Session-StateReveal is stronger than eCK's EphemeralKeyReveal: Using automatic analysis to attack the NAXOS protocol

Cas J.F. Cremers

Department of Computer Science, ETH Zurich
8092 Zurich, Switzerland
E-mail: cas.cremers@inf.ethz.ch

Preprint, January 2011

Abstract

In the paper "Stronger Security of Authenticated Key Exchange" [11, 12], a new security model for authenticated key exchange protocols (eCK) is proposed. The new model is suggested to be at least as strong as previous models for key exchange protocols, such as the CK model [5, 10]. The model includes a new notion of an EphemeralKeyReveal adversary query, which is claimed in e. g. [11, 17, 18] to be at least as strong as the Session-StateReveal query.

We investigate the relation between the two models by focusing on the difference in adversary queries. We formally model the NAXOS protocol and a variant of the eCK model, called eCK', in which the EphemeralKeyReveal query is replaced by the Session-StateReveal query. Using Scyther, a formal protocol analysis tool, we automatically find attacks on the protocol, showing that the protocol is insecure in the eCK' model. Our attacks prove that the Session-StateReveal query is stronger than the EphemeralKeyReveal query and that the eCK security model is incomparable to the CK model, disproving several claims made in the literature.

1 Introduction

In the area of secure key agreement protocols many security models [2, 4, 5, 6, 12, 15] and protocols have been proposed. Many of the proposed protocols have been shown to be secure in some particular security model, but have also shown to be insecure in others. In order to determine the exact properties that are required from such protocols, a single unified security model would be desirable. However, given the recent works such as [15], it seems that a single model is still not agreed upon.

In this paper we focus on a specific aspect of security models for key agreement protocols, namely, the ability of the adversary to learn the contents of the local state of an agent. For example, when an agent chooses a random value, or computes a hash function with certain input, the constituents of the computation reside temporarily in the local memory of the agent. It may be possible for the adversary to learn such information, even though he cannot learn the long-term private keys of the agent. A corresponding real-world scenario occurs when the long-term private keys of an agent reside in, e. g., a hardware security module (HSM), while the remainder of the protocol computations are done in regular (unprotected) memory. The ability of an adversary to learn the contents of the unprotected memory is captured in security models for key agreement protocols by the Session-StateReveal query [5, 10].

A drawback of the Session-StateReveal query in current security models is that it is underspecified in current models. For example, in the Canetti-Krawczyk (CK) model [5], Session-StateReveal is defined as giving the adversary the internal state of the Turing machine that executes the protocol. This internal state is not defined within the security model. Effectively, the definition of the internal state is postponed to the proof of a particular protocol.

To illustrate some of the subtleties involved, consider a protocol step that computes $H_2(H_1(sk, x))$, where H_1 and H_2 are hash functions, sk a long-term private key, and x is a freshly generated random bit string. Some constraints on the session state follow from straightforward assumptions: If an HSM is deployed, it should at least protect the long-term secrets, and we therefore assume that sk resides only in protected memory. As a result, H_1 cannot be computed in unprotected memory, because in that case its inputs, including sk, could be exposed. In general, the contents of the session state strongly depend on implementation details (which may be underspecified at design time): If the HSM is capable of generating random numbers or computing hashes, implementations may exist that respectively generate x or compute H_2 inside the HSM. If H_2 is not computed inside an HSM, it may be that its input resides only in a CPU register. Depending on the threat model, CPU registers may or may not be considered part of the session state. Additionally, any partial computations during the computation of H_2 may be considered part of the state. Even if we ignore these partial computations, the session state could reasonably be defined as any subset of $\{x, H_1(sk, x), H_2(H_1(sk, x))\}$: each of these choices would match to a particular implementation scenario and corresponding threat model.

Despite this high level of flexibility in modeling, the vast majority of existing proofs that use the Session-StateReveal query (e. g., [5, 10]) assume that the session state only contains freshly generated random bit strings. In the example above, this corresponds to defining the session state to be just $\{x\}$; a possible real-world interpretation is that all computations are performed in an HSM that does not generate fresh random numbers. Instead, the HSM retrieves the random numbers from unprotected memory. Any intermediate computations are assumed to be performed inside the HSM. Thus, though the query can be used to model many practically relevant scenarios, it is mainly used to model

limited adversaries that cannot reveal the result of intermediate computations.

In [11, 12] a new security model is proposed that is claimed to be stronger than existing AKE (Authenticated Key Exchange) security models. The model is based on the CK model, and is referred to in [12] as the Extended Canetti-Krawczyk (eCK) model. The main difference from the CK model is that in the eCK model the adversary is allowed to reveal part of the local state of participants for *all* sessions. A further difference is that the eCK model replaces the Session-StateReveal query from the CK model by a new EphemeralKeyReveal query.

Replacing the Session-StateReveal query by the EphemeralKeyReveal query in the eCK model addresses the underspecification of the session-state in the CK model, i. e., the eCK model explicitly defines what is revealed by the Ephemeral-KeyReveal query. In particular, the ephemeral key is defined to contain all secret session-specific information. The authors argue for the new EphemeralKeyReveal query that "by setting the ephemeral secret key equal to all session-specific secret information, we seem to cover all definitions of Session-StateReveal queries which exist in literature" [11, p. 2]. Similar arguments can be found in [14, 17, 18, 20]. Within the resulting eCK model, the NAXOS protocol is proposed and proven secure in [12].

Whereas most works argue that eCK is stronger than CK, it is informally argued in [3] that strictly speaking the eCK and CK models are incomparable. Regarding the difference between Session-StateReveal and EphemeralKeyReveal, it is remarked that "The important point to note is that the ephemeral-key does not include session state that has been computed using the long-term secret of the party. This is not the case in the CK model where, in principle, the adversary is allowed access to all the inputs (including the randomness, but excluding the long-term secret itself) and the results of all the computations done by a party as part of a session" [3, Section 3.1]. This observation in itself does not prove that there is a practical difference, i. e., that there exists a protocol that is secure in the eCK model but that is insecure in the CK model.

The aim of this work is to clarify the differences between the two queries and their practical implications.

Contributions

- We formally model the NAXOS protocol and a variant of the eCK model, called eCK', in which we replace EphemeralKeyReveal by Session-State-Reveal.
- We give a procedure to automatically determine the possible session-state contents of a protocol.
- Using Scyther, [7], a formal protocol analysis tool, we analyze the NAXOS protocol with respect to the eCK' model and using the automatically inferred session-state. The tool automatically finds two novel attacks on the NAXOS protocol. The attacks show that NAXOS is insecure in the eCK' model as well as the CK model.

- We show that contrary to the claims in [11, 14, 17, 18, 20], the Ephemeral-KeyReveal query is weaker than the Session-StateReveal query. Combined with the fact that the CK model does not allow the compromise of the ephemeral key of the tested session, we show that the CK and eCK models are incomparable.
- Finally, we show how our attacks can be extended to the KEA protocol and several of its descendants, and give practical interpretations of the Session-StateReveal and EphemeralKeyReveal queries.

We proceed as follows. In Section 2 we introduce notational conventions, and present the eCK security model and the NAXOS protocol. Then, in Section 3 we formalize the protocol and its session-state. We use the protocol analysis tool to find two attacks on this protocol that use Session-StateReveal, and discuss their implications. We provide some possible practical interpretations of Session-StateReveal and EphemeralKeyReveal in Section 4, and we conclude in Section 5. In the appendix we provide the full input files for the protocol analysis tool.

Acknowledgements This work is supported by the Hasler foundation within ManCom project number 2071 and by ETH Research Grant ETH-30 09-3. The author would like to thank David Basin, Mark Manulis, Kenny Paterson, Björn Tackmann, and Berkant Ustaoğlu for inspiring discussions.

