COMPARING THE PRE- AND POST-SPECIFIED PEER MODELS FOR KEY AGREEMENT

ALFRED MENEZES AND BERKANT USTAOGLU

Abstract. In the pre-specified peer model for key agreement, it is assumed that a party knows the identifier of its intended communicating peer when it commences a protocol run. On the other hand, a party in the post-specified peer model for key agreement does not know the identifier of its communicating peer at the outset, but learns the identifier during the protocol run. In this paper we compare the security assurances provided by the Canetti-Krawczyk security definitions for key agreement in the pre- and post-specified peer models. We give examples of protocols that are secure in one model but insecure in the other. We also enhance the Canetti-Krawczyk security models and definitions to encompass a class of protocols that are executable and secure in both the pre- and post-specified peer models.

1. Introduction

In 1993, Bellare and Rogaway [1] presented the first formal security model and security definition for key agreement. The model and associated definitions evolved over the years, culminating in the 2001 work of Canetti and Krawczyk [4] and its recent extension by LaMacchia, Lauter and Mityagin [13]. In all the aforementioned works, key agreement protocols are analyzed in the so-called *pre-specified peer model* wherein it is assumed that a party knows the identifier of its intended communicating peer when it commences a run of the protocol. That is, it is assumed that the exchange of identifiers, and possibly also the long-term public keys of the communicating parties, is handled by the application that invokes a run of the protocol.

In 2002, Canetti and Krawczyk [5] introduced the *post-specified peer model* wherein a party is activated to establish a session key knowing only a destination address (such as the IP address of a server) of the communicating peer, and only learns the peer's identifier during the execution of the protocol. According to [5], this scenario is common in practical settings where the peer's identifier is simply unavailable at the outset, or if one party wishes to conceal its identity from eavesdroppers or active adversaries. The IKE protocols [8, 9] (see also [10]) are important examples of key agreement protocols that provide the option of identity concealment.

In the remainder of this paper we will not consider the identity concealment attribute of key agreement protocols. We will often shorten 'pre-specified peer model' to 'pre model', and 'post-specified peer model' to 'post model'.

We say that a key agreement protocol designed for one of the pre or post models is *executable* in the other model if it can be run in the second model without requiring any additional message flows (and without making any fundamental changes to the protocol description). It is clear that any key agreement protocol designed for the post model is executable in the pre model. Indeed, if the peer's identifier (and long-term public key) is not needed at the start of the protocol, then the

Date: May 2, 2008.

protocol can also be executed given the peer's identifier. Canetti and Krawczyk observed that the Σ_0 key agreement protocol is secure in the post model with respect to the security definition given in [5], but not secure in the pre model with respect to the security definition given in [4]. Hence, even though any protocol designed for the post model can be executed in the pre model, security in the post model of [5] does not guarantee security in the pre model of [4].

In this paper we explore the executability and security in the post model of key agreement protocols that have been designed for and analyzed in the pre model. Of course any protocol designed for the pre model can be modified for the post model by adding message flows which include the identifiers and long-term public keys of the communicating parties; however such a modification does not conform to our notion of executability because of the additional message flows. We provide an example of a key agreement protocol that is secure in the pre model but is not executable in the post model. We also observe that the HMQV protocol [11], which has been proven to be secure in the pre model, is executable in the post model (without the addition of message flows) but not secure unless additional measures are taken. These examples illustrate the essential differences between the two models, and highlight the danger of running in the post model a protocol that has only been analyzed in the pre model.

It is natural then to ask when a protocol secure in one model is executable and secure in the other model. We identify a class of *modifiable* key agreement protocols that have been designed for the pre model but can be executed with minimal modifications in the post model. This class includes many of the protocols that have been proposed in the literature including station-to-station [7], UM [18, 16], MQV [15], Boyd-Mao-Paterson [2], HMQV [11], KEA+ [14], NAXOS [13], CMQV [19] and Okamoto [17]. (See [3] for an extensive list of key establishment protocols.) Such protocols have a *hybrid* description that combine the specification for the pre model and the specification of the modified protocol suitable for the post model. We develop a *combined* model and associated security definition that aims to simultaneously capture the security assurances (and more) of the extended Canetti-Krawczyk pre-specified peer model [13] and the Canetti-Krawczyk post-specified peer model [5]. The combined model has the feature that if a hybrid key agreement protocol is proven secure in that model, then its specializations are guaranteed to be secure when run in the pre and post models.

The remainder of this paper is organized as follows. In §2 we provide informal overviews of the Canetti-Krawczyk pre and post models and security definitions for key agreement. The differences between the two models are explored in §3. Protocol $\mathcal{P}$ is described in §3.1 as an example of a protocol that is secure in the pre model but not executable in the post model. In §3.2 we describe an attack on HMQV, demonstrating that the protocol is not secure in the post model. The Σ_0 protocol, which is an example of a protocol that is secure in the post model but insecure in the pre model, is revisited in §3.3. Our combined model and security definition are presented in §4. The NAXOS-C protocol is presented in §5 as an example of a protocol that is secure in the combined model. The appendix contains a reductionist security proof for the NAXOS-C protocol.

Notation and terminology. Let $G = \langle g \rangle$ denote a multiplicatively-written cyclic group of prime order q, and let $G^* = G \setminus \{1\}$. The *Computational Diffie-Hellman (CDH) assumption* in G is that computing $\mathrm{CDH}(U, V) = g^{uv}$ is infeasible given $U = g^u$ and $V = g^v$ where $u, v \in_R [1, q-1]$. The *Decisional Diffie-Hellman (DDH) assumption* in G is that distinguishing DH triples (g^a, g^b, g^{ab}) from random triples (g^a, g^b, g^c) is infeasible. The *Gap Diffie-Hellman (GDH) assumption* in G is

that the CDH assumption holds even when a CDH solver is given a DDH oracle that distinguishes DH triples from random triples.

We consider Diffie-Hellman type protocols where the two communicating parties exchange static (long-term) and ephemeral (one-time) public keys. Party $\hat{A}$'s static private key is an integer $a \in_R [1, q-1]$, and her corresponding static public key is $A = g^a$. Similarly, party $\hat{B}$ has a static key pair (b, B), and so on. A certifying authority (CA) issues certificates that binds a party's identifier to its static public key. We do not assume that the CA requires parties to prove possession of their static private keys, but we do insist that the CA verifies that static public keys belong to G^*. We restrict our attention to protocols where a party $\hat{A}$ called the *initiator* commences the protocol by selecting an ephemeral key pair and then sends the ephemeral public key (and possibly other data) to the second party. In our protocols, the ephemeral private key is either a randomly selected integer $x \in [1, q-1]$ or a randomly selected binary string $\tilde{x}$ which is used together with the static private key to derive an integer $x \in [1, q-1]$, and the corresponding ephemeral public key is $X = g^x$. Upon receipt of X, the *responder* $\hat{B}$ selects an ephemeral private key y or $\tilde{y}$ and sends $Y = g^y$ (and possibly other data) to $\hat{A}$. The parties may exchange some additional messages, after which they compute a session key. We use $\mathcal{I}$ and $\mathcal{R}$ to denote the constant strings "initiator" and "responder".

2. Security definitions for key agreement

We provide overviews of the Canetti-Krawczyk pre- and post-specified peer models for key agreement and the associated security definitions. For full details and further explanations refer to [4] and [5].

2.1. **Pre-specified peer model.**

Communications take place in a multi-party system, where the parties are identified by $\hat{A}$, $\hat{B}$, $\hat{C}$, ... At any given point in time, a party may be engaged in multiple instances of the protocol, each called a *session*. A session is created at $\hat{A}$ via a message containing at least three parameters $(\hat{A}, \hat{B}, s)$, where $\hat{A}$ is the session's *owner*, $\hat{B}$ is the intended *peer*, and s is a number that is unique among all sessions owned by $\hat{A}$. ($\hat{A}$ uses s to direct incoming messages to the appropriate session within $\hat{A}$.) Once created, a session is said to be *active* and maintains a *session state* where session-specific short-lived data such as an ephemeral private key is stored. The session processes incoming messages and produces outgoing messages. A session may *abort* without producing a session key, or may *complete* by accepting a session key and erasing its session state.

