J. Cryptology (2007) 20: 85–113
DOI: 10.1007/s00145-006-0361-5

Journal of
CRYPTOLOGY

Scalable Protocols for Authenticated Group Key Exchange[*]

Jonathan Katz
Department of Computer Science, University of Maryland,
College Park, MD 20742, U.S.A.
jkatz@cs.umd.edu

Moti Yung
RSA Laboratories
and
Department of Computer Science, Columbia University,
New York, NY 10027, U.S.A.
moti@cs.columbia.edu

Communicated by Dan Boneh

Received 7 December 2003 and revised 2 January 2006
Online publication 14 September 2006

Abstract. We consider the problem of authenticated group key exchange among n parties communicating over an insecure public network. A number of solutions to this problem have been proposed; however, all prior provably secure solutions do not scale well and, in particular, require $O(n)$ 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 $O(1)$ "full" modular exponentiations per user. Toward this goal (and adapting work of Bellare, Canetti, and Krawczyk), we first present an efficient 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.

Key words. Key exchange.

[*] A preliminary version of this work appeared as [32].

1. Introduction

Protocols for authenticated key exchange (AKE) allow parties communicating over an 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 and network security. 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. They are also 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 two-party AKE has been extensively investigated (see, e.g., [25], [12], [26], [10], [35], [6], [38], and [22]–[24]) and is fairly well-understood; furthermore, a variety of efficient and provably secure protocols for two-party AKE are known. Less attention has been given to the case of *group* AKE where a session key is to be established among $n > 2$ 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. [17], [15], [16] (building on [10], [11], and [8]) 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 in Relation to Prior Work*

The efficiency of a group AKE protocol should scale well with the number of users so as to remain practical even when the number of users is large. (In Section 1.3 we discuss in detail the complexity measures we use to evaluate the efficiency of protocols in this setting.) Unfortunately, prior work is somewhat limited in this respect; we may summarize the "state-of-the-art" for *provably secure* group AKE protocols as follows (we exclude here centralized protocols in which a designated group manager generates and distributes keys; such schemes place a high burden on one participant who is a single point of failure and who must also be trusted to generate keys properly):

- The most efficient solutions are those of Bresson et al. [17], [15], [16] which adapt previous work of Steiner et al. [40]. Unfortunately, these protocols do not scale well: to establish a key among n participants, they require n rounds and additionally require (for some players) $O(n)$ "full" 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*[1] [14]. This protocol does *not*

[1] The random oracle model [9] assumes a public random function to which all parties (including the adversary) have access. This random function is instantiated using a cryptographic hash function (e.g., SHA-1),

achieve forward secrecy (see [26]), and the exposure of a user's long-term secret key exposes all session keys previously generated by this user. The protocol is also not symmetric, and the initiator must perform $O(n)$ encryptions and send $O(n)$ communication.

The above solutions represent the best-known *provably secure* protocols even under various relaxations of the problem. For example, we are aware of no previous constant-round protocol with a full proof of security in the standard (i.e., non-random oracle) model even for the weaker case of security against a passive eavesdropper (but see footnote 2). Clearly, an $O(n)$-round protocol is not scalable and is unacceptable when the number of parties grows large or when network latency is the performance bottleneck.

Our main result is the first *constant-round* and *fully scalable* protocol for group AKE which is provably secure in the standard model (i.e., without assuming the existence of "random oracles"). Security is proved via reduction to the decisional Diffie–Hellman (DDH) assumption using the same security model as other recent work in this area [17], [15], [16], [14]. Our protocol also achieves forward secrecy [26] in the sense that exposure of principals' long-term secret keys does not compromise the secrecy of previous session keys. (We of course also require that exposure of multiple session keys does not compromise the secrecy of unexposed session keys; see the formal model in Section 2.) Our three-round protocol remains practical even for large groups: it requires each user to send only $O(1)$ communication and to compute only three "full" modular exponentiations and $O(n \log n)$ modular multiplications, generate two signatures, and perform $O(n)$ signature verifications (the cost of which can be improved by using signatures that allow for "batch verification" [7]).

The difficulty of analyzing protocols for group AKE has led to a number of ad hoc approaches to this problem and has seemingly hindered the development of practical and provably secure solutions (as evidenced by the many published protocols later found to be flawed; see the attacks given in [37] and [14]). To manage this complexity, we propose and analyze a scalable *compiler* for this setting which enables a modular approach and therefore greatly simplifies the design and analysis of group AKE protocols. Our compiler transforms any group key-exchange protocol which is secure against a *passive* eavesdropper to one which is secure against a stronger—and more realistic—*active* adversary who controls all communication in the network. If the original protocol achieves forward secrecy, the compiled protocol does too. Our compiler is inspired by earlier work of Bellare et al. [6], who show a compiler with similar functionality. Although their compiler does indeed apply to multi-party protocols, their primary motivation was the two-party setting. The compiler we show here is specifically tailored for the group setting, and it scales better than the compiler of [6]. Further discussion appears in Section 1.2.

As an immediate illustration of the advantages of a modular approach, note that our compiler may be applied to the group key-exchange protocol of Steiner et al. [40] to yield a group AKE protocol with comparable efficiency to that of [17] but with a much simpler security proof which holds even for groups of polynomial size.

appropriately modified to have the desired domain and range. Although proofs of security in this model provide heuristic evidence for the security of a given protocol, there exist schemes which are secure in the random oracle model but are *insecure* for any instantiation of the random oracle [21].

As an additional contribution, we investigate the security of the well-known Burmester–Desmedt protocol [19], [20] for *unauthenticated* group key exchange.[2] Adapting their work, we present a two-round group key-exchange protocol and rigorously prove its security—against a passive adversary—under the DDH assumption. Applying our above-mentioned compiler to this protocol gives our main result.

1.2. *Survey of Previous Work*

Group key exchange. A number of works have considered extending the two-party Diffie–Hellman protocol [25] to the multi-party setting [28], [39], [19], [40], [4], [33], [34]. Best known among these are perhaps the works of Ingemarsson et al. [28], Burmester and Desmedt [19], and Steiner et al. [40]. Ingemarsson et al. and Steiner et al. assume only a passive (eavesdropping) adversary; furthermore, no proofs of security appear to be given by Ingemarsson et al. (and we are not aware of any subsequent proof of security for their protocol). Security of the Burmester–Desmedt protocol is discussed in footnote 2 and Section 4.

Key-exchange (KE) protocols are intended to be secure against a passive adversary only. *Authenticated* key-exchange (AKE) protocols are designed to be secure against the stronger class of adversaries who—in addition to eavesdropping—control all communication in the network (see Section 2). A number of protocols for authenticated group key exchange have been suggested [29], [13], [2], [3], [41]; unfortunately, none of these works presents rigorous security proofs in a well-defined model (indeed, attacks on some of these protocols have been shown [37]). Tzeng and Tzeng [42] prove security of a group AKE protocol using a non-standard adversarial model and assuming a reliable broadcast channel (an assumption we do not make here). Their protocol does not achieve forward secrecy, and an explicit attack on their protocol has been identified [14].

Provably secure protocols. Bresson et al. [17], [15], [16], building on earlier work of Bellare and Rogaway in the two-party setting [10], [11], [8], give the first formal model of security for group AKE and the first provably secure protocols for this setting. (A stronger model of security for group AKE, based on work of Canetti and Krawczyk [23], has been proposed more recently [31].) The protocols of Bresson et al. are all based on work of Steiner et al. [40], and require $O(n)$ rounds to establish a key among n users. The initial work [17] deals with the case where the subset of participants sharing a key does not undergo frequent changes, and shows a protocol that achieves forward secrecy under the computational Diffie–Hellman assumption in the random oracle model (as noted there, however, the protocol can also be proven secure in the standard model using the DDH assumption). Unfortunately, the proof of security applies only for groups of *constant* size (see Theorem 1 of [17]).

Later work [15], [16] focuses on the dynamic case where users join or leave the group and the session key must be updated whenever this occurs. Although we do not

[2] Since no security proof appears in the proceedings version [19], the Burmester–Desmedt protocol has often been considered "heuristic" and not provably secure (see, e.g., [17] and [14]). Subsequent to our work we became aware of a security proof for a variant of their protocol against a passive adversary [18] (see also [20]). Section 4 contains further discussion.

Table 1. High-level comparison of provably secure protocols for group AKE. Efficiency comparisons are given in Table 2.

	Standard model?	Forward secrecy?	Polynomial group size?
Bresson et al. [17]	Yes	Yes	No
Boyd–Nieto [14]	No	No	Yes
Here	Yes	Yes	Yes

explicitly address this issue, note that dynamic group membership can always be handled by running the group AKE protocol from scratch among members of the new group. For the case when individual members join and leave, the complexity of the protocol given here is only slightly worse than the Join and Remove algorithms of [15] and [16]. The protocol given here seemingly performs even better relative to [15] and [16] when merging two large groups, or when partitioning a group into two groups of roughly equal size (see [1] for performance comparisons). Adapting the protocol presented here to handle dynamic membership even more efficiently, however, 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 [14]. We have already discussed in the previous section the various drawbacks of this protocol.

We summarize previous provably secure protocols, and compare them with the present work, in Table 1. We compare the efficiency of these protocols (with respect to various complexity measures) in Table 2, below.

Compilers for KE protocols. A modular approach such as that used here has been used previously in the design and analysis of KE protocols. For example, Mayer and Yung [36] give a compiler which converts any two-party protocol into a centralized (asymmetric) 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. [6] also show a compiler (the "BCK compiler") which converts unauthenticated protocols into authenticated protocols. (We remark, however, that their model is slightly different from ours—and thus their compiler does not directly apply to our setting—in that they assume parties agree a priori on unique, matching session ids.) At a high level, applying the BCK compiler to an n-party protocol can be viewed as providing $O(n^2)$ authenticated channels between each pair of parties. In contrast, our compiler may be viewed as providing a *single* authenticated "broadcast" channel to these n parties (although, as discussed below, we do not assume that broadcast is available as a primitive in our network). The compiler we show here is also more computationally efficient than $O(n^2)$ invocations of the BCK compiler.

Burmester and Desmedt [18] suggest a method (specific to their protocol) to achieve security against an active adversary. Their method requires zero-knowledge (ZK) proofs of knowledge, and it seems that (minimally) these ZK proofs need to be non-malleable and secure under concurrent executions in order for their protocol to be provably secure in the model considered here. The authenticated group KE protocol given here is computationally more efficient and has better round complexity than what could be achieved using their approach.

One can also view the work of Bresson et al. [17] as providing a "compiler" specific for the protocol of Steiner et al. [40]. Even for this application, their compiler results in an *exponential* degradation of the concrete security reduction; thus, their proof only applies to groups of *constant* size (see Theorem 1 of [17]). In contrast, our proofs apply to polynomial-size groups.

1.3. *Complexity Measures for Group AKE Protocols*

We measure the efficiency of group AKE protocols in a number of standard ways which we now describe for completeness. This is followed by a comparison of the efficiency of our main protocol with that obtained in previous work.

Round complexity. The round complexity of a protocol is simply the number of rounds until the protocol terminates. Note, however, that this must be more carefully defined in the presence of an active adversary who controls the scheduling of all messages in the network (indeed, the very notion of a "round" is not well-defined in this model, and an adversary can make the time complexity arbitrarily large by refusing to deliver any messages). We measure the round complexity assuming the "best-case" scenario: an adversary who delivers all messages intact to the appropriate recipient(s) as soon as they are sent. Messages which can be sent by parties simultaneously are considered to occur in the same round.