2 The eCK security model and the NAXOS protocol

2.1 Preliminaries

Let f be a function. We write $dom(f)$ and $ran(f)$ to denote f's domain and range. We write $f[b \hookleftarrow a]$ to denote f's update, i. e., the function f' where $f'(x) = b$ when $x = a$ and $f'(x) = f(x)$ otherwise. We write $f : X \nrightarrow Y$ to denote a partial function from X to Y. For any set S, $\mathcal{P}(S)$ denotes the power set of S and S^* denotes the set of finite sequences of elements from S. We write $\langle s_0, \ldots, s_n \rangle$ to denote the sequence of elements s_0 to s_n and we omit brackets when no confusion can result. For s a sequence of length $|s|$ and $i < |s|$, s_i denotes the i-th element. We write $s \hat{} s'$ for the concatenation of the sequences s and s'. For the sequence $s = \langle s_0, \ldots, s_n \rangle$ we define $set(s) = \{s_0, \ldots, s_n\}$. Abusing set notation, we write $e \in s$ to denote $e \in set(s)$ when s is a sequence.

Table 1 shows additional notational conventions used in protocol specifications and attack descriptions. We follow the notation from [12] where possible.

2.2 The eCK security model

The NAXOS protocol that we present in the next section was proven secure in the eCK model from [11, 12]. We recall the definition of the eCK model below.

$\mathcal{A}, \mathcal{B}$	The *initiator* and *responder* roles of the protocol.
a, b	Agents (participants) executing roles of the protocol.
G	A cyclic group of known prime order q.
g	A generator of the group G.
sk_{a}	The long-term private key of the agent a, where $sk_{\mathsf{a}} \in \mathbb{Z}_q$.
pk_{a}	The long-term public key of the agent a, where $pk_{\mathsf{a}} = g^{sk_{\mathsf{a}}}$.
λ	A constant.
H_1, H_2	Hash functions, where $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \to \{0,1\}^\lambda$.
$esk_{\mathsf{a}}, esk'_{\mathsf{a}}$	Two different ephemeral keys of the agent a, generated in different sessions.
$\circ$	Written in place of a (bigger) term that is not relevant for the explanation at that point.
$x \xleftarrow{\$} S$	The variable x is drawn uniformly from the set S.
$x \leftarrow e$	The variable x is assigned the result of the expression e.

Table 1: Notation

The core ingredient of most AKE security models is the definition of the security experiment. The experiment describes the possible interactions between the adversary and regular agents performing the protocol. In the experiment, the adversary should have a negligible advantage in distinguishing the session key from a random key. Additionally, it is required that when the protocol is executed among two participants in the absence of an adversary, the participants compute the same key.

Participants can perform roles of the protocol (such as initiator, $\mathcal{A}$, or responder, $\mathcal{B}$) multiple times, with various other partners. A single role instance performed by a participant is called a *session*. Each session is associated with a session identifier.

Definition 1. (Session identifier) The session identifier of a session sid is defined as the tuple $(role, ID, ID*, comm_1, \ldots, comm_n)$, where $role$ is the role performed by the session (here initiator or responder), ID is the name of the participant executing sid, $ID*$ the name of the intended communication partner, and $comm_1, \ldots, comm_n$ the list of messages that were sent and received.

Based on the session identifiers, we can define when two sessions are *matching*: when they are communicating as expected, i. e., without interference from the adversary.

Definition 2. (Matching sessions for two-party protocols) For a two-party protocol, sessions sid and sid' are said to match if and only if there exist roles $role, role'$ $(role \neq role')$, participants ID, ID', and message list $L = \langle comm_1, \ldots, comm_n \rangle$, such that the session identifier of sid is $(role, ID, ID', L)$ and the session identifier of sid' is $(role', ID', ID, L)$.

We will later use the definition of matching for (1) defining the adversary capabilities and (2) ensuring that the protocol computes the correct keys in the absence of an adversary.

We now turn to defining the security experiment that forms the core of the security model. As is common in AKE security models, security experiments are defined from the point of view of the adversary. An experiment allows for executions that are sequences of adversary *queries*. Such queries model both the adversary behaviour as well as the actions of regular participants. There are six different queries, which we classify into three categories.

Modeling normal participant behaviour There is a single query to model the normal participant behaviour.

- $\mathsf{Send}(\mathsf{a}, \mathsf{b}, comm)$. The adversary sends a message $comm$ to b on behalf of a, and receives b's response according to the protocol. Additionally, this allows the adversary to order a party to start an AKE session with another party.

Modeling indistinguishability of the session key The adversary can choose to attack any completed session. We identify the session under attack as the *Test* session. As we will later see, the adversary tries to distinguish the session key of the test session from a random bit string. Therefore, we allow the adversary to randomly get either the session key of this session or a random string.

- $\mathsf{Test}(Test)$. *Test* must be a completed session. A bit is randomly drawn: $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, let C be the session key of session *Test*, otherwise pick $C \xleftarrow{\$} \{0,1\}^\lambda$. In both cases C is returned to the adversary.
- $\mathsf{Guess}(b')$. The experiment ends after this query is made. If the guess b' is equal to b (as drawn in a previous Test query), return 1, otherwise return 0.

Modeling adversary capabilities The adversary is given additional capabilities to reveal session keys, long-term keys, or ephemeral keys. As we will see below, these queries can only be performed under certain conditions.

- $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{a})$. The adversary reveals the long-term key of the party a.
- $\mathsf{EphemeralKeyReveal}(sid)$. The adversary reveals an ephemeral key of the session sid. Note that sid is not required to be completed.
- $\mathsf{Reveal}(sid)$. The adversary reveals a session key of the completed session sid.

The above capabilities are restricted in the experiment by requiring that all considered executions are *clean*.

Definition 3. (*clean* for eCK) In an AKE experiment (e. g. as defined in Definition 4 below), let *sid* be a completed AKE session performed by a, supposedly with some party b. Then *sid* is said to be *clean* in the eCK model if all of the following conditions hold:

1. a and b are not adversary-controlled (the adversary does not choose or reveal both the long-term and ephemeral keys of the participant and performs on behalf of the participant.)
2. The experiment does not include $\mathsf{Reveal}(\mathsf{sid})$, i. e. the session key of session *sid* is not revealed.
3. The experiment does not include both $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{a})$ and $\mathsf{Ephemeral\text{-}KeyReveal}(\mathsf{sid})$.
4. If no session exists that matches *sid*, then the experiment does not include $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{b})$.
5. If a session $sid*$ exists that matches *sid*, then
   (a) the experiment does not include $\mathsf{Reveal}(\mathsf{sid}*)$, i. e. the session key of session $sid*$ is not revealed, and
   (b) the experiment does not include both $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{b})$ and $\mathsf{EphemeralKeyReveal}(\mathsf{sid}*)$.

In the eCK security model, the restriction that the test session is *clean* is meant to exclude the cases in which all protocols are trivially insecure, e. g., by performing a Reveal query on the test session or its partner.

Definition 4. (AKE security experiment for eCK) In the eCK AKE security experiment, the following steps are allowed.

- The adversary may perform $\mathsf{Send}(\mathsf{a}, \mathsf{b}, \mathsf{comm})$, $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{a})$, and $\mathsf{Reveal}(\mathsf{sid})$ queries.
- The adversary may perform an $\mathsf{EphemeralKeyReveal}(\mathsf{sid})$ query.
- The adversary performs a $\mathsf{Test}(\mathsf{sid})$ query on a single *clean* session *sid*. A coin is flipped: $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, the test query returns a random bit string. If $b = 1$, the query returns the session key of *sid*. This query can be performed only once.
- The adversary outputs a $\mathsf{Guess}(\mathsf{b}')$ bit b', after which the experiment ends.

Except for the final step Guess, steps may be performed in any order. Test and Guess may only be performed once. The other steps may be performed any number of times.

An adversary $\mathcal{M}$ *wins* the experiment if the $\mathsf{Guess}(\mathsf{b}')$ bit b' is equal to the bit b from the $\mathsf{Test}(\mathsf{sid})$ query.

Definition 5. (eCK security) The advantage of the adversary $\mathcal{M}$ in the eCK AKE experiment with AKE protocol Π is defined as

$$\mathbf{Adv}_{\Pi}^{AKE}(\mathcal{M}) = \Pr[\mathcal{M} \text{ wins}] - \frac{1}{2}.$$

An AKE protocol is said to be secure in the eCK model if and only if

1. matching sessions compute the same session keys, and
2. no efficient adversary $\mathcal{M}$ has more than a negligible advantage in winning the above experiment.

