One-Round Protocols for Two-Party Authenticated Key Exchange

Ik Rae Jeong* Jonathan Katz† Dong Hoon Lee*

July 15, 2004

Abstract

Cryptographic protocol design in a two-party setting has often ignored the possibility of *simultaneous* message transmission by each of the two parties (i.e., using a duplex channel). In particular, most protocols for two-party key exchange have been designed assuming that parties alternate sending their messages (i.e., assuming a bidirectional half-duplex channel). However, by taking advantage of the communication characteristics of the network it may be possible to design protocols with improved latency. This is the focus of the present work.

We present three provably-secure protocols for two-party authenticated key exchange (AKE) which require only a single round. Our first, most efficient protocol provides *key independence* but not forward secrecy. Our second scheme additionally provides *forward secrecy* but requires some additional computation. Security of these two protocols is analyzed in the random oracle model. Our final protocol provides the same strong security guarantees as our second protocol, but is proven secure in the standard model. This scheme is only slightly less efficient (from a computational perspective) than the previous ones. Our work provides the first provably-secure *one-round* protocols for two-party AKE which achieve forward secrecy.

Keywords: Authenticated key exchange, Forward secrecy, Round complexity, Diffie-Hellman key exchange.

1 Introduction

Key-exchange protocols are among the most basic and widely used cryptographic protocols. Such protocols are used to derive a common *session key* between two (or more) parties; this session key may then be used to communicate securely over an insecure public network. Thus, secure key-exchange protocols serve as basic building blocks for constructing secure, complex, higher-level protocols. For this reason, the computational efficiency, communication efficiency, and round complexity of key-exchange protocols are very important and have received much attention, both in the two-party [17, 22, 6, 18, 5, 4, 7, 8, 15] and multi-party (i.e., group) [19, 14, 26, 20, 2, 13, 10, 21] settings.

This paper concerns protocols for *authenticated* key exchange (AKE); achieving such authentication is only possible if some out-of-band initialization phase is assumed prior to execution of

*Center for Information Security Technologies (CIST), Korea University, Seoul, Korea. jir@cist.korea.ac.kr, donghlee@korea.ac.kr.

†Dept. of Computer Science, University of Maryland, College Park, MD, USA. jkatz@cs.umd.edu.

the protocol. One common assumption is that each communicating party has an associated public-/private-key pair, with the public key known to all other parties in the network (of course, this includes the adversary). We assume this model here.

Most protocols for two-party key exchange have been designed and analyzed assuming that parties alternate sending messages (equivalently, that the parties communicate over a bidirectional *half-duplex* channel). However, in many common scenarios parties can actually transmit messages simultaneously (i.e., they have access to a bidirectional *duplex* channel). Of course, any key-exchange protocol designed and proven secure in the former model will also be secure in the latter model[1]; however, it may be possible to design protocols with improved round complexity by fully exploiting the communication characteristics of the underlying network, and in particular the possibility of simultaneous message transmission.

As a simple example, consider the traditional Diffie-Hellman key-exchange protocol [17] which does *not* provide any authentication. Traditionally, this is described as a two-round protocol in which Alice first sends g^a and Bob then replies with g^b. However, in this particular case Alice and Bob can send their messages simultaneously, thereby "collapsing" this protocol to a single round. However, the situation is more complex when authentication is required. For instance, *authenticated* Diffie-Hellman key exchange typically involves one party signing messages sent by the other party; this may be viewed as a type of "challenge-response" mechanism. (For example, the work of Bellare, et al. [5, 4] suggests implementing "authenticated channels" in exactly this way.) When this is done, it is no longer possible to collapse the protocol to a single round.

Motivated by the above discussion, we explore the possibility of designing protocols for authenticated key exchange which can be implemented in only a single round (assuming simultaneous message transmission). Of course, we will also ensure that our protocols are efficient with respect to other measures, including communication complexity and computational efficiency.

1.1 Our Work in Relation to Prior Work

Before relating our work to prior work in this area, we briefly recall some of the various notions of security for key exchange protocols (formal definitions are given in Section 2.3). At the most basic level, an authenticated key-exchange scheme must provide secrecy of a generated session key. Yet to completely define a notion of security, we must define the class of adversarial behavior tolerated by the protocol. The minimum requirement is that a protocol should ensure secrecy of session keys for an adversary who passively eavesdrops on protocol executions and may also send messages of its choice to the various parties. A stronger notion of security (and the one that is perhaps most often considered in the cryptographic literature) is *key independence*, which means that session keys are computationally independent from each other. A bit more formally, key independence protects against "Denning-Sacco" attacks [16] involving compromise of multiple session keys for sessions other than the one whose secrecy is being considered. Lastly, protocols achieving *forward secrecy* [18] maintain secrecy of session keys even when an adversary is able to obtain long-term secret keys of principals who have previously generated a common session key (in an honest execution of the protocol, without any interference by the adversary).

Key exchange protocols may also be required to provide some form of authentication. We distinguish *implicit authentication*, whereby *only* the intended partner of a particular party A knows the session key held by A, and *explicit authentication*, whereby A knows that its intended partner indeed holds a matching session key. (See [5] for further discussion and formal definitions.)

[1] We stress, however, that this is not necessarily the case for 2-party protocols where one of the parties may be malicious.

	Boyd-Nieto (see note)	$\mathcal{TS}1$
Modular exponentiations (per party)	2	1
Communication (total)	$2\|p\| + \|q\|$	$2\|q\|$
Security	KI	KI
Assumptions	CDH in random oracle model	CDH in random oracle model

Table 1: Comparison of the Boyd-Nieto scheme [10] to $\mathcal{TS}1$. Efficiency of the Boyd-Nieto scheme depends on the instantiation of its generic components; the above are rough estimates assuming the random oracle model and "discrete-log-based" components using an order-q subgroup of $\mathbb{Z}_p^*$.

We note that it is also possible to convert a protocol providing implicit authentication to one providing explicit authentication, at the cost of one additional round, by adding a "challenge-response" exchange at the end of the protocol.

The original two-party key-exchange scheme of Diffie and Hellman [17] is secure against passive eavesdroppers, but not against active attacks; indeed, that protocol provides no authentication at all. Several variations of the scheme have been suggested to provide security against active attacks [22, 23, 24, 8]. There are only a few provably secure schemes in the literature which provide both key independence and forward secrecy. Most such schemes seem to be "overloaded" so as to provide explicit authentication as well. (For example, the schemes of [1, 7, 4] use signatures and/or message authentication codes to authenticate messages in a way that achieves explicit authentication.) However, in some cases explicit authentication may be unnecessary, or may be provided anyway by subsequent communication. Thus, one may wonder whether more efficient protocols (say, with reduced round complexity) are possible if explicit authentication is not a requirement.