Message complexity. We measure the message complexity in terms of the maximum number of messages sent *by any single user*; this seems more useful in determining the scalability of a protocol. We consider the message complexity in both the point-to-point and "broadcast" models. In the point-to-point model each message sent to a different party is counted separately, while in the "broadcast" model we assume that sending the *same* message to multiple parties incurs the same cost as sending that message to a single party; equivalently, we assume a "broadcast channel" and measure the number of messages a player sends to that channel. We stress, however, that the abstraction of a broadcast channel is used only for measuring the complexity, and we do not assume that a true broadcast channel is available when proving security of our protocols. In particular, an active adversary still has complete control over all communication in the network (and can deliver different messages to different parties), and if a player U is corrupted then that player can send different messages to different parties. Further discussion appears at the beginning of Section 3.1.

Communication complexity. As in the case of message complexity, we measure communication complexity in terms of the maximum number of bits communicated by any single party, and consider both point-to-point and "broadcast" models.

With the above in mind, we compare in Table 2 the complexity of our main protocol to that of previous work [17], [14] (recall that our main protocol is obtained by applying the compiler of Section 3 to the group KE protocol of Section 4). As for computational costs, the protocol of [17] requires some players to compute $O(n)$ "full" modular exponentiations (in addition to $O(1)$ signatures and $O(1)$ signature verifications); the protocol of [14] requires the initiator of the protocol to perform $O(n)$ public-key encryptions and

Table 2. Complexity of provably-secure protocols for group AKE in terms of the number of parties n. See text for explanation of what "point-to-point" and "broadcast" refer to.

		Messages (per user)		Communication (per user)	
	Rounds	Point-to-point	Broadcast	Point-to-point	Broadcast
Bresson et al. [17]	$O(n)$	2	2	$O(n)$	$O(n)$
Boyd–Nieto [14]	1	$O(n)$	1	$O(n^2)$	$O(n)$
Here	3	$O(n)$	3	$O(n)$	$O(1)$

one signature computation; and our protocol requires each player to compute $O(n \log n)$ modular multiplications and to verify $O(n)$ signatures (in addition to $O(1)$ signature computations and three "full" modular exponentiations). As we have already noted, the cost of signature verification in our protocol can be improved using techniques for batch verification [7].

1.4. *Outline*

In Section 2 we review the security model of Bresson et al. [17]. 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.

2. The Model and Preliminaries

For strings $x_1, \ldots, x_n$, we let $y = x_1 | \cdots | x_n$ be a string which unambiguously encodes $x_1, \ldots, x_n$ so that, in particular, it is possible to recover the $\{x_i\}$ from y correctly. As this can be achieved using standard techniques, we do not dwell on it further.

Our security model is the one of Bresson et al. [17] which builds on prior work from the two-party setting [10], [11], [8] and which has been widely used to analyze group KE protocols (see [15], [16], and [14]). 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 $\langle PK_i \rangle_{1 \leq i \leq |\mathcal{P}|}$ 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 our 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, pid_U^i, sid_U^i, and sk_U^i with the following semantics (following [8]):

- state_U^i represents the current (internal) state of instance Π_U^i. The variables term_U^i and acc_U^i take boolean values indicating whether the instance has terminated or accepted, respectively. When an instance has *terminated*, it is done sending and receiving messages. A terminated instance may also possibly *accept*; acceptance is meant to represent (informally) that the instance does not detect any incorrect behavior.
- pid_U^i, the *partner ID* of instance Π_U^i, is a set containing the identities of the players in the group with whom Π_U^i intends to establish a session key (including U itself). The *session ID* sid_U^i provides a means of determining which instances are taking part in a given execution of the protocol; its exact function is discussed further below. Finally, sk_U^i is the *session key* whose computation is the goal of the protocol.

For the most part, we only explicitly refer to pid_U^i, sid_U^i, and sk_U^i (the remaining variables are left implicit).

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. Since protocols in the group setting may wait to receive messages from multiple parties before generating any reply, we write $\mathsf{Send}(U, i, M_1|\cdots|M_\ell)$ to denote sending messages $M_1, \ldots, M_\ell$ to this instance (in response, the oracle generates the reply the instance would generate in response to this sequence of messages).

  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 $\mathsf{Send}(U, i, U_2|\cdots|U_n)$. In this case, pid_U^i is set to $\{U, U_2, \ldots, U_n\}$.
- $\mathsf{Execute}(U_1, i_1, \ldots, U_n, i_n)$—This executes the protocol between the (unused) instances $\{\Pi_{U_j}^{i_j}\}_{1\le j\le n}$, and outputs the transcript of the execution. The pid of each instance is set to $\{U_1, \ldots, U_n\}$. For simplicity, we refer to such a query by $\mathsf{Execute}(U_1, \ldots, U_n)$ and assume the $\{U_i\}$ are in lexicographic order.
- $\mathsf{Reveal}(U, i)$—This outputs session key sk_U^i for a terminated instance Π_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 Execute, Reveal, Corrupt, and Test oracles, while an *active adversary* is additionally given access to the Send oracle. Although the Execute oracle can be simulated via repeated calls to the Send oracle, allowing Execute queries allows for a tighter definition of forward secrecy as well as a more exact concrete security analysis.

Partnering. Partnering is defined via session IDs and partner IDs. The session ID sid_U^i for instance Π_U^i is a protocol-specified function of all communication sent and received by Π_U^i; for our purposes, we make the strictest definition and set sid_U^i equal to the concatenation of all messages sent and received by Π_U^i during its execution (where the messages are ordered by round, and within each round lexicographically by the identities of the purported senders). As we have already discussed above, pid_U^i is a set containing the identities of the players in the group with whom Π_U^i intends to establish a session key (including U itself), and is defined when the adversary initiates execution of the protocol via either the Send or $\mathsf{Execute}$ oracles. We say terminated instances Π_U^i and $\Pi_{U'}^j$ (with $U \neq U'$) 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 [17] since, in our protocols, all messages are sent to all other members of the group taking part in the protocol.

We remark that two instances Π_U^i, Π_U^j of the same user U *cannot* be partnered (i.e., we require $U \neq U'$ above). This definitional choice is motivated by our intuitive notion of "partnering," but has technical consequences as well. In particular, it implies that no *deterministic* protocol can be secure with respect to the definitions given here; see the discussion in Section 2.1.

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 [8], [17], and [30], 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 $\Pi_{U'}^j$ is partnered with Π_U^i; or (2) a query $\mathsf{Corrupt}(V)$ with $V \in \mathsf{pid}_U^i$ was asked *before* a query of the form $\mathsf{Send}(U', j, *)$, where $U' \in \mathsf{pid}_U^i$. We remark that our definition of freshness results in a *stronger* definition of forward secrecy than that given by [8].

Definitions of security. Event Succ occurs if the adversary $\mathcal{A}$ queries the Test oracle on an instance Π_U^i which is still fresh at the conclusion of the experiment and for which $\mathsf{acc}_U^i = \text{TRUE}$, and $\mathcal{A}$ correctly guesses the bit b used by the Test oracle in answering this query. The advantage of $\mathcal{A}$ 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 group authenticated 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 $\mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, q_{\text{ex}})$ 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 $\mathsf{Adv}_P^{\mathsf{AKE\text{-}fs}}(t, q_{\text{ex}}, q_{\text{s}})$ to be the maximum advantage of any active adversary attacking P, running in time t, making q_{ex} calls to the $\mathsf{Execute}$ oracle, and making q_{s} calls to the Send oracle.

Protocols without forward secrecy. The definitions above already incorporate the requirement of forward secrecy since the adversary has unrestricted access to the Corrupt oracle in each case. Our compiler may also be applied to KE protocols which do not achieve forward secrecy. 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 can only access the Corrupt oracle immediately after the initialization stage but before it makes any other oracle queries (this models "static" corruption of players before any executions of the protocol are carried out). This is stronger (and more realistic) than the usual model which simply disallows Corrupt queries altogether.[3]

Authentication. We do not define any notion of explicit authentication or, equivalently, confirmation that the other members of the group were the ones participating in the protocol (such a definition may be derived by adapting the definition of [10] from the two-party setting). However, it is not hard to see that any group AKE protocol resulting from our compiler does indeed achieve explicit authentication.

2.1. *Notes on the Definition*

Deterministic and non-interactive protocols. We claim that for any deterministic KE protocol P there exists an adversary $\mathcal{A}$ with $\mathsf{Adv}_P^{\mathsf{KE}}(t, 2) \approx 1$ (for some small value of t). To see this, consider the adversary who queries $\mathsf{Execute}(U_1, i, U_2, i, U_3, i)$ and $\mathsf{Execute}(U_1, j, U_2, j, U_3, j)$, followed by $\mathsf{Reveal}(U_1, i)$ and $\mathsf{Test}(U_1, j)$. Since P is deterministic, we have that $\mathsf{sk}_{U_1}^i = \mathsf{sk}_{U_1}^j$ and so the adversary can easily succeed with probability essentially 1 (assuming the session keys are long enough). Crucial to this attack is the fact that $\Pi_{U_1}^i$ and $\Pi_{U_1}^j$ are *not* partnered (and so $\Pi_{U_1}^j$ is fresh); note that if we were to allow two instances of the same user to be partnered then in fact *all* the relevant instances above (i.e., $\Pi_{U_1}^i, \Pi_{U_1}^j, \ldots, \Pi_{U_3}^i, \Pi_{U_3}^j$) would be partnered (since they all have identical values of sid and pid) and so the above attack would be ruled out.

A corollary of the above is that no *non-interactive* KE protocol is secure with respect to our definitions. We remark that there do exist non-interactive KE schemes satisfying weaker definitions of security.

Insider attacks. Although the definitions given above are standard for the analysis of group KE protocols [17], [15], [16], [14], there are a number of concerns they do *not* address. For one, the definitions do not offer complete protection against malicious insiders or users who do not honestly follow the protocol. The definitions also do not ensure any form of "agreement" (in the sense of [27]); in fact, since the model gives the adversary *complete* control over all communication in the network (i.e., in addition to delaying messages, the adversary may modify messages or refuse to deliver them at all!) full-fledged agreement is clearly impossible. Finally, the definitions do not protect against "denial of service" attacks in which, for example, an honest instance might "hang" indefinitely; this is a consequence of the adversary's ability to refuse to deliver messages.

[3] We thank an anonymous referee for suggesting that we use this stronger model.

Some of these concerns can be addressed, at least partially, by making suitable changes to existing protocols. For example, it seems that the following approach can be used following any group AKE protocol to achieve confirmation that all (non-corrupted) participants have computed a matching session key: after computing key sk, each player U_i computes $x_i = H(\mathsf{sk}|\mathsf{sid}|U_i)$, signs x_i, broadcasts x_i and the corresponding signature, and computes the "actual" session key $\mathsf{sk}' = H(\mathsf{sk}|\mathsf{sid}|\perp)$ (here, H is modeled as a random oracle and "$\perp$" represents some distinguished string); other players check the validity of the broadcast values in the obvious way. Although this does not provide agreement (since an adversary can still refuse to deliver messages to some of the participants), it *does* ensure the equality of any keys generated by partnered instances. For further discussion about insider attacks and a proof that a variant of the above approach is secure (in the standard model), see [31].

3. A Scalable Compiler for Group AKE Protocols

3.1. *Description of the Compiler*

We show here a compiler transforming any secure group KE protocol P to a secure group AKE protocol P' (recall that a group KE protocol protects against a passive adversary only, while a group AKE protocol additionally protects against an active adversary). Without loss of generality, we assume that in the original protocol P each message sent by an instance Π_U^i 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 jth message sent by an instance Π_U^i has the form $U|j|m$). Furthermore, because the KE protocol discussed in Section 4 has this property, we assume for simplicity that every message of the original protocol P 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. (We discuss briefly below the case when P does not satisfy this.) 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 can deliver different messages to different members of the group or refuse to deliver a message to some of the participants. (Also, if the adversary learns the secret key of a party V via a $\mathsf{Corrupt}$ query, the adversary can send different messages on behalf of V to different members of the group.)