The adversary $\mathcal{M}$, modeled as a probabilistic Turing machine, controls all communications between parties as well as the activation of sessions. In order to model the possible leakage of secret information, $\mathcal{M}$ is allowed to issue the following queries to parties:

- *SessionStateReveal*: $\mathcal{M}$ learns the contents of the session state for a (not yet completed) session of its choosing. The session can no longer be activated and stops producing output.
- *Expire*: $\mathcal{M}$ directs a completed session to delete its session key.
- *SessionKeyReveal*: $\mathcal{M}$ learns the session key held by a (completed but unexpired) session of its choosing.
- *Corrupt*: $\mathcal{M}$ learns all the secret information held by a party of its choosing, including the party's static private key, all session states, and all session keys. The party can no longer be activated and stops producing output.

The adversary's goal is to distinguish a session key from a random key. Obviously the adversary should not be allowed to learn the session key by trivial means, for example by asking for the session key via a *SessionKeyReveal* query. To this end, a session $(\hat{A}, \hat{B}, s)$ is said to be *locally exposed* if $\mathcal{M}$ issued a *SessionStateReveal* or *SessionKeyReveal* query to that session, or if $\mathcal{M}$ issued a *Corrupt* query to $\hat{A}$ before the session expired (this includes the case in which $\hat{A}$ is corrupted before the session is created). Moreover, the session $(\hat{B}, \hat{A}, s)$ is defined to be *matching* to the session $(\hat{A}, \hat{B}, s)$, and $(\hat{A}, \hat{B}, s)$ is said to be *unexposed* if neither this session nor its matching session are locally exposed. Now, $\mathcal{M}$ selects a session that is completed, unexpired, and unexposed, and issues a special *Test* query to that session. ($\mathcal{M}$ is not allowed to issue the *Test* query more than once.) In response, $\mathcal{M}$ is given with equal probability either the session key held by the test session or a random key. $\mathcal{M}$ can continue to issue queries, however must ensure that the test session remains unexposed. Finally, $\mathcal{M}$ is said to win its distinguishing game (and thereby break the protocol) if it guesses correctly whether the key is random or not with success probability significantly greater than $\frac{1}{2}$. A key agreement protocol is said to be *secure* (in the pre-specified peer model) if (i) uncorrupted parties who complete matching sessions compute the same session key (except with negligible probability); and (ii) there is no adversary $\mathcal{M}$ who wins the distinguishing game.

2.2. **Post-specified peer model.** The Canetti-Krawczyk post-specified peer model and associated security definition [5] are essentially the same as in the pre model, but there are two important differences.

First, a session at $\hat{A}$ is created via a message containing (at least) three parameters $(\hat{A}, \hat{d}, s)$, where $\hat{d}$ is a *destination address* to which outgoing messages should be delivered. That is, party $\hat{A}$ does not know the identifier of its peer when it starts the session. During the course of the protocol run, $\hat{A}$ learns the (alleged) identifier $\hat{B}$ of the communicating party; this party is referred to as $\hat{A}$'s peer for that session.

Second, the definition of a matching session is different. Let $(\hat{A}, s)$ be a session that has completed with peer $\hat{B}$. Then a session $(\hat{B}, s)$ is said to be *matching* to $(\hat{A}, s)$ if either (i) $(\hat{B}, s)$ has not yet completed; or (ii) $(\hat{B}, s)$ has completed and its peer is $\hat{A}$. Condition (i) is necessary because the incomplete session $(\hat{B}, s)$ may not yet have determined its peer and hence could have been communicating with $(\hat{A}, s)$, in which case exposure of $(\hat{B}, s)$ could possibly reveal non-trivial information about the session key held by $(\hat{A}, s)$.

3. Differences between the two models

This section presents three examples to illustrate the differences between the Canetti-Krawczyk security definitions for key agreement in the pre- and post-specified peer models. Protocol $\mathcal{P}$ is secure in the pre model, but cannot be executed in the post model. HMQV is an example of a protocol that is secure in the pre model, and executable but not secure in the post model. The Σ_0 protocol is secure in the post model but insecure in the pre model.

3.1. **Protocol $\mathcal{P}$.** We present a two-pass Diffie-Hellman key agreement protocol $\mathcal{P}$. The protocol can be proven secure in the pre-specified peer model under the GDH assumption and where H and H_2 are modeled as random functions. (The reductionist security argument is elementary but tedious, and hence is omitted.) Observe that the initiator $\hat{A}$ cannot prepare the first outgoing message without knowledge of the peer's identifier $\hat{B}$ and static public key B. Hence, unless

protocol $\mathcal{P}$ is modified in a fundamental way, it cannot be executed in the post-specified peer model without additional message flows to exchange identifiers and static public keys.

(1) On input $(\hat{A}, \hat{B}, s)$, party $\hat{A}$ (the initiator) does the following:
   (a) Create an active session $(\hat{A}, \hat{B}, s, \mathcal{I})$.
   (b) Select an ephemeral private key $x \in_R [1, q-1]$.
   (c) Compute $X = g^x$ and $t_A = H_2(B^a, \mathcal{I}, s, \hat{A}, \hat{B}, X)$.
   (d) Send $(\hat{B}, \hat{A}, s, X, t_A)$ to $\hat{B}$.

(2) Upon receiving $(\hat{B}, \hat{A}, s, X, t_A)$, party $\hat{B}$ (the responder) does the following:
   (a) Create an active session $(\hat{B}, \hat{A}, s, \mathcal{R})$.
   (b) Verify that $X \in G^*$.
   (c) Compute $\sigma_s = A^b$ and verify that $t_A = H_2(\sigma_s, \mathcal{I}, s, \hat{A}, \hat{B}, X)$.
   (d) Select an ephemeral private key $y \in_R [1, q-1]$.
   (e) Compute $Y = g^y$, $t_B = H_2(\sigma_s, \mathcal{R}, s, \hat{B}, \hat{A}, Y)$, and $k = H(X^y, X, Y)$.
   (f) Destroy y and σ_s.
   (g) Send $(\hat{A}, \hat{B}, s, \mathcal{I}, Y, t_B)$ to $\hat{A}$.
   (h) Complete the session $(\hat{B}, \hat{A}, s, \mathcal{R})$ and accept k as the session key.

(3) Upon receiving $(\hat{A}, \hat{B}, s, \mathcal{I}, Y, t_B)$, party $\hat{A}$ checks that she owns an active session with identifier $(\hat{A}, \hat{B}, s, \mathcal{I})$. If so, then $\hat{A}$ does the following:
   (a) Verify that $Y \in G^*$.
   (b) Verify that $t_B = H_2(B^a, \mathcal{R}, s, \hat{B}, \hat{A}, Y)$.
   (c) Compute $k = H(Y^x, X, Y)$.
   (d) Destroy x.
   (e) Complete the session $(\hat{A}, \hat{B}, s, \mathcal{I})$ by accepting k as the session key.

3.2. **HMQV protocol.** HMQV [11] is an efficient two-pass Diffie-Hellman key agreement protocol that has been proven to be secure in the pre-specified peer model under the CDH and KEA1 assumptions and where the hash functions employed are modeled as random functions.[1] The following informal description of the protocol omits some technical details that are not relevant to our analysis.[2]

Let $\overline{H}$ denote a hash function whose outputs are bitstrings of length l, where l is half the bitlength of the group order q. In HMQV, the initiator $\hat{A}$ sends $(\hat{B}, \hat{A}, X)$ to $\hat{B}$, who responds with $(\hat{A}, \hat{B}, Y)$. Party $\hat{A}$ computes the session key $k = H(\sigma_A)$, where $\sigma_A = (YB^e)^{x+da}$ and $d = \overline{H}(X, \hat{B})$ and $e = \overline{H}(Y, \hat{A})$. Party $\hat{B}$ computes the same session key as $k = H(\sigma_B)$, where $\sigma_B = (XA^d)^{y+eb}$.

Unlike protocol $\mathcal{P}$, HMQV is executable in the post-specified peer model. Indeed, the initiator can prepare the first message (which essentially consists of the ephemeral public key X) without knowledge of the peer's identifier $\hat{B}$ or static public key B. It is natural then to ask whether HMQV is secure in the post model. This is also important because the version of HMQV that is being considered for standardization by the P1363 working group [12] does not mandate that the protocol be executed in the pre model (i.e., there is no requirement that the communicating parties possess each other's identifiers and static public keys prior to a protocol run), and consequently the

[1]The security definition used in [11] is stronger than the security definition outlined in §2.1 in the sense that the adversary is granted certain additional capabilities. For example, the adversary is allowed to register a static key pair at any time thus allowing the modeling of attacks by malicious insiders.