We first propose and analyze a very simple one-round scheme, $\mathcal{TS}1$, which provides key independence but not forward secrecy, based on the computational Diffie-Hellman assumption in the random oracle model. In Table 1 we compare our scheme to the scheme of Boyd and Nieto [10] which achieves the same level of security in the same number of rounds. (Boyd and Nieto actually propose a protocol for group AKE, but their protocol can be suitable modified for the case of two parties.) Our scheme is (slightly) more efficient than the scheme of Boyd and Nieto and has other advantages as well: our protocol is simpler and is also symmetric with respect to the two parties.

We next propose a modification of this scheme, $\mathcal{TS}2$, which provides both key independence and forward secrecy yet still requires only a single round of communication (security is again proved based on the CDH assumption in the random oracle model). We are not aware of any previous one-round protocol achieving this level of security. $\mathcal{TS}2$ requires only 3 modular exponentiations per party and uses neither key confirmation nor digital signatures, and hence the protocol is more efficient than previous schemes in terms of computation and communication as well. A drawback of $\mathcal{TS}2$ is that its security is analyzed in the random oracle model. For this reason, we propose a third protocol, $\mathcal{TS}3$, which provides the same level of security in the same number of rounds but whose security can be analyzed in the standard model based on the stronger, but still standard, *decisional* Diffie-Hellman assumption. This protocol is only slightly less efficient than $\mathcal{TS}2$ (it uses message authentication codes, whose efficiency is negligible compared to modular exponentiations). We compare both of these protocols to previous work in Table 2.

	[1, 7]	Auth. DH (cf. [4])	Auth. DH (cf. [21])	$\mathcal{TS}2$	$\mathcal{TS}3$
Modular exponentiations (par party)	3	4	4	3	3
Rounds	3	3	2	1	1
Communication (total)	$2\|p\| + 2\|q\|$	$4\|p\|$	$4\|p\| + 2\|q\|$	$2\|p\|$	$2\|p\| + 2\|q\|$
Model	random oracle	standard	standard	random oracle	standard

Table 2: Comparison of key-exchange protocols achieving key independence and forward secrecy. Efficiency of some schemes depends on instantiation details; the above represent rough estimates assuming "discrete-log-based" instantiations using an order-q subgroup of $\mathbb{Z}_p^*$.

2 Preliminaries

2.1 The Computational Diffie-Hellman Problem

Let $\mathcal{GG}$ be an algorithm which on input 1^k outputs a (description of a) group G of prime order q (with $|q| = k$) along with a generator $g \in G$. The *computational Diffie-Hellman (CDH) problem* is the following: given g^{u_1}, g^{u_2} for random $u_1, u_2 \in \mathbb{Z}_q$, compute $g^{u_1 u_2}$. We say that $\mathcal{GG}$ satisfies the CDH assumption if this problem is infeasible for all PPT algorithms. More formally, for any PPT algorithm $\mathcal{A}$ consider the following experiment:

$\mathbf{Exp}^{\mathsf{cdh}}_{\mathcal{A},\mathcal{GG}}(k)$
$(G, q, g) \leftarrow \mathcal{GG}(1^k)$
$u_1, u_2 \leftarrow \mathbb{Z}_q$
$U_1 = g^{u_1}; U_2 = g^{u_2}$
$W \leftarrow \mathcal{A}(G, q, g, U_1, U_2)$
if $W = g^{u_1 u_2}$ return 1
else return 0

The advantage of an adversary $\mathcal{A}$ is defined as follows:

$$\mathsf{Adv}^{\mathsf{cdh}}_{\mathcal{A},\mathcal{GG}}(k) \stackrel{\text{def}}{=} \Pr\left[\mathbf{Exp}^{\mathsf{cdh}}_{\mathcal{A},\mathcal{GG}}(k) = 1\right].$$

We say that $\mathcal{GG}$ *satisfies the CDH assumption* if $\mathsf{Adv}^{\mathsf{cdh}}_{\mathcal{A},\mathcal{GG}}(k)$ is negligible for all PPT algorithms $\mathcal{A}$. When we are interested in a concrete security analysis, we drop the dependence on k and say that $\mathcal{GG}$ is (t, ϵ)*-secure with respect to the CDH problem* if $\mathsf{Adv}^{\mathsf{cdh}}_{\mathcal{A},\mathcal{GG}} \leq \epsilon$ for all $\mathcal{A}$ running in time at most t. (We will sometimes be informal and say that a group G output by $\mathcal{GG}$ satisfies the CDH assumption.)

2.2 The Decisional Diffie-Hellman Problem

Let $\mathcal{GG}$ be as in the previous section. The *decisional Diffie-Hellman assumption* is the following: given g^{u_1}, g^{u_2} for random $u_1, u_2 \in \mathbb{Z}_q$, distinguish between $g^{u_1 u_2}$ and a random group element. We say that $\mathcal{GG}$ satisfies the DDH assumption if this problem is infeasible for all PPT algorithms. More formally, for any PPT algorithm $\mathcal{A}$ consider the following experiment:

$\mathbf{Exp}^{\mathsf{ddh}}_{\mathcal{A},\mathcal{GG}}(k)$
$(G,q,g) \leftarrow \mathcal{GG}(1^k)$
$u_1, u_2 \leftarrow \mathbb{Z}_q$
$U_1 = g^{u_1}; U_2 = g^{u_2}$
$V_0 = g^{u_1 u_2}; V_1 \leftarrow G$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}(G,q,g,U_1,U_2,V_b)$
if $b' = b$ return 1
else return 0

The advantage of an adversary $\mathcal{A}$ is defined as follows:

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{A},\mathcal{GG}}(k) \stackrel{\text{def}}{=} \left| 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{ddh}}_{\mathcal{A},\mathcal{GG}}(k) = 1\right] - 1 \right|.$$

We say that $\mathcal{GG}$ *satisfies the DDH assumption* if $\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{A},\mathcal{GG}}(k)$ is negligible for all PPT algorithms $\mathcal{A}$. When we are interested in a concrete security analysis, we drop the dependence on k and say that $\mathcal{GG}$ is (t,ϵ)-*secure with respect to the DDH problem* if $\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{A},\mathcal{GG}} \leq \epsilon$ for all $\mathcal{A}$ running in time at most t. (We will sometimes be informal and say that a group G output by $\mathcal{GG}$ satisfies the DDH assumption.)

2.3 Message Authentication Codes

A message authentication code (MAC) consists of two algorithms $(\mathsf{Mac}, \mathsf{Vrfy})$. Given a random key sk, MAC computes a tag τ for a message M; we write this as $\tau \leftarrow \mathsf{Mac}_{sk}(M)$. Vrfy verifies the message-tag pair using the (shared) key, and returns 1 if the tag is valid or 0 otherwise. We require that for all keys sk, for all M, and for all τ output by $\mathsf{Mac}_{sk}(M)$ we have $\mathsf{Vrfy}_{sk}(M,\tau) = 1$.