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 assume that the signature length is independent of the length of the message signed; this is easy to achieve in practice by hashing the message (using a collision-resistant hash function) before signing. Given protocol P as above, our compiler constructs a new 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) generated 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 $\mathcal{U} = \{U_1, \ldots, U_n\}$. 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 Π_U^j (for $U \in \mathcal{U}$) sets $\mathsf{nonces}_U^j = U_1|r_1|\cdots|U_n|r_n$ and stores this 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 computes $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|\mathsf{nonces}_U^i)$ and then broadcasts $U|j|m|\sigma$.
   - When instance Π_U^i receives message $V|j|m|\sigma$, it checks that: (1) $V \in \mathsf{pid}_U^i \backslash \{U\}$, (2) j is the next expected sequence number for messages from V, and, finally, (3) $\mathsf{Vrfy}_{PK'_V}(j|m|\mathsf{nonces}_U^i, \sigma) = 1$. 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. Assuming it has not already aborted the protocol, each instance computes the session key as in P.

In case messages in P are not broadcast to the entire group, step 3 in the above may be modified as follows: Whenever instance Π_U^i is supposed to send $U|j|m$ to player U', the instance computes $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|U'|\mathsf{nonces}_U^i)$ and then sends $U|j|m|\sigma$ to U'. (Verification is changed in the obvious way.)

3.2. *Proof of Security*

We now prove that the compiler of the previous section converts any group KE protocol into a group AKE protocol. This holds regardless of whether or not the original protocol achieves forward secrecy (of course, if the original protocol does not achieve forward secrecy then the compiled one does not achieve it either).

Before giving the formal proof we present a high-level overview. In the proof we will transform any active adversary $\mathcal{A}'$ attacking protocol P' into a passive adversary $\mathcal{A}$ attacking protocol P. A natural first attempt is to have $\mathcal{A}$ query its $\mathsf{Execute}$ oracle for each unique value of nonces defined throughout the course of the experiment. This results in a transcript T of an execution of P, which can then be "patched" by $\mathcal{A}$ (by generating appropriate signatures) to yield a transcript T' of an execution of P'. The appropriate messages of T' can then be returned to $\mathcal{A}'$ as it makes the relevant Send queries. We would then like to claim that this yields a "good" simulation since, roughly speaking, $\mathcal{A}'$ is "limited" to sending messages already contained in T' (because $\mathcal{A}'$ cannot forge signatures and, except with negligible probability, nonces do not repeat).

Unfortunately, such an approach does not quite work since $\mathcal{A}'$ might make a $\mathsf{Corrupt}$ query and then send messages that are *not* contained in the "patched" transcript T'. That is, $\mathcal{A}'$ might query $\mathsf{Send}_1(U, \ell, *)$ and then, at some later point in time, corrupt a player $V \in \mathsf{pid}_U^\ell$. This will allow $\mathcal{A}'$ to sign messages on behalf of V and therefore to send arbitrary messages to, e.g., Π_U^ℓ. Note that although Π_U^ℓ is no longer fresh (and so $\mathcal{A}$ need not be concerned with the *secrecy* of the session key generated by that instance), the *messages* output by this instance must still be properly simulated by $\mathcal{A}$. Unfortunately, there does not seem to be any general way that $\mathcal{A}$ can perform such a simulation.

To avoid this potential problem, we adopt the following modified approach (informally): for all but (at most) *one* value of nonces, adversary $\mathcal{A}$ will respond to the Send queries of $\mathcal{A}'$ by running protocol P' *itself*. $\mathcal{A}$ will be able to perform this efficiently by making the necessary $\mathsf{Corrupt}$ queries to obtain any private information needed to execute the underlying protocol. For the (at most one) value of nonces that $\mathcal{A}$ does *not* simulate on its own, $\mathcal{A}$ will use its $\mathsf{Execute}$ oracle to obtain a transcript of P and simulate an execution of P' (as discussed above). In essence, $\mathcal{A}$ is guessing that $\mathcal{A}'$ will eventually issue a Test query for an instance associated with this singular value of nonces, which degrades the concrete security by a factor of roughly q_{s}. (We remark that when forward secrecy is not a concern the above issue does not arise and so a tighter security reduction is possible.) A formal proof follows.

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,*

$$\begin{aligned}\mathsf{Adv}_{P'}^{\mathsf{AKE\text{-}fs}}(t, q_{\mathrm{ex}}, q_{\mathrm{s}}) \;\le\; &\mathsf{Adv}_{P}^{\mathsf{KE\text{-}fs}}(O(t'), q_{\mathrm{ex}}) + \frac{q_{\mathrm{s}}}{2} \cdot \mathsf{Adv}_{P}^{\mathsf{KE\text{-}fs}}(O(t'), 1)\\ &+ |\mathcal{P}| \cdot \mathsf{Succ}_{\Sigma}(O(t')) + \frac{q_{\mathrm{s}}^2 + q_{\mathrm{ex}} q_{\mathrm{s}}}{2^k},\end{aligned}$$

where t' represents the maximum time required to execute an entire experiment involving an adversary running in time t attacking protocol P'.

For the case when P may not achieve forward secrecy, we have

$$\mathsf{Adv}_{P'}^{\mathsf{AKE}}(t, q_{\mathrm{ex}}, q_{\mathrm{s}}) \le \mathsf{Adv}_{P}^{\mathsf{KE}}\left(O(t'), q_{\mathrm{ex}} + \frac{q_{\mathrm{s}}}{2}\right) + |\mathcal{P}| \cdot \mathsf{Succ}_{\Sigma}(O(t')) + \frac{q_{\mathrm{s}}^2 + q_{\mathrm{ex}} q_{\mathrm{s}}}{2^k},$$

where t' is as above.

Proof. In the proof we focus on the case when P achieves forward secrecy and discuss briefly at the end the case when it does not. Given an *active* adversary $\mathcal{A}'$ attacking P', we will construct a *passive* adversary $\mathcal{A}$ attacking P; 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 respective probabilities. 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 $\mathsf{Corrupt}(U)$. (More formally, Forge is the event that $\mathcal{A}'$ makes a query of the form $\mathsf{Send}(V, i, U|j|m|\sigma)$ where $\mathsf{Vrfy}_{PK'_U}(j|m|\mathsf{nonces}_V^i, \sigma) = 1$ but σ was not previously output by any instance of the as-yet-uncorrupted player U as a signature on $j|m|\mathsf{nonces}_V^i$.) It is straightforward to show that $\Pr_{\mathcal{A}',P'}[\mathsf{Forge}] \le |\mathcal{P}| \cdot \mathsf{Succ}_{\Sigma}(O(t'))$.

Let Repeat be the event that a nonce used by any user in response to a Send query was used previously by that user (in response to either an $\mathsf{Execute}$ *or* a Send query). It is immediate that $\Pr_{\mathcal{A}',P'}[\mathsf{Repeat}] \le (q_{\mathrm{s}} q_{\mathrm{ex}} + q_{\mathrm{s}}^2)/2^k$.

Informally, we will consider separately the cases when $\mathcal{A}'$ asks its Test query to an instance initialized via an $\mathsf{Execute}$ query, and the case when $\mathcal{A}'$ asks its Test query to an instance initialized via a Send query. More formally, let Ex be the event that $\mathcal{A}'$ makes its query $\mathsf{Test}(U, i)$ to an instance Π_U^i such that $\mathcal{A}'$ never made a query of the form

$\mathsf{Send}(U, i, *)$ (and therefore *did* make an Execute query involving this instance). We also define $\mathsf{Se} \overset{\text{def}}{=} \overline{\mathsf{Ex}}$. We will separately bound the probabilities of $\Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Ex}]$ and $\Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Se}]$ by constructing appropriate passive adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ (each attacking P).

The initial behavior of adversaries $\mathcal{A}_1/\mathcal{A}_2$ is the same, and so we describe it once and for all. Recall that as part of the initial setup, adversary $\mathcal{A}_1/\mathcal{A}_2$ is given public keys $\{PK_U\}_{U \in \mathcal{P}}$ if any are defined as part of protocol P. First, $\mathcal{A}_1/\mathcal{A}_2$ obtains all secret keys $\{SK_U\}_{U \in \mathcal{P}}$ using multiple Corrupt queries. Next, $\mathcal{A}_1/\mathcal{A}_2$ 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 $\mathcal{A}'$. $\mathcal{A}_1/\mathcal{A}_2$ then runs $\mathcal{A}'$ and simulates the oracle queries of $\mathcal{A}'$ in the manner described below. We stress that $\mathcal{A}_1/\mathcal{A}_2$ now has the secret information of all parties in the network; yet, since P provides forward secrecy, this does not affect the security of P against passive attacks.[4]

We first describe the simulation provided by $\mathcal{A}_1$. At a high level, $\mathcal{A}_1$ simply runs $\mathcal{A}'$ as a subroutine and simulates the Execute queries of $\mathcal{A}'$ using queries to its own Execute oracle. All Send oracle queries of $\mathcal{A}'$ are answered by having $\mathcal{A}_1$ execute protocol P' itself; again, the key point is that $\mathcal{A}_1$ can do this since it has the secret keys of all the parties. If event Se occurs, $\mathcal{A}_1$ aborts and outputs a random bit; otherwise, it outputs whatever bit is eventually output by $\mathcal{A}'$. We now describe the simulation in more detail.

Execute queries. When $\mathcal{A}'$ makes a query $\mathsf{Execute}(U_1, \ldots, U_n)$ (with the U's assumed to be in lexicographic order), $\mathcal{A}_1$ sends the same query to its Execute oracle and receives in return a transcript T of an execution of P. Next, $\mathcal{A}_1$ chooses random $r_1, \ldots, r_n \in \{0, 1\}^k$ and sets $\mathsf{nonces} = U_1|r_1| \cdots |U_n|r_n$. To simulate a transcript T' of an execution of P', $\mathcal{A}_1$ sets the initial messages of T' to $\{U_i|0|r_i\}_{1 \le i \le n}$. Furthermore, for each message $U|j|m$ in transcript T, algorithm $\mathcal{A}_1$ computes signature $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|\mathsf{nonces})$ and places $U|j|m|\sigma$ in T'. When done, the complete transcript T' is given to $\mathcal{A}'$.

Send queries. As discussed earlier, when $\mathcal{A}'$ makes a Send query the response is computed by having $\mathcal{A}_1$ execute protocol P' itself (recording state as necessary).

Reveal/Test queries. Any Reveal queries made by $\mathcal{A}'$ to instances initiated via an Execute query are answered by having $\mathcal{A}_1$ make the appropriate query to its own Reveal oracle. Specifically, when $\mathcal{A}'$ queries $\mathsf{Reveal}(U, \ell)$ for an instance for which $\mathsf{acc}_U^\ell =$ TRUE, let $T' \overset{\text{def}}{=} \mathsf{sid}_U^\ell$ be the transcript of the execution of P' for instance Π_U^ℓ. Let T denote the underlying transcript of protocol P that is contained within transcript T', obtained by simply "stripping" the signatures from T'. Assuming Π_U^ℓ was initiated via an Execute query made by $\mathcal{A}'$, there exists an Execute query made by $\mathcal{A}_1$ which resulted in this transcript; $\mathcal{A}_1$ makes a Reveal query to the appropriate instance belonging to the lexicographically first user involved in this query, and then returns the result to $\mathcal{A}'$.

[4] Indeed, our definition of freshness ensures that all instances initialized by an Execute query are (potentially) fresh even if all parties involved have been corrupted.

Reveal queries made by $\mathcal{A}'$ to instances initiated via a Send query are answered in the obvious way by $\mathcal{A}_1$ (in particular, $\mathcal{A}_1$ simply computes the appropriate session key itself).