[2]In particular, we omit session identifiers and assume that all static and ephemeral public key are fully validated, i.e., verified as belonging to G^*.

protocol may in fact be executed in the post model in applications where the responder's identifier is not available to the initiator at the beginning of the protocol run.

We describe an attack which demonstrates that HMQV (without further modification such as the addition of message flows to exchange identifiers and static public keys) is not secure in the post model. The attack makes the following plausible assumptions: (i) the group order q is a 160-bit prime and so the outputs of $\overline{H}$ have bitlength 80; (ii) the best attack on the CDH problem in G takes approximately 2^{80} steps; (iii) there are at least 2^{20} honest (i.e., uncorrupted) parties; (iv) a party can select its own identifier; and (v) the certification authority does not require parties to prove knowledge of the static private keys corresponding to their static public keys during registration.[3] The attack proceeds as follows.

(1) The adversary $\mathcal{M}$ induces $\hat{A}$ to create a session with a destination address $\hat{d}$ controlled by $\mathcal{M}$. In response, $\hat{A}$ selects ephemeral key pair (x, X) and sends $(\hat{d}, \hat{A}, X)$.
(2) $\mathcal{M}$ intercepts $(\hat{d}, \hat{A}, X)$ and does the following:
   (a) Compute $S = \{(\hat{C}, \overline{H}(X, \hat{C})) \mid \hat{C}$ is an honest party$\}$.
   (b) Select an identifier $\hat{M}$ (not the same as the identifier of an honest party) such that $(\hat{B}, \overline{H}(X, \hat{M})) \in S$ for some $\hat{B}$.
   (c) Select $M = B$ as $\hat{M}$'s static public key (note that $\mathcal{M}$ does not know the corresponding private key).
   (d) Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.
(3) $\mathcal{M}$ intercepts $\hat{B}$'s reply $(\hat{A}, \hat{B}, Y)$ and sends $(\hat{A}, \hat{M}, Y)$ to $\hat{A}$.

Party $\hat{A}$ computes the session key $k = H(\sigma_A)$, where $\sigma_A = (YM^e)^{x+da}$ and $d = \overline{H}(X, \hat{M})$ and $e = \overline{H}(Y, \hat{A})$. Party $\hat{B}$ computes the session key $k' = H(\sigma_B)$, where $\sigma_B = (XA^{d'})^{y+e'b}$ and $d' = \overline{H}(X, \hat{B})$ and $e' = \overline{H}(Y, \hat{A})$. Since $d' = d$, $e' = e$, and $M = B$, we have $\sigma_A = \sigma_B$ and hence $k' = k$. The problem is that while $\hat{B}$ correctly believes that k is shared with $\hat{A}$, party $\hat{A}$ mistakenly believes that k is shared with $\hat{M}$. Thus $\mathcal{M}$ has successfully launched an 'unknown key-share' or 'identity misbinding' attack on HMQV in the post model. The expected running time of the attack is about 2^{60} (for step 2b). Since most of the work has to be done online, the attack cannot be considered practical. Nevertheless it demonstrates that HMQV does not attain an 80-bit security level in the post model as it presumably does in the pre model.

The mechanisms of the attack were outlined in Remark 7.2 of [11]. However, the adversary considered in [11] operates in a different setting, namely the pre model where party $\hat{A}$ precomputes and stores her ephemeral public keys X which are then inadvertently leaked to $\mathcal{M}$ *before* $\hat{A}$ uses them in a session. Three countermeasures were proposed in [11] for foiling this attack: (i) increase the output length of $\overline{H}$ to 160 bits; (ii) include the identifiers $\hat{A}$, $\hat{B}$ in the key derivation function whereby the session key is computed as $k = H(\sigma, \hat{A}, \hat{B})$; and (iii) include random nonces (which are not precomputed and stored) in the derivation of exponents d and e, whereby the exponents are computed as $d = \overline{H}(X, \hat{B}, \nu_A)$ and $e = \overline{H}(Y, \hat{A}, \nu_B)$ where ν_A and ν_B are $\hat{A}$'s and $\hat{B}$'s nonces, respectively. Countermeasures (i) and (ii) are successful in thwarting the attack described above on HMQV in the post model. However, it can easily be seen that countermeasure (iii) does *not* prevent the attack in the post model, thus demonstrating that the two attacks are indeed different.

[3] In [11] it is noted that the HMQV security proof does not depend on the CA performing any proof-of-possession checks.

The reason countermeasure (iii) fails is that, unlike in the pre model, the peer's identifier is not known to $\hat{A}$ when she creates the session in the post model.

3.3. **Σ_0 protocol.** The Σ_0 protocol [5] is a simplified version of one of the IKE key agreement protocols. In the protocol description below, PRF is a pseudorandom function family, MAC is a message authentication code algorithm, and sig_A and sig_B are the signing algorithms for $\hat{A}$ and $\hat{B}$, respectively.

(1) Party $\hat{A}$ (the initiator) selects an ephemeral key pair (x, X), initializes the session identifier to $(\hat{A}, s)$, and sends $(\hat{d}_B, \hat{d}_A, s, X)$. Here $\hat{d}_A$ and $\hat{d}_B$ are destination addresses for $\hat{A}$ and $\hat{B}$, respectively.
(2) Upon receipt of $(\hat{d}_B, \hat{d}_A, s, X)$, $\hat{B}$ (the responder) selects an ephemeral key pair (y, Y), and computes $\sigma = X^y$, $k = \mathrm{PRF}_\sigma(0)$, and $k' = \mathrm{PRF}_\sigma(1)$. $\hat{B}$ then destroys y and σ, initializes the session identifier to $(\hat{B}, s)$, and sends $m_1 = (\hat{d}_A, \hat{B}, s, Y, \mathrm{sig}_B(\mathcal{R}, s, X, Y), \mathrm{MAC}_{k'}(\mathcal{R}, s, \hat{B}))$.
(3) Upon receiving m_1, $\hat{A}$ computes $\sigma = Y^x$, $k = \mathrm{PRF}_\sigma(0)$, and $k' = \mathrm{PRF}_\sigma(1)$. $\hat{A}$ then verifies the signature and MAC tag in m_1, and sends $m_2 = (\hat{B}, \hat{A}, s, \mathrm{sig}_A(\mathcal{I}, s, Y, X), \mathrm{MAC}_{k'}(\mathcal{I}, s, \hat{A}))$. Finally, $\hat{A}$ accepts the session key k with peer $\hat{B}$, and erases the session state.
(4) Upon receiving m_2, $\hat{B}$ verifies the signature and MAC tag in m_2, accepts the session k with peer $\hat{A}$, and erases the session state.

In [5], the Σ_0 protocol is proven secure in the post-specified peer model provided that the DDH assumption holds in G and that the PRF, MAC, and sig primitives are secure. However, the following attack described in [5] shows that Σ_0 is not secure in the pre-specified peer model.

(1) Create a session $(\hat{A}, \hat{B}, s)$ at $\hat{A}$.
(2) Intercept $\hat{A}$'s outgoing message $(\hat{B}, \hat{A}, s, X)$ and send $(\hat{B}, \hat{M}, s, X)$ to $\hat{B}$.
(3) Intercept $\hat{B}$'s response $(\hat{M}, \hat{B}, s, Y, S_B, t_B)$, where $S_B = \mathrm{sig}_B(\mathcal{R}, s, X, Y)$ and $t_B = \mathrm{MAC}_{k'}(\mathcal{R}, s, \hat{B})$, and send $(\hat{A}, \hat{B}, s, Y, S_B, t_B)$ to $\hat{A}$.
(4) The session $(\hat{A}, \hat{B}, s)$ at $\hat{A}$ completes and accepts k as the session key.
(5) Intercept and delete $\hat{A}$'s final message, and issue a *SessionStateReveal* query to the session $(\hat{B}, \hat{M}, s)$ thus learning k and k'.
(6) Issue the *Test* query to the session $(\hat{A}, \hat{B}, s)$ and use knowledge of k to win the distinguishing game.