In defining the security of a MAC we use the standard definition of strong unforgeability under adaptive chosen-message attack. Namely, let M be a MAC scheme and $\mathcal{A}$ be an adversary, and consider the following experiment:

$\mathbf{Exp}^{\mathsf{suf}}_{\mathcal{A},M}(k)$
$sk \leftarrow \{0,1\}^k$
$(M,\tau) \leftarrow \mathcal{A}^{\mathsf{Mac}_{sk}(\cdot)}(1^k)$
if $\mathsf{Vrfy}_{sk}(M,\tau) = 1$ and oracle $\mathsf{Mac}_{sk}(\cdot)$
  never returned τ on input M then return 1
else return 0

The advantage of an adversary $\mathcal{A}$ is defined as: $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},M}(k) \stackrel{\text{def}}{=} \Pr\left[\mathbf{Exp}^{\mathsf{suf}}_{\mathcal{A},M}(k) = 1\right]$. We say that M is strongly unforgeable (SUF-secure) if $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},M}(k)$ is negligible for all PPT algorithms $\mathcal{A}$. When we are interested in a concrete security analysis, we drop the dependence on k and say that M is (t,q,ϵ)-*SUF-secure* if $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},M} \leq \epsilon$ for all $\mathcal{A}$ running in time t and making at most q queries to its Mac oracle. (We remark that allowing N queries to an oracle $\mathsf{Vrfy}_{sk}(\cdot,\cdot)$ cannot increase the advantage of an adversary by more than a factor of N.)

3 Security Model for Authenticated Key Exchange

We use the standard notion of security as defined in [5], taking into account forward secrecy following [18]. We assume that there are N parties, and each party's identity is denoted as P_i. Each party

P_i holds a pair of private and public keys, where the public key is assumed to be known to all other parties in the network (and the adversary, too). We consider key-exchange protocols in which two parties want to exchange a session key using their public keys to provide authentication. Π_i^k represents the k-th instance of player P_i. If a key-exchange protocol terminates, then Π_i^k generates a session key sk_i^k. A session identifier of an instance, denoted sid_i^k, is a string different from those of all other sessions in the system (with high probability). Without loss of generality, we assume that sid_i^k is simply the concatenation of all messages sent and received by a particular instance Π_i^k, where the order of these messages is determined *by the lexicographic ordering of the two parties' identities*. (Note that ordering messages according to the time they were sent cannot be used in our setting, because the two parties may send their messages simultaneously.)

Consider instance Π_i^k of player P_i. The *partner* of this instance is the player with whom P_i believes it is interacting. We say that two instances Π_i^k and $\Pi_j^{k'}$ are *partnered* if $\mathsf{sid}_i^k = \mathsf{sid}_i^{k'}$, P_j is the partner of Π_i^k, and P_i is the partner of $\Pi_j^{k'}$. Any protocol should satisfy the following correctness condition: two partnered instances compute the same session key.

To define a notion of security, we define the capabilities of an adversary. We allow the adversary to potentially control all communication in the network via access to a set of oracles as defined below. We consider an *experiment* in which the adversary asks queries to oracles, and the oracles answer back to the adversary. Oracle queries model attacks which an adversary may use in the real system. We consider the following types of queries in this paper.

- The query $\mathsf{Initiate}(i, j)$ is used to "prompt" party P_i to initiate execution of the protocol with partner P_j. This query will result in P_i sending a message, which is given to the adversary.
- A query $\mathsf{Send}(i, k, M)$ is used to send a message M to instance Π_i^k (this models active attacks on the part of the adversary). When Π_i^k receives M, it responds according to the key-exchange protocol.
- A query $\mathsf{Execute}(i, j)$ represents passive eavesdropping of the adversary on an execution of the protocol by parties P_i and P_j. In response to this query, parties P_i and P_j execute the protocol without any interference from the adversary, and the adversary is given the resulting transcript of the execution. (Although the actions of the Execute query can be simulated via repeated Initiate and Send oracle queries, this particular query is needed to distinguish between passive and active attacks in the definition of forward secrecy.)
- A query $\mathsf{Reveal}(i, k)$ models *known key* attacks (or Denning-Sacco attacks) in the real system. The adversary is given the session key sk_i^k for the specified instance.
- A query $\mathsf{Corrupt}(P_i)$ models exposure of the long-term key held by player P_i. The adversary is assumed to be able to obtain long-term keys of players, but cannot control the behavior of these players directly (of course, once the adversary has asked a query $\mathsf{Corrupt}(P_i)$, the adversary may impersonate P_i in subsequent Send queries.)
- A query $\mathsf{Test}(i, k)$ is used to define the advantage of an adversary. When an adversary $\mathcal{A}$ asks a Test query to an instance Π_i^k, a coin b is flipped. If b is 1, then the session key sk_i^k is returned. Otherwise, a random session key (i.e., one chosen uniformly from the space of session keys) is returned. The adversary is allowed to make a single Test query to a *fresh* instance (see below), at any time during the experiment.

To define a meaningful notion of security, we must first define *freshness*:

Definition 1 An instance Π_i^k (partnered with instance $\Pi_j^{k'}$) is *fresh* if the following conditions are true at the conclusion of the experiment described above:

(a) The adversary has not queried $\mathsf{Reveal}(i, k)$ or $\mathsf{Reveal}(j, k')$.

(b) If the adversary has queried $\mathsf{Corrupt}(P_i)$ or $\mathsf{Corrupt}(P_j)$, then it has not queried $\mathsf{Send}(i, k, *)$ or $\mathsf{Send}(j, k', *)$.[2]

In all the notions of security considered below, the adversary $\mathcal{A}$ outputs a bit b' at the end of the experiment above. The advantage of $\mathcal{A}$, denoted $\mathsf{Adv}_{\mathcal{A}}(k)$, is defined as $|2 \cdot \Pr[b' = b] - 1|$. Generically speaking, a protocol is called "secure" if the advantage of any PPT adversary is negligible. The following notions of security may then be considered, depending on the types of queries the adversary is allowed to ask:

(1) The most basic level of security does not allow adversary $\mathcal{A}$ to make any Reveal or $\mathsf{Corrupt}$ queries.

(2) KI (Key Independence): An adversary $\mathcal{A}$ can ask Reveal queries, but can not ask $\mathsf{Corrupt}$ queries.

(3) FS (Forward Secrecy): An adversary $\mathcal{A}$ is now allowed to ask any of the above queries (i.e., including $\mathsf{Corrupt}$).

Note that forward secrecy implies key independence.