When $\mathcal{A}$ makes its Test query $\mathsf{Test}(U, i)$, $\mathcal{A}_1$ checks whether $\mathcal{A}'$ has ever made a query of the form $\mathsf{Send}(U, i, *)$ (i.e., whether event Se has occurred). If so, $\mathcal{A}_1$ aborts and outputs a random bit. Otherwise (namely, if event Ex has occurred), $\mathcal{A}_1$ issues the appropriate Test query to its own oracle, returns the result to $\mathcal{A}'$, and outputs whatever $\mathcal{A}'$ outputs.[5]

The simulation provided by $\mathcal{A}_1$ is perfect (even if Repeat or Forge occurs) unless $\mathcal{A}_1$ aborts due to the occurrence of event Se. Thus,

$$\Pr_{\mathcal{A}_1,P}[\mathsf{Succ}] = \Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Ex}] + \tfrac{1}{2} \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Se}]. \tag{1}$$

Furthermore, $|2 \cdot \Pr_{\mathcal{A}_1,P}[\mathsf{Succ}] - 1| \leq \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), q_{\mathrm{ex}})$ because $\mathcal{A}_1$ makes at most q_{ex} queries to its $\mathsf{Execute}$ oracle.

Before describing $\mathcal{A}_2$, we introduce some terminology. Denote the initial Send query to an instance (denoting a request for protocol initiation) by Send_0; note that for any particular instance Π_U^ℓ this query always has the form $\mathsf{Send}_0(U, \ell, U_1|\cdots|U_n)$. Denote the second Send query to an instance by Send_1; for an instance Π_U^ℓ as before, we may assume without loss of generality that this query has the form $\mathsf{Send}_1(U, \ell, (U_1|0|r_1)|\cdots|(U_n|0|r_n))$ with $r_i \in \{0,1\}^k$ for all i and the $\{U_i\}$ appearing in lexicographic order. Following a Send_1 query to instance Π_U^ℓ, define $\mathsf{nonces}_U^\ell = U_1|r_1|\cdots|U|r_U^\ell|\cdots|U_n|r_n$ where r_U^ℓ is the nonce generated by Π_U^ℓ and the position of $U|r_U^\ell$ is chosen so as to maintain the lexicographic ordering of the U's.

We now describe $\mathcal{A}_2$. At a high level, $\mathcal{A}_2$ guesses that Se will occur and furthermore guesses in advance the *first* Send_1 query corresponding to the eventual Test query of $\mathcal{A}'$. This Send query to some instance Π_U^ℓ (and all Send queries to instances partnered with Π_U^ℓ) will be simulated by having $\mathcal{A}_2$ make the appropriate query to its $\mathsf{Execute}$ oracle to obtain a transcript T, "patch" T (by generating appropriate signatures) to obtain a modified transcript T', and then use T' in the natural way (i.e., send the appropriate message from T' in response to the Send queries of $\mathcal{A}'$). All other Send queries of $\mathcal{A}'$ (as well as all $\mathsf{Execute}$ queries) will be answered by having $\mathcal{A}_2$ simulate execution of P' itself. If Ex, Forge, or Repeat occurs, $\mathcal{A}_2$ aborts and outputs a random bit; otherwise, it makes the appropriate Test query, forwards the result to $\mathcal{A}'$, and outputs whatever $\mathcal{A}'$ outputs. The key point here is that as long as neither Forge nor Repeat occurs, $\mathcal{A}'$ is (informally) "limited" to sending to Π_U^ℓ (and partnered instances) messages contained in T'; adversary $\mathcal{A}'$ can do otherwise only if it queries $\mathsf{Corrupt}(V)$ for some $V \in \mathsf{pid}_U^\ell$, but then Π_U^ℓ is no longer fresh (we assume without loss of generality that $\mathcal{A}'$ only makes a Test query to a fresh instance, since it cannot succeed otherwise).

We now provide the details. First, $\mathcal{A}_2$ chooses a random $\alpha \in \{1, \ldots, q_{\mathrm{s}}/2\}$ (note that $q_{\mathrm{s}}/2$ is an upper bound on the number of Send_1 queries made by $\mathcal{A}'$). It then simulates the

[5] There is a slight subtlety here, in that $\mathcal{A}_1$ must make sure that its Test query is made to a *fresh* instance (and it is possible that two $\mathsf{Execute}$ queries of $\mathcal{A}_1$ result in the same transcript T). This is ensured by always having $\mathcal{A}_1$ direct its $\mathsf{Reveal}/\mathsf{Test}$ queries to an appropriate instance of the *lexicographically first* user involved in the corresponding $\mathsf{Execute}$ query, using the fact that two instances associated with the same user cannot be partnered (by definition).

oracle calls of $\mathcal{A}'$ as described below (to aid the simulation, $\mathcal{A}_2$ maintains a list Nonces whose function will become clear). $\mathcal{A}_2$ aborts immediately (and outputs a random bit) if Repeat or Forge occurs.

Execute queries. All $\mathsf{Execute}$ queries of $\mathcal{A}'$ are answered by having $\mathcal{A}_2$ execute protocol P' itself ($\mathcal{A}_2$ can do this efficiently because it has the secret keys of all players), resulting in a value nonces and a transcript T'. $\mathcal{A}_2$ stores $(\mathsf{nonces}, \emptyset)$ in Nonces and returns T' to $\mathcal{A}'$.

Send queries. These queries are handled as follows:

- On query $\mathsf{Send}_0(U, \ell, *)$, $\mathcal{A}_2$ chooses random $r_U^\ell \in \{0, 1\}^k$ and replies with $U|0|r_U^\ell$.
- On the αth Send_1 query (say, to an instance Π_U^ℓ), the values of pid_U^ℓ and nonces_U^ℓ are defined (by the previous Send_0 query to this instance and the current query, respectively). $\mathcal{A}_2$ first looks in Nonces for an entry of the form $(\mathsf{nonces}_U^\ell, \emptyset)$; if such an entry exists, $\mathcal{A}_2$ aborts and outputs a random bit. Also, if $\mathcal{A}'$ had previously asked a $\mathsf{Corrupt}$ query to any user $V \in \mathsf{pid}_U^\ell$ then $\mathcal{A}_2$ aborts and outputs a random bit. Otherwise, $\mathcal{A}_2$ queries $\mathsf{Execute}(\mathsf{pid}_U^\ell)$, receives in return a transcript T, and stores $(\mathsf{nonces}_U^\ell, T)$ in Nonces. $\mathcal{A}_2$ then finds the message of the form $U|1|m$ in T, computes the signature $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(1|m|\mathsf{nonces}_U^\ell)$, and returns $U|1|m|\sigma$ to $\mathcal{A}'$.
- On any other Send query to an instance Π_U^i, $\mathcal{A}_2$ looks in Nonces for an entry of the form (nonces_U^i, T). If no such entry exists, $\mathcal{A}_2$ stores $(\mathsf{nonces}_U^i, \emptyset)$ in Nonces. In this case or if $T = \emptyset$, $\mathcal{A}_2$ responds by executing protocol P' *itself*; recall again that $\mathcal{A}_2$ can do this since it has the secret information of all players. On the other hand, if $T \neq \emptyset$ this implies that $\mathcal{A}_2$ has received T in response to its single $\mathsf{Execute}$ query. Now, $\mathcal{A}_2$ verifies correctness of the incoming messages as in the specification of the compiler and terminates the instance if verification fails. Additionally, $\mathcal{A}_2$ aborts (and outputs a random bit) if $\mathcal{A}'$ had previously asked a $\mathsf{Corrupt}$ query to any user $V \in \mathsf{pid}_U^i$. Otherwise, $\mathcal{A}_2$ locates the appropriate message $U|j|m$ in transcript T, computes the signature $\sigma \leftarrow \mathsf{Sign}_{SK'_U}(j|m|\mathsf{nonces}_U^\ell)$, and replies to $\mathcal{A}'$ with $U|j|m|\sigma$.

Corrupt queries. These are answered in the obvious way.

Reveal queries. When $\mathcal{A}'$ queries $\mathsf{Reveal}(U, \ell)$ for a terminated instance Π_U^ℓ, the value of nonces_U^ℓ is defined. $\mathcal{A}_2$ finds an entry $(\mathsf{nonces}_U^\ell, T)$ in Nonces and aborts (and outputs a random bit) if $T \neq \emptyset$. Otherwise, if $T = \emptyset$ it means that $\mathcal{A}_2$ has executed protocol P' itself; $\mathcal{A}_2$ can therefore compute the appropriate session key and return it to $\mathcal{A}'$.

Test query. When $\mathcal{A}'$ queries $\mathsf{Test}(U, i)$ for a terminated instance, $\mathcal{A}_2$ finds an entry $(\mathsf{nonces}_U^\ell, T)$ in Nonces. If $T = \emptyset$, then $\mathcal{A}_2$ aborts and outputs a random bit. Otherwise, $\mathcal{A}_2$ makes the appropriate Test query (for one of the instances it activated via its single $\mathsf{Execute}$ query) and returns the result to $\mathcal{A}'$.

Let Bad denote $\mathsf{Forge} \cup \mathsf{Repeat}$, and recall that $\mathcal{A}_2$ aborts immediately if Bad occurs. From the above simulation, we may also note that $\mathcal{A}_2$ aborts if Ex occurs (since in this case $T = \emptyset$ when $\mathcal{A}'$ makes its Test query). On the other hand, if neither Bad nor Ex

occur, then $\mathcal{A}_2$ does not abort (i.e., it correctly guesses the value of α) with probability exactly $2/q_s$. Furthermore, the simulation is perfect in case $\mathcal{A}_2$ does not abort. Putting this together (and letting Guess denote the probability that $\mathcal{A}_2$ correctly guesses α) implies

$$\Pr_{\mathcal{A}_2,P}[\mathsf{Succ}] = \frac{2}{q_s} \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Succ} \wedge \mathsf{Se} \wedge \overline{\mathsf{Bad}}] + \tfrac{1}{2} \cdot \Pr[\mathsf{Ex} \vee \overline{\mathsf{Guess}} \vee \mathsf{Bad}]. \tag{2}$$

Furthermore, $\left|2 \cdot \Pr_{\mathcal{A}_2,P}[\mathsf{Succ}] - 1\right| \le \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), 1)$ since $\mathcal{A}_2$ makes only a single query to its $\mathsf{Execute}$ oracle.