2.3 The NAXOS key exchange protocol

The NAXOS protocol, as defined in [11, 12], is shown in Figure 1. NAXOS builds on ideas from the KEA and KEA+ protocols [13, 16]. The purpose of the NAXOS protocol is to establish a shared symmetric key between two parties. Both parties have a long-term private key, e. g. sk_{a}, and initially know the public key of all other participants, e. g. pk_{b}. For full details of the protocol we refer the reader to [11, 12].

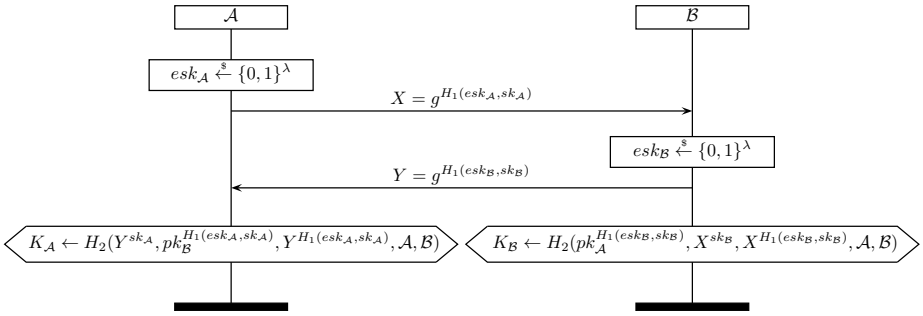

Figure 1: The NAXOS protocol. At the end of a normal execution we have that $K_{\mathcal{A}} = K_{\mathcal{B}}$ (recall that $pk_x = g^{sk_x}$).

The protocol is designed to be secure in a very strong sense: the adversary is assumed to have the capability of learning long-term private keys, and also has the capability of learning short term data generated during a protocol session that does not include the private key.

The intuition behind the design of the protocol is that by hashing the long-term private key together with the short term ephemeral key, the adversary would need to have both of these elements to construct an attack. For example, the protocol should be secure if the adversary either (a) learns the long-term key of a participant during a session, or (b) learns the short-term data (except for the long-term key) of a participant during a session. A typical scenario for (b) is that the participant stores the long-term key on an HSM or tamper-proof module, and computes other operations in unprotected memory.

Consider a normal execution where the attacker is passive. As depicted in Figure 1, we have $X = g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$ and $Y = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}$. Based on the properties of the modular exponentiation the following equivalences hold.

$$X^{sk_{\mathcal{B}}} = pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} \tag{1}$$

$$Y^{sk_{\mathcal{A}}} = pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})} \tag{2}$$

$$Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} = X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})} \tag{3}$$

At the end of a normal protocol execution, the session key is computed by both parties as

$$H_2(g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})sk_{\mathcal{A}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})sk_{\mathcal{B}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B}). \tag{4}$$

The NAXOS protocol was proven secure in the eCK model in [11, 12].

3 Automatic analysis of NAXOS

NAXOS was proven secure in the eCK model, and it is therefore not vulnerable to attacks based on EphemeralKeyReveal. To examine the relation between EphemeralKeyReveal and Session-StateReveal, we define the eCK' model that allows the Session-StateReveal query instead of the EphemeralKeyReveal query. Next we use formal analysis methods to establish that NAXOS is insecure in the eCK' model.

3.1 Security model eCK'

We define the eCK' security model, which is similar to the eCK model. We replace the EphemeralKeyReveal query by the Session-StateReveal query throughout the security definition, and add the restriction that the Session-StateReveal query cannot be performed on the test session or its partner. This restriction stems from [5] and is required because otherwise all protocols that store the session key in the session-state would be trivially insecure.

The Session-StateReveal query reveals the current state of a session. For the NAXOS protocol, we require that whenever $H_2(x_1, \ldots, x_n)$ is computed, $x_1, \ldots, x_n$ are part of the session state just before the computation, and can therefore be revealed by a Session-StateReveal query. An example of an execution model where this condition holds, is an HSM setting in which $H_2(x_1, \ldots, x_n)$ is computed in local memory, whereas all other computations (such as $H_1(x)$ and g^x) are performed inside the HSM. In contrast, applying the EphemeralKeyReveal query to a session of the agent a in the eCK model (and original NAXOS proof) from [12] reveals only the ephemeral key esk_{a}.

Definition 6. (*clean* for eCK') In an AKE experiment, let *sid* be a completed AKE session performed by a, supposedly with some party b. Then *sid* is said to be *clean* in the eCK' model if all of the following conditions hold:

1. a and b are not adversary-controlled (the adversary does not choose or reveal both the long-term and ephemeral keys of the participant and performs on behalf of the participant.)
2. The experiment does not include $\mathsf{Reveal}(\mathsf{sid})$, i. e. the session key of session sid is not revealed.
3. The experiment does not include $\mathsf{Session\text{-}StateReveal}(\mathsf{sid})$.
4. If no session exists that matches sid, then the experiment does not include $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{b})$.
5. If a session $sid*$ exists that matches sid, then
   (a) the experiment does not include $\mathsf{Reveal}(\mathsf{sid}*)$, i. e. the session key of session $sid*$ is not revealed, and
   (b) the experiment does not include $\mathsf{Session\text{-}StateReveal}(\mathsf{sid}*)$.

Definition 7. (AKE security experiment for eCK') In the eCK' AKE security experiment, the following steps are allowed:

- The adversary may perform $\mathsf{Send}(\mathsf{a}, \mathsf{b}, \mathsf{comm})$, $\mathsf{Long\text{-}TermKeyReveal}(\mathsf{a})$, and $\mathsf{Reveal}(\mathsf{sid})$ queries.
- The adversary may perform a $\mathsf{Session\text{-}StateReveal}(\mathsf{sid})$ query. (This query replaces $\mathsf{EphemeralKeyReveal}(\mathsf{sid})$ in the definition from [12].)
- The adversary performs a $\mathsf{Test}(\mathsf{sid})$ query on a single *clean* session sid. A coin is flipped: $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, the test query returns a random bit string. If $b = 1$, the query returns the session key of sid. This query can be performed only once.
- The adversary outputs a $\mathsf{Guess}(\mathsf{b}')$ bit b', after which the experiment ends.

Except for the final step Guess, steps may be performed in any order. Test and Guess may only be performed once. The other steps may be performed any number of times.

An adversary $\mathcal{M}$ *wins* the experiment if the $\mathsf{Guess}(\mathsf{b}')$ bit b' is equal to the bit b from the $\mathsf{Test}(\mathsf{sid})$ query.

Definition 8. (eCK' security) The advantage of the adversary $\mathcal{M}$ in the eCK' AKE experiment with AKE protocol Π is defined as

$$\mathbf{Adv}_{\Pi}^{AKE}(\mathcal{M}) = \Pr[\mathcal{M} \text{ wins}] - \frac{1}{2}.$$

We say that an AKE protocol is secure in the eCK' model if matching sessions compute the same session keys and no efficient adversary $\mathcal{M}$ has more than a negligible advantage in winning the above experiment.

We prove a theorem for the eCK' model that will simplify our automatic analysis later.

Theorem 1. (Irrelevance of session-state erasure point for the eCK' model) *Let P be a protocol with a role R in which some element s of the session-state is erased at some point during the session (and hence not used afterwards). Let P' be the same protocol as P, except that the s is erased from the session-state as late as possible within R, i. e., as its last action. Then, we have that P is eCK'-secure if and only if P' is eCK'-secure.*

Proof. Proof of Theorem 1 We prove the theorem by showing that P is insecure in eCK' if and only if P' is insecure in eCK'.

