Scalable Protocols for Authenticated Group Key Exchange

Jonathan Katz[1] and Moti Yung[2]

[1] Dept. of Computer Science, University of Maryland, College Park, MD
jkatz@cs.umd.edu

[2] Dept. of Computer Science, Columbia University, New York, NY
moti@cs.columbia.edu

Abstract. We consider the fundamental problem of authenticated group key exchange among n parties within a larger and insecure public network. A number of solutions to this problem have been proposed; however, all provably-secure solutions thus far are not scalable and, in particular, require n rounds. Our main contribution is the first *scalable* protocol for this problem along with a rigorous proof of security in the standard model under the DDH assumption; our protocol uses a constant number of rounds and requires only $O(1)$ modular exponentiations per user (for key derivation). Toward this goal and of independent interest, we first present a scalable compiler that transforms any group key-exchange protocol secure against a passive eavesdropper to an *authenticated* protocol which is secure against an active adversary who controls all communication in the network. This compiler adds only one round and $O(1)$ communication (per user) to the original scheme. We then prove secure — against a passive adversary — a variant of the two-round group key-exchange protocol of Burmester and Desmedt. Applying our compiler to this protocol results in a provably-secure three-round protocol for *authenticated* group key exchange which also achieves forward secrecy.

1 Introduction

Protocols for authenticated key exchange (AKE) allow a group of parties within a larger and completely insecure public network to establish a common secret key (a *session key*) and furthermore to be guaranteed that they are indeed sharing this key with *each other* (i.e., with their intended partners). Protocols for securely achieving AKE are fundamental to much of modern cryptography. For one, they are crucial for allowing symmetric-key cryptography to be used for encryption/authentication of data among parties who have no alternate "out-of-band" mechanism for agreeing upon a common key. Furthermore, they are instrumental for constructing "secure channels" on top of which higher-level protocols can be designed, analyzed, and implemented in a modular manner. Thus, a detailed understanding of AKE — especially the design of provably-secure protocols for achieving it — is critical.

The case of 2-party AKE has been extensively investigated (e.g., [19, 8, 20, 6, 25, 4, 30, 15–17] and others) and is fairly well-understood; furthermore, a variety of efficient and provably-secure protocols for 2-party AKE are known. Less attention has been given to the important case of *group* AKE where a session key is to be established among n parties; we survey relevant previous work in the sections that follow. Group AKE protocols are essential for applications such as secure video- or tele-conferencing, and also for collaborative (peer-to-peer) applications which are likely to involve a large number of users. The recent foundational papers of Bresson, et al. [13, 11, 12] (building on [6, 7, 5]) were the first to present a formal model of security for group AKE and the first to give rigorous proofs of security for particular protocols. These represent an important initial step, yet much work remains to be done to improve the efficiency and scalability of existing solutions.

1.1 Our Contributions

We may summarize the prior "state-of-the-art" for group AKE as follows (see Section 1.2 for a more detailed discussion of previous work):

– The best-known provably-secure solutions in the standard model are those of [13, 11, 12], building on [31]. These protocols do not scale well: to establish a key among n participants, they require n rounds and additionally require (for some players) $O(n)$ modular exponentiations and $O(n)$ communication.
– Subsequent to the present work, a constant-round protocol for group AKE has been proven secure *in the random oracle model* [10]. Unfortunately, this protocol does *not* achieve forward secrecy (an explicit attack is known [10]). The protocol is also not symmetric; furthermore, the initiator of the protocol must perform $O(n)$ encryptions and send $O(n)$ communication.

Our main result is the first *constant-round* and *fully-scalable* protocol for group AKE which is provably-secure in the standard model. Security is proved (in the same security model used in previous work [13, 11, 12, 10]) via reduction to the decisional Diffie-Hellman (DDH) assumption. The protocol also achieves forward secrecy [20] in the sense that exposure of principals' long-term secret keys does not compromise the security of previous session keys.[3] Our 3-round protocol remains practical even for large groups: it requires only $O(1)$ communication, 3 modular exponentiations, and $O(n)$ signature verifications per user.

The difficulty of analyzing protocols for group AKE has seemingly hindered the development of practical and provably-secure solutions, and has led to the proposal of some protocols which were later found to be flawed (see, e.g., the attacks given in [29, 10]). To manage this complexity, we take a modular approach which greatly simplifies the design and analysis of group AKE protocols and should therefore prove useful for future work. Specifically, we show a *compiler* that transforms any group key-exchange protocol secure against a passive eavesdropper to one secure against a stronger (and more realistic) *active* adversary who controls all communication in the network. If the original protocol achieves

[3] We of course also require that exposure of (multiple) session keys does not compromise the security of unexposed session keys; see the formal model in Section 2.

forward secrecy, the compiled protocol does too. Adapting work of Burmester and Desmedt [14], we then present a 2-round group key-exchange protocol and rigorously prove its security — against a passive adversary — under the DDH assumption.[4] Applying our compiler to this protocol gives our main result.

We note two additional and immediate applications of the compiler presented here. First, the compiler may be applied to the group key-exchange protocols of [31] to yield a group AKE protocol similar to that of [13] but with a much simpler security proof which holds for groups of polynomial size (the proof given in [13] holds only for groups of *constant* size). Second, we may compile the 1-round, 3-party key-exchange protocol of Joux [23] to obtain a simple, 3-party AKE protocol requiring 2 rounds. The simplicity of the resulting security proof in these cases makes a modular approach of this sort compelling, especially when this approach is compared to the largely *ad hoc* methods which are often used when analyzing group AKE protocols (as in, e.g., [1, 26, 27]).

1.2 Previous Work

Group key exchange. A number of works have considered the problem of extending the 2-party Diffie-Hellman protocol [19] to the multi-party setting. Most well-known among these are perhaps the works of Ingemarsson, et al. [22], Burmester and Desmedt [14], and Steiner, et al. [31]. These works all assume a passive (eavesdropping) adversary, and only [31] provides a rigorous proof of security (but see footnote 4).

Authenticated protocols are designed to be secure against the stronger class of adversaries who —in addition to eavesdropping — control all communication in the network (cf. Section 2). A number of protocols for authenticated group key exchange have been suggested [24, 9, 2, 3, 32]; unfortunately, none of these works present rigorous security proofs and thus confidence in these protocols is limited. Indeed, attacks on some of these protocols have been presented [29], emphasizing the need for rigorous proofs in a well-defined model. Tzeng and Tzeng [33] prove security of a group AKE protocol using a non-standard adversarial model; an explicit attack on their protocol has recently been identified [10].

Provably-secure protocols. As mentioned earlier, only recently have Bresson, et al. [13, 11, 12] given the first formal model of security and the first provably-secure protocols for the group AKE setting. Their security model builds on earlier work of Bellare and Rogaway in the 2-party setting [6, 7] as extended by Bellare, et al. [5] to handle (among other things) forward secrecy.

