One-Round Protocols for Two-Party Authenticated Key Exchange

Ik Rae Jeong[*] Jonathan Katz[†] Dong Hoon Lee[*]

January 15, 2008

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 even stronger security guarantees than our second protocol, but does not require random oracles. 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.

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/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 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.

[*]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`.

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; 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). 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. We define all these notions formally in Section 3.

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.) Note that achieving explicit authentication is not possible using a 1-round in the setting we consider, since the message of either party can always be replayed. Of course, it is well known how to convert any protocol providing implicit authentication to one providing explicit authentication, but only at the expense of additional rounds of communication.

	Boyd-Nieto (see note)	$\mathcal{TS}1$
Modular exponentiations (per party)	2	1
Communication (total)	$2\lvert p\rvert + \lvert q\rvert$	$2\lvert q\rvert$
Security	Key independence	Key independence
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 $\mathbb{Z}_p^*$.

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 designed 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.) In some cases explicit authentication may be unnecessary, however, 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, and whose security is based on the computational Diffie-Hellman (CDH) 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. (The Boyd-Nieto protocol is actually for group AKE, but it can be suitably modified for the case of two parties.) Our scheme is 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 $\mathcal{TS}2$. We compare both these protocols to previous work in Table 2.

	[1, 7]	[4]	[21]	$\mathcal{TS}2$	$\mathcal{TS}3$
Modular exponentiations (per party)	3	4	4	3	3
Rounds	3	3	2	1	1
Communication (total)	$2\|p\| + 2\|q\|$	$4\|p\|$	$4\|p\| + 2\|q\|$	$2\|p\|$	$2\|p\| + 2\|q\|$
Random oracle?	Yes	No	No	Yes	No

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

We rely on the standard computational and decisional Diffie-Hellman assumptions, as well as message authentication codes. The purpose of the present section is merely to fix some notation.

2.1 The Diffie-Hellman Problems

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:

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

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.)

Letting $\mathcal{GG}$ be as above, we may define a *DDH tuple* to be a tuple of the form $(g, g^{u_1}, g^{u_2}, g^{u_1 u_2})$ and a *random tuple* to be a tuple of the form $(g, g^{u_1}, g^{u_2}, g^{u_3})$. The *decisional Diffie-Hellman assumption* is to distinguish a random DDH tuple from a random tuple. 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:

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

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.)

2.2 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 $\tau = \mathsf{Mac}_{sk}(M)$. The verification algorithm Vrfy takes as input the message/tag pair and the (shared) key, and returns 1 if the tag is valid and 0 otherwise. We require that for all keys sk and all M, we have $\mathsf{Vrfy}_{sk}(M, \mathsf{Mac}_{sk}(M)) = 1$.