Using (1) and (2) we obtain (in the following, $\Pr[\cdot]$ always denotes $\Pr_{\mathcal{A}',P'}[\cdot]$)

$$\begin{aligned}
&|2 \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Succ}] - 1| \\
&\quad = 2 \cdot |\Pr[\mathsf{Succ} \wedge \mathsf{Ex}] + \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \mathsf{Bad}] + \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \overline{\mathsf{Bad}}] - \tfrac{1}{2}| \\
&\quad \le \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), q_{ex}) \\
&\qquad + 2 \cdot |\Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \overline{\mathsf{Bad}}] + \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \mathsf{Bad}] - \tfrac{1}{2}\Pr[\mathsf{Se}]| \\
&\quad \le \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), q_{ex}) \\
&\qquad + \frac{q_s}{2} \cdot \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), 1) \\
&\qquad + \left| 2 \cdot \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \mathsf{Bad}] - \Pr[\mathsf{Se}] - \frac{q_s}{2}\Pr[\mathsf{Ex} \vee \mathsf{Bad}] \right. \\
&\qquad\quad \left. - \left(\frac{q_s}{2} - 1\right)\Pr[\mathsf{Se} \wedge \overline{\mathsf{Bad}}] + \frac{q_s}{2} \right|,
\end{aligned}$$

using the fact that

$$\begin{aligned}
\Pr[\mathsf{Ex} \vee \overline{\mathsf{Guess}} \vee \mathsf{Bad}] &= \Pr[\mathsf{Ex} \vee \mathsf{Bad}] + \Pr[\overline{\mathsf{Guess}} \wedge \mathsf{Se} \wedge \overline{\mathsf{Bad}}] \\
&= \Pr[\mathsf{Ex} \vee \mathsf{Bad}] + \left(1 - \frac{2}{q_s}\right) \cdot \Pr[\mathsf{Se} \wedge \overline{\mathsf{Bad}}].
\end{aligned}$$

Continuing, and using $\Pr[\mathsf{Ex} \vee \mathsf{Bad}] + \Pr[\mathsf{Se} \wedge \overline{\mathsf{Bad}}] = 1$, we have

$$\begin{aligned}
&|2 \cdot \Pr_{\mathcal{A}',P'}[\mathsf{Succ}] - 1| \\
&\quad \le \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), q_{ex}) + \frac{q_s}{2} \cdot \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), 1) \\
&\qquad + |2 \cdot \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \mathsf{Bad}] - \Pr[\mathsf{Se}] + \Pr[\mathsf{Se} \wedge \overline{\mathsf{Bad}}]| \\
&\quad = \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t)', q_{ex}) + \frac{q_s}{2} \cdot \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), 1) \\
&\qquad + |2 \cdot \Pr[\mathsf{Succ} \wedge \mathsf{Se} \wedge \mathsf{Bad}] - \Pr[\mathsf{Se} \wedge \mathsf{Bad}]| \\
&\quad \le \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), q_{ex}) + \frac{q_s}{2} \cdot \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(O(t'), 1) + \Pr[\mathsf{Bad}].
\end{aligned}$$

Since $\Pr[\mathsf{Bad}] \le \Pr[\mathsf{Forge}] + \Pr[\mathsf{Repeat}]$, this gives the desired result.

For the case when P does not achieve forward secrecy, we can construct an adversary $\mathcal{A}$ that "merges" the behavior of $\mathcal{A}_1$ and $\mathcal{A}_2$; furthermore, $\mathcal{A}$ will not have to guess (as $\mathcal{A}_2$ does) the first Send_1 query corresponding to the eventual Test query of $\mathcal{A}'$. In more detail: $\mathcal{A}$ issues $\mathsf{Corrupt}$ queries in exactly the same way that $\mathcal{A}_1/\mathcal{A}_2$ do (note that all these queries are made at the outset, before any other oracle queries are made). Then $\mathcal{A}$ simulates the experiment for $\mathcal{A}'$ in the following way: $\mathsf{Execute}$ queries of $\mathcal{A}'$ are handled as $\mathcal{A}_1$ does; Send queries of $\mathcal{A}'$ when no members of pid are corrupted are all handled as $\mathcal{A}_2$ does for Send queries corresponding to its "guess"; and Send queries of $\mathcal{A}'$ when some member of pid *is* corrupted are handled by having $\mathcal{A}$ run the protocol itself. (The key point is that $\mathcal{A}$ does not have to worry that $\mathcal{A}'$ will later corrupt some player since all of the $\mathsf{Corrupt}$ queries of $\mathcal{A}'$ must now be made at the outset.) $\mathcal{A}$ aborts (and outputs a random bit) if either Forge or Repeat occurs; otherwise, $\mathcal{A}$ outputs whatever $\mathcal{A}'$ outputs. Since $\mathcal{A}$ makes at most $q_{\mathrm{ex}} + q_{\mathrm{s}}/2$ queries to its $\mathsf{Execute}$ oracle, the claimed bound follows. $\square$

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.

4. A Constant-Round Group KE Protocol

In this section we describe an efficient, two-round group KE protocol which achieves forward secrecy. Applying the compiler of the previous section to this protocol immediately yields (via Theorem 1) an efficient, three-round group AKE protocol achieving forward secrecy. The security of the present protocol is based on the decisional Diffie–Hellman (DDH) assumption [25], which we describe now. Let $\mathbb{G}$ be a cyclic group of prime order q and let g be a fixed generator of $\mathbb{G}$. Informally, the DDH problem is to distinguish between tuples of the form (g^x, g^y, g^{xy}) (called *Diffie–Hellman tuples*) and (g^x, g^y, g^z) where $z \in \mathbb{Z}_q \setminus \{xy\}$ is random[6] (called *random tuples*); $\mathbb{G}$ is said to satisfy the DDH assumption if these two distributions are computationally indistinguishable. More formally, define $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ as the maximum value, over all distinguishing algorithms D running in time at most t, of

$$|\Pr[x, y \leftarrow \mathbb{Z}_q : D(g^x, g^y, g^{xy}) = 1] - \Pr[x, y \leftarrow \mathbb{Z}_q;\, z \leftarrow \mathbb{Z}_q \setminus \{xy\} : D(g^x, g^y, g^z) = 1]|,$$

where we assume that g is fixed and known to algorithm D. Given the above, $\mathbb{G}$ satisfies the DDH assumption if $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ is "small" for "reasonable" values of t. (This definition is appropriate for a concrete security analysis; for asymptotic security one would consider an infinite sequence of groups $\mathcal{G} = \{\mathbb{G}_k\}_{k \geq 1}$ and require that $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_k}(t(k))$ be negligible in k for all polynomials t.) One standard way to generate a group assumed to satisfy the DDH assumption is to choose primes p, q such that $p = \beta q + 1$ and let $\mathbb{G}$ be the subgroup of order q in $\mathbb{Z}_p^*$. However, other choices of $\mathbb{G}$ are also possible.

[6] Our definition is slightly non-standard in that we restrict random tuples to have $z \neq xy$. Since this occurs with probability $1/q$ (for randomly chosen z), our modification has essentially no consequence. We introduce the modification because it simplifies the proof of Theorem 2 somewhat.

The protocol presented here is essentially the protocol of Burmester and Desmedt [19], except that here $\mathbb{G}$ is a cyclic group of prime order assumed to satisfy the DDH assumption (in [19], $\mathbb{G}$ was taken to be an arbitrary cyclic group assumed to satisfy the *computational* Diffie–Hellman (CDH) assumption). Our work was originally motivated by the fact that no proof of security appears in the proceedings version of [19]; furthermore, subsequent work in this area (e.g., [17] and [14]) has implied that the Burmester–Desmedt protocol was "heuristic" and had not been proven secure. 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] (see also [20]). Even so, we note the following:

- Burmester and Desmedt show only that an adversary cannot compute the *entire* session key; in contrast, we show that the session key is indistinguishable from random.
- Burmester and Desmedt give a proof of security only for an *even* number of participants n. They recommend using an asymmetric modified protocol (in which one player simulates the actions of two players) for the case of n odd.
- Finally, Burmester and Desmedt make no effort to optimize the concrete security of the reduction (and do not achieve a tight reduction to the CDH problem); indeed, this issue was not generally considered at that time.

For these reasons, we believe it is important to present a full proof of security for this protocol (taking care to achieve as tight a security reduction as possible) in a precise and widely accepted model.

As required by our compiler, the protocol below 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). For simplicity, in describing our protocol we assume a group $\mathbb{G}$ and a generator $g \in \mathbb{G}$ have been fixed in advance and are known to all parties in the network; however, 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 remains secure follows 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

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

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

Note that each user computes only three full-length exponentiations in $\mathbb{G}$ since $n \ll q$ in practice (typically, $q \approx 2^{160}$ while $n \ll 2^{32}$). 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.

Theorem 2. *The above protocol is a secure group KE protocol achieving forward secrecy. Namely,*

$$\mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, q_{\mathrm{ex}}) \le 4 \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t') + \frac{2 \cdot q_{\mathrm{ex}}}{|\mathbb{G}|},$$

where $t' = t + O(|\mathcal{P}| \cdot q_{\mathrm{ex}} \cdot t_{\mathrm{exp}})$ *and* t_{exp} *is the time to perform exponentiations in* $\mathbb{G}$.

Proof. The overall structure of our proof is as follows: assume for now an adversary who eavesdrops on a single execution of the protocol, and consider the joint distribution of $(\mathsf{T}, \mathsf{sk})$, where $\mathsf{T} = (\{z_i\}, \{X_i\})$ is a transcript of an execution of the protocol and sk is the resulting session key; we refer to this distribution as Real. We first show that, under the DDH assumption, no efficient adversary can distinguish between the distributions Real and Fake, where Fake is a distribution in which (as in the case of Real) all the $\{z_i\}$ are uniformly distributed in $\mathbb{G}$, but in which (in contrast to the case of Real) all the $\{X_i\}$ are uniformly distributed in $\mathbb{G}$ subject to the constraint $\prod_i X_i = 1$ (in this informal discussion, we do not describe the distribution of the session key in Fake). We then show that in distribution Fake, the value of the session key is uniformly distributed in $\mathbb{G}$, *independent* of the value of the transcript. Security of the original protocol, at least for an adversary making only a single $\mathsf{Execute}$ query, follows immediately. The case of an adversary making multiple $\mathsf{Execute}$ queries can be dealt with using a straightforward hybrid argument, but we show an improved analysis resulting in a tighter concrete security reduction.

A natural approach for the first step described above (namely, showing that Real and Fake are indistinguishable) is to proceed via a sequence of n hybrid distributions in which (roughly speaking) one X_i at a time is replaced with a random group element (subject to the above-mentioned constraint). To obtain a tighter concrete security reduction, we instead use the random self-reducibility [38], [5] of the DDH assumption. Due to the dependence between the $\{X_i\}$ we were unable to show a "one step" reduction. Instead, we proceed in two steps via an intermediate distribution Fake', where each step modifies the distribution of a subset of the $\{X_i\}$. We now give the details.

Let $\varepsilon(\cdot) \overset{\mathrm{def}}{=} \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\cdot)$. We consider first an adversary making a single $\mathsf{Execute}$ query and show that $\mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, 1) \le 4 \cdot \varepsilon(t'')$, where $t'' = t + O(n \cdot t_{\mathrm{exp}})$ and n is the number of parties involved in the $\mathsf{Execute}$ query. We then discuss how to extend the proof for the case of multiple $\mathsf{Execute}$ queries without affecting the tightness of the security reduction.

Since there are no public keys, we simply 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; since the protocol is symmetric and there are no public keys, however, the identities of the parties are unimportant for our discussion.