The provably-secure protocols of Bresson, et al. [13, 11, 12] are based on the protocols of Steiner, et al. [31], and require n rounds to establish a key among a group of n users. The initial work [13] deals with the static case, and shows a

[4] Because no proof of security appears in [14], the Burmester-Desmedt protocol has been considered "heuristic" and not provably-secure (see, e.g., [13, 10]). Subsequent to our work we became aware that a proof of security for a variant of the Burmester-Desmedt protocol (in a weaker model than that considered here) appears in the pre-proceedings of Eurocrypt '94 [18]. See Section 4 for further discussion.

protocol which is secure (and achieves forward secrecy) under the DDH assumption.[5] Unfortunately, the given proof of security applies only when n is *constant*; in contrast, the proofs given here allow $n = \mathsf{poly}(k)$.

Later work [11, 12] focuses on the dynamic case where users join or leave and the session key must be updated whenever this occurs. Although we do not explicitly address this case, note that dynamic group membership can be handled efficiently — when using a constant-round protocol — by running the group AKE protocol from scratch among members of the new group. For the protocol given here, the complexity of this approach is roughly equivalent[6] to the Join and Remove protocols of [11, 12]. Yet, handling dynamic membership even more efficiently remains an interesting topic for future research.

More recently (in work subsequent to ours), a constant-round group AKE protocol with a security proof in the random oracle model has been shown [10]. The given protocol does not provide forward secrecy; in fact (as noted by the authors) an *explicit attack* is possible when long-term keys are exposed. Finally, the protocol is not symmetric but instead requires a "group leader" to perform $O(n)$ encryptions and send $O(n)$ communication each time a group key is established.

Compilers for key-exchange protocols. A modular approach such as ours has previously been used in the design and analysis of key-exchange protocols. Mayer and Yung [28] give a compiler which converts any 2-party protocol into a centralized (non-contributory) group protocol; their compiler invokes the original protocol $O(n)$ times, however, and is therefore not scalable. In work with similar motivation as our own, Bellare, et al. [4] show a compiler which converts unauthenticated protocols into authenticated protocols in the 2-party setting. Their compiler was not intended for the group setting and does not scale as well as ours; extending [4] to the group setting gives a compiler which triples the number of rounds and furthermore requires n signature computations/verifications and an $O(n)$ increase in communication per player per round. In contrast, the compiler presented here adds only a *single* round and introduces an overhead of 1 signature computation, n signature verifications, and $O(1)$ communication per player per round. (In fact, the compiler introduced here is slightly more efficient than that of [4] even in the 2-party case.)

1.3 Outline

In Section 2, we review the security model of Bresson, et al. [13]. We present our compiler in Section 3 and a two-round protocol secure against passive adversaries in Section 4. Applying our compiler to this protocol gives our main result: an efficient, fully-scalable, and constant-round group AKE protocol.

[5] The given reduction is in the random oracle model using the CDH assumption but they mention that the protocol can be proven secure in the standard model under the DDH assumption.

[6] For example, the Join algorithm of [11, 12] requires 2 rounds when one party joins and $O(n)$ rounds when n parties join; running our group AKE protocol from scratch requires only 3 rounds regardless of the number of parties who have joined.

2 The Model and Preliminaries

Our security model is the standard one of Bresson, et al. [13] which builds on prior work from the 2-party setting [6, 7, 5] and which has been widely used to analyze group key-exchange protocols (e.g., [11, 12, 10]). We explicitly define notions of security for both passive and active adversaries; this will be necessary for stating and proving meaningful results about our compiler in Section 3.

Participants and initialization. We assume for simplicity a fixed, polynomial-size set $\mathcal{P} = \{U_1, \ldots, U_\ell\}$ of potential participants. Any subset of $\mathcal{P}$ may decide at any point to establish a session key, and we do not assume that these subsets are always the same size or always include the same participants. Before the protocol is run for the first time, an initialization phase occurs during which each participant $U \in \mathcal{P}$ runs an algorithm $\mathcal{G}(1^k)$ to generate public/private keys (PK_U, SK_U). Each player U stores SK_U, and the vector $\langle PK_i \rangle_{1 \le i \le |\mathcal{P}|}$ is known by all participants (and is also known by the adversary).

Adversarial model. In the real world, a protocol determines how principals behave in response to signals from their environment. In the model, these signals are sent by the adversary. Each principal can execute the protocol multiple times with different partners; this is modeled by allowing each principal an unlimited number of *instances* with which to execute the protocol. We denote instance i of user U as Π_U^i. A given instance may be used only once. Each instance Π_U^i has associated with it the variables state_U^i, term_U^i, acc_U^i, used_U^i, sid_U^i, pid_U^i, and sk_U^i; the last of these is the *session key* whose computation is the goal of the protocol, while the function of the remaining variables is as in [5].

The adversary is assumed to have complete control over all communication in the network. An adversary's interaction with the principals in the network (more specifically, with the various instances) is modeled by the following *oracles*:

- $\mathsf{Send}(U, i, M)$ — This sends message M to instance Π_U^i, and outputs the reply generated by this instance. We allow the adversary to prompt the unused instance Π_U^i to initiate the protocol with partners $U_2, \ldots, U_n$ by calling $\mathsf{Send}(U, i, \langle U_2, \ldots, U_n \rangle)$.
- $\mathsf{Execute}(U_1, \ldots, U_n)$ — This executes the protocol between unused instances of players $U_1, \ldots, U_n \in \mathcal{P}$ and outputs the transcript of the execution. The number of group members and their identities are chosen by the adversary.
- $\mathsf{Reveal}(U, i)$ — This outputs session key sk_U^i.
- $\mathsf{Corrupt}(U)$ — This outputs the long-term secret key SK_U of player U.
- $\mathsf{Test}(U, i)$ — This query is allowed only once, at any time during the adversary's execution. A random bit b is generated; if $b = 1$ the adversary is given sk_U^i, and if $b = 0$ the adversary is given a random session key.

A *passive adversary* is given access to the $\mathsf{Execute}$, Reveal, $\mathsf{Corrupt}$, and Test oracles, while an *active adversary* is additionally given access to the Send oracle. (Even though the $\mathsf{Execute}$ oracle can be simulated via repeated calls to the Send oracle, allowing the adversary access to the $\mathsf{Execute}$ oracle allows for a tighter definition of forward secrecy.)