In defining the security of a MAC we use the standard definition of strong unforgeability under adaptive chosen-message attack. Namely, let $\Pi = (\mathsf{Mac}, \mathsf{Vrfy})$ be a MAC 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 $\tau$ on input $M$ then return 1
else return 0
```

The advantage of an adversary $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},\Pi}(k) \overset{\text{def}}{=} \Pr\left[\mathbf{Exp}^{\mathsf{suf}}_{\mathcal{A},\Pi}(k) = 1\right]$. We say that $\Pi = (\mathsf{Mac}, \mathsf{Vrfy})$ is *strongly unforgeable* (SUF-secure) if $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},\Pi}(k)$ 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 $\mathsf{Adv}^{\mathsf{suf}}_{\mathcal{A},\Pi} \leq \epsilon$ 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 if Mac is deterministic and verification is done by re-computing the MAC. We assume this in our constructions.)

3 Security Model for Authenticated Key Exchange

We use the standard notion of security for key-exchange protocols 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, and we assume a given instance is used only once. 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).

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 (of uncorrupted parties) compute the same session key.

To define 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, *specialized for the case of 1-round protocols.*

- The query $\mathsf{Initiate}(P_i, k, P_j)$ is used to "prompt" the un-used instance Π_i^k of party P_i to initiate execution of the protocol with partner $P_j \neq P_i$. This query will result in P_i sending a message, which is given to the adversary.

- A query $\mathsf{Send}(P_i, k, M)$ is used to send a message M to instance Π_i^k; this models active attacks on the part of the adversary. We assume without loss of generality that an adversary always queries $\mathsf{Initiate}(P_i, k, \star)$ before querying $\mathsf{Send}(P_i, k, M)$; this corresponds to assuming that the adversary always "rushes" the messages of honest parties, which only gives the adversary more power. Since we are dealing with 1-round protocols in this paper, a Send query does not result in any output (but it does cause instance Π_i^k to compute a session key).

- A query $\mathsf{Execute}(P_i, P_j)$ represents passive eavesdropping of the adversary on an execution of the protocol by parties P_i and P_j (with $P_i \neq P_j$). 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 $\mathsf{Execute}$ query can be simulated via repeated $\mathsf{Initiate}$ and Send oracle queries, this particular query is used to distinguish between passive and active attacks.)

- A query $\mathsf{Reveal}(P_i, k)$ models *known key* attacks (or Denning-Sacco attacks) in the real system. In response to this query, 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 $\mathsf{Test}(P_i, k)$ is used to define the advantage of an adversary. In response to this query, 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 may 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 also define *freshness*:

Definition 1 An instance Π_i^k, having partner P_j and (possibly) partnered with instance $\Pi_j^{k'}$, is *fresh* if the following conditions are true at the conclusion of the experiment:

(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 never queried $\mathsf{Send}(P_i, k, \star)$.

In all the notions of security considered below, the adversary $\mathcal{A}$ outputs a bit b' at the end of the experiment, and 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 (in addition to the single Test query):

- KI (Key independence): An adversary $\mathcal{A}$ can ask $\mathsf{Initiate}, \mathsf{Send}, \mathsf{Execute}$, and Reveal queries, but can not ask $\mathsf{Corrupt}$ queries.

- FS (Forward secrecy): An adversary $\mathcal{A}$ is now allowed to ask all queries, 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 $t = \text{poly}(k)$.

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 $H : \{0,1\}^* \rightarrow \{0,1\}^k$. (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. (For the case of $\mathcal{TS}3$, parties' public keys may include other information as well; see further below.) 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.

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):

<u>$\mathcal{TS}1$</u>