Assume that P' is insecure in eCK'. There are two possible causes. If P' is insecure because matching sessions do not compute the same session key, then it is trivial to see that P is insecure in eCK' as well. We therefore consider the case in which there exists an efficient adversary that can win the experiment with non-negligible probability. Then there exists a *run*, i. e., a fixed execution of the security experiment that occurs with non-negligible probability, in which the adversary correctly guesses the bit b. Let $r1$ be such a run of P'. We will use $r1$ to construct a valid run $r2$ of protocol P in which the adversary correctly guesses the bit, thereby showing that P is insecure in eCK'. In particular, we define $r2$ to be identical to $r1$, except where $r1$ includes a Session-StateReveal(sid) query to reveal s, at a point in the session where s is already erased in P, but not yet in P'. We will move the Session-StateReveal query of s to an earlier point in the run and leave all other queries unchanged. In particular, we move the Session-StateReveal event of session sid from $r1$ to a position in $r2$ just before s is erased from the session-state of sid in P. In case s is not the entire session-state but only part of it, we don't move the Session-StateReveal query. Rather, we remove s from the Session-StateReveal query and add an additional (earlier) Session-StateReveal event that reveals the earlier state that does include s. Observe that because the adversary guesses the bit correctly in the run $r1$, the test session must be *clean* in $r1$ (as in Definition 6), and therefore sid cannot be the test session or its matching session. Because we don't change queries other than Session-StateReveal and do not change the target session of Session-StateReveal, sid is also not the test session or its matching session in run $r2$. Therefore, the Session-StateReveal query is allowed on sid, and the test session is *clean* in $r2$ as well. Consequently, $r2$ is a valid run of protocol P and occurs with non-negligible probability, showing that P is insecure in eCK'.

Assume that P is insecure in eCK'. Because session-state is only erased later in P', any run of P in which the adversary guesses the bit correctly is also a valid run of P', and therefore P' is also insecure in the eCK' model. □

Note that a similar theorem and proof can be given for the CK model. Intuitively, the condition under which a Session-StateReveal query is allowed (in both the CK and eCK' model) is only defined in terms of *which* session is queried. The point at which the query is performed is irrelevant for this

condition and therefore Session-StateReveal queries in a particular session can be performed earlier or later as desired.

3.2 Automatic analysis

We use an automatic protocol analysis tool, *Scyther*, to analyze the security of the Naxos protocol with respect to a symbolic version of the eCK' model.

3.2.1 Background on the Scyther tool

The Scyther tool [7] is a formal protocol analysis tool based on an operational semantics for security protocols [1]. In this framework, protocol messages are modeled as abstract terms. The adversaries' capabilities are explicitly modeled as operations on terms. These operations abstract from cryptographic details by assuming that cryptography is "perfect". For example, the adversary learns nothing from an encrypted message unless he knows the decryption key.

Because the capabilities of the adversary are explicitly modeled, all possible interactions between protocol participants and the adversary can be modeled by a (labeled) transition system, giving rise to a set of *traces*. Each trace represents a possible execution history of the protocol. Security properties are modeled as trace properties. For example, a message m is said to be secret if for all traces of the protocol, the adversary never learns m.

This abstract model of cryptographic protocols, as used in the Scyther framework, is a version of the standard Dolev-Yao model [9]. Any attacks found in the abstract model can be translated to attacks on the protocol in computational models such as eCK or eCK'. The converse does not hold: the absence of attacks in the abstract model does not imply that the protocol is secure in a computational sense. However, this is not a concern here as we only use the tool to establish attacks.

For full details on the Scyther tool, its underlying protocol model, and the analysis method we refer the reader to [1, 7, 8].

3.2.2 Modeling the NAXOS protocol

We provide below an abstract protocol specification for readability; in Appendix A we provide the concrete input file in the input language of the Scyther tool.

To specify the protocol and in particular the messages it sends and receives, we define the set of terms *Term*. We assume given the infinite sets *Agent*, *Role*, *Fresh*, *Var*, and *Func* of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, and function names.

Definition 9. Terms

$$\begin{aligned} \mathit{Term} ::= {} & \mathit{Agent} \mid \mathit{Role} \mid \mathit{Fresh} \mid \mathit{Var} \mid (\mathit{Term}, \mathit{Term}) \\ & \mid \mathit{sk}(\mathit{Term}) \mid \mathit{Term}^{\mathit{Term}} \mid \mathit{Func}(\mathit{Term}^*) \end{aligned}$$

For each $X \in \mathit{Agent}$, $sk(X)$ denotes its long-term private key. The notation $t^{t'}$ denotes modular exponentiation. The set Func is used to model other cryptographic functions, such as hash functions. We model constants as 0-ary functions.

Protocols are specified as a set of roles, where each role consists of a sequence of protocol events.

Definition 10. Protocol events

$$\begin{aligned} \mathit{Event} ::= {} & \mathsf{send}(\mathit{Term}) \mid \mathsf{recv}(\mathit{Term}) \\ & \mid \mathsf{state}(\mathcal{P}(\mathit{Term})) \mid \mathsf{sessionkey}(\mathit{Term}) \end{aligned}$$

The first two events are standard and model the basic agent actions, i. e., sending a message and receiving a message. The message in the send and receive events does not include explicit sender or recipient fields although, if desired, they can be given as subterms of the message. Note that we do not explicitly specify any intermediate computations: it is the responsibility of the protocol modeler to specify sent terms such that they can be computed during protocol execution. Similarly, there is no explicit event to generate fresh terms. Fresh terms, identified by their type, are generated upon first occurrence. The parsing of messages is modeled as pattern matching: terms in protocol events may contain variables, and terms in receive events are pattern matched against any messages the adversary can produce during protocol execution.

The other two events are used to annotate the traces with information that is relevant for the verification process. The *state* event marks a set of messages as the contents of the session-state, which can possibly be compromised by the adversary. The *sessionkey* event serves two purposes. It marks a term as the session key and also specifies that this term should be secret, thereby specifying the security property.

We model the NAXOS protocol by individually modeling its two roles, initiator $\mathcal{A}$ and responder $\mathcal{B}$, by the following role scripts, where $esk_{\mathcal{A}}, esk_{\mathcal{B}} \in \mathit{Fresh}$ and $X, Y \in \mathit{Var}$.

Role description 1 NAXOS($\mathcal{A}$)

1: $\mathsf{send}(\mathcal{A}, \mathcal{B}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})})$
2: $\mathsf{recv}(\mathcal{B}, \mathcal{A}, Y)$
3: $\mathsf{sessionkey}(H_2(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \mathcal{A}, \mathcal{B}))$

Role description 1 specifies the $\mathcal{A}$ role of the NAXOS protocol. As in the protocol description in Figure 1, $\mathcal{A}$ starts by implicitly drawing a fresh nonce $esk_{\mathcal{A}}$. In the abstract protocol specification, the security parameter λ is not modeled. The first event is the sending of the message $g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$ from $\mathcal{A}$ to $\mathcal{B}$. The second event models receiving a message Y. The third event encodes both the computation of the session key and its security property. In step 3, $\mathcal{A}$ computes the session key that should not become known to the adversary.

The $\mathcal{B}$ role is modeled similarly and is specified in role description 2.

Role description 2 NAXOS($\mathcal{B}$)

1: $\mathsf{recv}(\mathcal{A}, \mathcal{B}, X)$
2: $\mathsf{send}(\mathcal{B}, \mathcal{A}, g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})})$
3: $\mathsf{sessionkey}(H_2(pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B}))$

3.2.3 Adversary model

The default adversary model in the framework of the Scyther tool is known as the Dolev-Yao adversary model, i. e., the adversary learns all messages sent, can generate any number of fresh values, and can inject any message that he can infer from his knowledge, where the inference relation $\cdot \vdash \cdot$ is given as the smallest relation satisfying

$$\begin{aligned} t \in M &\Rightarrow M \vdash t \\ M \vdash t_1 \wedge M \vdash t_2 &\Rightarrow M \vdash (t_1, t_2) \\ M \vdash (t_1, t_2) &\Rightarrow M \vdash t_1 \wedge M \vdash t_2 \\ M \vdash t_1 \wedge M \vdash t_2 &\Rightarrow M \vdash {t_1}^{t_2} \\ \Big(\bigwedge_{0 \le i \le n} M \vdash t_i \Big) &\Rightarrow M \vdash f(t_0, \ldots, t_n) \end{aligned}$$

We write IK_{init} to denote the initial adversary knowledge, containing all public knowledge of the protocol, such as the agent names, their long-term public keys, and the generator g.

3.2.4 Protocol execution

In the execution model any number of instances of protocol roles can be executed in parallel. Send events $\mathsf{send}(m)$ imply that the adversary learns the message m. Receive events $\mathsf{recv}(m)$ can be executed if and only if the adversary can infer a message m' that matches the expected message m (i. e., by instantiating the free variables occurring in m).

