\tilde{\pi}$ in the real world, or with $\mathcal{S}$ and dummy players communicating with $\tilde{\mathcal{F}}_{\mathcal{GKE}}$ in the ideal world. Environment machine $\mathcal{Z}$ and real-life adversary $\mathcal{A}$ proceed as follows:

1. When $\tilde{\mathcal{A}}$ asks $\mathsf{Send}(U, i, (\mathsf{ssid}, \mathsf{pid}))$, this message is forwarded to $\mathcal{Z}$ who then invokes player $U$ with input $(\texttt{new-session}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pid})$. Also, $\mathcal{Z}$ and $\mathcal{A}$ record $(U, i, \mathsf{ssid}, \mathsf{pid})$ as an uncompleted and unexposed session.

2. When $\tilde{\mathcal{A}}$ asks $\mathsf{Send}(U, i, M)$, $\mathcal{A}$ checks if there is a uncompleted session $(U, i, \mathsf{ssid}, \mathsf{pid})$. If not, $\mathcal{A}$ returns "invalid query". Otherwise, $\mathcal{A}$ delivers $M$ to the appropriate session of player $U$, and returns to $\tilde{\mathcal{A}}$ the response of $U$.

3. When a player $U$ outputs $(\mathsf{ssid}, \mathsf{pid}, \kappa)$, $\mathcal{Z}$ records $(U, i, \mathsf{ssid}, \mathsf{pid}, \kappa)$ as a completed session.

4. When $\tilde{\mathcal{A}}$ asks $\mathsf{Reveal}(U, i)$, $\mathcal{A}$ forwards this message to $\mathcal{Z}$ who then checks if there is a $\mathsf{completed}$ session $(U, i, \mathsf{ssid}, \mathsf{pid}, \kappa)$. If not, $\mathcal{Z}$ tells $\mathcal{A}$ to return "invalid query". Otherwise, $\mathcal{Z}$ gives $\kappa$ to $\tilde{\mathcal{A}}$ (via $\mathcal{A}$) and marks the session $\mathsf{exposed}$.

5. When $\tilde{\mathcal{A}}$ asks $\mathsf{StrongCorrupt}(U)$, $\mathcal{A}$ corrupts player $U$ and provides $\tilde{\mathcal{A}}$ with the internal state of $U$. $\mathcal{Z}$ (who finds out about this corruption) marks $U$ and all the $\mathsf{uncompleted}$ sessions belonging to $U$ as $\mathsf{corrupted}$. Moreover, any sessions invoked by $U$ in the future will be marked $\mathsf{exposed}$.

6. When $\tilde{\mathcal{A}}$ asks $\mathsf{Test}(U, i)$, this message is forwarded to $\mathcal{Z}$ who checks if there is a record $(U, i, \mathsf{ssid}, \mathsf{pid}, \kappa)$ marked as $\mathsf{completed}$ and $\mathsf{unexposed}$. If not, $\mathcal{Z}$ outputs a random bit and halts. Otherwise, $\mathcal{Z}$ flips a coin $b \leftarrow \{0, 1\}$. If $b = 0$, $\mathcal{Z}$ provides $\tilde{\mathcal{A}}$ (via $\mathcal{A}$) with a random session key. If $b = 1$, $\mathcal{Z}$ provides $\tilde{\mathcal{A}}$ (via $\mathcal{A}$) with $\kappa$. Also, $\mathcal{Z}$ marks the session as $\mathsf{test}$.

7. When $\tilde{\mathcal{A}}$ outputs a guess bit $b'$, this is forwarded to $\mathcal{Z}$ who proceeds as follows:

   (a) First, $\mathcal{Z}$ finds the $\mathsf{test}$ session record $(U, i, \mathsf{ssid}, \mathsf{pid}, \kappa)$ and then finds all the sessions whose session ID is $\mathsf{ssid}$ and marks them as $\mathsf{matching}$.

   (b) $\mathcal{Z}$ checks if any of sessions marked as $\mathsf{test}$ or $\mathsf{matching}$ is $\mathsf{exposed}$. If it is, $\mathcal{Z}$ outputs a random bit. Otherwise, $\mathcal{Z}$ outputs 1 if $b' = b$ and outputs 0 if $b' \neq b$.

First, we consider the case when $\mathcal{Z}$ is interacting with $\tilde{\mathcal{A}}$ (via $\mathcal{A}$) in the real world with players running $\tilde{\pi}$. Since it is clear that a $\mathsf{fresh}$ instance (according to the AKE-security definition) is $\mathsf{unexposed}$ and $\mathsf{completed}$, then if $\tilde{\mathcal{A}}$ distinguishes a real session key (of a fresh instance) from a random session key with probability non-negligibly better than $\frac{1}{2}$, then $\mathcal{Z}$ outputs 1 with probability non-negligibly greater than $\frac{1}{2}$.