Let $n = 3s + k$ where $k \in \{3, 4, 5\}$ and $s \ge 0$ is an integer (we assume $n > 2$ since the protocol reduces to the standard Diffie–Hellman protocol [25] for the case $n = 2$). A detailed proof requires a slightly different analysis for each of the possible values of k and depending on whether or not $s = 0$; we describe here the proof for $k = 5, s > 0$ (proofs for the other cases follow using similar arguments). Considering an execution of the protocol as described above, define $\Gamma_{i,i+1} = g^{r_i r_{i+1}}$ and note that

$$X_i \overset{\text{def}}{=} \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i} = \frac{g^{r_i r_{i+1}}}{g^{r_{i-1} r_i}} = \frac{\Gamma_{i,i+1}}{\Gamma_{i-1,i}}.$$

Furthermore, the (common) session key is equal to

$$\mathsf{sk}_1 \overset{\text{def}}{=} (z_n)^{n r_1} X_1^{n-1} \cdots X_{n-1} = (\Gamma_{n,1})^n X_1^{n-1} \cdots X_{n-1}.$$

Thus, in a real execution of the protocol the distribution of the transcript T and the resulting session key sk is given by the following, denoted Real:

$$\mathsf{Real} \overset{\text{def}}{=} \left\{ \begin{array}{l} r_1, \ldots, r_n \leftarrow \mathbb{Z}_q; \\ z_1 = g^{r_1},\ z_2 = g^{r_2},\ \ldots,\ z_n = g^{r_n}; \\ \Gamma_{1,2} = g^{r_1 r_2},\ \Gamma_{2,3} = g^{r_2 r_3},\ \ldots,\ \Gamma_{n-1,n} = g^{r_{n-1} r_n},\ \Gamma_{n,1} = g^{r_n r_1}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}.$$

We next define a distribution Fake' in the following way: random exponents $\{r_i\}$ are selected as in the case of Real. The $\{z_i\}$ are also computed exactly as in Real. However, the values $\Gamma_{1,2}, \Gamma_{2,3}, \Gamma_{3,4}$, as well as every value $\Gamma_{j,j+1}$ with $j \geq 6$ a multiple of 3 (i.e., every third subsequent Γ value), are now chosen uniformly at random from $\mathbb{G}$ (rather than computed as in Real). Formally (recall $n = 3s + 5$ and $s \geq 1$):

$$\mathsf{Fake}' \overset{\text{def}}{=} \left\{ \begin{array}{l} r_1, \ldots, r_n \leftarrow \mathbb{Z}_q; \\ z_1 = g^{r_1},\ z_2 = g^{r_2},\ \ldots,\ z_n = g^{r_n}; \\ \Gamma_{1,2}, \Gamma_{2,3}, \Gamma_{3,4} \leftarrow \mathbb{G};\ \Gamma_{4,5} = g^{r_4 r_5}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad \Gamma_{j-1,j} = g^{r_{j-1} r_j},\ \Gamma_{j,j+1} \leftarrow \mathbb{G},\ \Gamma_{j+1,j+2} = g^{r_{j+1} r_{j+2}}; \\ \Gamma_{n,1} = g^{r_n r_1}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}.$$

Claim. *For any algorithm $\mathcal{A}$ running in time t we have*

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

Proof. Given algorithm $\mathcal{A}$, consider the following algorithm D which takes as input a triple $(g_1, g_2, g_3) \in \mathbb{G}^3$ (where furthermore a generator $g \in \mathbb{G}$ is fixed): D generates $(\mathsf{T}, \mathsf{sk})$ according to distribution Dist', runs $\mathcal{A}(\mathsf{T}, \mathsf{sk})$, and outputs whatever $\mathcal{A}$

outputs. Distribution Dist' is defined as follows (note that this distribution depends on g_1, g_2, g_3):

$$\mathsf{Dist}' \overset{\text{def}}{=} \left\{ \begin{array}{l} \alpha_0, \beta_0, \alpha_0', \beta_0', r_0, \{\alpha_i, \beta_i, \gamma_i, r_i\}_{i=1}^s \leftarrow \mathbb{Z}_q; \\ z_1 = g^{\alpha_0} g_2^{\beta_0},\ z_2 = g_1,\ z_3 = g_2,\ z_4 = g^{\alpha_0'} g_1^{\beta_0'},\ z_5 = g^{r_0}; \\ \Gamma_{1,2} = g_1^{\alpha_0} g_3^{\beta_0},\ \Gamma_{2,3} = g_3,\ \Gamma_{3,4} = g_2^{\alpha_0'} g_3^{\beta_0'},\ \Gamma_{4,5} = z_4^{r_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\gamma_i} g_1,\ z_{j+1} = g^{\alpha_i} g_2^{\beta_i},\ z_{j+2} = g^{r_i}; \\ \quad \Gamma_{j-1,j} = z_j^{r_{i-1}},\ \Gamma_{j,j+1} = g_1^{\alpha_i} g_3^{\beta_i} z_{j+1}^{\gamma_i},\ \Gamma_{j+1,j+2} = z_{j+1}^{r_i}; \\ \Gamma_{n,1} = z_1^{r_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}.$$

We first examine the above distribution when (g_1, g_2, g_3) is chosen uniformly at random from the set of Diffie–Hellman tuples (i.e., $g_1 = g^x$, $g_2 = g^y$, and $g_3 = g^{xy}$); we refer to the resulting distribution as $\mathsf{Dist}'_{\mathsf{ddh}}$:

$$\mathsf{Dist}'_{\mathsf{ddh}} = \left\{ \begin{array}{l} x, y, \alpha_0, \beta_0, \alpha_0', \beta_0', r_0, \{\alpha_i, \beta_i, \gamma_i, r_i\}_{i=1}^s \leftarrow \mathbb{Z}_q; \\ z_1 = g^{\alpha_0} g^{y\beta_0},\ z_2 = g^x,\ z_3 = g^y,\ z_4 = g^{\alpha_0'} g^{x\beta_0'},\ z_5 = g^{r_0}; \\ \Gamma_{1,2} = g^{x\alpha_0} g^{xy\beta_0},\ \Gamma_{2,3} = g^{xy},\ \Gamma_{3,4} = g^{y\alpha_0'} g^{xy\beta_0'},\ \Gamma_{4,5} = z_4^{r_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\gamma_i} g^x,\ z_{j+1} = g^{\alpha_i} g^{y\beta_i},\ z_{j+2} = g^{r_i}; \\ \quad \Gamma_{j-1,j} = z_j^{r_{i-1}},\ \Gamma_{j,j+1} = g^{x\alpha_i} g^{xy\beta_i} z_{j+1}^{\gamma_i},\ \Gamma_{j+1,j+2} = z_{j+1}^{r_i}; \\ \Gamma_{n,1} = z_1^{r_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\},$$

where all we have done is substitute $g_1 = g^x$, $g_2 = g^y$, and $g_3 = g^{xy}$ (for $x, y \in \mathbb{Z}_q$ chosen at random at the beginning of the experiment) into the definition of Dist'. We claim that $\mathsf{Dist}'_{\mathsf{ddh}}$ is identical to Real. To see this, first notice that $\{X_i\}$, T, and sk are computed identically in both, and so it suffices to look at the distribution of the z's and the Γ's. It is not hard to see that in $\mathsf{Dist}'_{\mathsf{ddh}}$, the $\{z_k\}_{k=1}^n$ are uniformly and independently distributed in $\mathbb{G}$, exactly as in Real. It remains to show that for all $k \in \{1, \ldots, n\}$, the tuple $(z_k, z_{k+1}, \Gamma_{k,k+1})$ in $\mathsf{Dist}'_{\mathsf{ddh}}$ is a Diffie–Hellman tuple (as in Real). This can be verified by inspection. We conclude that

$$\Pr[x, y \leftarrow \mathbb{Z}_q : D(g^x, g^y, g^{xy}) = 1] = \Pr[(\mathsf{T}, \mathsf{sk}) \leftarrow \mathsf{Real} : \mathcal{A}(\mathsf{T}, \mathsf{sk}) = 1]. \qquad (3)$$

We now examine distribution Dist' in case (g_1, g_2, g_3) is chosen uniformly from the space of random tuples (we refer to the resulting distribution as $\mathsf{Dist}'_{\mathsf{rand}}$):

$$\mathsf{Dist}'_{\mathsf{rand}} = \left\{ \begin{array}{l} x, y, \alpha_0, \beta_0, \alpha'_0, \beta'_0, r_0, \{\alpha_i, \beta_i, \gamma_i, r_i\}_{i=1}^{s} \leftarrow \mathbb{Z}_q;\ z \leftarrow \mathbb{Z}_q \backslash \{xy\}; \\ z_1 = g^{\alpha_0} g^{y\beta_0},\ z_2 = g^x,\ z_3 = g^y,\ z_4 = g^{\alpha'_0} g^{x\beta'_0},\ z_5 = g^{r_0}; \\ \Gamma_{1,2} = g^{x\alpha_0} g^{z\beta_0},\ \Gamma_{2,3} = g^z,\ \Gamma_{3,4} = g^{y\alpha'_0} g^{z\beta'_0},\ \Gamma_{4,5} = z_4^{r_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\gamma_i} g^x,\ z_{j+1} = g^{\alpha_i} g^{y\beta_i},\ z_{j+2} = g^{r_i}; \\ \quad \Gamma_{j-1,j} = z_j^{r_{i-1}},\ \Gamma_{j,j+1} = g^{x\alpha_i} g^{z\beta_i} z_{j+1}^{\gamma_i},\ \Gamma_{j+1,j+2} = z_{j+1}^{r_i}; \\ \Gamma_{n,1} = z_1^{r_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\},$$

where all we have done is substitute $g_1 = g^x$, $g_2 = g^y$, and $g_3 = g^z$ (for x, y, z chosen at the beginning of the experiment) into the definition of Dist'. We claim that $\mathsf{Dist}'_{\mathsf{rand}}$ is statistically close to Fake'. As before, we need only examine the distribution of the z's and the Γ's. It is again not hard to see that in $\mathsf{Dist}'_{\mathsf{rand}}$ the $\{z_k\}$ are uniformly and independently distributed in $\mathbb{G}$, as in Fake'. By inspection, we see also that $\Gamma_{4,5}$, $\{\Gamma_{j-1,j}, \Gamma_{j+1,j+2}\}_{j=3i+3;\, i=1,\ldots,s}$, and $\Gamma_{n,1}$ are distributed identically in $\mathsf{Dist}'_{\mathsf{rand}}$ and Fake' (in each case these values are completely determined by the $\{z_k\}$). It only remains to show that in $\mathsf{Dist}'_{\mathsf{rand}}$, the distribution on each of the remaining Γ's is statistically close to the uniform one. Take $\Gamma_{j,j+1}$ (for some $i \in \{1, \ldots, s\}$ and $j = 3i+3$) as a representative example. Since α_i, β_i are used only in computing z_{j+1} and $\Gamma_{j,j+1}$, the joint distribution of $z_{j+1}, \Gamma_{j,j+1}$—conditioned on arbitrary values of all other Γ's and z's—is given by

$$\begin{aligned} \log_g z_{j+1} &= \alpha_i + y \cdot \beta_i, \\ \log_g \Gamma_{j,j+1} &= x \cdot \alpha_i + z \cdot \beta_i + \gamma_i \log_g z_{j+1} \\ &= (x+1) \cdot \alpha_i + (z+y) \cdot \beta_i, \end{aligned}$$

where α_i, β_i are chosen uniformly and independently from $\mathbb{Z}_q$. Using the fact that $z \neq xy$, the above (viewed as linear equations in the variables α_i, β_i) are linearly independent. It follows that $\Gamma_{j,j+1}$ is uniformly distributed in $\mathbb{G}$, independent of z_{j+1} and everything else. (We remark that this argument is essentially the same as that used in [38] and [5, Lemma 5.2].) A similar argument applies for $\Gamma_{1,2}$ and $\Gamma_{3,4}$. The only value left is $\Gamma_{2,3}$. Because of our restriction on the choice of z, the value $\log_g \Gamma_{2,3}$ in $\mathsf{Dist}'_{\mathsf{rand}}$ is distributed uniformly in $\mathbb{Z}_q \backslash \{xy\}$. This is statistically close (within a factor of $1/|\mathbb{G}|$) to the distribution of

$\log_g \Gamma_{2,3}$ in Fake'. We conclude that

$$\begin{aligned}&\Pr[x, y \leftarrow \mathbb{Z}_q; z \leftarrow \mathbb{Z}_q \backslash \{xy\} : D(g^x, g^y, g^z) = 1] \\ &\qquad \geq \Pr[(\mathsf{T}, \mathsf{sk}) \leftarrow \mathsf{Fake}' : \mathcal{A}(\mathsf{T}, \mathsf{sk}) = 1] - \frac{1}{|\mathbb{G}|}. \end{aligned} \tag{4}$$

The running time of D is t'', the running time of $\mathcal{A}$ plus $O(n)$ exponentiations in $\mathbb{G}$. The claim now follows easily from (3) and (4) and the definition of ε. □