Notice that the attack is legitimate in the pre-specified peer model since the exposed session $(\hat{B}, \hat{M}, s)$ is not matching to the test session $(\hat{A}, \hat{B}, s)$. On the other hand, such an attack is not permitted in the post-specified peer model because in step 5 of the attack the session $(\hat{B}, s)$ is still incomplete and therefore matching to the *Test* session (and thus cannot be exposed). This is all rather counterintuitive since one would expect that if a protocol is secure when the initiator does not have a priori knowledge of the peer's identifier, then it should remain secure when the peer's identifier is known at the outset.

One feature of both the pre and post models is that an exposed session does not produce any further output. In practice, however, one might desire the assurance that a particular session is secure even if the adversary learns some secret state information (such as an ephemeral private key) associated with that session or its matching session. For this reason, the security models in recent papers such as [11], [13] and [19] permit exposed sessions to continue producing output, and furthermore allow the adversary to issue a *SessionStateReveal* query (or its equivalent) to the *Test* session and its matching session (cf. §4.3). However, if the adversary $\mathcal{M}$ were equipped with these

extra capabilities, then the Σ_0 protocol would be insecure in both the pre and post models since $\mathcal{M}$ could issue a *SessionStateReveal* query to $(\hat{A}, s)$ after step 1 to learn x and thereafter compute the session key. Furthermore, the Σ_0 protocol falls in the post model to the following analogue of the attack described above. The attack is a little more realistic than the attack described above in the pre model because we now assume that the *SessionStateReveal* query does not yield the session key k (which may be stored in secure memory). $\mathcal{M}$'s actions are the following:

(1) Create a session $(\hat{A}, s)$ at $\hat{A}$ with peer destination address $\hat{d}_B$.
(2) Intercept $\hat{A}$'s outgoing message $(\hat{d}_B, \hat{d}_A, s, X)$ and send $(\hat{d}_B, \hat{d}_M, s, X)$ to $\hat{B}$.
(3) Intercept $\hat{B}$'s response $(\hat{d}_M, \hat{B}, s, Y, S_B, t_B)$, where $S_B = \text{sig}_B(\mathcal{R}, s, X, Y)$ and $t_B = \text{MAC}_{k'}(\mathcal{R}, s, \hat{B})$, and send $(\hat{d}_A, \hat{B}, s, Y, S_B, t_B)$ to $\hat{A}$.
(4) Intercept $\hat{A}$'s final message and delete it. The session $(\hat{A}, s)$ completes with peer $\hat{B}$ and session key k.
(5) Issue a *SessionStateReveal* query to the incomplete session $(\hat{B}, s)$ and learn the MAC key k'.
(6) Compute $S_M = \text{sig}_M(\mathcal{I}, s, Y, X)$ and $t_M = \text{MAC}_{k'}(\mathcal{I}, s, \hat{M})$, and send $(\hat{B}, \hat{M}, s, S_M, t_M)$ to $\hat{B}$.

The session $(\hat{B}, s)$ completes with peer $\hat{M}$ and session key k. Thus $\mathcal{M}$ has successfully launched an unknown key-share attack on Σ_0 in the post model. The two attacks demonstrate that a protocol proven secure in the post-specified peer model of [5] may no longer be secure if exposed sessions are allowed to continue producing output.

4. Combining the two models

In this section we introduce the notion of a modifiable key agreement protocol — protocols designed for the pre-specified peer model but which can be adapted with minor changes to be executable in the post-specified peer model. We also introduce the notion of a hybrid key agreement protocol, which simultaneously describes a modifiable protocol and its modification suitable for the post model. We then develop a security definition that, if satisfied by a hybrid protocol, guarantees that the associated protocols are secure in the pre and post models.

4.1. Modifiable protocols.

Consider a key agreement protocol Π designed for the pre model where the first outgoing message prepared by the initiator $\hat{A}$ is of the form $(\hat{B}, \hat{A}, \mathit{RoundOne})$. Then Π is said to be *modifiable* if *RoundOne* can be computed before the session is created at $\hat{A}$; in particular, this means that *RoundOne* does not depend on $\hat{B}$'s identifier or static public key.

A modifiable protocol Π can be easily adapted for the post-specified peer model by incorporating identity establishment into the protocol flows. The required changes are the following. The initiator $\hat{A}$, who is activated to create a session with a destination address $\hat{d}$ (and without knowledge of the recipient's identifier or static public key), sends $(\hat{d}, \hat{A}, \mathit{RoundOne})$ as her first outgoing message. Since this message contains the identifier $\hat{A}$, the responder has all the information he needs to prepare his first outgoing message as specified by Π. The responder appends his identifier to this outgoing message (if the message does not already contain the identifier). After $\hat{A}$ receives this reply, both $\hat{A}$ and the responder can proceed with Π without any further modifications. Notice that the modified protocol Π' has the same number of message flows as the original protocol Π; except for appending a public value to the first outgoing message, the remainder of the protocol remains the same.

As mentioned in §1, the class of modifiable key agreement protocols includes many of the protocols that have been proposed in the literature. However, not all key agreement protocols are modifiable; for example, protocol $\mathcal{P}$ defined in §3.1 is not modifiable. Furthermore, as demonstrated by the attack on HMQV in §3.2, security of a modifiable protocol Π in the pre model does not imply security of the modified protocol Π' in the post model.

4.2. **Hybrid protocols.** Suppose that Π is a modifiable key agreement protocol, and Π' its modification suitable for the post model. The specification of Π and Π' can be combined as described below, resulting in a protocol $\tilde{\Pi}$ called a *hybrid* protocol.

We use $\tilde{A}$ to denote either an identifier $\hat{A}$ or a destination address $\hat{d}$ that can be used to send messages to some party $\hat{A}$ whose identifier is not known to the sender; note that the address $\hat{d}$ may not necessarily be under $\hat{A}$'s control. In the description of $\tilde{\Pi}$, a session is created at initiator $\hat{A}$ via a message containing $(\hat{A}, \tilde{B})$. The first outgoing message from $\hat{A}$ is $(\tilde{B}, \hat{A}, \mathit{RoundOne})$. The responder $\hat{B}$ includes the identifiers $\hat{A}$ and $\hat{B}$ in his response, and the remainder of the protocol description is the same as for Π.

A hybrid protocol $\tilde{\Pi}$ can be specialized for the pre model by using an identifier $\hat{B}$ for $\tilde{B}$. Protocol $\tilde{\Pi}$ can also be specialized for the post model by using a destination address for $\tilde{B}$. An example of a hybrid protocol is given in §5.

4.3. **Combined security model.** This section describes a "combined" model and associated security definition that aims to simultaneously capture the security assurances of the pre- and post-specified peer models. That is, if a hybrid protocol $\tilde{\Pi}$ is proven secure with respect to the new definition, then its specializations Π and Π' are guaranteed to be secure when run in the pre and post models, respectively. More precisely, when run in the pre model, Π satisfies the extended Canetti-Krawczyk (eCK) definition [13] suitably enhanced to capture attacks where an adversary is able to learn ephemeral public keys of parties *before* they are actually used in a protocol session.[4] Such attacks were considered by Krawczyk [11], but were not incorporated into his security model. When run in the post model, the modified protocol Π' satisfies a strengthened version of the Canetti-Krawczyk definition from [5], suitably enhanced to offer security assurances similar to the eCK definition (including resistance to attacks where the adversary learns ephemeral private keys of the session being attacked) and to capture attacks where the adversary learns ephemeral public keys before they are actually used.

Instead of using pre-determined session numbers s to identify sessions (cf. §2.1), our session identifiers will consist of the identities of the communicating parties together with a concatenation of the messages exchanged during a protocol run. As shown in [6], this notion of session identifier yields a security model for key agreement that is at least as strong as other security models.

Notation. We assume that messages are represented as binary strings. If m is a vector then $\#m$ denotes the number of its components. We say that two vectors m_1 and m_2 are *matched*, written $m_1 \sim m_2$, if the first $t = \min\{\#m_1, \#m_2\}$ components of the vectors are pairwise equal as binary strings. We write $\hat{A} \equiv \tilde{D}$ if either $\tilde{D} = \hat{A}$ or if $\tilde{D}$ is a destination address that can be used to send messages to $\hat{A}$.

[4] As discussed in [11], such attacks may be possible in situations where a party precomputes ephemeral public keys in order to improve on-line performance.