Setup: Assume P_i wants to establish a session key with P_j, and $P_i < P_j$. Recall P_i has public key $y_i = g^{x_i}$ and secret key x_i, and knows P_j's public key y_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 $\mathsf{sid} = r_i || r_j$. Party P_i computes the session key $sk_i = H(P_i \| P_j \| \mathsf{sid} \| y_j^{x_i})$.

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

$$\mathsf{sid} = r_1 || r_2$$
$$\mathsf{sk} = H(P_1||P_2||\mathsf{sid}||g^{x_1 x_2}) = H(P_1||P_2||\mathsf{sid}||y_2^{x_1}) = H(P_1||P_2||\mathsf{sid}||y_1^{x_2})$$

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

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

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*

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

where t is the maximum 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 instances initiated in the experiment.

Proof 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 $sk_i^k = H(P_i||P_j||\mathsf{sid}_i^k||g^{x_i x_j})$ for a fresh instance: either the adversary queries the random oracle on the point $P_i||P_j||\mathsf{sid}_i^k||g^{x_i x_j}$, 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 $q_s^2/2^k$, 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 coll 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 $H(P_i||P_j||\star||W)$ with $W = g^{x_i x_j}$. Observe that $\Pr_{\mathcal{A}}[b' = b \mid \overline{\mathsf{query}} \wedge \overline{\mathsf{coll}}] = \frac{1}{2}$, we may write

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

Now, as noted previously, $\Pr_{\mathcal{A}}[\mathsf{coll}]$ 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. $\mathcal{F}$ begins by selecting random distinct $i^*, j^* \in \{1, \ldots, N\}$, and letting U_1 and U_2 be the public keys for P_{i^*} and P_{j^*}, respectively. Assume without loss of generality that $P_{i^*} < P_{j^*}$. Public keys of other players are chosen in the specified way, so that $\mathcal{F}$ knows their secret keys.

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 $H(P_i||P_j||\mathsf{sid}||W)$ 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 $\mathsf{Initiate}(P_i, P_j)$, 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 $\mathsf{Send}(P_i, k, M)$, compute sid and check whether there is any other session $\Pi_j^{k'}$ with the same session id. (This can only happen if $\Pi_j^{k'}$ is partnered with Π_i^k). If so, then set the session key sk_i^k equal to the value of $\mathsf{sk}_j^{k'}$. Otherwise, set the session key sk_i^k equal to a random value in $\{0,1\}^k$.
   - For queries $\mathsf{Execute}(P_i, P_j)$ 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 sk_i^k and $\mathsf{sk}_j^{k'}$ both equal to the same random value in $\{0,1\}^k$.
   - Queries of the form $\mathsf{Reveal}(P_i, k)$ and $\mathsf{Test}(P_i, k)$ 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 $\Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{coll}}]/N^2 q_h$, 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 $\Pr_{\mathcal{A}}[\mathsf{query} \wedge \overline{\mathsf{coll}}] \leq q_h N^2 \epsilon$. This concludes the proof of the theorem. ■

We remark that the concrete security reduction in Theorem 1 can be improved using a more careful analysis, but we have not made any attempt to do so here.

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):

$\underline{\mathcal{TS}2}$

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. It receives a value g^{α_j}, presumably 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 $\mathsf{sid} = g^{\alpha_i} || g^{\alpha_j}$. P_i computes the session key $\mathsf{sk}_i = H(P_i || P_j || \mathsf{sid} || (g^{\alpha_j})^{\alpha_i} || y_j^{x_i})$.

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

$$\mathsf{sid} = g^{\alpha_1} || g^{\alpha_2}$$
$$\mathsf{sk} = H(P_1 || P_2 || \mathsf{sid} || g^{\alpha_1 \alpha_2} || g^{x_1 x_2})$$

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 experiment time including the adversary's execution time, and the adversary makes q_{re} Reveal queries, q_{co} $\mathsf{Corrupt}$ queries, and q_h hash queries. N is an upper bound on the number of parties, and q_s is an upper bound on the number of sessions initiated in the experiment.