On the other hand, when $\mathcal{Z}$ is interacting with $\mathcal{S}$ and dummy players in an ideal execution with$\tilde{\mathcal{F}}_{\mathcal{GKE}}$, the key of any $\mathsf{unexposed}$ and $\mathsf{completed}$ session has the same distribution as a random key since it is chosen uniformly at random by $\tilde{\mathcal{F}}_{\mathcal{GKE}}$. Therefore, there is no way for $\mathcal{S}$ to distinguish a real session key from a random key for $\mathsf{unexposed}$ sessions. This implies that no ideal adversary $\mathcal{S}$ can skew the output of $\mathcal{Z}$ from a random bit.

Since the above cannot happen if $\tilde{\pi}$ is UC-secure, it follows that $\tilde{\pi}$ is AKE-secure.

## B.2 Proof of Claim 3

Let $\pi$ be a UC-secure group key exchange protocol, and let $\tilde{\pi}$ be the multi-session extension of $\pi$ [12, 14, 15] which UC-securely realizes $\tilde{\mathcal{F}}_{\mathcal{GKE}}$, the multi-session extension of $\mathcal{F}_{\mathcal{GKE}}$. Assume to the contrary that $\tilde{\pi}$ is not secure against insider attacks. Then there exists an adversary $\mathcal{A}_{in}$ violating insider security of $\tilde{\pi}$. We use $\tilde{\mathcal{A}}$ to construct an environment machine $\mathcal{Z}$ and a real-life adversary $\mathcal{A}$ so that for *any* ideal adversary $\mathcal{S}$, $\mathcal{Z}$ can distinguish whether it interacts with $\mathcal{A}$ and players running $\tilde{\pi}$ in the real world, or with $\mathcal{S}$ and dummy players communicating with $\tilde{\mathcal{F}}_{\mathcal{GKE}}$ in the ideal world. Environment machine $\mathcal{Z}$ and real-life adversary $\mathcal{A}$ proceed as follows:

1. When $\mathcal{A}_{in}$ asks $\mathsf{Send}$, $\mathsf{Reveal}$, or $\mathsf{StrongCorrupt}$, $\mathcal{Z}$ and $\mathcal{A}$ proceed as described in the proof of Claim 2 (cf. Appendix B.1).

2. During the execution, $\mathcal{Z}$ outputs 1 and halts if the following event happens: there exist players $U, U'$ and a $\mathsf{completed}$ and $\mathsf{unexposed}$ session $(U, i, \mathsf{ssid}, \mathsf{pid})$ such that $U'$ was not corrupted before this session was completed, $U' \in \mathsf{pid}$, but there is no invoked session of $U'$ holding $(U', j, \mathsf{ssid}, \mathsf{pid})$ (for any value of $j$).

3. During the execution, $\mathcal{Z}$ outputs 1 and halts if the following event happens: there exist two players $U$ and $U'$ and sessions $(U, i, \mathsf{ssid}, \mathsf{pid}, \kappa)$, $(U', j, \mathsf{ssid}, \mathsf{pid}, \kappa')$ which are completed and unexposed but $\kappa \neq \kappa'$.

4. Otherwise, $\mathcal{Z}$ outputs 0.

First, we consider the case when $\mathcal{Z}$ is interacting with $\mathcal{A}_{in}$ (via $\mathcal{A}$) in the real world with players running $\tilde{\pi}$. Clearly, whenever $\mathcal{A}_{in}$ violates security against insider impersonation attacks or violates agreement, $\mathcal{Z}$ outputs 1. So, if $\mathcal{A}_{in}$ violates security of $\tilde{\pi}$ against insider attacks with non-negligible probability, then $\mathcal{Z}$ outputs 1 with non-negligible probability.

On the other hand, when $\mathcal{Z}$ is interacting with $\mathcal{S}$ and dummy players in an ideal execution with $\tilde{\mathcal{F}}_{\mathcal{GKE}}$, we claim that $\mathcal{Z}$ never outputs 1. First of all, the event in step 2 does not happen in the ideal world because a copy of $\mathcal{F}_{\mathcal{GKE}}$ running within $\tilde{\mathcal{F}}_{\mathcal{GKE}}$ does not proceed in the session ssid until it receives $(\texttt{new-session}, \mathsf{sid}, \mathsf{ssid}, U, \mathsf{pid})$ from *all* the players $U \in \mathsf{pid}$ (regardless of how many players are corrupted). Therefore, $\mathcal{S}$ cannot force a uncorrupted player in a session to output a key when at least one of players in pid has not invoked the session with $(\mathsf{ssid}, \mathsf{pid})$.

Next, let us see why the event in step 3 does not happen in the ideal world, either. No matter whether $\mathcal{S}$ or $\mathcal{F}_{\mathcal{GKE}}$ has chosen the key value, $\mathcal{F}_{\mathcal{GKE}}$ distributes the same session key to all the players who receive a key in any particular session. Once a key is chosen, even though $\mathcal{S}$ can instruct $\tilde{\mathcal{F}}_{\mathcal{GKE}}$ to deliver or not to deliver the key to each player, $\mathcal{S}$ cannot modify the key value delivered to uncorrupted players. Therefore, any uncorrupted players who output a session key will output the same session key.

It follows that $\mathcal{Z}$ never outputs 1 in the latter case. Since the above claims would contradict the UC-security of $\tilde{\pi}$, it follows that $\tilde{\pi}$ is secure against insider attacks.