Partnering. Partnering is defined via *session IDs* and *partner IDs*. The session ID for instance Π_U^i (denoted sid_U^i) is a protocol-specified function of all communication sent and received by Π_U^i; for our purposes, we will simply set sid_U^i equal to the concatenation of all messages sent and received by Π_U^i during the course of its execution. The partner ID for instance Π_U^i (denoted pid_U^i) consists of the identities of the players in the group with whom Π_U^i intends to establish a session key, including U itself; note that these identities are always clear from the initial call to the Send or $\mathsf{Execute}$ oracles. We say instances Π_U^i and $\Pi_{U'}^j$ are *partnered* iff (1) $\mathsf{pid}_U^i = \mathsf{pid}_{U'}^j$ and (2) $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j$. Our definition of partnering is much simpler than that of [13] since, in our protocols, all messages are sent to all other members of the group taking part in the protocol.

Correctness. Of course, we wish to rule out "useless" protocols from consideration. In the standard way, we require that for all U, U', i, j such that $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j$, $\mathsf{pid}_U^i = \mathsf{pid}_{U'}^j$, and $\mathsf{acc}_U^i = \mathsf{acc}_{U'}^j = \text{TRUE}$ it is the case that $\mathsf{sk}_U^i = \mathsf{sk}_{U'}^j \neq \text{NULL}$.

Freshness. Following [5, 13], we define a notion of *freshness* appropriate for the goal of forward secrecy. An instance Π_U^i is *fresh* unless one of the following is true: (1) at some point, the adversary queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Reveal}(U', j)$ where Π_U^i and $\Pi_{U'}^j$ are partnered; or (2) a $\mathsf{Corrupt}$ query was asked before a query of the form $\mathsf{Send}(U, i, *)$.

Definitions of security. We say event Succ occurs if the adversary queries the Test oracle on a fresh instance and correctly guesses the bit b used by the Test oracle in answering this query. The advantage of an adversary $\mathcal{A}$ in attacking protocol P is defined as $\mathsf{Adv}_{\mathcal{A},P}(k) \stackrel{\text{def}}{=} |2 \cdot \Pr[\mathsf{Succ}] - 1|$. We say protocol P is a *secure group key exchange (KE) protocol* if it is secure against a passive adversary; that is, for any PPT passive adversary $\mathcal{A}$ it is the case that $\mathsf{Adv}_{\mathcal{A},P}(k)$ is negligible. We say protocol P is a *secure authenticated group key exchange (AKE) protocol* if it is secure against an active adversary; that is, for any PPT active adversary $\mathcal{A}$ it is the case that $\mathsf{Adv}_{\mathcal{A},P}(k)$ is negligible.

To enable a concrete security analysis, we define $\mathsf{Adv}_P^{\mathsf{KE-fs}}(t, q_{\text{ex}})$ to be the maximum advantage of any passive adversary attacking P, running in time t and making q_{ex} calls to the $\mathsf{Execute}$ oracle. Similarly, we define $\mathsf{Adv}_P^{\mathsf{AKE-fs}}(t, q_{\text{ex}}, q_{\text{s}})$ to be the maximum advantage of any active adversary attacking P, running in time t and making q_{ex} calls to the $\mathsf{Execute}$ oracle and q_{s} calls to the Send oracle.

Protocols without forward secrecy. Throughout this paper we will be concerned primarily with protocols achieving forward secrecy; the definitions above already incorporate this requirement since the adversary has access to the $\mathsf{Corrupt}$ oracle in each case. However, our compiler may also be applied to KE protocols which do not achieve forward secrecy (cf. Theorem 2). For completeness, we define $\mathsf{Adv}_P^{\mathsf{KE}}(t, q_{\text{ex}})$ and $\mathsf{Adv}_P^{\mathsf{AKE}}(t, q_{\text{ex}}, q_{\text{s}})$ in a manner completely analogous to the above, with the exception that the adversary in each case no longer has access to the $\mathsf{Corrupt}$ oracle.

Authentication. We do not define any notion of explicit authentication or, equivalently, confirmation that the other members of the group have computed the common key. Indeed, our protocols do not explicitly provide such confirmation. However, explicit authentication in our protocols can be achieved at little additional cost. Previous work (e.g., [13, Sec. 7]) shows how to achieve explicit authentication for any secure group AKE protocol using one additional round and minimal extra computation. (Although [13] use the random oracle model, their techniques can be extended to the standard model by replacing the random oracle with a pseudorandom function.) Applying their transformation to our final protocol will result in a constant-round group AKE protocol with explicit authentication.

2.1 Notes on the Definition

Although the above definition is standard for the analysis of group key-exchange protocols — it is the definition used, e.g., in [13, 11, 10] — there are a number of concerns that it does *not* address. For one, it does not offer any protection against malicious insiders, or users who do not honestly follow the protocol. Similarly, the definition is not intended to ensure any form of "agreement" and thus secure protocols for group AKE do not contradict known impossibility results for asynchronous distributed computing (e.g., [21]). (Actually, since the public-key model is assumed here, many of these impossibility results do not apply.) Finally, the definition inherently does not protect against "denial of service" attacks, and cannot prevent the adversary from causing an honest instance to "hang" indefinitely; this is simply because the model allows the adversary to refuse to deliver messages to any instance.

Some of these concerns can be addressed — at least partially — within the model above. For example, to achieve confirmation that all intended participants have computed the (correct, matching) session key following execution of a protocol, we may augment *any* group AKE protocol in the following way: after computing key sk, each player U_i computes $x_i = F_{\mathsf{sk}}(U_i)$, signs x_i, broadcasts x_i and the corresponding signature, and computes the "actual" session key $\mathsf{sk}' = F_{\mathsf{sk}}(\perp)$ (here, F represents a pseudorandom function and "$\perp$" represents some distinguished string); other players check the validity of these values in the obvious way.[7] Although this does not provide agreement (since an adversary can refuse to deliver messages to some of the participants), it *does* prevent a corrupted user from sending different messages to different parties, thereby causing them to generate and use non-matching keys.

Addressing the other concerns mentioned above represents an interesting direction for future work.

3 A Scalable Compiler for Group AKE Protocols

We show here a compiler transforming any secure group KE protocol P to a secure group AKE protocol P'. Without loss of generality, we assume the following about P: (1) Each message sent by an instance Π_U^i during execution of

[7] This is slightly different from the approach of [13, Sec. 7] in that we require a signature on the broadcast value x_i.

P includes the sender's identity U as well as a sequence number which begins at 1 and is incremented each time Π_U^i sends a message (in other words, the j^{th} message sent by an instance Π_U^i has the form $U|j|m$); (2) every message of the protocol is sent — via point-to-point links — to every member of the group taking part in the execution of the protocol (that is, Π_U^i sends each message to all users in pid_U^i). For simplicity, we refer to this as "broadcasting a message" but stress that we do *not* assume a broadcast channel and, in particular, an active adversary or a corrupted user can deliver different messages to different members of the group. Note that any secure group KE protocol $\widetilde{P}$ can be readily converted to a secure group KE protocol P in which the above assumptions hold (recall, security of a KE protocol is with respect to a passive adversary only).

Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme which is strongly unforgeable under adaptive chosen message attack (where "strong" means that an adversary is also unable to forge a new signature for a previously-signed message), and let $\mathsf{Succ}_\Sigma(t)$ denote the maximum advantage of any adversary running in time t in forging a new message/signature pair. We furthermore assume that the signature length is independent of the length of the message signed; this is easy to achieve by hashing the message (using a collision-resistant hash function) before signing. Given P as above, our compiler constructs protocol P' as follows:

1. During the initialization phase, each party $U \in \mathcal{P}$ generates the verification/signing keys (PK'_U, SK'_U) by running $\mathsf{Gen}(1^k)$. This is in addition to any keys (PK_U, SK_U) needed as part of the initialization phase for P.
2. Let $U_1, \ldots, U_n$ be the identities (in lexicographic order) of users wishing to establish a common key, and let $\mathcal{U} = U_1|\cdots|U_n$. Each user U_i begins by choosing a random nonce $r_i \in \{0,1\}^k$ and broadcasting $U_i|0|r_i$ (note we assign this message the sequence number "0"). After receiving the initial broadcast message from all other parties, each instance stores $\mathcal{U}$ and $r_1|\cdots|r_n$ as part of its state information.
3. The members of the group now execute P with the following changes:
   - Whenever instance Π_U^i is supposed to broadcast $U|j|m$ as part of protocol P, the instance instead signs $j|m|\mathcal{U}|r_1|\cdots|r_n$ using SK'_U to obtain signature σ, and then broadcasts $U|j|m|\sigma$.
   - When instance Π_U^i receives message $V|j|m|\sigma$, it checks that: (1) $V \in \mathsf{pid}_U^i$, (2) j is the next expected sequence number for messages from V, and (3) (using PK'_V) σ is a correct signature of V on $j|m|\mathcal{U}|r_1|\cdots|r_n$. If any of these are untrue, Π_U^i aborts the protocol and sets $\mathsf{acc}_U^i = \text{FALSE}$ and $\mathsf{sk}_U^i = \text{NULL}$. Otherwise, Π_U^i continues as it would in P upon receiving message $V|j|m$.
4. Each non-aborted instance computes the session key as in P.

Theorem 1. *If P is a secure group KE protocol achieving forward secrecy, then P' given by the above compiler is a secure group AKE protocol achieving forward secrecy. Namely:*

$$\mathsf{Adv}_{P'}^{\mathsf{AKE-fs}}(t, q_{\text{ex}}, q_{\text{s}}) \leq (q_{\text{ex}}+q_{\text{s}})\cdot\mathsf{Adv}_P^{\mathsf{KE-fs}}(t,1)+|\mathcal{P}|\cdot\mathsf{Succ}_\Sigma(t)+\frac{q_{\text{s}}^2+2q_{\text{ex}}q_{\text{s}}+|\mathcal{P}|q_{\text{ex}}^2}{2^{k+1}}.$$

Proof. Given an *active* adversary $\mathcal{A}'$ attacking P', we will construct a *passive* adversary $\mathcal{A}$ attacking P where $\mathcal{A}$ makes only a single $\mathsf{Execute}$ query; relating the success probabilities of $\mathcal{A}'$ and $\mathcal{A}$ gives the stated result.

Before describing $\mathcal{A}$, we first define events Forge and Repeat and bound their probabilities of occurrence. Let Forge be the event that $\mathcal{A}'$ outputs a new, valid message/signature pair with respect to the public key PK'_U of some user $U \in \mathcal{P}$ *before* querying $\mathsf{Corrupt}(U)$, and let $\Pr[\mathsf{Forge}]$ denote $\Pr_{\mathcal{A}',P'}[\mathsf{Forge}]$ for brevity. Using $\mathcal{A}'$, we may construct an algorithm $\mathcal{F}$ that forges a signature with respect to signature scheme Σ as follows: given a public key PK, algorithm $\mathcal{F}$ chooses a random $U \in \mathcal{P}$, sets $PK'_U = PK$, and honestly generates all other public/private keys for the system. $\mathcal{F}$ simulates the oracle queries of $\mathcal{A}'$ in the natural way (accessing its signing oracle when necessary); this results in a perfect simulation unless $\mathcal{A}'$ queries $\mathsf{Corrupt}(U)$. If this occurs, $\mathcal{F}$ simply aborts. Otherwise, if $\mathcal{A}'$ ever outputs a new, valid message/signature pair with respect to $PK'_U = PK$, then $\mathcal{F}$ outputs this pair as its forgery. The success probability of $\mathcal{F}$ is exactly $\frac{\Pr[\mathsf{Forge}]}{|\mathcal{P}|}$; this immediately implies that

$$\Pr[\mathsf{Forge}] \leq |\mathcal{P}| \cdot \mathsf{Succ}_{\Sigma}(t).$$

Let Repeat be the event that a nonce is used twice by a particular user; i.e., that there exists a user $U \in \mathcal{P}$ and i, j $(i \neq j)$ such that the nonce used by instance Π_U^i is equal to the nonce used by instance Π_U^j. A straightforward "birthday problem" calculation shows that $\Pr[\mathsf{Repeat}] \leq \frac{|\mathcal{P}|(q_{\rm ex}+q_{\rm s})^2}{2^{k+1}}$, since each user $U \in \mathcal{P}$ chooses at most $(q_{\rm ex} + q_{\rm s})$ nonces from $\{0,1\}^k$. A more careful analysis (omitted in the present abstract) in fact shows that

$$\Pr[\mathsf{Repeat}] \leq \frac{q_{\rm s}^2 + 2q_{\rm ex}q_{\rm s} + |\mathcal{P}|q_{\rm ex}^2}{2^{k+1}}.$$

We now construct our passive adversary $\mathcal{A}$ attacking protocol P. Recall that as part of the initial setup, adversary $\mathcal{A}$ is given public keys $\{PK_U\}_{U\in\mathcal{P}}$ if any are defined as part of protocol P. We first have $\mathcal{A}$ obtain all secret keys $\{SK_U\}_{U\in\mathcal{P}}$ using multiple $\mathsf{Corrupt}$ queries. Next, $\mathcal{A}$ runs $\mathsf{Gen}(1^k)$ to generate keys (PK'_U, SK'_U) for each $U \in \mathcal{P}$; the set of public keys $\{PK'_U, PK_U\}_{U\in\mathcal{P}}$ is then given to $\mathcal{A}'$. We now have $\mathcal{A}$ run $\mathcal{A}'$, simulating the oracle queries of $\mathcal{A}'$ as described below.