Proof 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 $sk_i^k = H(P_i||P_j||\mathsf{sid}||g^{\alpha_i\alpha_j}||g^{x_ix_j})$: either the adversary queries the random oracle on the point $P_i||P_j||\mathsf{sid}||g^{\alpha_i\alpha_j}||g^{x_ix_j}$, or the value sid has repeated at some point during the experiment (for the same pair of users). The latter case happens with probability upper bounded by $\frac{q_s^2}{q}$, 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 coll 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 query to an instance which is corrupted, or whose partner is corrupted; i.e., it asks query $\mathsf{Test}(P_i, k)$ 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, if $\mathsf{corrupt}$ occurs then it must 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 $P_i||P_j|| \star || \star ||W$, neither P_i nor P_j are corrupted in the entire course of the experiment, and $W = g^{x_ix_j}$. Let query_2 be the event that, for some $i, j \in [N]$, the adversary at some point queries the random oracle on $P_i||P_j||\mathsf{sid}||X||\star$, and it holds that:

- $\mathsf{sid} = \mathsf{sid}_i^k$ for some k
- Π_i^k was initiated via a call $\mathsf{Execute}(P_i, P_j)$ (and hence $\mathsf{sid} = \mathsf{sid}_j^{k'}$ for some k' as well)
- $\mathsf{sid} = \mathsf{sid}_i^k = g^{\alpha_i}||g^{\alpha_j}$ and $X = g^{\alpha_i\alpha_j}$.

It is not hard to see that

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

and

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

Thus:

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

As in the previous theorem, $\Pr_{\mathcal{A}}[\mathsf{coll}] \leq q_s^2/q$. Furthermore, following essentially the same argument as in the proof of the previous theorem (with some small modifications), we may show that

$$\Pr_{\mathcal{A}}[\mathsf{query}_1 \wedge \overline{\mathsf{corrupt}}] \leq \Pr_{\mathcal{A}}[\mathsf{query}_1] \leq q_h N^2 \epsilon.$$

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 $H(P_i||P_j||\mathsf{sid}||X||W)$, return a random value $v \in \{0,1\}^k$. (We assume $\mathcal{A}$ does not make the same oracle query twice.) Store (sid, X) in a list $\mathsf{dh\text{-}tuples}$.
    - $\mathsf{Initiate}$, Send, Reveal, $\mathsf{Corrupt}$, and Test queries are answered honestly (in particular, when a session key must be computed the appropriate random oracle query is answered as discussed above, maintaining consistency in the obvious way).
    - For queries $\mathsf{Execute}(P_i, P_j)$ 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 (sid, X) from its list $\mathsf{dh\text{-}tuples}$. If an $\mathsf{Execute}$ query was previously answered using session id sid, then $\mathcal{F}$ takes the values a, b used in answering that query and outputs $X/U_2^a U_1^b g^{ab}$. (Note that, in this case, $\mathsf{sid} = U||V$ with $U = U_1 g^a$ and $V = U_2 g^b$.)

Note that the simulation is perfect until the point, if any, that query_2 occurs. Furthermore, in case query_2 occurs then with probability at least $1/q_h$ it is the case that $\mathcal{F}$ outputs a correct answer to the given instance of the CDH problem. 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. We now present protocol $\mathcal{TS}3$ which is proven secure in the standard model (but using the stronger DDH assumption):

$\underline{\mathcal{TS}3}$

Setup: Same as in $\mathcal{TS}1$, though see below for a discussion regarding additional values that may be included in the parties' public keys.

Round 1: P_i computes $k_{i,j} = y_j^{x_i}$ which it will use as a key for a message authentication code. (Note that $k_{i,j}$ is a group element rather than a string. See below for further discussion.) 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. It receives $g^{\alpha_j}||\tau_j$, presumably from P_j.

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 $\mathsf{sid}_i = g^{\alpha_i}||\tau_i||g^{\alpha_j}||\tau_j$.

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

In our description of the protocol, we have assumed a MAC that can be keyed by *group elements* rather than strings. Doing so is mainly a conceptual simplification that we have made so as to focus on the security of the protocol (as our aim is not to provide a detailed specification for

	$P_1(x_1, y_2)$	$P_2(x_2, y_1)$
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 $\mathcal{TS}3$.

implementation purposes). Nevertheless, since existing MACs assume a random string as a key (and not a random group element), we mention for completeness two ways our protocol can be modified in practice:

- An easy way to resolve the discrepancy is to simply hash a given group element to a string suitable for use as a MAC key. If we treat the hash H as a random oracle, and assume the underlying MAC is secure (when using a random string as the key), then it is trivial to see that this derived MAC is secure when using a random group element as a key.