We now introduce one final distribution in which *all* the Γ's are chosen uniformly and independently at random from $\mathbb{G}$:

$$\mathsf{Fake} \overset{\text{def}}{=} \left\{ \begin{array}{l} r_1, \ldots, r_n \leftarrow \mathbb{Z}_q; \\ z_1 = g^{r_1}, \ \ldots, \ z_n = g^{r_n}; \\ \Gamma_{1,2}, \ldots, \Gamma_{n-1,n}, \Gamma_{n,1} \leftarrow \mathbb{G}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1}, \ X_2 = \Gamma_{2,3}/\Gamma_{1,2}, \ \ldots, \ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n); \ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}.$$

Claim. *For any algorithm $\mathcal{A}$ running in time t we have*

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

Proof. Given algorithm $\mathcal{A}$, consider the following algorithm D which takes as input a triple $(g_1, g_2, g_3) \in \mathbb{G}^3$ (where furthermore a generator $g \in \mathbb{G}$ is fixed): D generates $(\mathsf{T}, \mathsf{sk})$ according to distribution Dist, runs $\mathcal{A}(\mathsf{T}, \mathsf{sk})$, and outputs whatever $\mathcal{A}$ outputs. Distribution Dist is defined as follows (note that this distribution depends on g_1, g_2, g_3):

$$\mathsf{Dist} \overset{\text{def}}{=} \left\{ \begin{array}{l} r_1, r_2, \{\alpha_i, \beta_i, \alpha'_i, \beta'_i, \gamma_i\}_{i=0}^{s} \leftarrow \mathbb{Z}_q; \\ z_1 = g^{\alpha_0} g_2^{\beta_0}, \ z_2 = g^{r_1}, \ z_3 = g^{r_2}, \ z_4 = g^{\alpha'_0} g_2^{\beta'_0}, \ z_5 = g^{\gamma_0} g_1; \\ \Gamma_{1,2}, \Gamma_{2,3}, \Gamma_{3,4} \leftarrow \mathbb{G}; \ \Gamma_{4,5} = g_1^{\alpha'_0} g_3^{\beta'_0} z_4^{\gamma_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\alpha_i} g_2^{\beta_i}, \ z_{j+1} = g^{\alpha'_i} g_2^{\beta'_i}, \ z_{j+2} = g^{\gamma_i} g_1; \\ \quad \Gamma_{j-1,j} = g_1^{\alpha_i} g_3^{\beta_i} z_j^{\gamma_{i-1}}, \ \Gamma_{j,j+1} \leftarrow \mathbb{G}, \ \Gamma_{j+1,j+2} = g_1^{\alpha'_i} g_3^{\beta'_i} z_{j+1}^{\gamma_i}; \\ \Gamma_{n,1} = g_1^{\alpha_0} g_3^{\beta_0} z_1^{\gamma_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1}, \ X_2 = \Gamma_{2,3}/\Gamma_{1,2}, \ \ldots, \ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n); \ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\}.$$

The remainder of the proof is very similar to the proof of the previous claim. We first examine the above distribution when (g_1, g_2, g_3) is chosen uniformly at random from the set of Diffie–Hellman tuples (we refer to the resulting distribution as $\mathsf{Dist}_{\mathsf{ddh}}$):

$$\mathsf{Dist}_{\mathsf{ddh}} = \left\{ \begin{array}{l} x, y, r_1, r_2, \{\alpha_i, \beta_i, \alpha'_i, \beta'_i, \gamma_i\}_{i=0}^{s} \leftarrow \mathbb{Z}_q; \\ z_1 = g^{\alpha_0} g^{y\beta_0},\ z_2 = g^{r_1},\ z_3 = g^{r_2},\ z_4 = g^{\alpha'_0} g^{y\beta'_0},\ z_5 = g^{\gamma_0} g^{x}; \\ \Gamma_{1,2}, \Gamma_{2,3}, \Gamma_{3,4} \leftarrow \mathbb{G};\ \Gamma_{4,5} = g^{x\alpha'_0} g^{xy\beta'_0} z_4^{\gamma_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\alpha_i} g^{y\beta_i},\ z_{j+1} = g^{\alpha'_i} g^{y\beta'_i},\ z_{j+2} = g^{\gamma_i} g^{x}; \\ \quad \Gamma_{j-1,j} = g^{x\alpha_i} g^{xy\beta_i} z_j^{\gamma_{i-1}}, \\ \quad \Gamma_{j,j+1} \leftarrow \mathbb{G},\ \Gamma_{j+1,j+2} = g^{x\alpha'_i} g^{xy\beta'_i} z_{j+1}^{\gamma_i}; \\ \Gamma_{n,1} = g^{x\alpha_0} g^{xy\beta_0} z_1^{\gamma_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\},$$

where all we have done is substitute $g_1 = g^x$, $g_2 = g^y$, and $g_3 = g^{xy}$ (for $x, y \in \mathbb{Z}_q$ chosen at random at the beginning of the experiment) into the definition of Dist. We claim that $\mathsf{Dist}_{\mathsf{ddh}}$ is identical to Fake'. As in the proof of the previous claim, we need only focus on the z's and the Γ's. It is again easy to see that in $\mathsf{Dist}_{\mathsf{ddh}}$ the $\{z_k\}$ are uniformly and independently distributed in $\mathbb{G}$ (as in Fake'). It is also immediate that in both distributions we have $\Gamma_{1,2}$, $\Gamma_{2,3}$, $\Gamma_{3,4}$, and $\{\Gamma_{j,j+1}\}_{j=3i+3;\, i=1,\ldots,s}$ distributed uniformly and independently in $\mathbb{G}$. As for the remaining Γ's, take $\Gamma_{j-1,j}$ (for some $i \in \{1, \ldots, s\}$ and $j = 3i + 3$) as a representative example. We claim that in $\mathsf{Dist}_{\mathsf{ddh}}$ the tuple $(z_{j-1}, z_j, \Gamma_{j-1,j})$ is a Diffie–Hellman tuple, as is the case in Fake'. To see this, note that $z_{j-1} = g^{\gamma_{i-1}+x}$, $z_j = g^{\alpha_i + y\beta_i}$ and

$$\Gamma_{j-1,j} = g^{x\alpha_i + xy\beta_i} z_j^{\gamma_{i-1}} = g^{x\alpha_i + xy\beta_i + \gamma_{i-1}\alpha_i + \gamma_{i-1}y\beta_i}.$$

Thus, $(z_{j-1}, z_j, \Gamma_{j-1,j})$ is a Diffie–Hellman tuple as desired. We conclude that

$$\Pr[x, y \leftarrow \mathbb{Z}_q : D(g^x, g^y, g^{xy}) = 1] = \Pr[(\mathsf{T}, \mathsf{sk}) \leftarrow \mathsf{Fake}' : \mathcal{A}(\mathsf{T}, \mathsf{sk}) = 1]. \tag{5}$$

We now examine distribution Dist when (g_1, g_2, g_3) is chosen uniformly from the set of random tuples (we refer to the resulting distribution as $\mathsf{Dist}_{\mathsf{rand}}$):

$$\mathsf{Dist}_{\mathsf{rand}} = \left\{ \begin{array}{l} x, y, r_1, r_2, \{\alpha_i, \beta_i, \alpha'_i, \beta'_i, \gamma_i\}_{i=0}^{s} \leftarrow \mathbb{Z}_q;\ z \leftarrow \mathbb{Z}_q \backslash \{xy\}; \\ z_1 = g^{\alpha_0} g^{y\beta_0},\ z_2 = g^{r_1},\ z_3 = g^{r_2},\ z_4 = g^{\alpha'_0} g^{y\beta'_0},\ z_5 = g^{\gamma_0} g^{x}; \\ \Gamma_{1,2}, \Gamma_{2,3}, \Gamma_{3,4} \leftarrow \mathbb{G};\ \Gamma_{4,5} = g^{x\alpha'_0} g^{z\beta'_0} z_4^{\gamma_0}; \\ \text{for } i = 1 \text{ to } s: \\ \quad \text{let } j = 3i + 3 \\ \quad z_j = g^{\alpha_i} g^{y\beta_i},\ z_{j+1} = g^{\alpha'_i} g^{y\beta'_i},\ z_{j+2} = g^{\gamma_i} g^{x}; \\ \quad \Gamma_{j-1,j} = g^{x\alpha_i} g^{z\beta_i} z_j^{\gamma_{i-1}}, \\ \quad \Gamma_{j,j+1} \leftarrow \mathbb{G},\ \Gamma_{j+1,j+2} = g^{x\alpha'_i} g^{z\beta'_i} z_{j+1}^{\gamma_i}; \\ \Gamma_{n,1} = g^{x\alpha_0} g^{z\beta_0} z_1^{\gamma_s}; \\ X_1 = \Gamma_{1,2}/\Gamma_{n,1},\ X_2 = \Gamma_{2,3}/\Gamma_{1,2},\ \ldots,\ X_n = \Gamma_{n,1}/\Gamma_{n-1,n}; \\ \mathsf{T} = (z_1, \ldots, z_n, X_1, \ldots, X_n);\ \mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} \end{array} : (\mathsf{T}, \mathsf{sk}) \right\},$$

where all we have done is substitute $g_1 = g^x$, $g_2 = g^y$, and $g_3 = g^z$ (for x, y, z chosen at the beginning of the experiment) into the definition of Dist. We claim that $\mathsf{Dist}_{\mathsf{rand}}$ is identical to Fake. It is easy to verify that in $\mathsf{Dist}_{\mathsf{rand}}$ the $\{z_k\}$ are uniformly and independently distributed in $\mathbb{G}$. We need to show that the Γ's are also uniformly distributed, independent of each other and the z's (as is the case in Fake). This clearly holds for $\Gamma_{1,2}$, $\Gamma_{2,3}$, $\Gamma_{3,4}$, and $\{\Gamma_{j,j+1}\}_{j=3i+3;\, i=1,\ldots,s}$. For the remaining Γ's, take $\Gamma_{4,5}$ as a representative example. Since α_0', β_0' are used only in computing z_4 and $\Gamma_{4,5}$, the joint distribution of z_4 and $\Gamma_{4,5}$—conditioned on arbitrary values of all other z's and Γ's—is given by

$$\begin{aligned}\log_g z_4 &= \alpha_0' + y \cdot \beta_0',\\ \log_g \Gamma_{4,5} &= x \cdot \alpha_0' + z \cdot \beta_0' + \gamma_0 \log_g z_4\\ &= (x+1) \cdot \alpha_0' + (z+y) \cdot \beta_0',\end{aligned}$$

where α_0', β_0' are uniformly and independently distributed in $\mathbb{Z}_q$. Using the fact that $z \neq xy$, the above (viewed as linear equations in the variables α_0', β_0') are linearly independent. It follows that $\Gamma_{4,5}$ is uniformly distributed in $\mathbb{G}$, independent of z_4 and everything else. We conclude that

$$\begin{aligned}&\Pr[x, y \leftarrow \mathbb{Z}_q; z \leftarrow \mathbb{Z}_q \backslash \{xy\} : D(g^x, g^y, g^z) = 1]\\ &\quad = \Pr[(\mathsf{T}, \mathsf{sk}) \leftarrow \mathsf{Fake} : \mathcal{A}(\mathsf{T}, \mathsf{sk}) = 1]. \qquad (6)\end{aligned}$$

The running time of D is t'':, the running time of $\mathcal{A}$ plus $O(n)$ exponentiations in $\mathbb{G}$. The claim now follows readily from (5) and (6) and the definition of ε. □

In experiment Fake, let $w_{i,i+1} \overset{\text{def}}{=} \log_g \Gamma_{i,i+1}$ for $1 \leq i \leq n$. Given T, the values $w_{1,2}, \ldots, w_{n,1}$ are constrained by the following n equations (only $n-1$ of which are linearly independent):

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

Furthermore, $\mathsf{sk} = g^{w_{1,2}+w_{2,3}+\cdots+w_{n,1}}$; 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, sk is independent of T. This implies that even for a computationally unbounded adversary $\mathcal{A}$,

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

Combining this with the previous two claims shows that $\mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, 1) \leq 4 \cdot \varepsilon(t'') + 2/|\mathbb{G}|$.