Session creation. A party $\hat{A}$ can be activated via an incoming message to create a session. The incoming message has one of the following forms: (i) $(\hat{A}, \tilde{B})$ or (ii) $(\tilde{A}, \hat{B}, In)$. If $\hat{A}$ was activated with $(\hat{A}, \tilde{B})$ then $\hat{A}$ is the session initiator; otherwise $\hat{A}$ is the session responder.

Session initiator. If $\hat{A}$ is the session initiator then $\hat{A}$ creates a separate session state where session-specific short-lived data is stored, and prepares a reply Out that includes an ephemeral public key X. The session is labeled active and identified via a (temporary and incomplete) session identifier $s = (\hat{A}, \tilde{B}, \mathcal{I}, Comm)$ where $Comm$ is initialized to Out. The outgoing message is $(\tilde{B}, \hat{A}, Out)$.

Session responder. If $\hat{A}$ is the session responder then $\hat{A}$ creates a separate session state and prepares a reply Out that includes an ephemeral public key X. The session is labeled active and identified via a (temporary and incomplete) session identifier $s = (\hat{A}, \hat{B}, \mathcal{R}, Comm)$ where $Comm = (In, Out)$. The outgoing message is $(\hat{B}, \hat{A}, \mathcal{I}, In, Out)$.

Session update. A party $\hat{A}$ can be activated to update a session via an incoming message of the form $(\hat{A}, \hat{B}, role, Comm, In)$, where $role \in \{\mathcal{I}, \mathcal{R}\}$. Upon receipt of this message, $\hat{A}$ checks that she owns an active session with identifier $s = (\hat{A}, \hat{B}, role, Comm)$ or $s = (\hat{A}, \hat{d}, role, Comm)$ where $\hat{d}$ is a destination address; except with negligible probability, $\hat{A}$ can own at most one such session. If no such session exists then the message is rejected. If a session $s = (\hat{A}, \hat{d}, role, Comm)$ or $s = (\hat{A}, \hat{B}, role, Comm)$ exists, then in the former case $\hat{A}$ updates the session identifier to $s = (\hat{A}, \hat{B}, role, Comm)$; in either case, $\hat{A}$ updates s by appending In to $Comm$. If the protocol requires a response by $\hat{A}$, then $\hat{A}$ prepares the required response Out; the outgoing message is $(\hat{B}, \hat{A}, role, Comm, Out)$ where $role$ is $\hat{B}$'s role as perceived by $\hat{A}$, and the session identifier is updated by appending Out to $Comm$. If the protocol specifies that no further messages will be received, then the session completes and accepts a session key.

Matching sessions. Since ephemeral public keys are selected at random on a per-session basis, session identifiers are unique except with negligible probability. Party $\hat{A}$ is said to be the owner of a session $(\hat{A}, \tilde{B}, *, *)$. For a session $(\hat{A}, \hat{B}, *, *)$ we call $\hat{B}$ the session *peer*; together $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. Let $s = (\hat{A}, \tilde{B}, role_A, Comm_A)$ be a session owned by $\hat{A}$, where $role_A \in \{\mathcal{I}, \mathcal{R}\}$. A session $s^* = (\hat{C}, \tilde{D}, role_C, Comm_C)$, where $role_C \in \{\mathcal{I}, \mathcal{R}\}$, is said to be *matching* to s if $\hat{C} \equiv \tilde{B}$, $\hat{A} \equiv \tilde{D}$, $role_A \neq role_C$, and $Comm_C \sim Comm_A$. It can be seen that the session s, except with negligible probability, can have more than one matching session if and only if $Comm_A$ has exactly one component, i.e., is comprised of a single outgoing message.

Aborted sessions. A protocol may require parties to perform some checks on incoming messages. For example, a party may be required to perform some form of public key validation or verify a signature. If a party is activated to create a session with an incoming message that does not meet the protocol specifications, then that message is rejected and no session is created. If a party is activated to update an active session with an incoming message that does not meet the protocol specifications, then the party deletes all information specific to that session (including the session state and the session key if it has been computed) and *aborts* the session; such an abortion occurs before the session identifier can be updated. At any point in time a session is in exactly one of the following states: active, completed, aborted.

Adversary. The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. In particular, this means that $\hat{A} \equiv \hat{d}$ for all parties $\hat{A}$ and all destination addresses $\hat{d}$. Parties submit outgoing messages to $\mathcal{M}$, who makes decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of parties. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information $\mathcal{M}$ is allowed to make the following queries:

- *StaticKeyReveal*($\hat{A}$): $\mathcal{M}$ obtains $\hat{A}$'s static private key.
- *EphemeralKeyReveal*(s): $\mathcal{M}$ obtains the ephemeral private key held by session s.[5] We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that hold an ephemeral private key.
- *SessionKeyReveal*(s): If s has completed then $\mathcal{M}$ obtains the session key held by s. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that have completed.
- *EphemeralPublicKeyReveal*($\hat{A}$): $\mathcal{M}$ obtains the ephemeral public key that $\hat{A}$ will use the next time a session is created within $\hat{A}$.
- *EstablishParty*($\hat{A}, A$): This query allows $\mathcal{M}$ to register an identifier $\hat{A}$ and a static public key A on behalf of a party. The adversary totally controls that party, thus permitting the modeling of attacks by malicious insiders. Parties that were established by $\mathcal{M}$ using *EstablishParty* are called *corrupted* or *adversary controlled*. If a party is not corrupted it is said to be *honest*.

Adversary goal. To capture the indistinguishability requirement, $\mathcal{M}$ is allowed to make a special query *Test*(s) to a 'fresh' session s. In response, $\mathcal{M}$ is given with equal probability either the session key held by s or a random key. $\mathcal{M}$ meets its goal if it guesses correctly whether the key is random or not. Note that $\mathcal{M}$ can continue interacting with the parties after issuing the *Test* query, but must ensure that the test session remains fresh throughout $\mathcal{M}$'s experiment.

Definition 1. Let s be the identifier of a completed session, owned by an honest party $\hat{A}$ with peer $\hat{B}$, who is also honest. Let s^* be the identifier of the matching session of s, if it exists. Define s to be *fresh* if none of the following conditions hold:

(1) $\mathcal{M}$ issued *SessionKeyReveal*(s) or *SessionKeyReveal*(s^*) (if s^* exists).
(2) s^* exists and $\mathcal{M}$ issued one of the following:
  (a) Both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(s).
  (b) Both *StaticKeyReveal*($\hat{B}$) and *EphemeralKeyReveal*(s^*).
(3) s^* does not exist and $\mathcal{M}$ issued one of the following:
  (a) Both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(s).
  (b) *StaticKeyReveal*($\hat{B}$).

Definition 2. A key agreement protocol is *secure* if the following conditions hold:

(1) If two honest parties complete matching sessions then, except with negligible probability, they both compute the same session key.
(2) No polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than $\frac{1}{2}$ plus a negligible fraction.

[5]The *EphemeralKeyReveal* query can be made functionally equivalent to the *SessionStateReveal* query by defining the ephemeral private key to consist of all ephemeral secret data that a session may hold.

5. NAXOS-C protocol

In this section we present the hybrid version of the NAXOS-C key agreement protocol, which is essentially the NAXOS protocol of [13] augmented with key confirmation. In the protocol description, λ is the security parameters, and $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda \times \{0,1\}^\lambda$, $H_1 : \{0,1\}^* \rightarrow [1, q-1]$, and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ are hash functions. NAXOS-C can be specialized to run in either the pre or the post model. Moreover, it can be proven secure in the combined model of §4.3 provided that the GDH assumption holds in G and that the hash functions H, H_1 and H_2 are modeled as random functions (see Appendix A). Hence NAXOS-C is secure in both the pre- and post-specified peer models.

The purpose of presenting the NAXOS-C protocol is to demonstrate that the security definition of §4.3 is useful (and not too restrictive) in the sense that there exist practical protocols that meet the definition under reasonable assumptions. The protocol was designed to allow a straightforward (albeit tedious) reductionist security argument, and has not been optimized. In particular, not all the inputs to the hash functions H, H_1 and H_2 may be necessary for security, and in practice H_2 would be implemented as a MAC algorithm (with secret key k').

(1) Party $\hat{A}$ (the initiator) does the following:
   (a) Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\lambda$, and compute $x = H_1(a, \tilde{x})$ and $X = g^x$.
   (b) Destroy x.
   (c) Initialize the session identifier to $(\hat{A}, \tilde{B}, \mathcal{I}, X)$.
   (d) Send $(\tilde{B}, \hat{A}, X)$ to $\tilde{B}$.