  Of course, the whole point of this protocol is to avoid using the random oracle model in the first place! However, we note that the full power of the random oracle is not needed, and we may instead simply *assume* that H and the underlying MAC are such that using $H(g)$ as the key (for g a random group element) is secure. This is a well-defined (though non-standard) complexity assumption that is reasonable when H is a standard cryptographic hash function such as, e.g., (truncated) SHA-1.

- If one prefers to stick to standard assumptions, another solution is to use a *strong extractor*. Roughly speaking, an extractor is a keyed function Ext for which $\mathsf{Ext}_s(g)$ is statistically close to uniform when s is chosen uniformly at random and g has high min-entropy (but is not necessarily uniform); a *strong* extractor has this property even conditioned on the value of s. Strong extractors can be constructed unconditionally.

  In our setting, a strong extractor with publicly-known key s can be applied to a random group element to yield a random string that can then be used as a key for some underlying MAC. The only remaining question is how s is generated. (We stress that it can be public, but must be chosen uniformly at random. Therefore we may not leave it under adversarial control.) One possibility is to simply assume that s is generated as some system-wide parameter, along with a description of the group, etc. If one wishes to avoid this, then one may alternately assume that each user chooses their own value s and publishes it as part of their public key. Then two parties P_i, P_j with $P_i < P_j$ would use $\mathsf{Ext}_{s_i}(g^{x_1 x_2})$ as their MAC key.

The above justifies treating the MAC as being keyed by a random group element, and we simply assume this in the proof below.

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

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

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} Corrupt queries. N is an upper bound on the number of parties, and q_s is an upper bound of the number of instances an adversary initiates.

Proof The intuition is as follows: under the DDH assumption, the adversary cannot have non-negligible advantage unless it can choose a value g^α for which it knows α, send this to an instance, and have that instance compute a valid session key using this value (i.e., if the instance sends $Y = g^{\alpha'}$ then the adversary knows the computed session key Y^α). 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 coll 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 $\mathsf{Send}(P_i, k, M)$ and (1) neither P_i nor P_j was previously corrupted; (2) M was never sent by P_j to P_i (formally, was never output in response to a query $\mathsf{Initiate}(P_j, \star, P_i)$ and never appeared in a transcript output by a query $\mathsf{Execute}(P_i, P_j)$ or $\mathsf{Execute}(P_j, P_i)$); and (3) the tag contained in M is valid. One can fairly easily derive:

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

As in the previous proofs, we have $\Pr[\mathsf{coll}] \leq q_s^2/q$. 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. ■

The two claims that follow provide upper bounds for the final two terms of the equation above.

Claim $\left|\Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{coll}} \wedge \mathsf{Ex}] - \frac{1}{2} \cdot \Pr_{\mathcal{A}}[\overline{\mathsf{coll}} \wedge \mathsf{Ex}]\right| \leq \epsilon'$.

Proof of claim: Intuitively, we prove the claim by embedding a DDH instance into every $\mathsf{Execute}$ query. Formally, consider an adversary $\mathcal{F}$ who proceeds as follows:

1. Public keys for all the players are generated honestly.

2. Given a tuple (g, X, Y, Z) as input, $\mathcal{F}$ will 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. We refer to [3] for a description of how this is done.

3. For each query $\mathsf{Execute}(P_i, P_j)$, 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.

4. All $\mathsf{Initiate}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{Corrupt}$, and Test queries are answered normally.

5. When $\mathcal{A}$ is done, $\mathcal{F}$ checks that coll has not occurred and that Ex has occurred. If this is not the case, then $\mathcal{F}$ outputs a random bit. Otherwise, $\mathcal{F}$ outputs '1' iff $\mathcal{A}$ correctly guesses the value b used to answer the Test query.