For an adversary $\mathcal{A}$ attacking a scheme in the sense of XX (where XX is either KI or FS), we denote the advantage of this adversary by $\mathsf{Adv}_{\mathcal{A}}^{XX}(k)$. For a protocol P, we define its security as:

$$\mathsf{Adv}_P^{XX}(k, t) = \max_{\mathcal{A}}\{\mathsf{Adv}_{\mathcal{A}}^{XX}(k)\},$$

where the maximum is taken over all adversaries running in time t. A scheme P is said to be XX-secure if $\mathsf{Adv}_P^{XX}(k, t)$ is negligible (in k) for any $t = \mathrm{poly}(k)$.

4 One-Round Protocols for Authenticated Key Exchange

For each of the protocols we present, we assume that parties can be ordered by their names (e.g., lexicographically) and write $P_i < P_j$ to denote this ordering. Let k be a security parameter, and let G be a group of prime order q (where $|q| = k$) with generator g. Let H be a hash function such that $H : \{0,1\}^* \to \{0,1\}^k$. (We assume that G, q, g, and H are fixed in advance and known to the entire network.) We assume that each party P_i has a public-/private-key pair $(y_i = g^{x_i}, x_i)$, and that the public keys of all parties are known to all other parties in the network (and the adversary as well). Recall that the standard definition of security (discussed above) does not include the possibility of "malicious insiders"; thus, in particular, we assume that all public-/secret-keys are honestly generated.

[2] Note that this is a slightly weaker definition of forward secrecy than has sometimes been considered, in which the instance is considered fresh as long as the adversary has not queried $\mathsf{Send}(i, k, *)$ or $\mathsf{Send}(j, k', *)$ *after* a query $\mathsf{Corrupt}(P_i)$ or $\mathsf{Corrupt}(P_j)$.

4.1 Protocol $\mathcal{TS}1$

We now present our first protocol $\mathcal{TS}1$. The protocol is described from the perspective of P_i, but its partner P_j behaves analogously (i.e., the protocol is symmetric):

$\mathcal{TS}1$

Setup: Assume P_i wants to establish a session key with P_j, and $P_i < P_j$.

Round 1: P_i selects a random number $r_i \in_R \{0,1\}^k$, transmits it, and receives a random r_j which is supposedly from P_j.

Computation of session key: P_i forms a session identifier by concatenating the messages according to the ordering of P_i, P_j; i.e., it sets $\mathsf{sid}_i = r_i||r_j$. Party P_i computes the session key $sk_i = H(i||j||\mathsf{sid}_i||y_j^{x_i})$.

An example of an execution of $\mathcal{TS}1$ is shown in Fig. 1 (which assumes $P_1 < P_2$). The following theorem states the security achieved by this protocol.

	$P_1(x_1, y_2)$	$P_2(x_2, y_1)$
Round 1	r_1	r_2

$\mathsf{sid} = r_1||r_2$

$\mathsf{sk} = H(1||2||\mathsf{sid}||g^{x_1x_2}) = H(1||2||\mathsf{sid}||y_2^{x_1}) = H(1||2||\mathsf{sid}||y_1^{x_2})$

Figure 1: An example of an execution of $\mathcal{TS}1$.

Theorem 1 *Assuming G satisfies the CDH assumption, $\mathcal{TS}1$ is a KI-secure key-exchange protocol when H is modeled as a random oracle. Concretely, if G is generated by $\mathcal{GG}$ which is (t,ϵ)-secure with respect to the CDH problem, then*

$$\mathsf{Adv}^{KI}_{\mathcal{TS}1}(k,t,q_{re},q_h) \quad \leq \quad q_q N^2 \cdot \epsilon + \frac{q_s^2}{2^k},$$

where t is the maximum total experiment time including the adversary's execution time, and the adversary makes q_{re} Reveal queries and q_h hash queries. Here, N is an upper bound on the number of parties, and q_s is an upper bound on the number of the sessions in the experiment.

Proof of Theorem 1.
Consider an adversary $\mathcal{A}$ attacking $\mathcal{TS}1$ in the sense of key independence. Informally, and using the fact that we work in the random oracle model, there are only two ways an adversary can get information about a particular session key $sk_i^k = H(i||j||\mathsf{sid}_i^k||g^{x_ix_j})$ for a fresh instance: either the adversary queries the random oracle on the point $i||j||\mathsf{sid}_i^k||g^{x_ix_j}$, or the value sid_i^k has repeated (for the same pair of users) at some point during the experiment. The latter case happens with probability upper-bounded by $\frac{q_s^2}{2^k}$, while the former case allows us to solve the computational Diffie-Hellman problem with probability related to that of the adversary's success probability. We now proceed with a more formal proof.

Let col denote the event that a value of sid repeats at some point during the experiment, and let query be the event that, for some $i, j \in [N]$, the adversary at some point makes an oracle query $H(i||j|| * ||W)$ with $W = g^{x_i x_j}$. Using the fact that $\Pr_{\mathcal{A}}[b' = b \mid \overline{\mathsf{query}} \wedge \overline{\mathsf{col}}] = \frac{1}{2}$, we may write

$$\begin{aligned}
\left|\Pr_{\mathcal{A}}[b' = b] - \tfrac{1}{2}\right| &= \big|\Pr_{\mathcal{A}}[b' = b \wedge \mathsf{col}] + \Pr_{\mathcal{A}}[b' = b \wedge \mathsf{query} \wedge \overline{\mathsf{col}}] \\
&\quad + \Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{query}} \wedge \overline{\mathsf{col}}] - \tfrac{1}{2}\big| \\
&= \big|\Pr_{\mathcal{A}}[b' = b \mid \mathsf{col}] \cdot \Pr_{\mathcal{A}}[\mathsf{col}] + \Pr_{\mathcal{A}}[b' = b \mid \mathsf{query} \wedge \overline{\mathsf{col}}] \cdot \Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}] \\
&\quad + \tfrac{1}{2} \cdot \Pr_{\mathcal{A}}[\overline{\mathsf{query}} \wedge \overline{\mathsf{col}}] - \tfrac{1}{2}\big| \\
&= \big|\Pr_{\mathcal{A}}[b' = b \mid \mathsf{col}] \cdot \Pr_{\mathcal{A}}[\mathsf{col}] + \Pr_{\mathcal{A}}[b' = b \mid \mathsf{query} \wedge \overline{\mathsf{col}}] \cdot \Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}] \\
&\quad + \tfrac{1}{2} \cdot \big(1 - \Pr_{\mathcal{A}}[\mathsf{col}] - \Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}]\big) - \tfrac{1}{2}\big| \\
&\leq \tfrac{1}{2} \cdot \big(\Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}] + \Pr_{\mathcal{A}}[\mathsf{col}]\big) . \qquad (1)
\end{aligned}$$

Now, as noted previously, $\Pr_{\mathcal{A}}[\mathsf{col}]$ is bounded from above by $q_s^2/2^k$ by a "birthday problem" calculation, since for this event to occur two random nonces of length k generated by some player(s) in separate instances must repeat. We now bound the first term in Eq. (1).