(2) Upon receiving $(\tilde{B}, \hat{A}, X)$, party $\hat{B}$ (the responder) does the following:
   (a) Verify that $X \in G^*$.
   (b) Select an ephemeral private key $\tilde{y} \in_R \{0,1\}^\lambda$, and compute $y = H_1(b, \tilde{y})$ and $Y = g^y$.
   (c) Compute $\sigma_1 = A^y$, $\sigma_2 = X^b$ and $\sigma_e = X^y$.
   (d) Compute $(k, k') = H(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$ and $t_B = H_2(k', \mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
   (e) Destroy $\tilde{y}, y, \sigma_1, \sigma_2$ and σ_e.
   (f) Initialize the session identifier to $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, t_B)$.
   (g) Send $(\hat{A}, \hat{B}, X, Y, t_B)$ to $\hat{A}$.

(3) Upon receiving $(\hat{A}, \hat{B}, X, Y, t_B)$, party $\hat{A}$ checks that she owns an active session with identifier $(\hat{A}, \tilde{B}, \mathcal{I}, X)$. If so, then $\hat{A}$ does the following:
   (a) Verify that $Y \in G^*$.
   (b) Compute $x = H_1(a, \tilde{x})$, $\sigma_1 = Y^a$, $\sigma_2 = B^x$ and $\sigma_e = Y^x$.
   (c) Compute $(k, k') = H(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$.
   (d) Destroy $\tilde{x}, x, \sigma_1, \sigma_2$ and σ_e.
   (e) Verify that $t_B = H_2(k', \mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
   (f) Compute $t_A = H_2(k', \mathcal{I}, \hat{A}, \hat{B}, X, Y)$.
   (g) Destroy k'.
   (h) Send $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$ to $\hat{B}$.
   (i) Update the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, X, Y, t_B, t_A)$ and complete the session by accepting k as the session key.

(4) Upon receiving $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$, party $\hat{B}$ checks that he owns an active session with identifier $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, t_B)$. If so, then $\hat{B}$ does the following:
   (a) Verify that $t_A = H_2(k', \mathcal{I}, \hat{A}, \hat{B}, X, Y)$.

(b) Destroy k'.
(c) Update the session identifier to $(\hat{B}, \hat{A}, \mathcal{R}, X, Y, t_B, t_A)$ and complete the session by accepting k as the session key.

6. Conclusions

We compared the Canetti-Krawczyk pre- and post-specified peer models for key agreement, and demonstrated that security in one model does not guarantee security or even executability in the other model. We also presented a combined security model and definition that simultaneously encompasses strengthened versions of the Canetti-Krawczyk definitions. The new definition is stronger in that it permits the adversary to learn ephemeral public keys before they are used, and to learn secret information from the session being attacked. Useful directions for future research would be the development of an optimized protocol that satisfies the new security definition, perhaps modified to allow for identity concealment, and the extension of the definition to capture a wider class of key agreement protocols.

References

[1] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science, 775 (1994), 232-249. Full version available at http://www.cs.ucdavis.edu/~rogaway/papers/eakd-abstract.html.
[2] C. Boyd, W. Mao and K. Paterson, "Key agreement using statically keyed authenticators", *Applied Cryptography and Network Security – ACNS 2004*, Lecture Notes in Computer Science, 3089 (2004), 248-262.
[3] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*, Springer, 2003.
[4] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels", *Advances in Cryptology – EUROCRYPT 2001*, Lecture Notes in Computer Science, 2045 (2001), 453-474. Full version available at http://eprint.iacr.org/2001/040.
[5] R. Canetti and H. Krawczyk, "Security analysis of IKE's signature based key-exchange protocol", *Advances in Cryptology – CRYPTO 2002*, Lecture Notes in Computer Science, 2442 (2002), 143-161. Full version available at http://eprint.iacr.org/2002/120.
[6] K. Choo, C. Boyd and Y. Hitchcock, "Examining indistinguishability-based proof models for key establishment protocols" *Advances in Cryptology – ASIACRYPT 2005*, Lecture Notes in Computer Science, 3788 (2005), 585-604.
[7] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2 (1992), 107-125.
[8] D. Harkins and D. Carrel, "The internet key exchange (IKE)", RFC 2409, Internet Engineering Task Force, 1998.
[9] C. Kaufman (editor), "Internet key exchange (IKEv2) protocol", RFC 4306, Internet Engineering Task Force, 2005.
[10] H. Krawczyk, "SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols", *Advances in Cryptology – CRYPTO 2003*, Lecture Notes in Computer Science, 2729 (2003), 400-425.
[11] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol", Cryptology ePrint Archive, Report 2005/176, available at http://eprint.iacr.org/2005/176. A short version appears in *Advances in Cryptology – CRYPTO 2005*, Lecture Notes in Computer Science, 3621 (2005), 546-566.
[12] H. Krawczyk, "HMQV in IEEE P1363", submission to the IEEE P1363 working group, July 7 2006. Available at http://grouper.ieee.org/groups/1363/P1363-Reaffirm/submissions/krawczyk-hmqv-spec.pdf.
[13] B. LaMacchia, K. Lauter and A. Mityagin, "Stronger security of authenticated key exchange", *ProvSec 2007*, Lecture Notes in Computer Science, 4784 (2007), 1-16.
[14] K. Lauter and A. Mityagin, "Security analysis of KEA authenticated key exchange", *Public Key Cryptography – PKC 2006*, Lecture Notes in Computer Science, 3958 (2006), 378-394.

[15] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", *Designs, Codes and Cryptography*, 28 (2003), 119-134.
[16] A. Menezes and B. Ustaoglu, "Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard", Proceedings of *ASIACCS '08*, ACM Press, 261-270.
[17] T. Okamoto, "Authenticated key exchange and key encapsulation in the standard model", *Advances in Cryptology – ASIACRYPT 2007*, Lecture Notes in Computer Science, 4833 (2007), 474-484.
[18] SP 800-56A *Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, National Institute of Standards and Technology, March 2006.
[19] B. Ustaoglu, "Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS", *Designs, Codes and Cryptography*, 46 (2008), 329-342.

Appendix A. Security proof for the NAXOS-C protocol

Theorem 1. *If H, H_1 and H_2 are modeled as random oracles, and G is a group where the GDH assumption holds, then NAXOS-C is secure in the combined model.*

Proof. Verifying that NAXOS-C satisfies condition 1 of Definition 2 is straightforward. We now verify that condition 2 of Definition 2 is satisfied — that no polynomially bounded adversary can distinguish the session key of a fresh session from a randomly chosen session key. Let λ denote the security parameter, and let $\mathcal{M}$ be a polynomially (in λ) bounded adversary. The adversary $\mathcal{M}$ is said to be successful with non-negligible probability if $\mathcal{M}$ wins the distinguishing game with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible. The event that $\mathcal{M}$ is successful is denoted by M. Let the test session be $s = (\hat{A}, \hat{B}, \mathcal{I}, X, Y, t_B, t_A)$ or $s = (\hat{B}, \hat{A}, \mathcal{R}, X, Y, t_B, t_A)$, and let H^* be the event that $\mathcal{M}$ queries H with $(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$. Let $\overline{H^*}$ be the complement of event H^*, and let s^* be any completed session owned by an honest party such that $s^* \neq s$ and s^* is non-matching to s. Since s^* and s are distinct and non-matching, it can be seen that the inputs to the key derivation function H are different for s and s^*. And, since H is a random oracle, it follows that $\mathcal{M}$ cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence $\Pr(M \wedge \overline{H^*}) \leq \frac{1}{2}$ and

$$\Pr(M) = \Pr(M \wedge H^*) + \Pr(M \wedge \overline{H^*}) \leq \Pr(M \wedge H^*) + \frac{1}{2},$$

whence $\Pr(M \wedge H^*) \geq p(\lambda)$. We will henceforth denote the event $M \wedge H^*$ by M^*.

Assume further that $\mathcal{M}$ succeeds in an environment with n parties, activates at most t sessions within a party, makes at most h, h_1, h_2 queries to oracles H, H_1, H_2 respectively, and terminates after time at most $\mathcal{T}_{\mathcal{M}}(\lambda)$.