3.2.5 Specifying the eCK and eCK' models in Scyther

Recent versions of the Scyther tool support protocol analysis with respect to symbolic equivalents of Session-StateReveal, EphemeralKeyReveal and Reveal queries, as defined in [1]. We describe one important difference between the eCK/eCK' models and the model used in the automatic tool. To verify that the test session is clean in the automatic analysis, we use a definition of matching sessions that differs from the definition in the eCK model.

Definition 11. (Matching for two-party protocols in Scyther) Let $msg(sid)$ denote the sequence of messages (sent and received) by the session sid in a security experiment. Let $role(sid)$ denote the role (here $\mathcal{A}$ or $\mathcal{B}$) performed by the session sid in the security experiment. We say that two sessions

sid, sid' are partners if and only if

$$role(sid) \neq role(sid') \wedge msg(sid) \neq \emptyset \wedge msg(sid') \neq \emptyset \wedge \exists l.\big(msg(sid)\hat{}l = msg(sid') \vee msg(sid')\hat{}l = msg(sid)\big).$$

Definition 11 deviates slightly from Definition 2 (which is part of eCK definition given in [11, 12]). Definition 11 allows incomplete sessions also to be partners of the test session. For example, if a session sid sends m_1 to the test session, and the test session responds with m_2 but this second message m_2 is not (yet) received by session sid, we consider sid to be a partner of the test session anyway. In contrast, Definition 2 does not consider sid to be a partner in this case. Our definition is in line eCK follow-up works such as [18], in which a star "$*$" or cross "$\times$" notation is used (instead of our l) in the list of exchanged messages to match any unreceived messages with all possible sent messages.

3.2.6 Modeling the session-state

In the previously given role specifications, it is not defined what the session-state is, as in the original protocol description. To analyze the protocol with respect to the Session-StateReveal query we need to specify the session state. The Scyther tool provides support for this query by allowing manual specification of the session state by manually inserting state events. Still, this leaves the analyst with the task of specifying the session state of a given protocol.

The Session-StateReveal query models an adversary capability, e. g., what an adversary may learn from a side-channel attack. This depends not only on the protocol specification (e. g., computing g^x or $H(x)$), but also on implementation details of cryptographic primitives (e. g., what are intermediate products of computing $H(x)$?), as well as platform specifics (e. g., if a side-channel attack requires data to be in memory for a certain amount of time, the possibly-revealed data may depend on caching mechanisms, erasure/overwrite times, etc.)

In many cases, such implementation and platform details are not known at design time. This is a common situation for many protocol proposals (including NAXOS). We can however make a safe approximation of the possible session-state contents in *any* implementation by extracting intermediate computation products that are the result of the protocol *design* rather than its *implementation*.

We give a procedure for automatic session-state inference, which automatically inserts state events in a given protocol description.

We start off by defining the auxiliary function $unpair$ and relation $\sqsubseteq$.

Definition 12. (Unpair) The function $unpair : Term \to \mathcal{P}(Term)$ is defined as

$$unpair(t) \stackrel{\text{def}}{=} \begin{cases} unpair(t_1) \cup unpair(t_2) & \text{if } t = (t_1, t_2), \\ \{t\} & \text{otherwise.} \end{cases}$$

We extend the domain of *unpair* to sets of terms, i. e., for T a set of terms, we define

$$unpair(T) \stackrel{\text{def}}{=} \bigcup_{t \in T} unpair(t).$$

Definition 13. (Subterms) The syntactic subterm relation $\sqsubseteq$ is defined as the smallest relation satisfying:

$$t \sqsubseteq t \qquad t_1 \sqsubseteq (t_1, t_2) \qquad t \sqsubseteq f(\ldots, t, \ldots) \qquad t_1 \sqsubseteq {t_1}^{t_2}$$
$$t_2 \sqsubseteq (t_1, t_2) \qquad t \sqsubseteq sk(t) \qquad t_2 \sqsubseteq {t_1}^{t_2}.$$

Our aim is to determine which terms are intermediate products of a computed term. The main constraint from [5] is that the state does not contain any long-term private keys. We assume that all terms that involve freshly generated terms and variables must be computed locally.

We define a function $\psi : Event^* \to \mathcal{P}(Term)$ that identifies all subterms occurring in a sequence of events that must be computed locally, and are therefore possibly part of the session-state.

$$\psi(s) \stackrel{\text{def}}{=} unpair\Big(\big\{t' \mid \exists T, t, t'', ev \,.\, t'' \in Var \cup Fresh \wedge t'' \sqsubseteq t' \wedge t' \sqsubseteq t \wedge t \in unpair(T) \wedge ev(T) \in s\big\}\Big)$$

We define the function *seen* that identifies all terms in a sequence of events that the adversary learns after the sequence is executed and the adversary performs Session-StateReveal queries. We define *seen* as

$$seen(s) \stackrel{\text{def}}{=} unpair\Big(IK_{init} \cup \big\{t' \mid \exists T, t, ev \,.\, t' \sqsubseteq t \wedge t \in unpair(T) \wedge ev \neq \mathsf{sessionkey} \wedge ev(T) \in s\big\}\Big).$$

The functions ψ and *seen* are used to define the augmentation function $\phi : Event^* \to Event^*$. Given a sequence of events, i. e., a role description, it returns a modified sequence of events that includes state events. The intuition is that the state contains all terms given by ψ, but we omit any terms the adversary could have trivially learnt at any earlier point. Theorem 1 allows us to ignore the details of the points at which session-state is erased.

Definition 14. (Automatic symbolic state inference) Let s be a sequence of events. We define the augmented sequence by $\phi(s)$, where ϕ is defined as

$$\phi(\langle\rangle) \stackrel{\text{def}}{=} \langle\rangle$$
$$\phi(s^\frown\langle e\rangle) \stackrel{\text{def}}{=} \phi(s)^\frown\langle e\rangle^\frown\Big\langle \mathsf{state}\Big(\psi\big(\langle e\rangle\big) \setminus seen\big(\phi(s)\big)\Big)\Big\rangle$$

Finally, any empty state events (i. e., events of the form $\mathsf{state}(\emptyset)$) are removed from the sequence.

For the NAXOS protocol, Scyther's automatic state inference yields the augmented role descriptions 3 and 4.

Role description 3 NAXOS($\mathcal{A}$) with automatically inferred state

1: $\mathsf{send}(\mathcal{A}, \mathcal{B}, g^{H_1(esk_\mathcal{A}, sk_\mathcal{A})})$
2: $\mathsf{state}(\{esk_\mathcal{A}, H_1(esk_\mathcal{A}, sk_\mathcal{A})\})$
3: $\mathsf{recv}(\mathcal{B}, \mathcal{A}, Y)$
4: $\mathsf{sessionkey}(H_2(Y^{sk_\mathcal{A}}, pk_\mathcal{B}^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, Y^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, \mathcal{A}, \mathcal{B}))$
5: $\mathsf{state}(\{Y^{sk_\mathcal{A}}, pk_\mathcal{B}^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, Y^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, H_2(Y^{sk_\mathcal{A}}, pk_\mathcal{B}^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, Y^{H_1(esk_\mathcal{A}, sk_\mathcal{A})}, \mathcal{A}, \mathcal{B})\})$

Role description 4 NAXOS($\mathcal{B}$) with automatically inferred state

1: $\mathsf{recv}(\mathcal{A}, \mathcal{B}, X)$
2: $\mathsf{state}(\{esk_\mathcal{B}, H_1(esk_\mathcal{B}, sk_\mathcal{B})\})$
3: $\mathsf{send}(\mathcal{B}, \mathcal{A}, g^{H_1(esk_\mathcal{B}, sk_\mathcal{B})})$
4: $\mathsf{sessionkey}(H_2(pk_\mathcal{A}^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, X^{sk_\mathcal{B}}, X^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, \mathcal{A}, \mathcal{B}))$
5: $\mathsf{state}(\{pk_\mathcal{A}^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, X^{sk_\mathcal{B}}, X^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, H_2(pk_\mathcal{A}^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, X^{sk_\mathcal{B}}, X^{H_1(esk_\mathcal{B}, sk_\mathcal{B})}, \mathcal{A}, \mathcal{B})\})$

3.3 Two attacks on the Naxos protocol

We used the Scyther tool to analyse the NAXOS protocol with automatically inserted session-state events with respect to the eCK' model. The protocol description files provided to the tool are given in the appendix. The tool automatically finds two attacks. One attack performs a test query on a session of the initiator role and the other on the responder role. Finding both attacks takes 24.7 seconds on a dual core Intel Centrino, running at 1.2 GHz, using Scyther-Compromise version 0.3. The appendix provides the information needed to reproduce our results.