Consider the following algorithm $\mathcal{F}$ which attempts to solve the CDH problem using $\mathcal{A}$ as a subroutine. $\mathcal{F}$ is given $(U_1 = g^{u_1}, U_2 = g^{u_2})$, an instance of the CDH problem. $\mathcal{F}$ randomly selects two parties and uses U_1 and U_2 as their public keys. $\mathcal{F}$ simulates the random oracle H, and tries to find $g^{u_1 u_2}$ from the hash queries made by $\mathcal{A}$. A complete description of $\mathcal{F}$ follows:

1. $\mathcal{F}$ is given $U_1, U_2 \in G$. It begins by selecting random i^* and j^* from $\{1, \ldots, N\}$, and letting U_1 and U_2 be the public keys for P_{i^*} and P_{j^*}, respectively. Assume $i^* < j^*$. Public keys of other players are chosen in the specified way.

2. $\mathcal{F}$ then runs $\mathcal{A}$ (giving it the vector of public keys for all N parties). The oracle queries of $\mathcal{A}$ are answered as follows:

   - For queries $H(i||j||\mathsf{sid}||W)$ return a random value $v \in \{0,1\}^k$. If $i = i^*$ and $j = j^*$, store W in a list $\mathsf{dh\text{-}tuples}$. (We assume $\mathcal{A}$ does not make the same query to H twice.)
   - For queries of the form $\mathsf{Initiate}(i, j)$, first choose a random nonce $r_i \in \{0,1\}^k$. If this nonce has been chosen before by any party, abort. Otherwise, return r_i to $\mathcal{A}$.
   - After a query of the form $\mathsf{Send}(i, k, M)$, simply set the session key sk_i^k equal to a random value in $\{0,1\}^k$.
   - For queries $\mathsf{Execute}(i, j)$ choose random $r_i, r_j \in \{0,1\}^k$ and abort if either of these values has been chosen as a nonce before (by any party). Otherwise, set $\mathsf{sk}_i^k = \mathsf{sk}_j^{k'}$ equal to a random value in $\{0,1\}^k$.
   - Queries of the form $\mathsf{Reveal}(i, k)$ and $\mathsf{Test}(i, k)$ are answered in the correct way.

3. Once the experiment has concluded (i.e., $\mathcal{A}$ is done) choose a random element in $\mathsf{dh\text{-}tuples}$ and output it.

The probability that $\mathcal{F}$ returns the correct answer is at least $\Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}]/N^2 q_h$, since the simulation is perfect until the point, if any, that query occurs. Furthermore, since the running time of $\mathcal{F}$ is essentially the same as the running time of $\mathcal{A}$ we must have $\Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{col}}] \leq q_h N^2 \epsilon$. This concludes the proof of the theorem. ■

We remark that the concrete security reduction in Theorem 1 can be improved using the random self-reducibility of the Diffie-Hellman problem.

4.2 Protocol $\mathcal{TS}2$

It is easy to see that $\mathcal{TS}1$ does not provide forward secrecy. Forward secrecy can be achieved by adding an ephemeral Diffie-Hellman key exchange to $\mathcal{TS}1$. The resulting protocol, $\mathcal{TS}2$, is described below, again from the point of view of player P_i wanting to exchange a key with player P_j (and P_j acts symmetrically):

$\mathcal{TS}2$

Setup: Same as in $\mathcal{TS}1$.

Round 1: P_i selects a random number $\alpha_i \in_R \mathbb{Z}_q$ and sends g^{α_i} to the other party.

Computation of session key: P_i forms a session identifier by concatenating the messages according to the ordering of P_i, P_j; i.e., it sets $\mathsf{sid}_i = g^{\alpha_i}||g^{\alpha_j}$. P_i computes the session key $\mathsf{sk}_i = H(i||j||\mathsf{sid}_i||(g^{\alpha_j})^{\alpha_i}||y_j^{x_i})$.

	$P_1(x_1, y_2)$	$P_2(x_2, y_1)$
Round 1	g^{α_1}	g^{α_2}

$\mathsf{sid} = g^{\alpha_1}||g^{\alpha_2}$

$\mathsf{sk} = H(1||2||\mathsf{sid}||g^{\alpha_1\alpha_2}||g^{x_1x_2})$

Figure 2: An example of an execution of $\mathcal{TS}2$.

An example of an execution of $\mathcal{TS}2$ is shown in Fig. 2 (where we assume $P_1 < P_2$). The following characterizes the security of $\mathcal{TS}2$.

Theorem 2 *Assuming G satisfies the CDH assumption, $\mathcal{TS}2$ is an FS-secure key-exchange protocol when H is modeled as a random oracle. Concretely, if G is generated by $\mathcal{GG}$ which is (t, ϵ)-secure with respect to the CDH problem, then*

$$\mathsf{Adv}^{FS}_{\mathcal{TS}2}(k, t, q_{re}, q_{co}, q_h) \quad \leq \quad q_h \cdot (N^2 + 1) \cdot \epsilon + \frac{q_s^2}{q},$$

where t is the maximum total experiment time including the adversary's execution time, and the adversary makes q_{re} Reveal queries, q_{co} Corrupt queries, and q_h hash queries. N is an upper bound of the number of parties, and q_s is the upper bound on the number of the sessions in the experiment.

Proof of Theorem 2.
Consider an adversary $\mathcal{A}$ attacking $\mathcal{TS}2$ in the sense of forward secrecy. Informally, and again using the fact that we work in the random oracle model, there are only two ways an adversary can get information about a particular session key $sk_i^k = H(i||j||\mathsf{sid}_i^k||g^{\alpha_i\alpha_j}||g^{x_ix_j})$: either the adversary queries the random oracle on the point $i||j||\mathsf{sid}_i^k||g^{\alpha_i\alpha_j}||g^{x_ix_j}$, or the value sid_i^k has repeated at some point during the experiment (for the same pair of users). The latter case happens with probability upper bounded by $\frac{q_s^2}{q}$ (where q is the size of the group G), while the former case allows us to solve the computational Diffie-Hellman problem with probability related to that of the adversary's success probability. We now proceed with a more formal proof.