Before describing the details, we provide a high-level overview. Let $Q = q_{\rm ex} + q_{\rm s}$ denote the total number of $\mathsf{Execute}$ and Send queries made by $\mathcal{A}'$. Intuitively, $\mathcal{A}$ chooses an $\alpha \in \{1, \ldots, Q\}$ representing a guess as to which Send/$\mathsf{Execute}$ query of $\mathcal{A}'$ activates the instance for which $\mathcal{A}'$ will ask its Test query. For the $\alpha^{\rm th}$ such query of $\mathcal{A}'$, we will have $\mathcal{A}$ respond by making an $\mathsf{Execute}$ query, obtaining a transcript of an execution of P, modifying this transcript to obtain a valid transcript for P', and then returning an appropriate response to $\mathcal{A}'$. (We also need to ensure that this provides $\mathcal{A}'$ with a consistent view; these details are discussed below.) For all other (unrelated) Send/$\mathsf{Execute}$ queries of $\mathcal{A}'$, we have $\mathcal{A}$ respond by directly running protocol P'; note that $\mathcal{A}$ can do this since it has the secret keys for all players. $\mathcal{A}$ aborts and outputs a random bit if it determines that its guess α was incorrect, or if events Forge or Repeat occur. Otherwise, $\mathcal{A}$ outputs whatever bit is output by $\mathcal{A}'$. We now describe the simulation of the oracle queries of $\mathcal{A}'$ in detail.

Execute queries. If an $\mathsf{Execute}$ query is *not* the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, then $\mathcal{A}$ simply generates on its own a transcript of an execution of P' and returns this to $\mathcal{A}'$ (as noted above, $\mathcal{A}$ can do this since it knows all the secret keys for all players). If an $\mathsf{Execute}$ query *is* the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, let the query be $\mathsf{Execute}(U_1, \ldots U_n)$ and let $\mathcal{U} = U_1|\cdots|U_n$. Adversary $\mathcal{A}$ sends the same query to its $\mathsf{Execute}$ oracle and receives in return a transcript T of an execution of P. To simulate a transcript T' of an execution of P', $\mathcal{A}$ first chooses random $r_1, \ldots, r_n \in \{0,1\}^k$. The initial messages of T' are set to $\{U_i|0|r_i\}_{1 \le i \le n}$. Then, for each message $U|j|m$ in transcript T, $\mathcal{A}$ computes $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|\mathcal{U}|r_1|\cdots|r_n)$ and places $U|j|m|\sigma$ in T'. When done, the complete transcript T' is given to $\mathcal{A}'$.

Send queries. If a Send query is not the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, the intuition is to have $\mathcal{A}$ simulate by itself the actions of this instance and thus generate the appropriate responses for $\mathcal{A}'$. On the other hand, if a Send query is the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, then $\mathcal{A}$ should obtain a transcript T from its $\mathsf{Execute}$ oracle and generate responses for $\mathcal{A}'$ by modifying T using the signature keys $\{SK'_U\}_{U \in \mathcal{P}}$. The actual simulation is slightly more difficult, since we need to ensure consistency in the view of $\mathcal{A}'$.

Consider an arbitrary instance Π_U^ℓ. Denote the initial Send query to this instance (i.e., protocol initiation) by Send_0; this query always has the form $\mathsf{Send}_0(U, \ell, \langle U_1, \ldots, U_n \rangle)$ for some n. We set $\mathcal{U}_U^\ell = U|U_1|\cdots|U_n$, where we assume without loss of generality that these are in lexicographic order. We denote the second Send query to the instance by Send_1; this query always has the form $\mathsf{Send}(U, \ell, U_1|0|r_1, \ldots, U_n|0|r_n)$. After a Send_1 query, we may set $\mathcal{R}_U^\ell = r_U^\ell|r_1|\cdots|r_n$, where r_U^ℓ is the nonce generated by instance Π_U^ℓ. To aid the simulation, $\mathcal{A}$ will maintain a list Nonces whose function will become clear below.

If a Send query is *not* the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, then:

- On query $\mathsf{Send}_0(U, \ell, *)$, $\mathcal{A}$ simply chooses a random nonce r_U^ℓ and replies to $\mathcal{A}'$ with $U|0|r_U^\ell$. Note that $\mathcal{U}_U^\ell$ is now defined.
- If the query is not a Send_0 query and has the form $\mathsf{Send}(U, \ell, M)$, then $\mathcal{R}_U^\ell$ is defined (either by the present query or by some previous query). $\mathcal{A}$ looks in Nonces for an entry of the form $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, c)$. There are two cases to consider:
  - If such an entry exists and $c = 1$ then $\mathcal{A}$ has already queried its $\mathsf{Execute}$ oracle and received in return a transcript T. First, $\mathcal{A}$ verifies correctness of the current incoming message(s) as in the description of the compiler (and aborts execution of Π_U^ℓ if verification fails). $\mathcal{A}$ then finds the appropriate message $U|j|m$ in T, computes $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|\mathcal{U}_U^\ell|\mathcal{R}_U^\ell)$, and replies to $\mathcal{A}'$ with $U|j|m|\sigma$.
  - If no such entry exists, $\mathcal{A}$ stores $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, 0)$ in Nonces. In this case or if the entry exists and $c = 0$, then $\mathcal{A}$ simulates on its own the actions of this instance ($\mathcal{A}$ can do this since it knows all relevant secret keys).

If a Send query *is* the α^{th} $\mathsf{Send}/\mathsf{Execute}$ query of $\mathcal{A}'$, then:

- If the query is *not* a Send_1 query, then $\mathcal{A}$ aborts (and outputs a random bit) since its guess α was incorrect.
- If a $\mathsf{Corrupt}$ query has previously been made by $\mathcal{A}'$, then $\mathcal{A}$ aborts. The current instance is no longer fresh, and therefore the guess α is incorrect.
- If the query is a Send_1 query to instance Π_U^ℓ, then $\mathcal{U}_U^\ell$ and $\mathcal{R}_U^\ell$ are now both defined. $\mathcal{A}$ looks in Nonces for an entry of the form $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, 0)$. If such an entry exists, then $\mathcal{A}$ aborts (and outputs a random bit). Otherwise, $\mathcal{A}$ stores $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, 1)$ in Nonces. Next, $\mathcal{A}$ queries $\mathsf{Execute}(\mathcal{U}_U^\ell)$, obtains in return a transcript T (which is stored for later use), and finds the message $U|1|m$ in T. The signature $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(1|m|\mathcal{U}_U^\ell|\mathcal{R}_U^\ell)$ is computed, and the message $U|1|m|\sigma$ is returned to $\mathcal{A}'$.

Corrupt queries. On query $\mathsf{Corrupt}(U)$, $\mathcal{A}$ returns (SK_U, SK'_U) (recall that $\mathcal{A}$ has obtained SK_U already, and knows SK'_U since it was generated by $\mathcal{A}$).