3.3.1 Attacking the initiator

In Figure 2 we show an attack for a test query on an initiator session of NAXOS. The attack requires an active adversary capable of Session-StateReveal.

The adversary can compute K_a on the basis of the revealed information (based on the algebraic properties of the group exponentiation, which are required for the core of the protocol).

The attack proceeds as follows.

1. a starts an initiator instance, wanting to communicate with b.
2. a chooses her ephemeral key esk_a, and sends out $X_\mathsf{a} = g^{H_1(esk_\mathsf{a}, sk_\mathsf{a})}$. The adversary learns this message.
3. b also starts an initiator instance, wanting to communicate with a.
4. b chooses her ephemeral key esk_b, and sends out $X_\mathsf{b} = g^{H_1(esk_\mathsf{b}, sk_\mathsf{b})}$. The adversary learns this message.

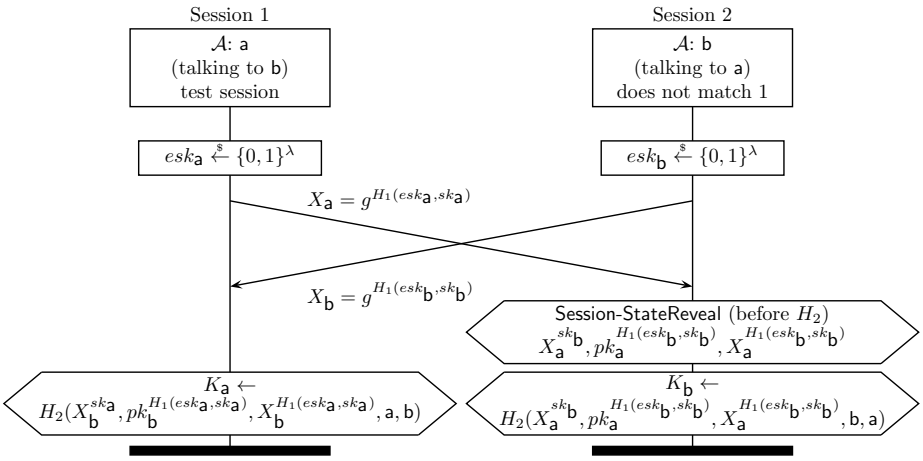

Figure 2: Attacking an initiator session. Note that $K_{\mathsf{a}} \neq K_{\mathsf{b}}$. The adversary can compute K_{a} after compromising the local state of b.

5. The adversary sends the message X_{b} to a.
6. a computes the session key K_{a} as

$$H_2(X_{\mathsf{b}}^{sk_{\mathsf{a}}}, pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{a}, \mathsf{b}). \tag{5}$$

7. The adversary sends the message X_{a} to b.
8. b computes the session key K_{b} as

$$H_2(X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, \mathsf{b}, \mathsf{a}). \tag{6}$$

   During the computation of K_{b}, the adversary uses Session-StateReveal to learn the input to H_2. In particular, the adversary learns $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$, $pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$, and $X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$.
9. The adversary now knows

$$pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = g^{sk_{\mathsf{a}} H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = X_{\mathsf{b}}^{sk_{\mathsf{a}}},$$
$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})},$$
$$X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}.$$

   The three terms on the right-hand side are the first three components of the session key K_{a} from Formula 5.

10. The adversary combines the elements with the names a and b, and applies H_2, resulting in K_{a}.

The above sequence of actions forms an attack on the protocol, because the adversary can learn the session key of the initiator a by revealing the local state of the second session. Furthermore, the test session is *clean* according to Definition 6 on page 9 because (1) neither a nor b are adversary-controlled, (2) no Reveal queries are performed, (3) no long-term keys are revealed, and (4) session 2 is not a partner to the test session 1. Therefore, the attack violates security in the eCK' model.

Some further observations regarding this attack:

- The sessions compute different session keys: $K_{\mathsf{a}} \neq K_{\mathsf{b}}$, because the order of the participant names a, b is reversed.
- The adversary does not need to learn any ephemeral keys for this attack.
- Even in other existing interpretations of the partner function (or freshness) from literature (matching conversations, external session identifiers, explicit session identifiers, etc.) the two sessions are not partners. Consequently, the NAXOS protocol is therefore also not secure in other models that allow Session-StateReveal, such as the CK model [5].

3.3.2 Attacking the responder

Second, we show an attack for a test query on a responder session in Figure 3.

The attack proceeds as follows.

1. The adversary chooses an arbitrary bit string κ.
2. The adversary computes g^{κ} and sends the result to a responder instance of a, with sender address b.
3. a receives the message and assigns $X_{\mathsf{b}} = g^{\kappa}$.
4. a chooses her ephemeral key esk_{a}, and sends out $X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$. The adversary learns this message.
5. a computes the session key K_{a} as

   $$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{sk_{\mathsf{a}}}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}) \tag{7}$$

   which is equal to

   $$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, g^{\kappa\, sk_{\mathsf{a}}}, g^{\kappa\, H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}). \tag{8}$$

6. The adversary redirects X_{a} to a responder instance of b. The adversary can insert an arbitrary participant name in the sender field of the message, which b takes to be the origin of the message.

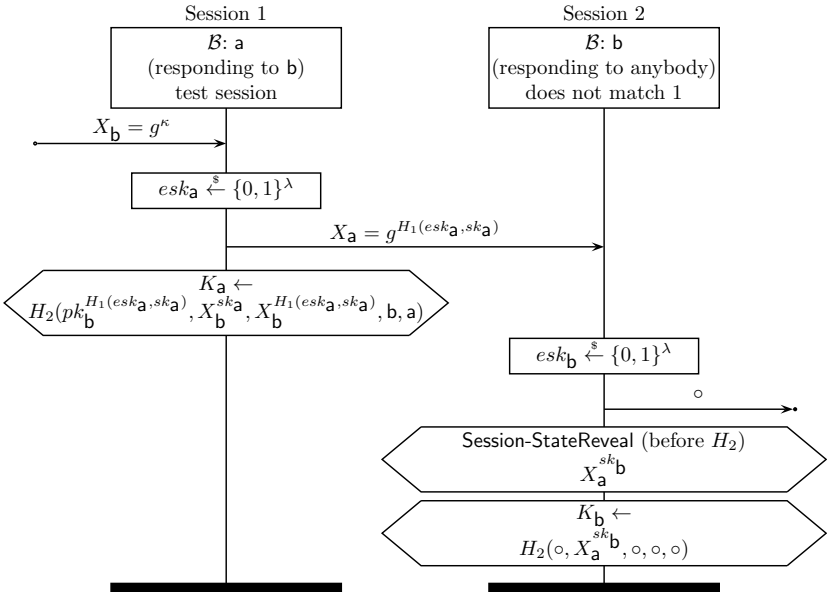

Figure 3: Attack on a responder session. We have $K_{\mathsf{a}} \neq K_{\mathsf{b}}$. The adversary can compute (and even contribute to) K_{a} after revealing the local state of b.

7. b computes his ephemeral secret, combines it with his long term key, and sends out the corresponding message.

8. b computes his session key K_{b} (which differs from K_{a}). Before applying H_2, b computes the second component $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$.

9. The adversary uses Session-StateReveal on the session of b directly before the application of H_2 to learn $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$.

10. The adversary knows κ, X_{a}, and $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$. Furthermore, as the public keys are public, the adversary also knows pk_{a}. Hence the adversary also knows, or can compute:

$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \tag{9}$$

$$(pk_{\mathsf{a}})^{\kappa} = g^{sk_{\mathsf{a}}\kappa} = X_{\mathsf{b}}^{sk_{\mathsf{a}}}, \tag{10}$$

$$(X_{\mathsf{a}})^{\kappa} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})\kappa} = X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}. \tag{11}$$

    The three terms on the right-hand side are the first three components of the session key K_{a} from Formula 7.