Let col be the event that a value of sid repeats at some point during the experiment. Let $\mathsf{corrupt}$ be the event that the adversary makes its Test to a corrupted instance; i.e., that it asks query $\mathsf{Test}(i,k)$ and at some point during the experiment asks either $\mathsf{Corrupt}(P_i)$ or $\mathsf{Corrupt}(P_j)$ (where P_j is the partner of instance Π_i^k). (Note that, by definition of freshness, it must then be the case that the instance Π_i^k and its partner instance were initiated by an $\mathsf{Execute}$ query.) Let query_1 be the event that, for some $i,j \in [N]$, the adversary at some point queries the random oracle at a point $i||j||U||V||X||W$, neither P_i nor P_j are corrupted in the entire course of the experiment, and $W = g^{x_i x_j}$. Let query_2 be the event that, for some $i,j \in [N]$, the adversary at some point queries the random oracle at a point $i||j||U||V||X||W$ and, for some k, $\mathsf{sid}_i^k = U||V = g^{\alpha_i}||g^{\alpha_j}$, instance Π_i^k was initiated via a call $\mathsf{Execute}(i,j)$ (and hence $\mathsf{sid}_j^{k'} = U||V$ for some k' as well), and $X = g^{\alpha_i \alpha_j}$. Since

$$\Pr_{\mathcal{A}}[b' = b \mid \overline{\mathsf{col}} \wedge \overline{\mathsf{query}_1} \wedge \overline{\mathsf{corrupt}}] = \Pr_{\mathcal{A}}[b' = b \mid \overline{\mathsf{col}} \wedge \overline{\mathsf{query}_2} \wedge \mathsf{corrupt}] = \tfrac{1}{2},$$

we have:

$$|\Pr_{\mathcal{A}}[b' = b] - \tfrac{1}{2}| \;\leq\; \tfrac{1}{2} \cdot (\Pr_{\mathcal{A}}[\mathsf{col}] + \Pr_{\mathcal{A}}[\mathsf{query}_1 \wedge \overline{\mathsf{corrupt}}] + \Pr_{\mathcal{A}}[\mathsf{query}_2 \wedge \mathsf{corrupt}]) \qquad (2)$$

Now, as in the previous theorem, we have $\Pr_{\mathcal{A}}[\mathsf{col}] \leq q_s^2/q$. Furthermore, following exactly the same argument as in the previous theorem, we may show that $\Pr_{\mathcal{A}}[\mathsf{query}_1 \wedge \overline{\mathsf{corrupt}}] \leq q_h N^2 \epsilon$. We therefore concentrate on upper-bounding the last term of Eq. (2).

Consider the following algorithm $\mathcal{F}$ which attempts to solve the CDH problem using $\mathcal{A}$ as a subroutine. $\mathcal{F}$ is given $(U_1 = g^{u_1}, U_2 = g^{u_2})$, an instance of the CDH problem. $\mathcal{F}$ will "embed" this instance into all $\mathsf{Execute}$ oracle calls and we will use the random self-reducibility of the CDH problem to argue that in case query_2 occurs then $\mathcal{F}$ can solve the given CDH instance with probability at least $1/q_h$. Details follow.

1. $\mathcal{F}$ is given $U_1, U_2 \in G$. It begins by choosing public keys for all parties normally (i.e., choosing a random x_i and letting the public key of P_i be g^{x_i}).

2. $\mathcal{F}$ runs $\mathcal{A}$, answering its oracle queries as follows:

   - For queries $H(i||j||U||V||X||W)$, if this query was asked before then return the answer given previously. Otherwise, return a random value $v \in \{0,1\}^k$. Furthermore, if there is an instance Π_i^k with $\mathsf{sid}_i^k = U||V$ and this instance was initiated via a call $\mathsf{Execute}(i,j)$, then store (U,V,X) in a list $\mathsf{dh\text{-}tuples}$.
   - $\mathsf{Initiate}$, Send, Reveal, Test, and $\mathsf{Corrupt}$ queries are answered honestly (in particular, when a session key must be computed the appropriate random oracle query is answered as discussed above).
   - For queries $\mathsf{Execute}(i,j)$ proceed as follows: choose random $a,b \in \mathbb{Z}_q$ and return the transcript $(U_1 g^a || U_2 g^b)$. The session keys $\mathsf{sk}_i^k = \mathsf{sk}_j^{k'}$ are set equal to a random value in $\{0,1\}^k$.

3. Once the experiment has concluded (i.e., $\mathcal{A}$ is done), $\mathcal{F}$ chooses a random tuple (U,V,X) from its list $\mathsf{dh\text{-}tuples}$. It finds a,b such that $U = U_1 g^a$ and $V = U_2 g^b$ and outputs $X/U_2^a U_1^b g^{ab}$.

Note that the simulation is perfect until the point, if any, that query_2 occurs. Furthermore, in case query_2 occurs then with probability $1/q_h$ it is the case that $\mathcal{F}$ selects a tuple (U,V,X) for which $X = g^{\alpha_1 \alpha_2}$, where $\alpha_1 \stackrel{\text{def}}{=} \log_g U$ and $\alpha_2 \stackrel{\text{def}}{=} \log_g V$ (i.e., this tuple is a Diffie-Hellman

tuple). Assuming this is the case, we may write $X = g^{u_1 u_2} g^{ab} U_1^b U_2^a$ where $u_1 \stackrel{\text{def}}{=} \log_g U_1$ and $u_2 \stackrel{\text{def}}{=} \log_g U_2$; therefore, $\mathcal{F}$ indeed outputs a correct solution to its CDH instance. Thus, we see that $\Pr_{\mathcal{A}}[\mathsf{query}_2] \leq q_h \epsilon$. Plugging this into Eq. (2) gives the result of the theorem. ■

4.3 Protocol $\mathcal{TS}3$

The security of $\mathcal{TS}2$ (and $\mathcal{TS}1$, for that matter) is proven in the random oracle model. Next, we present protocol $\mathcal{TS}3$ which is proven secure in the standard model (but using the stronger DDH assumption):

$\mathcal{TS}3$

Setup: Same as in $\mathcal{TS}1$.

Round 1: P_i computes $k_{i,j} = y_j^{x_i}$ which it will use as a key for a message authentication code (of course, $k_{i,j}$ may need to be hashed before being used; we ignore this technicality here). Next, P_i chooses a random number $\alpha_i \in_R \mathbb{Z}_q$, computes $\tau_i \leftarrow \mathsf{Mac}_{k_{i,j}}(i||j||g^{\alpha_i})$, and sends $g^{\alpha_i}||\tau_i$ to the other party.

Computation of session key: P_i verifies the tag of the received message using $k_{i,j}$. If verification fails, no session key is computed. Otherwise, P_i computes a session key $\mathsf{sk}_i = (g^{\alpha_j})^{\alpha_i}$. The session identifier is $\mathsf{sid}_i = g^{\alpha_i}||\tau_i||g^{\alpha_j}||\tau_j$.