Reveal queries. When $\mathcal{A}'$ queries $\mathsf{Reveal}(U, i)$ for a terminated instance it must be the case that $\mathcal{U}_U^\ell$ and $\mathcal{R}_U^\ell$ are both defined. $\mathcal{A}$ locates the entry $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, c)$ in Nonces and aborts (and outputs a random bit) if $c = 1$ since its guess α was incorrect. Otherwise, $c = 0$ implies that this instance was simulated by $\mathcal{A}$ itself; thus $\mathcal{A}$ can compute the appropriate key sk_U^ℓ and returns this key to $\mathcal{A}'$.

Test queries. When $\mathcal{A}'$ queries $\mathsf{Test}(U, i)$ for a terminated instance it must be the case that $\mathcal{U}_U^\ell$ and $\mathcal{R}_U^\ell$ are both defined. $\mathcal{A}$ finds the entry $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, c)$ in Nonces and aborts (and outputs a random bit) if $c = 0$ since its guess α was incorrect. Otherwise, $c = 1$ implies that this instance corresponds to an instance for which $\mathcal{A}$ had asked its single $\mathsf{Execute}$ query. So, $\mathcal{A}$ asks its own Test query for any such instance (it does not matter which, since they are all partnered and all hold the same key) and returns the result to $\mathcal{A}'$.

Let Guess denote the event that $\mathcal{A}$ correctly guesses α. We claim that as long as Guess and $\overline{\mathsf{Forge}}$ and $\overline{\mathsf{Repeat}}$ occur, the above simulation is perfect. Indeed, assuming Guess occurs the only difference between the simulation and a real execution of $\mathcal{A}'$ occurs for those instances Π_U^ℓ for which $(\mathcal{U}_U^\ell|\mathcal{R}_U^\ell, 1) \in \mathsf{Nonces}$. Here, the simulation is perfect unless A' forges a signature or can "splice in" a message from a different execution. However, neither of these events can happen as long as neither Forge nor Repeat occur.

Letting $\mathsf{Good} \stackrel{\text{def}}{=} \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Repeat}}$ and $\mathsf{Bad} \stackrel{\text{def}}{=} \overline{\mathsf{Good}}$, and recalling that $Q = q_{\text{ex}} + q_{\text{s}}$ denotes the total number of $\mathsf{Send}/\mathsf{Execute}$ queries asked by $\mathcal{A}'$, a straightforward probability calculation shows that:

$$
\begin{aligned}
&2 \cdot \left|\Pr_{\mathcal{A},P}[\mathsf{Succ}] - \tfrac{1}{2}\right| \\
&= 2 \cdot \left|\Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Guess} \wedge \mathsf{Good}] + \tfrac{1}{2}\Pr_{\mathcal{A}',P'}[\overline{\mathsf{Guess}} \vee \mathsf{Bad}] - \tfrac{1}{2}\right| \\
&= 2 \cdot \Big|\tfrac{1}{Q}\Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Good}] + \tfrac{1}{2}\Pr_{\mathcal{A}',P'}[\mathsf{Bad}] \\
&\qquad + \tfrac{1}{2}\Pr[\overline{\mathsf{Guess}}|\overline{\mathsf{Bad}}]\Pr_{\mathcal{A}',P'}[\overline{\mathsf{Bad}}] - \tfrac{1}{2}\Big|
\end{aligned}
$$

$$= 2 \cdot \Big| \tfrac{1}{Q} \Pr_{\mathcal{A}',P'}[\mathsf{Succ}] - \tfrac{1}{Q} \Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Bad}] + \tfrac{1}{2} \Pr_{\mathcal{A}',P'}[\mathsf{Bad}]$$
$$+ \tfrac{1}{2}(\tfrac{Q-1}{Q})(1 - \Pr_{\mathcal{A}',P'}[\mathsf{Bad}]) - \tfrac{1}{2} \Big|$$
$$\geq \tfrac{1}{Q} \cdot |2 \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Succ}] - 1| - \tfrac{1}{Q} \left| 2 \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Bad}] - \Pr_{\mathcal{A}',P'}[\mathsf{Bad}] \right| .$$

Since $2 \left| \Pr_{\mathcal{A},P}[\mathsf{Succ}] - \frac{1}{2} \right| \leq \mathsf{Adv}_P^{\mathsf{KE-fs}}(t, 1)$ by assumption, we obtain:

$$\mathsf{Adv}_{P'}^{\mathsf{AKE-fs}}(t, q_{\mathrm{ex}}, q_{\mathrm{s}}) \leq Q \cdot \mathsf{Adv}_P^{\mathsf{KE-fs}}(t, 1) + \Pr_{\mathcal{A}',P'}[\mathsf{Forge}] + \Pr[\mathsf{Repeat}],$$

which immediately yields the statement of the theorem. ■

We remark that the above theorem is a generic result that applies to the invocation of the compiler on an *arbitrary* group KE protocol P. For specific protocols, a better exact security analysis may be obtainable. Furthermore, the compiler above may also be applied to KE protocols that do not achieve forward secrecy. In this case, we obtain the following tighter security reduction.

Theorem 2. *If P is a secure group KE protocol (without forward secrecy), then P' given by the above compiler is a secure group AKE protocol (without forward secrecy). Namely:*

$$\mathsf{Adv}_{P'}^{\mathsf{AKE}}(t, q_{\mathrm{ex}}, q_{\mathrm{s}}) \leq \mathsf{Adv}_P^{\mathsf{KE}}(t, q_{\mathrm{ex}} + q_{\mathrm{s}}) + |\mathcal{P}| \cdot \mathsf{Succ}_{\Sigma}(t) + \frac{q_{\mathrm{s}}^2 + 2q_{\mathrm{ex}}q_{\mathrm{s}} + |\mathcal{P}|q_{\mathrm{ex}}^2}{2^{k+1}}.$$

The proof is largely similar to that of Theorem 1, and will appear in the full version of this paper.

4 A Constant-Round Group KE protocol

Let $\mathbb{G}$ be any finite cyclic group of prime order q (e.g., letting p, q be prime such that $p = \beta q + 1$ we may let $\mathbb{G}$ be the subgroup of order q in $\mathbb{Z}_p^*$), and let g be an arbitrary generator of $\mathbb{G}$. We define $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t)$ as the maximum value, over all adversaries A running in time at most t, of:

$$\left| \Pr[x, y \leftarrow \mathbb{Z}_q : A(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow \mathbb{Z}_q : A(g, g^x, g^y, g^z) = 1] \right| .$$

Informally, we say the *DDH assumption holds in* $\mathbb{G}$ if $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t)$ is "small" for "reasonable" values of t. We now describe an efficient, two-round group KE protocol whose security is based on the DDH assumption in $\mathbb{G}$. Applying the compiler of the previous section to this protocol immediately yields an efficient, three-round group AKE protocol.