Let us analyze the behavior of $\mathcal{F}$. If its input is a random DDH tuple, then $\mathcal{F}$ provides a perfect simulation for $\mathcal{A}$. Thus, the probability that $\mathcal{F}$ outputs '1' in this case is given by:

$$\tfrac{1}{2} \cdot \left(1 - \Pr_{\mathcal{A}}[\overline{\mathsf{coll}} \wedge \mathsf{Ex}]\right) + \Pr_{\mathcal{A}}[b = b' \wedge \overline{\mathsf{coll}} \wedge \mathsf{Ex}].$$

On the other hand, if the input given to $\mathcal{F}$ is a random tuple then the probability that $\mathcal{F}$ outputs '1' is exactly 1/2 (since regardless of whether Ex occurs or not, $\mathcal{F}$ outputs '1' with probability 1/2). We therefore have

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{F},\mathcal{GG}} = \left|\Pr_{\mathcal{A}}[b = b' \wedge \overline{\mathsf{coll}} \wedge \mathsf{Ex}] - \tfrac{1}{2} \cdot \Pr_{\mathcal{A}}[\overline{\mathsf{coll}} \wedge \mathsf{Ex}]\right|.$$

The claim follows. ■

Claim $\left|\Pr_{\mathcal{A}}[b' = b \wedge \overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] - \frac{1}{2} \cdot \Pr_{\mathcal{A}}[\overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}]\right| \leq q_s^2 \epsilon'$.

Proof of claim: 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 distinct, random $\alpha, \beta \in \{1, \ldots, q_s\}$. Recall that q_s is a bound on the number of instances initiated by $\mathcal{A}$; intuitively, $\mathcal{F}$ is guessing that $\mathcal{A}$ will take the message sent by the α^{th} instance, forward this message to the β^{th} instance, and then make its Test query on the β^{th} instance.

2. $\mathcal{F}$ chooses public keys for all parties normally, and stores the associated secret keys.

3. $\mathcal{F}$ runs $\mathcal{A}$, answering its oracle queries as described below. In the course of this experiment, $\mathcal{F}$ will sequentially assign a unique number in the range $\{1, \ldots, q_s\}$ to each instance initiated by $\mathcal{A}$. Note that $\mathcal{A}$ initiates an instance in two ways: either via an $\mathsf{Initiate}$ query (when a single instance is initiated), or via an $\mathsf{Execute}$ query (when two instances are initiated). We now describe how the oracle queries of $\mathcal{A}$ are handled:

   In response to a query $\mathsf{Execute}(P_i, P_j)$, where ctr_1, ctr_2 are the numbers assigned by $\mathcal{F}$ to the two instances thus initiated, $\mathcal{F}$ responds as follows:

   - If either $\mathsf{ctr}_1 = \beta$ or $\mathsf{ctr}_2 = \beta$, then $\mathcal{F}$ aborts (since its guess for β was wrong) and outputs a random bit.
   - If $\mathsf{ctr}_1 \neq \alpha$ and $\mathsf{ctr}_2 \neq \alpha$, then this query is answered normally.

- If $\mathsf{ctr}_1 = \alpha$, then $\mathcal{F}$ chooses random $\gamma \in \mathbb{Z}_q$ and responds to this query with the transcript $X||\tau||g^\gamma||\tau'$, where X is taken from the input instance and τ, τ' are computed normally. Also, the session key for both instances is set to X^γ.
- If $\mathsf{ctr}_2 = \alpha$, then $\mathcal{F}$ responds as above except with the transcript $g^\gamma||\tau||X||\tau'$.

In response to a query $\mathsf{Initiate}(P_i, k, P_j)$, where ctr is the number assigned by $\mathcal{F}$ to instance Π_i^k initialized by this query, $\mathcal{F}$ responds as follows:

- If $\mathsf{ctr} \notin \{\alpha, \beta\}$, then this query is answered normally.
- If $\mathsf{ctr} = \alpha$, then $\mathcal{F}$ responds to this query with $X||\tau$, where X is taken from the input instance and τ is computed normally.
- If $\mathsf{ctr} = \beta$, then $\mathcal{F}$ responds to this query with $Y||\tau$, where Y is taken from the input instance and τ is computed normally.

In response to a query $\mathsf{Send}(P_i, k, M)$, where ctr is the number assigned by $\mathcal{F}$ to instance Π_i^k, adversary $\mathcal{F}$ responds as follows:

- If $\mathsf{ctr} \notin \{\alpha, \beta\}$, then $\mathcal{F}$ responds to this query normally. (It can do this since it responded normally to the corresponding $\mathsf{Initiate}$ query for this instance.)
- If $\mathsf{ctr} = \alpha$, then the message sent by this instance was $X||\tau$. Let $M = \hat{X}||\tau'$. Then:
  - If τ' is an invalid MAC tag, then $\mathcal{F}$ simply sets $\mathsf{sk}_i^k = \perp$.
  - Say $\hat{X}||\tau'$ was not output by $\mathcal{F}$ in response to an $\mathsf{Initiate}$ or $\mathsf{Execute}$ query, but τ is valid. Then either a MAC forgery has occurred or else P_i or its partner has been corrupted; in any case, $\mathcal{F}$ aborts and outputs a random bit.