	$P_1(x_1)$	$P_2(x_2)$
Round 1	$g^{\alpha_1}\|\|\tau_1$	$g^{\alpha_2}\|\|\tau_2$

$k_{1,2} = g^{x_1 x_2}$

$\tau_1 \leftarrow \mathsf{Mac}_{k_{1,2}}(1||2||g^{\alpha_1}); \tau_2 \leftarrow \mathsf{Mac}_{k_{1,2}}(2||1||g^{\alpha_2})$

$\mathsf{sid} = g^{\alpha_1}||\tau_1||g^{\alpha_2}||\tau_2$

$\mathsf{sk} = g^{\alpha_1 \alpha_2}$

Figure 3: An example of an execution of $\mathcal{TS}3$.

An example of an execution of $\mathcal{TS}3$ is shown in Fig. 3. In the example we assume that $P_1 < P_2$. The following characterizes the security of $\mathcal{TS}3$.

Theorem 3 Assuming the MAC used above is SUF-secure and G satisfies the DDH assumption, $\mathcal{TS}3$ is an FS-secure key-exchange protocol. Concretely, if the MAC is (t, q_s, ϵ)-SUF-secure and G is generated by $\mathcal{GG}$ which is (t, ϵ')-secure with respect to the DDH problem, then

$$\mathsf{Adv}^{FS}_{\mathcal{TS}3}(k, t, q_{re}, q_{co}) \quad \leq \quad (4 + 2N^2 + 2q_s^2) \cdot \epsilon' + 2N^2 \cdot \epsilon + \frac{2q_s^2}{q},$$

where t is the maximum total experiment time including an adversary's execution time, and an adversary makes q_{re} Reveal queries and q_{co} $\mathsf{Corrupt}$ queries. N is an upper bound on the number of parties, and q_s is an upper bound of the number of the sessions an adversary initiates.

Proof of Theorem 3.
The intuition is as follows: under the DDH assumption, the adversary cannot have non-negligible advantage unless it can choose a value g^x for which it knows x, send this to an instance, and have that instance compute a session key using this value (i.e., if the instance sends $Y = g^\alpha$ then the adversary knows the computed session key Y^x). However, this cannot happen with more than negligible probability by the security of the MAC. Details follow.

Consider an adversary $\mathcal{A}$ attacking $\mathcal{TS}3$ in the sense of forward secrecy. Let col be the event that a value g^{α_i} is used twice (possibly by different parties), and let Ex be the event that the adversary makes its Test query to an instance that was initiated via an $\mathsf{Execute}$ query. Let Forge be the event that, for some instance Π_i^k with partner P_j, the adversary queries $\mathsf{Send}(i, k, M)$ and (1) neither P_i nor P_j were ever corrupted; (2) M was never sent by P_j (formally, was never output in response to a query $\mathsf{Initiate}(j, i)$ and never appeared in a transcript output by a query $\mathsf{Execute}(i, j)$); and (3) Π_i^k computes a valid session key. As usual, we may write:

$$\begin{aligned} |\Pr_{\mathcal{A}}[b' = b] - \tfrac{1}{2}| \;\leq\; & \Pr_{\mathcal{A}}[\mathsf{col}] + \Pr_{\mathcal{A}}[\mathsf{Forge}] + \Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{col}} \wedge \mathsf{Ex}] \\ & + \Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{col}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}]. \end{aligned}$$

As in the previous proofs, we have $\Pr[\mathsf{col}] \leq q_s^2/q$. We next upper-bound $\Pr_{\mathcal{A}}[\mathsf{Forge}]$:

Claim $\Pr_{\mathcal{A}}[\mathsf{Forge}] \leq N^2(\epsilon + \epsilon')$.

Proof of claim (sketch): Let $\mathsf{Forge}_{i,j}$ be the probability that Forge occurs for a specific pair of parties i, j. Clearly, we have $\Pr_{\mathcal{A}}[\mathsf{Forge}] \leq N^2 \Pr[\mathsf{Forge}_{i,j}]$. Now, if we replace the key $k_{i,j} = g^{x_i x_j}$ by a random element from G, we claim that this does not affect $\Pr_{\mathcal{A}}[\mathsf{Forge}_{i,j}]$ by more than ϵ' since we can embed an instance of the DDH problem in the public keys y_i, y_j of these players (note that $k_{i',j'}$ for all pairs of players $\{i', j'\} \neq \{i, j\}$ can be computed correctly, and the rest of the experiment can be carried out honestly). However, the probability that $\mathsf{Forge}_{i,j}$ occurs when $k_{i,j}$ is truly random is at most ϵ by the security of the MAC. The claim follows. ■

Next, we claim that $\Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{col}} \wedge \mathsf{Ex}] \leq 2\epsilon'$. To see this, consider an adversary $\mathcal{F}$ who embeds a DDH problem into all $\mathsf{Execute}$ queries as follows:

1. Given a tuple (g, X, Y, Z) as input, algorithm $\mathcal{F}$ can generate polynomially-many tuples (g, X_i, Y_i, Z_i) with the following properties: if (g, X, Y, Z) is a DDH tuple, then each tuple (g, X_i, Y_i, Z_i) is a random and independently-distributed DDH tuple. On the other hand, if the original input is a random tuple, then each tuple (g, X_i, Y_i, Z_i) is a random and independently-distributed random tuple. (This uses the random self-reducibility properties of the DDH problem; see, e.g., [3].)

2. For each query $\mathsf{Execute}(i, j)$, algorithm $\mathcal{F}$ generates a tuple (g, X_i, Y_i, Z_i) as above, outputs the transcript $X_i||\tau||Y_i||\tau'$ (where the tags are computed as expected), and sets the session key equal to Z_i.

3. All other queries are answered normally. In particular, note that even if an adversary replays a message that previously appeared in a transcript output by an $\mathsf{Execute}$ query, a correct session key can be computed (since $\mathcal{F}$ will know one of the necessary discrete logarithms).

4. Assuming col does not occur, if the adversary correctly guesses the value b used for the Test oracle query and Ex occurs, output "1" (this represents a guess that the initial input was a Diffie-Hellman tuple). In any other case (i.e., if col occurs, or Ex does not occur, or the adversary's guess is incorrect), output a random bit.

One can easily verify that if the initial input to $\mathcal{F}$ is a Diffie-Hellman tuple then the simulation is perfect and hence the probability that $\mathcal{F}$ outputs 1 in this case is

$$\Pr[b' = b \wedge \mathsf{Ex} \wedge \overline{\mathsf{col}}] + \frac{1}{2} \cdot (\Pr[b' \neq b \vee \overline{\mathsf{Ex}} \vee \mathsf{col}]).$$

On the other hand, if the input tuple is a random tuple then the adversary $\mathcal{A}$ has no information about the bit b used by the Test oracle and so the probability that $\mathcal{F}$ outputs 1 in this case is exactly 1/2. Since the difference between these probabilities can be at most ϵ', the claim follows.