The protocol presented here is essentially the protocol of Burmester and Desmedt [14], except we assume that $\mathbb{G}$ is a finite, cyclic group of prime order in which the DDH assumption holds. Our work was originally motivated by the fact that no proof of security appears in the proceedings version of [14]; furthermore, subsequent work in this area (e.g., [13, 10]) implied that the Burmester-Desmedt protocol was "heuristic" and had not been proven secure. (Indeed, presumably for this reason the group AKE protocols of [13, 11, 12] are based on the

$O(n)$-round group KE protocol of Steiner, et al. [31] rather than the Burmester-Desmedt protocol.) Subsequent to our work, however, we became aware that a proof of security for a variant of the Burmester-Desmedt protocol appears in the pre-proceedings of Eurocrypt '94 [18].[8] Even so, we note the following:

– The given proof shows only that an adversary cannot compute the *entire* session key; in contrast to our work, it says nothing about whether the key is indistinguishable from random. On the other hand, for this reason their proof uses only the weaker CDH assumption.
– A proof of security is given only for an *even* number of participants n. A modified, asymmetric protocol (which is slightly less efficient) is introduced and proven secure for the case of n odd.
– Finally, the previously-given proof of security makes no effort to optimize the concrete security of the reduction (since this issue was not generally considered at that time).

As required by the compiler of the previous section, our protocol ensures that players send every message to all members of the group via point-to-point links; although we refer to this as "broadcasting" we stress that no broadcast channel is assumed (in any case, the distinction is moot since we are dealing here with a passive adversary). In our protocol P, no public keys are required but for simplicity we assume a group $\mathbb{G}$ and generator $g \in \mathbb{G}$ have been fixed in advance and are known to all parties in the network. Note that this assumption can be avoided at the expense of an additional round in which the first player simply generates and broadcasts these values (that this is secure is clear from the fact that we are now considering a *passive* adversary). When n players $U_1, \ldots, U_n$ wish to generate a session key, they proceed as follows (the indices are taken modulo n so that player U_0 is U_n and player U_{n+1} is U_1):

Round 1 Each player U_i chooses a random $r_i \in \mathbb{Z}_q$ and broadcasts $z_i = g^{r_i}$.
Round 2 Each player U_i broadcasts $X_i = (z_{i+1}/z_{i-1})^{r_i}$.
Key computation Each player U_i computes their session key as:

$$K_i = (z_{i-1})^{nr_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2}.$$

(It may be easily verified that all users compute the same key $g^{r_1r_2+r_2r_3+\cdots+r_nr_1}$.)

We do not explicitly include sender identities and sequence numbers as required by the compiler of the previous section; however, as discussed there, it is easy to modify the protocol to include this information. Note that each user only computes three (full-length) exponentiations since $n \ll q$ in practice.

Theorem 3. *Protocol P is a secure group KE protocol achieving forward secrecy. Namely:*

$$\mathsf{Adv}_P^{\mathsf{KE-fs}}(t, q_{\mathrm{ex}}) \leq 4 \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t).$$

[8] We are happy to publicize this, especially since it appears to have been unknown to many others in the cryptographic community as well!

Proof. Let $\varepsilon(t) \stackrel{\text{def}}{=} \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$. We provide here a proof for the case of an adversary making only a single $\mathsf{Execute}$ query, and show the weaker result that $\mathsf{Adv}^{\mathsf{KE-fs}}_{P}(t,1) \leq 2|\mathcal{P}|\varepsilon(t)$. Note that this is sufficient for the purposes of applying Theorem 1, and also immediately yields (via a standard hybrid argument) that $\mathsf{Adv}^{\mathsf{KE-fs}}_{P}(t,q_{\text{ex}}) \leq 2q_{\text{ex}}|\mathcal{P}|\varepsilon(t)$. A proof of the stronger result stated in the theorem can be obtained using random self-reducibility properties of the DDH problem (following [30]), and will appear in the full version.

Since there are no public keys in the protocol, we may ignore $\mathsf{Corrupt}$ queries. Assume an adversary $\mathcal{A}$ making a single query $\mathsf{Execute}(U_1,\ldots,U_n)$ (we stress that the number of parties n is chosen by the adversary; however, since the protocol is symmetric and there are no public keys the identities of the parties are unimportant). The distribution of the transcript T and the resulting session key sk is given by:

$$\mathsf{Real} \stackrel{\text{def}}{=} \left\{ \begin{array}{c} r_1,\ldots,r_n \leftarrow \mathbb{Z}_q;\ z_1 = g^{r_1},\ z_2 = g^{r_2},\ \ldots,\ z_n = g^{r_n} \\ X_1 = \frac{g^{r_2 r_1}}{g^{r_n r_1}},\ X_2 = \frac{g^{r_3 r_2}}{g^{r_1 r_2}},\ \ldots,\ X_n = \frac{g^{r_1 r_n}}{g^{r_{n-1} r_n}} \\ \mathsf{T} = (z_1,\ldots,z_n,X_1,\ldots,X_n) \\ \mathsf{sk} = (g^{r_1 r_2})^n \cdot (X_2)^{n-1} \cdots X_n \end{array} : (\mathsf{T},\mathsf{sk}) \right\}.$$

Consider the following modified distribution:

$$\mathsf{Fake}_1 \stackrel{\text{def}}{=} \left\{ \begin{array}{c} w_{1,2}, r_1,\ldots,r_n \leftarrow \mathbb{Z}_q;\ z_1 = g^{r_1},\ z_2 = g^{r_2},\ \ldots,\ z_n = g^{r_n} \\ X_1 = \frac{g^{w_{1,2}}}{(g^{r_1})^{r_n}},\ X_2 = \frac{(g^{r_2})^{r_3}}{g^{w_{1,2}}},\ \ldots,\ X_n = \frac{(g^{r_1})^{r_n}}{g^{r_{n-1} r_n}} \\ \mathsf{T} = (z_1,\ldots,z_n,X_1,\ldots,X_n) \\ \mathsf{sk} = (g^{w_{1,2}})^n \cdot (X_2)^{n-1} \cdots X_n \end{array} : (\mathsf{T},\mathsf{sk}) \right\}.$$

A standard argument shows that for any algorithm $\mathcal{A}'$ running in time t we have:

$$|\Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Real} : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1] - \Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Fake}_1 : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1]| \leq \varepsilon(t).$$