11. The adversary combines the elements and applies H_2, resulting in K_{a}.

This sequence forms an attack on the protocol, because the adversary can use data revealed from session 2 in order to compute the session key of the test session 1. The test session is also clean according to Definition 6. In practical terms, this attack even allows the adversary to determine a part of the session key of a.

Some observations with respect to this attack:

- The responder session of b is not a partner to the session of a in terms of matching sessions. Also, in other partner existing interpretations from literature (external session identifiers, explicit session identifiers, etc.) they would also not match.
- The adversary chooses κ, and can therefore influence the session key.
- The adversary does not learn any long term private keys or ephemeral keys in this attack.
- The attack is also valid in the CK model: the sessions are not partners for a number of reasons, for example because their choice of agents differs. Session 1 has $\{\mathsf{a}, \mathsf{b}\}$ and session 2 has $\{\mathsf{b}, z\}$ where z is an arbitrary participant. Hence the adversary can choose $z \neq \mathsf{a}$.

3.4 Implications of our attacks

Implications for the eCK model The existence of our NAXOS attacks in the eCK' model shows that the Session-StateReveal query models attacks that the EphemeralKeyReveal query cannot: there exist attacks that strictly depend on intermediate products (of non-partners) being revealed, and not only their randomness. In that sense, the Session-StateReveal query is stronger than EphemeralKeyReveal.

However, this does not lead to a direct conclusion about the relation between the security models CK and eCK. In the security models there is a strong interaction between the notion of a *clean* (or *fresh*) session and the queries that the adversary is allowed to make. In particular, because the session state may contain the session key, Session-StateReveal cannot be allowed on the test session and its partner. Whereas the EphemeralKeyReveal query is allowed on all sessions, the Session-StateReveal query is only allowed on non-partner sessions. In that sense, and as shown in [12], attacks that exploit the randomness of the test session or its matching session are not covered by, e. g., the CK model.

Theorem 2. *The eCK security model is* incomparable *to the CK security model: there exist protocols that are secure in CK but not in eCK, and conversely, there exist protocols that are secure in eCK but not in CK.*

Proof. Proof The proof is based on the analysis of two protocols, one for each direction of the proof.

First, consider the 2-message signed Diffie-Hellman protocol that was proven secure in the CK model in [5] under the assumption that the session-state only

contains the random numbers. All other intermediate products of the computation (e. g. conform the state-inference procedure proposed here) are either long-term private keys, which are by definition excluded from the session-state, or public information during protocol execution. Therefore, adding these intermediate products to the state does not give the adversary additional possibilities. As a result, this protocol is secure in the CK model for any definition of session-state. However, the protocol is insecure in the eCK model as shown by the attack described in [12]: revealing the randomness of the test session allows the adversary to reconstruct the session key.

Second, consider the NAXOS protocol, as proven secure in the eCK model in [11, 12]. Assume that the session-state of the protocol contains the inputs to H_2. Then, the NAXOS protocol is insecure in the CK model, as shown by the attacks presented earlier. □

Implications for the NAXOS protocol For our attacks we use the NAXOS protocol exactly as specified in [11, 12]. We assume that the protocol is implemented such that when a participant in the NAXOS protocol computes $H_2(x)$, where H_2 is a particular hash function in the NAXOS protocol, then x is in the session state just before the computation. As a result, performing a Session-StateReveal query just before the computation of $H_2(x)$ reveals x. This assumption does not require changing the protocol. Rather, we make the contents of the session state explicit, as would be required for a proof in the CK model.

Our attacks are valid for a certain class of implementations of the NAXOS protocol, in which the long-term private keys are protected by an HSM, but other computations, in particular the computation of H_2, are done in unprotected memory. If an adversary can get access to the unprotected memory of an honest participant b, he can trigger the second attack at will with any agent, and learn the session key. This effectively allows him to impersonate as b in the communication even though he never learns b's long-term private key and does not reveal anything of the test session or its partner.

The attack is of a theoretical nature: from a practical point of view one may wonder why an adversary capable of accessing b's unprotected memory doesn't just compromise the partner to the test session, in which case all protocols that store the session key in protected memory are trivially insecure.

3.5 Extending the scope of the attacks

The structure of our attacks can be generalized to attack some protocols that were proven secure in the CK model [5].

In [13], the KEA+ protocol is proven secure in the CK model from [5]. KEA+ can be viewed as a predecessor of the NAXOS protocol, and uses a similar setup. Two other variants of this protocol are KEA+C from [13] and KEA from [16]. All three protocols compute the session key using a hash function, which takes as inputs components built by the communication partners. Each of the crucial inputs is a modular exponentiation that includes in the exponent both

randomness and the long-term private keys of one of the participants. The proof in [13] of the security of KEA+ in the CK model assumes that Session-State-Reveal is defined such that only the ephemeral keys are revealed.

The attacks presented in this paper on the NAXOS protocol also work within the CK model for the KEA, KEA+, and KEA+C protocols after minimal modifications. The attacks use the same scenarios and exploit the same Session-State-Reveal definition, in which the inputs to the final hash function are part of the session state.

The existence of these attacks shows the importance of explicitly specifying the definition of session state as it is used in a proof: e. g. KEA+ is not secure in the CK model if the inputs to the final hash function are part of the session state. It would be more precise to say that in [13] KEA+ is proven secure with respect to the CK model if the session state only includes the ephemeral keys.

4 Interpreting the Session-StateReveal and Ephemeral-KeyReveal queries

4.1 Practical interpretations of the Session-StateReveal query

Session-StateReveal allows the adversary to reveal the internal state of the protocol, which intuitively corresponds to an adversary with read-only access to the memory of a participant. However, because the long-term private keys are explicitly excluded from the session state, the access is of a special type, which distinguishes between long-term keys and other session state. This suggests that the long-term keys are stored at a different protection level than the session state.

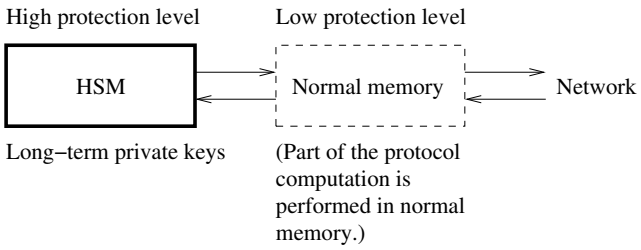

Figure 4: Practical scenarios captured by Session-StateReveal. The contents of the session-state specify which parts of the computation are stored at a lower protection level.

Figure 4 depicts a typical scenario that can be modeled using Session-State-Reveal. The long-term private keys are stored at a high protection level, e. g. in an HSM. Other computations of the protocol are stored at a lower protection

level, e. g. in normal client memory. In such a scenario, the definition of the session-state models which parts of the computation are performed where. The assumption is that it is easier for an adversary to retrieve the contents of the memory stored at a lower protection level than to retrieve the contents of the HSM.

Because the Session-StateReveal query models a strictly passive adversary capability, it does not seem to correspond well to an adversary that actively attacks the memory with a lower protection level. Rather, the Session-State-Reveal query seems well suited to model a class of side-channel attacks.

4.2 Practical interpretations of EphemeralKeyReveal

In contrast to the Session-StateReveal query, the EphemeralKeyReveal query is defined to exactly reveal the randomness generated within the session.

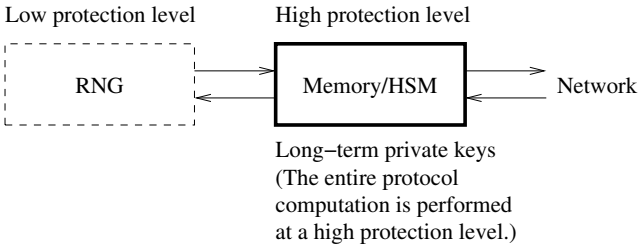

Figure 5: Practical scenarios captured by EphemeralKeyReveal. The adversary can reveal data generated by the random number generator (RNG).

Figure 5 depicts a typical scenario that can be modeled well using Ephemeral-KeyReveal. The data generated by the random-number generator (RNG) can be revealed using a side-channel attack, but the adversary has no access to the long-term private keys or any other intermediate computations.