The following conventions will be used for the remainder of the security argument. The DDH oracle on input (g^a, g^b, g^c) returns the bit 1 if $g^{ab} = g^c$ and the bit 0 otherwise. Also, $\xi : G \times G \rightarrow G$ is a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$ for all $X, Y \in G$. The algorithm $\mathcal{S}$, which simulates $\mathcal{M}$'s environment, will use $\xi(X, Y)$ to "represent" $\mathrm{CDH}(X, Y)$ in situations where $\mathcal{S}$ may not know $\log_g X$ and $\log_g Y$. Except with negligible probability, $\mathcal{M}$ will not detect that $\xi(X, Y)$ is being used instead of $\mathrm{CDH}(X, Y)$.

We use $\mathcal{M}$ to construct a CDH solver $\mathcal{S}$ that succeeds with non-negligible probability. We begin by considering the following complementary events:

E_1: There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries H_1 with $(b, *)$ before issuing a *StaticKeyReveal*$(\hat{B})$ query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*$(\hat{B})$ query.)

E_2: During its execution, for every party $\hat{B}$ for which $\mathcal{M}$ queries H_1 with $(b, *)$, $\mathcal{M}$ issued *StaticKeyReveal*($\hat{B}$) before the first $(b, *)$ query to H_1.

Since $\Pr(M^*)$ is non-negligible, it must be the case that either event $E_1 \wedge M^*$ or $E_2 \wedge M^*$ occurs with non-negligible probability. These possibilities are considered in §A.1 and §A.2.

A.1. **Event $E_1 \wedge M^*$.** We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability provided that event $E_1 \wedge M^*$ occurs with non-negligible probability.

$\mathcal{S}$ begins by establishing n parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs. For each honest party $\hat{A}$, $\mathcal{S}$ maintains a list of at most t ephemeral key pairs, and two markers – a party marker and an adversary marker. The list is initially empty, and the markers initially point to the first entry of the list. Whenever $\hat{A}$ is activated to create a new session, $\mathcal{S}$ checks if the party marker points to an empty entry. If so then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in step 1 or 2 of the NAXOS-C protocol. If the list entry is not empty, then $\mathcal{S}$ uses the ephemeral key pair in that list entry for the newly created session. In either case the party marker is updated to point to the next list entry, and the adversary marker is also advanced if it points to an earlier entry. If $\mathcal{M}$ issues an *EphemeralPublicKeyReveal* query, then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in step 1 or 2 of the NAXOS-C protocol. $\mathcal{S}$ stores the key pair in the entry pointed to by the adversary marker, returns the public key as the query response, and advances the adversary marker.

$\mathcal{S}$ activates the adversary $\mathcal{M}$ on this set of parties and awaits the actions of $\mathcal{M}$. We next describe the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*($\hat{A}, \tilde{B}$): $\mathcal{S}$ executes step 1 of the protocol.
(2) *Send*($\tilde{B}, \hat{A}, X$) issued to $\hat{B}$: $\mathcal{S}$ executes step 2 of the protocol. However, if $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $\sigma_2 = \xi(V, X)$.
(3) *Send*($\hat{A}, \hat{B}, X, Y, t_B$): $\mathcal{S}$ executes step 3 of the protocol. However, if $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $\sigma_1 = \xi(V, Y)$.
(4) *Send*($\hat{B}, \hat{A}, X, Y, t_B, t_A$): $\mathcal{S}$ executes step 4 of the protocol.
(5) $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way, by returning a random value from the range of H_2 for new queries and replaying answers if the queries were made before.
(6) $H_1(z, *)$: For every new query, $\mathcal{S}$ computes $Z = g^z$. If $Z = V$, then $\mathcal{S}$ aborts and outputs $\mathrm{CDH}(U, V) = U^z$; otherwise $\mathcal{S}$ simulates a random oracle in the usual way.
(7) $H(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:
   (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(V, Y, \sigma_1)$.
      (i) If $\tau_1 = 1$, $\mathcal{S}$ returns $H(\hat{A}, \hat{B}, X, Y, \xi(V, Y), \sigma_2, \sigma_e)$.
      (ii) If $\tau_1 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(V, X, \sigma_2)$.
      (i) If $\tau_2 = 1$, $\mathcal{S}$ returns $H(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(V, X), \sigma_e)$.
      (ii) If $\tau_2 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
(8) *EphemeralPublicKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.
(9) *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

(10) *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
(11) *StaticKeyReveal*$(\hat{A})$: If $\hat{A} = \hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
(12) *EstablishParty*$(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
(13) *Test*(s): $\mathcal{S}$ responds to the query faithfully.
(14) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. If $\mathcal{S}$ assigns V to an honest party $\hat{B}$ for whom $\mathcal{M}$ will query $H_1(v, *)$ without first issuing a *StaticKeyReveal*$(\hat{B})$ query, then $\mathcal{S}$ is successful as described in step 6 and abortions as in steps 11 and 14 do not occur. Hence, if event $E_1 \wedge M^*$ occurs with probability p_1, then $\mathcal{S}$ is successful with probability $\Pr(S)$ that is bounded by

$$\Pr(S) \geq \frac{1}{n} p_1. \tag{1}$$

During the simulation, $\mathcal{S}$ performs group exponentiations, accesses the DDH oracle, and simulates three random oracles. Let $q = \Theta(2^\lambda)$. Then a group exponentiation takes time $\mathcal{T}_G = O(\lambda)$. We assume that a DDH oracle call takes polynomial time $\mathcal{T}_{\text{DDH}}(\lambda)$. Similarly, simulating oracles H, H_1 and H_2 take polynomially bounded time $\mathcal{T}_H(\lambda)$, $\mathcal{T}_{H_1}(\lambda)$ and $\mathcal{T}_{H_2}(\lambda)$, respectively. Therefore the running time $\mathcal{T}_{\mathcal{S}}$ of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (5\mathcal{T}_G + 2\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \tag{2}$$

A.2. **Event $E_2 \wedge M^*$.** Let T_m be the event "the test session has a matching session owned by an honest party". We further subdivide event $E_2 \wedge M^*$ into the following complementary events: (i) $E_{2a} = E_2 \wedge M^* \wedge T_m$ and (ii) $E_{2b} = E_2 \wedge M^* \wedge \overline{T_m}$. Let $p_2 = \Pr(E_2 \wedge M^*)$, $p_{2a} = \Pr(E_{2a})$, and $p_{2b} = \Pr(E_{2b})$. Since E_{2a} and E_{2b} are complementary events we have $p_2 = p_{2a} + p_{2b}$. Therefore, if event $E_2 \wedge M^*$ occurs with non-negligible probability, then either E_{2a} or E_{2b} occurs with non-negligible probability. The two events are considered in §A.2.1 and §A.2.2.

A.2.1. *Event E_{2a}.* We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability, provided that event E_{2a} occurs with non-negligible probability.

$\mathcal{S}$ establishes n parties that are assigned random static key pairs. $\mathcal{S}$ randomly selects two parties $\hat{C}, \hat{D}$ and two integers $i, j \in_R [1, t]$ subject to the condition that $(\hat{C}, i) \neq (\hat{D}, j)$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in §A.1, with the following exceptions. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\lambda$; note that $\mathcal{S}$ does not know $u = \log_g U = H_1(c, \tilde{u})$. Similarly, the jth ephemeral public key selected on behalf of $\hat{D}$ is V and the corresponding ephemeral private key is $\tilde{v} \in_R \{0,1\}^\lambda$; $\mathcal{S}$ does not know $v = \log_g V = H_1(d, \tilde{v})$. The sessions that are subsequently activated with ephemeral public keys U and V are denoted by s^U and s^V, respectively. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*$(\hat{A}, \tilde{B})$: $\mathcal{S}$ executes step 1 of the protocol.
(2) *Send*$(\tilde{B}, \hat{A}, X)$ issued to $\hat{B}$: $\mathcal{S}$ executes step 2 of the protocol. If the session created is s^U or s^V, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_1 = \xi(A, Y)$ and $\sigma_e = \xi(X, Y)$ where $Y \in \{U, V\}$ is the ephemeral public key of the created session.