Completing the proof, we claim that $\Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{col}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] \leq q_s^2 \epsilon'$. To see this, consider the following adversary $\mathcal{F}$:

1. $\mathcal{F}$ receives a tuple (g, X, Y, Z) which is either a Diffie-Hellman tuple or a random tuple. $\mathcal{F}$ chooses random $t_1, t_2 \in [q_s]$ with $t_1 < t_2$ (recall, q_s is a bound on the number of sessions initiated by $\mathcal{A}$). $\mathcal{F}$ chooses public keys for all parties normally.

2. $\mathcal{F}$ runs $\mathcal{A}$, answering its oracle queries as follows:

   - $\mathsf{Initiate}$ queries are answered normally except for the t_1^{th} and t_2^{th} such queries, which are answered as follows: The t_1^{th} query $\mathsf{Initiate}(i^*, j^*)$ is answered with $X|\tau$ and assume that the instance of i^* that is activated is $\Pi_{i^*}^k$. Let the t_2^{th} query be $\mathsf{Initiate}(j', i')$. If $i' \neq i^*$ or $j' \neq j^*$ then $\mathcal{F}$ aborts (and outputs a random bit). Otherwise, the query is answered with $Y|\tau'$. Let the instance of $j' = j^*$ that is activated by this query be $\Pi_{j^*}^{k'}$. (In both cases, the tags are computed normally, using the appropriate key.)
   - Send queries are answered normally, except for queries of the form $\mathsf{Send}(i^*, k, \tilde{Y}||\tilde{\tau})$ and $\mathsf{Send}(j^*, k', M')$ (if these queries are ever made). We focus on how the former query is answered, as the latter query is answered analogously. If $\tilde{Y}||\tilde{\tau} = Y|\tau'$ then $\mathcal{F}$ sets the session key of instance $\Pi_{i^*}^k$ to be Z and we call instance $\Pi_{i^*}^k$ good. If $\tilde{\tau}$ is not a valid tag for $\tilde{Y}$, then the instance terminates without a valid session key. If the tag is valid but $\tilde{Y}||\tilde{\tau}$ does not correspond to a message that was previously output by party j^*, then event Forge has occurred and $\mathcal{F}$ aborts (and outputs a random bit). Finally, if the tag is valid and $\tilde{Y}||\tilde{\tau}$ corresponds to a message previously output by j^* then $\mathcal{F}$ knows $\log_g \tilde{Y}$ and can compute a valid session key for instance $\Pi_{i^*}^k$.
   - $\mathsf{Execute}$ queries are answered normally.
   - Reveal and Test are answered normally except in the following cases: if a Reveal query is made to a good instance, then $\mathcal{F}$ aborts. If a Test query is *not* made to a good instance, then $\mathcal{F}$ aborts. If $\mathcal{F}$ aborts, it outputs a random bit. (In either eventuality, $\mathcal{F}$'s guess of t_1, t_2 was incorrect.)
   - $\mathsf{Corrupt}$ queries are answered normally except that if $\mathcal{A}$ ever queries $\mathsf{Corrupt}(P_{i^*})$ or $\mathsf{Corrupt}(P_{j^*})$ then $\mathcal{F}$ aborts and outputs a random bit (as its guess for t_1, t_2 was incorrect).

3. At the conclusion of the experiment, if neither col nor Ex have occurred and the adversary correctly guesses the value of the bit b used by the Test oracle, then output 1 and say that event Good occurs. In any other case, output a random bit.

The probability that $\mathcal{F}$ outputs 1 is $\Pr[\mathsf{Good}] + \frac{1}{2}(1 - \Pr[\mathsf{Good}])$. When the input to $\mathcal{F}$ is a Diffie-Hellman tuple then $\Pr[\mathsf{Good}]$ is at least $\Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{col}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}]/q_s^2$. On the other hand,

when the input to $\mathcal{F}$ is a random tuple then Pr[Good] is exactly 1/2. The hardness of the DDH problem in G thus implies the stated claim. ■

A variant. In the above description of $\mathcal{TS}3$, each party computes a key $k_{i,j}$ which it then uses to authenticate its message using a message authentication code. It is also possible to have each party P_i *sign* its messages using, for example, its public key y_i as a public key for, e.g., the Schnorr signature scheme (in fact, any signature scheme could be used assuming the parties have established the appropriate public keys). In this case, the party must sign (P_i, P_j, g^{α_i}) (in particular, it should sign the recipient's identity as well) to ensure that the signed message will be accepted only by the intended partner. The proof of security for this modified version is completely analogous to (and, in fact, slightly easier than) the proof of $\mathcal{TS}3$.

References

[1] R. Ankney, D. Johnson, and M. Matyas. The Unified Model. Contribution to ANSI X9F1, October 1995.

[2] G. Ateniese, M. Steiner, and G. Tsudik. New Multi-Party Authentication Services and Key Agreement Protocols. *IEEE J. on Selected Areas in Communications* 18(4): 628–639 (2000).

[3] M. Bellare, A. Boldyreva, and S. Micali. Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements. *Adv. in Cryptology — Eurocrypt 2000.*

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

[5] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Adv. in Cryptology — Crypto '93.*

[6] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *IEEE J. on Selected Areas in Communications* 11(5): 679–693 (1993).

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

[8] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. *Selected Areas in Cryptography*, 1998.

[9] C. Boyd. On Key Agreement and Conference Key Agreement. *ACISP 1997.*

[10] C. Boyd and J.M.G. Nieto. Round-Optimal Contributory Conference Key Agreement. *Public Key Cryptography*, 2003.

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

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

[13] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. *ACM Conf. on Computer and Communications Security*, 2001.

[14] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. *Advances in Cryptology — Eurocrypt '94.*

[15] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Adv. in Cryptology — Eurocrypt 2002.*

[16] D. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. *Comm. ACM* 24(8): 533–536 (1981).

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

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

[19] I. Ingemarasson, D.T. Tang, and C.K. Wong. A Conference Key Distribution System. *IEEE Trans. on Information Theory* 28(5): 714–720 (1982).

[20] M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. *Adv. in Cryptology — Asiacrypt '96.*

[21] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. *Adv. in Cryptology — Crypto 2003.*

[22] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. Technical report CORR 98-05, University of Waterloo, 1988.

[23] T. Matsumoto, Y. Takashima, and H. Imai. On Seeking Smart Public-Key Distribution Systems. *Trans. of the IECE of Japan*, E69, pp. 99–106, 1986.

[24] National Security Agency. SKIPJACK and KEA Algorithm Specification. Version 2.0, May 29, 1998.

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

[26] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. *ACM Conf. on Computer and Communications Security*, 1996.

[27] W.-G. Tzeng. A Practical and Secure-Fault-Tolerant Conference-Key Agreement Protocol. *Public Key Cryptography*, 2000.