For the case of $q_{\mathrm{ex}} > 1$, a standard hybrid argument immediately shows that

$$\mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, q_{\mathrm{ex}}) \leq q_{\mathrm{ex}} \cdot \mathsf{Adv}_P^{\mathsf{KE\text{-}fs}}(t, 1).$$

Tighter concrete security can be obtained by again using the random self-reducibility of the DDH problem [5, Lemma 5.2]. In particular, given a tuple $(g_1, g_2, g_3) \in \mathbb{G}^3$ and a fixed generator g, one can efficiently generate $q_{\rm ex}$ tuples $L = \{(g_1^1, g_2^1, g_3^1), \ldots, (g_1^{q_{\rm ex}}, g_2^{q_{\rm ex}}, g_3^{q_{\rm ex}})\}$ such that (1) if (g_1, g_2, g_3) is a Diffie–Hellman tuple, then all tuples in L are Diffie–Hellman tuples whose first two components are randomly distributed in $\mathbb{G}^2$ (independently of anything else); (2) if (g_1, g_2, g_3) is a random tuple, then all tuples in L are randomly distributed in $\mathbb{G}^3$ (again, independently of anything else). In the second case, with all but probability $q_{\rm ex}/|\mathbb{G}|$, it will be the case that $\log_g g_3^i \neq \log_g g_1^i \cdot \log_g g_2^i$ for all i.

Paralleling the preceding proof, we may define distributions $\mathsf{Real}_{q_{\rm ex}}$, $\mathsf{Fake}'_{q_{\rm ex}}$, $\mathsf{Dist}'_{q_{\rm ex}}$, $\mathsf{Fake}_{q_{\rm ex}}$, and $\mathsf{Dist}_{q_{\rm ex}}$ which simply consist of $q_{\rm ex}$ (independent) copies of each of the corresponding distributions; in the case of $\mathsf{Dist}'_{q_{\rm ex}}$ and $\mathsf{Dist}_{q_{\rm ex}}$ we use the corresponding tuple (g_1^i, g_2^i, g_3^i) for the ith copy. Corresponding to the first claim, one can then show that for any algorithm $\mathcal{A}$ running in time t,

$$|\Pr[(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) \leftarrow \mathsf{Real}_{q_{\rm ex}} : \mathcal{A}(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) = 1] - \Pr[(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) \leftarrow \mathsf{Fake}'_{q_{\rm ex}} : \mathcal{A}(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) = 1]| \leq \varepsilon(t'),$$

where t' is as in the statement of the theorem. (We remark that we no longer lose the factor $O(1/|\mathbb{G}|)$ since the tuples in L are now completely random when (g_1, g_2, g_3) is not a Diffie–Hellman tuple.) Corresponding to the second claim, one could show that for any algorithm $\mathcal{A}$ running in time t,

$$\begin{aligned} &|\Pr[(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) \leftarrow \mathsf{Fake}'_{q_{\rm ex}} : \mathcal{A}(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) = 1] - \Pr[(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) \leftarrow \mathsf{Fake}_{q_{\rm ex}} : \mathcal{A}(\vec{\mathsf{T}}, \vec{\mathsf{sk}}) = 1]| \\ &\qquad \leq \ \varepsilon(t') + \frac{q_{\rm ex}}{|\mathbb{G}|}. \end{aligned}$$

Finally, using the same techniques as above, it is straightforward to see that even for a computationally unbounded $\mathcal{A}$,

$$\Pr[(\vec{\mathsf{T}}, \vec{\mathsf{sk}}_0) \leftarrow \mathsf{Fake}; \vec{\mathsf{sk}}_1 \leftarrow \mathbb{G}^{q_{\rm ex}}; b \leftarrow \{0,1\} : \mathcal{A}(\mathsf{T}, \vec{\mathsf{sk}}_b) = b] = \tfrac{1}{2}.$$

Putting these together gives the result of the theorem. □

References

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the Performance of Group Key Agreement Protocols. *Proc*. 22*nd International Conference on Distributed Computing Systems* (*ICDCS*), IEEE, Piscataway, NJ, 2002, pp. 463–464. Full version available at http://www.cnds.jhu.edu/publications/.

[2] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. *Proc*. 5*th Annual ACM Conference on Computer and Communications Security*, ACM, New York, 1998, pp. 17–26.

[3] G. Ateniese, M. Steiner, and G. Tsudik. New Multi-Party Authentication Services and Key Agreement Protocols. *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4 (2000), pp. 628–639.

[4] C. Becker and U. Wille. Communication Complexity of Group Key Distribution. *Proc*. 5*th Annual ACM Conference on Computer and Communication Security*, ACM, New York, 1998, pp. 1–6.

[5] M. Bellare, A. Boldyreva, and S. Micali. Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements. *Advances in Cryptology—Eurocrypt* 2000, LNCS vol. 1807, B. Preneel, ed., Springer-Verlag, Berlin, 2000, pp. 259–274.

[6] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. *Proc*. 30*th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1998, pp. 419–428.

[7] M. Bellare, J. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. *Advances in Cryptology—Eurocrypt* '98, LNCS vol. 1403, K. Nyberg ed., Springer-Verlag, Berlin, 1998, pp. 236–250.

[8] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Advances in Cryptology—Eurocrypt* 2000, LNCS vol. 1807, B. Preneel, ed., Springer-Verlag, Berlin, 2000, pp. 139–155.

[9] M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. *Proc*. 1*st Annual ACM Conference on Computer and Communications Security*, ACM, New York, 1993, pp. 62–73.

[10] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Advances in Cryptology—Crypto* '93, LNCS vol. 773, D. R. Stinson, ed., Springer-Verlag, Berlin, 1993, pp. 232–249.

[11] M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: The Three Party Case. *Proc*. 27*th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1995, pp. 57–66.

[12] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5 (1993), pp. 679–693.

[13] C. Boyd. On Key Agreement and Conference Key Agreement. *Australasian Conference on Information Security and Privacy—ACISP* '97, LNCS vol. 1270, V. Varadharajan, J. Pieprzyk, and Y. Mu, eds., Springer-Verlag, Berlin, 1997, pp. 294–302.

[14] C. Boyd and J.M.G. Nieto. Round-Optimal Contributory Conference Key Agreement. *Public-Key Cryptography*, LNCS vol. 2567, Y. Desmedt, ed., Springer-Verlag, Berlin, 2003, pp. 161–174.

[15] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie–Hellman Key Exchange—The Dynamic Case. *Advances in Cryptology—Asiacrypt* 2001, LNCS vol. 2248, C. Boyd, ed., Springer-Verlag, Berlin, 2001, pp. 290–309.

[16] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie–Hellman Key Exchange under Standard Assumptions. *Advances in Cryptology—Eurocrypt* 2002, LNCS vol. 2332, L. Knudsen, ed., Springer-Verlag, Berlin, 2002, pp. 321–336.

[17] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie–Hellman Key Exchange. *Proc*. 8*th Annual ACM Conference on Computer and Communications Security*, ACM, New York, 2001, pp. 255–264.

[18] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. Pre-proceedings of Eurocrypt '94, Scuola Superiore Guilielmo Reiss Romoli, 1994, pp. 279–290. Available at http://www.cs.fsu.edu/∼burmeste/pubs.html.

[19] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. *Advances in Cryptology—Eurocrypt* '94, LNCS vol. 950, A. De Santis, ed., Springer-Verlag, Berlin, 1995, pp. 275–286.

[20] M. Burmester and Y. Desmedt. A Secure and Scalable Group Key Exchange System. *Information Processing Letters*, vol. 94, no. 3 (2005), pp. 137–143.

[21] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *Proc*. 30*th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1998, pp. 209–218.

[22] R. Canetti and H. Krawczyk. Key-Exchange Protocols and Their Use for Building Secure Channels. *Advances in Cryptology—Eurocrypt* 2001, LNCS vol. 2045, B. Pfitzmann, ed., Springer-Verlag, Berlin, 2001, pp. 453–474.

[23] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Advances in Cryptology—Eurocrypt* 2002, LNCS vol. 2332, L. Knudsen, ed., Springer-Verlag, Berlin, 2002, pp. 337–351.

[24] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. *Advances in Cryptology—Crypto* 2002, LNCS vol. 2442, M. Yung, ed., Springer-Verlag, Berlin, 2002, pp. 143–161.

[25] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. 22, no. 6 (1976), pp. 644–654.

[26] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, vol. 2, no. 2 (1992), pp. 107–125.

[27] M. Fischer, N. Lynch, and M. Patterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, vol. 32, no. 2 (1985), pp. 374–382.

[28] I. Ingemarsson, D.T. Tang, and C.K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, vol. 28, no. 5 (1982), pp. 714–720.

[29] M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. *Advances in Cryptology—Asiacrypt* 1996, LNCS vol. 1163, K. Kim and T. Matsumoto, eds., Springer-Verlag, Berlin, 1996, pp. 36–49.

[30] J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-Only Key-Exchange Protocols. *Security in Communication Networks* (*SCN* 2002), LNCS vol. 2576, S. Cimato, C. Galdi, and G. Persiano, eds., Springer-Verlag, Berlin, 2002, pp. 29–44.

[31] J. Katz and J.S. Shin. Modeling Insider Attacks on Group Key Exchange Protocols. *Proc*. 12*th ACM Conference on Computer and Communications Security*, ACM, New York, 2005, pp. 180–189.

[32] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. *Advances in Cryptology—Crypto* 2003, LNCS vol. 2729, Dan Boneh, ed., Springer-Verlag, Berlin, 2003, pp. 110–125.

[33] Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. *Proc*. 7*th Annual ACM Conference on Computer and Communication Security*, ACM, New York, 2000, pp. 235–244.

[34] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. *Proc*. *IFIP TC*11 16*th Annual Working Conference on Information Security* (*IFIP/SEC*), Kluwer, Dordrecht, 2001, pp. 229–244.

[35] H. Krawczyk. SKEME: A Versatile Secure Key-Exchange Mechanism for the Internet. *Proc*. *Internet Society Symposium on Network and Distributed System Security*, Feb. 1996, pp. 114–127.

[36] A. Mayer and M. Yung. Secure Protocol Transformation via "Expansion": From Two-Party to Groups. *Proc*. 6*th ACM Conference on Computer and Communication Security*, ACM, New York, 1999, pp. 83–92.

[37] O. Pereira and J.-J. Quisquater. Some Attacks Upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, vol. 11, no. 4 (2003), pp. 555–580.

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

[39] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A Secure Audio Teleconference System. *Advances in Cryptology—Crypto* '98, LNCS vol. 403, Springer-Verlag, Berlin, 1990, pp. 520–528.

[40] M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8 (2000), pp. 769–780.

[41] W.-G. Tzeng. A Practical and Secure Fault-Tolerant Conference Key Agreement Protocol. *Public-Key Cryptography*, LNCS vol. 1751, H. Imai and Y. Zheng, eds., Springer-Verlag, Berlin, 2000, pp. 1–13.

[42] W.-G. Tzeng and Z.-J. Tzeng. Round Efficient Conference Key Agreement Protocols with Provable Security. *Advances in Cryptology—Asiacrypt* 2000, LNCS vol. 1976, T. Okamoto, ed., Springer-Verlag, Berlin, 2000, pp. 614–628.