  - Otherwise, $\hat{X}||\tau'$ was output by $\mathcal{F}$ in response to an $\mathsf{Initiate}$ or $\mathsf{Execute}$ query corresponding to some instance $\Pi_j^{k'}$ numbered ctr'. If $\mathsf{ctr}' = \alpha = \mathsf{ctr}$, then this is a forgery (since $\tau' = \mathsf{Mac}_{k_{i,j}}(i||j||\hat{X})$ but τ' is also a valid tag for the message $j||i||\hat{X}$) and so $\mathcal{F}$ aborts and outputs a random bit. If $\mathsf{ctr}' = \beta$ then $\hat{X} = Y$ and $\mathcal{F}$ sets $\mathsf{sk}_i^k = Z$. If $\mathsf{ctr}' \notin \{\alpha, \beta\}$, then $\mathcal{F}$ knows $\gamma = \log_g \hat{X}$ and computes the session key $\mathsf{sk}_i^k = X^\gamma$.
- If $\mathsf{ctr} = \beta$, then there are two cases:
  - If M is the message output by the α^{th} instance (i.e., $M = X||\tau$) and τ is a valid tag (i.e., the user associated with the α^{th} instance is the intended partner of the present instance), then $\mathcal{F}$ sets $\mathsf{sk}_i^k = Z$.
  - In any other case, $\mathcal{F}$ aborts and outputs a random bit.

Reveal, $\mathsf{Corrupt}$, and Test queries are answered normally.

4. At the conclusion of the experiment, $\mathcal{F}$ outputs a random bit if any of the following are true: (1) coll or Forge or Ex has occurred, or (2) $\mathcal{A}$ did not ask its Test query on the β^{th} instance. Otherwise, $\mathcal{F}$ outputs '1' if $\mathcal{A}$ guesses b correctly, and '0' otherwise.

Let us first analyze the behavior of $\mathcal{F}$ under the assumption that its input is a random Diffie-Hellman tuple. Define Good to be the event that neither Forge nor coll occur, and furthermore $\mathcal{A}$ asks its Test query on the β^{th} instance, which was initialized via an $\mathsf{Initiate}$ query and was sent the message output by the α^{th} instance. (Note that if Good occurs then in particular Ex does not occur.) We claim that $\mathcal{F}$ provides a perfect simulation for $\mathcal{A}$ until such point as it is clear that Good will not occur. To see this, let P_β be the user associated with the β^{th} instance, and let P'_β be the partner of the β^{th} instance. Let P_α be the user associated with the α^{th} instance, and let P'_α be the partner of the α^{th} instance. First observe that if $P_\alpha \neq P'_\beta$ or $P'_\alpha \neq P_\beta$, then Good cannot occur. So assume $P_\alpha = P'_\beta$ and $P'_\alpha = P_\beta$ (with $P_\alpha \neq P_\beta$). Assume also that the β^{th} instance was initialized via an $\mathsf{Initiate}$ query (or else Good also cannot occur). Then:

- If $\mathcal{F}$ aborts in response to a query $\mathsf{Send}(P_\alpha, k, M)$ with $\mathsf{ctr} = \alpha$, there are two cases:
  - A forgery has occurred, in which case Good cannot occur.
  - P_α or P_β were corrupted; then the β^{th} instance is not fresh and Good cannot occur.
- If $\mathcal{F}$ aborts in response to a query $\mathsf{Send}(P_\beta, k, M)$ with $\mathsf{ctr} = \beta$, clearly Good cannot occur.

$\mathcal{F}$ outputs '1' in this case with probability

$$\Pr[b' = b \wedge \mathsf{Good}] + \frac{1}{2} \cdot (1 - \Pr[\mathsf{Good}]).$$

Furthermore, we conclude from the above that

$$\Pr[\mathsf{Good}] = \frac{\Pr[\overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}]}{q_s(q_s - 1)}$$

and

$$\Pr[b' = b \wedge \mathsf{Good}] = \frac{\Pr[b' = b \wedge \overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}]}{q_s(q_s - 1)}.$$

Summarizing, when the input to $\mathcal{F}$ is a random Diffie-Hellman tuple then $\mathcal{F}$ outputs '1' with probability

$$\frac{1}{2} + \frac{1}{q_s(q_s - 1)} \cdot \left(\Pr[b' = b \wedge \overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] - \frac{1}{2} \cdot \Pr[\overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] \right).$$

On the other hand, when the input to $\mathcal{F}$ is a random tuple then $\mathcal{F}$ outputs '1' with probability exactly $\frac{1}{2}$. Therefore,

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{F},\mathcal{GG}} = \frac{1}{q_s(q_s - 1)} \cdot \left| \left(\Pr[b' = b \wedge \overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] - \tfrac{1}{2} \cdot \Pr[\overline{\mathsf{coll}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ex}}] \right) \right|,$$

from which the claim follows. ■

This completes the proof of the theorem. ■

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 $\mathcal{TS}3$.

Acknowledgments

We thank Kenny Paterson and Colin Boyd for pointing out some errors in the proofs contained in a previous version of this paper.

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.