(3) *Send*$(\hat{A}, \hat{B}, X, Y, t_B)$: $\mathcal{S}$ executes step 3 of the protocol. However, if $X \in \{U, V\}$, then $\mathcal{S}$ deviates by not computing $x = H_1(a, \tilde{x})$ in step 3(b) and by setting $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.
(4) *Send*$(\hat{B}, \hat{A}, X, Y, t_B, t_A)$: $\mathcal{S}$ executes step 4 of the protocol.
(5) $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
(6) $H_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of s^U, or if $\tilde{x} = \tilde{v}$ and $\hat{A}$ is the owner of s^V, in which case $\mathcal{S}$ aborts with failure.
(7) $H(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:
   (a) If $\{X, Y\} = \{U, V\}$ and $\mathrm{DDH}(X, Y, \sigma_e) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\sigma_e = \mathrm{CDH}(U, V)$.
   (b) If $X \in \{U, V\}$ and either $\sigma_2 \neq \xi(B, X)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau_2 = 1$ if either $\mathrm{DDH}(B, X, \sigma_2) = 1$ or $\sigma_2 = \xi(B, X)$; otherwise set $\tau_1 = 0$. Similarly, set $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$, and $\tau_e = 0$ otherwise.
      (i) If $\tau_2 = 1$ and $\tau_e = 1$, $\mathcal{S}$ returns $(H(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(B, X), \xi(X, Y))$.
      (ii) If $\tau_2 \neq 1$ or $\tau_e \neq 1$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) If $Y \in \{U, V\}$ and either $\sigma_1 \neq \xi(A, Y)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau_1 = 1$ if either $\mathrm{DDH}(A, Y, \sigma_1) = 1$ or $\sigma_1 = \xi(A, Y)$; otherwise set $\tau_1 = 0$. Similarly, set $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$, and $\tau_e = 0$ otherwise.
      (i) If $\tau_1 = 1$ and $\tau_e = 1$, $\mathcal{S}$ returns $H(\hat{A}, \hat{B}, X, Y, \xi(A, Y), \sigma_2, \xi(X, Y))$.
      (ii) If $\tau_1 \neq 1$ or $\tau_e \neq 1$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
(8) *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
(9) *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
(10) *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
(11) *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
(12) *EstablishParty*$(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
(13) *Test*(s): If the ephemeral public keys used in s are not $\{U, V\}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
(14) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects s^U or s^V as the test session and the other as its matching session is at least $2/(nt)^2$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 13), and suppose that event E_{2a} occurs. Let $\hat{A}$ and $\hat{B}$ denote the communicating parties for the test session and, without loss of generality, let $\hat{A}$ be the test session owner and U her ephemeral public key. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an H_1 query that includes $\tilde{u}$. Similarly, $\mathcal{M}$ must obtain $\tilde{v}$ from the matching session via an *EphemeralKeyReveal* query before making an H_1 query that includes $\tilde{v}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in each of the pairs $(a, \tilde{u})$ and $(b, \tilde{v})$; hence $\mathcal{S}$ does not abort as described in step 6. Under event M^*, except with negligible probability of guessing $\xi(U, V)$, $\mathcal{S}$ is successful as described in step 7(a)

and does not fail as described in step 14. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{2}{(nt)^2} p_{2a}. \tag{3}$$

The running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (4\mathcal{T}_G + 3\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\,\mathcal{T}_{\mathcal{M}}. \tag{4}$$

A.2.2. *Event $E_{2b} \wedge M^*$.* We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability, provided that event $E_{2b} \wedge M^*$ occurs with non-negligible probability.

$\mathcal{S}$ establishes n parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects a party $\hat{C}$ and integer $i \in_R [1, t]$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in §A.1, with the following exception. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\lambda$; note that $\mathcal{S}$ does not know $u = \log_g U = H_1(c, \tilde{u})$. The session that is subsequently activated with ephemeral public key U is denoted by s^U. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*$(\hat{A}, \tilde{B})$: $\mathcal{S}$ executes step 1 of the protocol.
(2) *Send*$(\tilde{B}, \hat{A}, X)$ issued to $\hat{B}$: $\mathcal{S}$ executes step 2 of the protocol. However, if $\hat{B} = \hat{V}$ then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_2 = \xi(V, X)$. Also, if the session created is s^U, then $\mathcal{S}$ sets $\sigma_1 = \xi(A, Y)$ and $\sigma_e = \xi(X, Y)$ where $Y = U$ is the ephemeral public key of the created session.
(3) *Send*$(\hat{A}, \hat{B}, X, Y, t_B)$: $\mathcal{S}$ executes step 3 of the protocol. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ deviates by setting $\sigma_1 = \xi(A, Y)$. Also, if $X = U$, then $\mathcal{S}$ deviates by not computing $x = H_1(a, \tilde{x})$ in step 3(b) and sets $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.
(4) *Send*$(\hat{B}, \hat{A}, X, Y, t_B, t_A)$: $\mathcal{S}$ executes step 4 of the protocol.
(5) $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
(6) $H_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of s^U, in which case $\mathcal{S}$ aborts with failure.
(7) $H(\hat{A}, \hat{B}, X, Y, \sigma_1, \sigma_2, \sigma_e)$:
   (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, $\mathcal{S}$ obtains $\tau_1 = \text{DDH}(V, Y, \sigma_1)$.
      (i) If $\tau_1 = 1$ and $Y = U$, $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_1$.
      (ii) If $\tau_1 = 1$ and $Y \neq U$, $\mathcal{S}$ returns $H(\hat{A}, \hat{B}, X, Y, \xi(V, Y), \sigma_2, \sigma_e)$.
      (iii) If $\tau_1 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, $\mathcal{S}$ obtains $\tau_2 = \text{DDH}(V, X, \sigma_2)$.
      (i) If $\tau_2 = 1$ and $X = U$, $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_2$.
      (ii) If $\tau_2 = 1$ and $X \neq U$, $\mathcal{S}$ returns $H(\hat{A}, \hat{B}, X, Y, \sigma_1, \xi(V, X), \sigma_e)$.
      (iii) If $\tau_2 = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
(8) *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ to the query faithfully.
(9) *EphemeralKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.
(10) *SessionKeyReveal*(s): $\mathcal{S}$ responds to the query faithfully.

(11) *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
(12) *EstablishParty*($\hat{E}, E$): $\mathcal{S}$ responds to the query faithfully.
(13) *Test*(s): If the peer of s is not $\hat{V}$ or the outgoing ephemeral public key is not U then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
(14) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. The probability that the test session has peer $\hat{V}$ and outgoing ephemeral public key U is at least $1/(n^2t)$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 13), and suppose that event E_{2b} occurs. Let $\hat{A}$ be the owner of the test session s^U. The experiment may fail if $\mathcal{M}$ queries H_1 with $(a, \tilde{u})$. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an H_1 query that includes $\tilde{u}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in the pair $(a, \tilde{u})$; hence $\mathcal{S}$ does not abort as described in step 6. Since the test session is fresh, $\mathcal{M}$ is not allowed to issue a *StaticKeyReveal* query to $\hat{V}$; hence $\mathcal{S}$ does not abort as described in step 11.

Now, $\hat{A}$ receives an incoming tag t_V before the test session completes. In event $\overline{T_m}$ the test session has no matching session. Since key confirmation tags are not repeated except with negligible probability, $\mathcal{M}$ must have computed tag t_V. Since t_V is an output of a random oracle H_2, $\mathcal{M}$ must have obtained the input k' by querying H with $\mathrm{CDH}(U, V)$. In this case, $\mathcal{S}$ is successful as described in step 7 and does not abort in step 14.

The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{1}{(n^2t)} p_{2b}, \tag{5}$$

and the running time of $\mathcal{S}$ is bound by

$$\mathcal{T}_{\mathcal{S}} \leq (4\mathcal{T}_G + 2\mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \tag{6}$$

A.3. **Overall analysis.** Combining the results from §A.1, §A.2.1 and §A.2.2, we conclude that for every adversary $\mathcal{M}$ there is an algorithm $\mathcal{S}$ that solves the GDH instance with running time $\mathcal{T}_{\mathcal{S}}$ and success probability $\Pr(\mathcal{S})$, where

$$\Pr(\mathcal{S}) \geq \max\left\{ \frac{1}{n} p_1, \frac{2}{(nt)^2} p_{2a}, \frac{1}{n^2 t} p_{2b} \right\} \tag{7}$$

and

$$\mathcal{T}_{\mathcal{S}} \leq (5\mathcal{T}_G + 3\mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \quad \square \tag{8}$$

Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
E-mail address: ajmeneze@uwaterloo.ca

Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
E-mail address: bustaoglu@uwaterloo.ca