We next make the following additional modification:

$$\mathsf{Fake}_2 \stackrel{\text{def}}{=} \left\{ \begin{array}{c} w_{1,2}, w_{2,3}, r_1,\ldots,r_n \leftarrow \mathbb{Z}_q;\ z_1 = g^{r_1},\ z_2 = g^{r_2},\ \ldots,\ z_n = g^{r_n} \\ X_1 = \frac{g^{w_{1,2}}}{g^{r_1 r_n}},\ X_2 = \frac{g^{w_{2,3}}}{g^{w_{1,2}}},\ \ldots,\ X_n = \frac{g^{r_1 r_n}}{g^{r_{n-1} r_n}} \\ \mathsf{T} = (z_1,\ldots,z_n,X_1,\ldots,X_n) \\ \mathsf{sk} = (g^{w_{1,2}})^n \cdot (X_2)^{n-1} \cdots X_n \end{array} : (\mathsf{T},\mathsf{sk}) \right\},$$

where, again, a standard argument shows that:

$$|\Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Fake}_1 : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1] - \Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Fake}_2 : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1]| \leq \varepsilon(t).$$

Continuing in this way, we obtain the distribution:

$$\mathsf{Fake}_n \stackrel{\text{def}}{=} \left\{ \begin{array}{c} w_{1,2}, w_{2,3},\ldots,w_{n-1,n}, w_{n,1}, r_1,\ldots,r_n \leftarrow \mathbb{Z}_q \\ X_1 = \frac{g^{w_{1,2}}}{g^{w_{n,1}}},\ X_2 = \frac{g^{w_{2,3}}}{g^{w_{1,2}}},\ \ldots,\ X_n = \frac{g^{w_{n,1}}}{g^{w_{n-1,n}}} \\ \mathsf{T} = (g^{r_1},\ldots,g^{r_n},X_1,\ldots,X_n) \\ \mathsf{sk} = (g^{w_{1,2}})^n \cdot (X_2)^{n-1} \cdots X_n \end{array} : (\mathsf{T},\mathsf{sk}) \right\},$$

such that, for any $\mathcal{A}'$ running in time t we have (via standard hybrid argument):

$$|\Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Real} : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1] - \Pr[(\mathsf{T},\mathsf{sk}) \leftarrow \mathsf{Fake}_n : \mathcal{A}'(\mathsf{T},\mathsf{sk}) = 1]| \leq n \cdot \varepsilon(t). \quad (1)$$

In experiment Fake_n, the values $w_{1,2}, \ldots, w_{n,1}$ are constrained by T according to the following n equations

$$\begin{aligned} \log_g X_1 &= w_{1,2} - w_{n,1} \\ &\vdots \\ \log_g X_n &= w_{n,1} - w_{n-1,n}, \end{aligned}$$

of which only $n-1$ of these are linearly independent. Furthermore, sk may be expressed as $g^{w_{1,2}+w_{2,3}+\cdots+w_{n,1}}$; equivalently, we have

$$\log_g \mathsf{sk} = w_{1,2} + w_{2,3} + \cdots + w_{n,1}.$$

Since this final equation is linearly independent from the set of equations above, the value of sk is independent of T. This implies that, for any adversary $\mathcal{A}$:

$$\Pr[(\mathsf{T},\mathsf{sk}_0) \leftarrow \mathsf{Fake}_n; \mathsf{sk}_1 \leftarrow \mathbb{G}; b \leftarrow \{0,1\} : \mathcal{A}(\mathsf{T},\mathsf{sk}_b) = b] = 1/2,$$

which — combined with Equation (1) and the fact that $n \leq |\mathcal{P}|$ — yields the desired result $\mathsf{Adv}_P^{\mathsf{KE-fs}}(t,1) \leq 2|\mathcal{P}|\varepsilon(t)$.

References

1. S.S. Al-Riyami and K.G. Paterson. Tripartite Authenticated Key Agreement Protocols from Pairings. Available at http://eprint.iacr.org/2002/035/.
2. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. ACM CCCS '98.
3. G. Ateniese, M. Steiner, and G. Tsudik. New Multi-Party Authentication Services and Key Agreement Protocols. *IEEE Journal on Selected Areas in Communications*, 18(4): 628–639 (2000).
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. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. Crypto '93.
7. M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. STOC '95.
8. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *IEEE J. on Selected Areas in Communications*, 11(5): 679–693 (1993). A preliminary version appeared in Crypto '91.
9. C. Boyd. On Key Agreement and Conference Key Agreement. ACISP '97.

10. C. Boyd and J.M.G. Nieto. Round-Optimal Contributory Conference Key Agreement. PKC 2003.
11. E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange — The Dynamic Case. Asiacrypt 2001.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. Eurocrypt 2002.
13. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. ACM CCCS 2001.
14. M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. Eurocrypt '94.
15. R. Canetti and H. Krawczyk. Key-Exchange Protocols and Their Use for Building Secure Channels. Eurocrypt 2001.
16. R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. Eurocrypt 2002.
17. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. Crypto 2002.
18. Y. Desmedt. Personal communication (including a copy of the pre-proceedings version of [14]), March 2003.
19. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6): 644–654 (1976).
20. W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, 2(2): 107–125 (1992).
21. M. Fischer, N. Lynch, and M. Patterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32(2): 374–382 (1985).
22. I. Ingemarsson, D.T. Tang, and C.K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5): 714–720 (1982).
23. A. Joux. A One Round Protocol for Tripartite Diffie Hellman. ANTS 2000.
24. M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. Asiacrypt '96.
25. H. Krawczyk. SKEME: A Versatile Secure Key-Exchange Mechanism for the Internet. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, Feb. 1996, pp. 114–127.
26. H.-K. Lee, H.-S. Lee, and Y.-R. Lee. Multi-Party Authenticated Key Agreement Protocols from Multilinear Forms. Available at http://eprint.iacr.org/2002/166/.
27. H.-K. Lee, H.-S. Lee, and Y.-R. Lee. An Authenticated Group Key Agreement Protocol on Braid groups. Available at http://eprint.iacr.org/2003/018/.
28. A. Mayer and M. Yung. Secure Protocol Transformation via "Expansion": From Two-Party to Groups. ACM CCCS '99.
29. O. Pereira and J.-J. Quisquater. A Security Analysis of the Cliques Protocol Suites. *IEEE Computer Security Foundations Workshop*, June 2001.
30. V. Shoup. On Formal Models for Secure Key Exchange. Draft, 1999. Available at `http://eprint.iacr.org/1999/012`.
31. M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Trans. on Parallel and Distributed Systems* 11(8): 769–780 (2000). A preliminary version appeared in ACM CCCS '96.
32. W.-G. Tzeng. A Practical and Secure Fault-Tolerant Conference Key Agreement Protocol. PKC 2000.
33. W.-G. Tzeng and Z.-J. Tzeng. Round Efficient Conference Key Agreement Protocols with Provable Security. Asiacrypt 2000.