In the context of the eCK model, the EphemeralKeyReveal query models leakage of data of a random number generator that generates numbers in the same way as a normal RNG. Furthermore, numbers can only be revealed *after* they were generated, as also pointed out in [19]. As a result, predictable or malicious random number generators are not well modeled by the eCK model's EphemeralKeyReveal query. Similarly, the loss of pre-computed randomness is also not well modeled.

4.3 On extending the definition of EphemeralKeyReveal

In later works that use variants of the eCK model it has been suggested that the ephemeral secrets, as revealed by the EphemeralKeyReveal query, can be redefined to contain everything modeled by a state-reveal query. However, the

session-state contents evolve during protocol execution. The timing of the query is therefore relevant for the session-state contents. Defining the ephemeral secrets as the session-state at the end of a session, possibly involving computations with received data, leads to modeling inconsistencies when the EphemeralKeyReveal query is performed on incomplete sessions.

5 Conclusion

In common definitions of AKE security the Session-StateReveal query is underspecified. The definition of Session-StateReveal is only made explicit in particular protocol proofs. This approach turns the exact definition of Session-StateReveal into a parameter of the exact security provided by the protocol. As a result, stating that two protocols are provably secure in e. g. the CK model does not mean they meet exactly the same property.

In the eCK model [11, 12] the Session-StateReveal query is replaced by the EphemeralKeyReveal query, which is claimed to be at least as strong as Session-StateReveal. Thus, the notion of Session-StateReveal is reduced to EphemeralKeyReveal. Reducing Session-StateReveal to EphemeralKeyReveal simplifies proofs significantly: one does not need to define what exactly is part of the ephemeral key, but one only needs to prove that no information about the ephemeral key is revealed [12, 17, 18]. However, the validity of this reduction has not been proven.

The validity of the reduction is informally argued in [11], and similar arguments can be found in other works that use the eCK model [17, 18], e. g. in [18, p. 333]: "In general, by specifying that the session specific private information (the session state) is part of the ephemeral private key, the Session-StateReveal and EphemeralKeyReveal queries can be made functionally equivalent".

Here we have shown that the reduction is invalid, that is, a security model with EphemeralKeyReveal (eCK) is not stronger than a model with Session-StateReveal (eCK'). The attacks presented here on the NAXOS protocol, which was proven secure for EphemeralKeyReveal in [12], strictly depend on the use of the Session-StateReveal query.

The attacks fall just outside the eCK security model, and they therefore do not indicate a problem with the proofs in [12]. Instead, what the attacks indicate is that the eCK security model, and similarly the property that is proved correct, is not as strong as suggested in e. g. [12]. Furthermore, the attacks are also valid in the CK model, which shows that the difference between CK and eCK is in fact meaningful in practice. In particular, we have shown that one can prove real protocols secure in eCK which are not secure in CK, and are vulnerable to attacks where the session state is revealed. Consequently, the CK and eCK models are not only theoretically, but also practically incomparable.

Our automatic analysis extends recent techniques from the formal protocol verification field. This approach allowed us to efficiently find intricate attacks that were previously unreported, and allowed us to show the incomparability of the models.

The structure of our attacks on NAXOS can be translated to attacks on the KEA, KEA+, and KEA+C protocols from [13, 16]. As a result, also these protocols are not CK-secure if the session state includes the inputs to the final hash function.

The idea behind the NAXOS protocol (which is already found in KEA and KEA+) is appealing: by strongly connecting the long- and short-term information, the adversary would be required to know both elements to perform an attack. However, in order to use the combination of these elements securely in the protocol, in particular for transmission, there are further computations needed. These subsequent computations often influence the session state. This effect is not captured by the definition of EphemeralKeyReveal, which is the ultimate problem with the reduction from Session-StateReveal to EphemeralKeyReveal, as was already noted in [3]. The attacks presented here exploit exactly this difference.

A possible practical interpretation of the difference between the models is the following. The CK model considers an HSM implementation, where parts of the protocol are computed in unprotected memory, specified by the contents of the session-state, but the long-term private keys are protected by the HSM. The adversary may be able to learn the contents of the unprotected memory at some point, but not necessarily all the time. In contrast, the eCK model considers an information-leaking random number generator, which implies that the adversary learns all ephemeral keys, but assumes the protocol computations are performed at a higher protection level (e. g. in an HSM).

The guarantees provided by proofs in the CK, eCK, and similar models would be significantly clearer if the proofs would be accompanied by (a) a clear specification of the contents of the session state, and (b) a specification of implementation restrictions under which the proof holds.

References

[1] D. Basin and C. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Proceedings of the 15th European Symposium on Research in Computer Security, ESORICS 2010, Athens, Greece, September 20-22, 2010*, Lecture Notes in Computer Science. Springer-Verlag, 2010. To appear.

[2] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, Lecture Notes in Computer Science, pages 139–155. Springer-Verlag, 2000.

[3] C. Boyd, Y. Cliff, J. Nieto, and K. Paterson. Efficient one-round key exchange in the standard model. In *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 69–83. Springer-Verlag, 2008.

[4] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264, New York, USA, 2001. ACM.

[5] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, 2001.

[6] K.-K. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment proofs. In *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 624–643. Springer-Verlag, 2005.

[7] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer-Verlag, 2008.

[8] C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128, New York, NY, USA, 2008. ACM.

[9] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12):198–208, Mar. 1983.

[10] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer-Verlag, 2005.

[11] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073, 2006. http://eprint.iacr.org/.

[12] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2007.

[13] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 378–394. Springer-Verlag, 2006.

[14] J. Lee and C. Park. An efficient authenticated key exchange protocol with a tight security reduction. Cryptology ePrint Archive, Report 2008/345, 2008. http://eprint.iacr.org/, retrieved on January 12, 2009.

[15] A. Menezes and B. Ustaoğlu. Comparing the pre- and post-specified peer models for key agreement. In *Proceedings of ACISP 2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 53–68. Springer-Verlag, 2008.

[16] NIST. SKIPJACK and KEA algorithm specification, 1998. http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.

[17] T. Okamoto. Authenticated key exchange and key encapsulation in the standard model. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 474–484. Springer-Verlag, 2007.

[18] B. Ustaoğlu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Des. Codes Cryptography*, 46(3):329–342, 2008.

[19] B. Ustaoğlu. Comparing SessionStateReveal and EphemeralKeyReveal for Diffie-Hellman protocols. In J. Pieprzyk and F. Zhang, editors, *Provable Security: Third International Conference, ProvSec 2009*, volume 5848 of *Lecture Notes in Computer Science*, pages 183–197. Springer-Verlag, 2009.

[20] J. Xia, J. Wang, L. Fang, Y. Ren, and S. Bian. Formal proof of relative strengths of security between ECK2007 model and other proof models for key agreement protocols. Cryptology ePrint Archive, Report 2008/479, 2008. http://eprint.iacr.org/, retrieved on January 12, 2009.

A Protocol model

The protocol model below was analyzed in Scyther using the following command:

```
scyther-linux \
  --SSR=1 --SSRinfer=2 --SKR=1 \
  --LKRaftercorrect=1 --LKRactor=1 \
  --max-runs=4 \
  naxos.spdl
```

where `naxos.spdl` is the filename of the protocol description given below.

The first five switches set up the correct security model abstraction for the eCK' model, see [1] for details. In particular, the first two switches enable the Session-StateReveal query and trigger the automatic inference of state. The third switch, `SKR`, enables the Reveal query. The `LKRaftercorrect` switch enables long-term key reveal after a session in which the adversary did not interfere, modeling weak Perfect Forward Secrecy. The `LKRactor` switch enables detection of key-compromise impersonation attacks.

The `max-runs` switch bounds the depth of the search space thereby guaranteeing termination of the tool.

naxos.spdl

```
/*
 * NAXOS protocol
 *
 * Input file for Scyther-compromise v0.3.
 *
 * Modeled by Cas Cremers 2009/2010.
 *
 * Original protocol description in "Stronger
 * Security of Authenticated Key Exchange" by
 * LaMacchia, Lauter, and Mityagin, 2006.
 *
 * To find the attacks in the paper:
 *
 * scyther-linux \
 *   --SSR=1 --SSRinfer=2 --SKR=1 \
 *   --LKRaftercorrect=1 --LKRactor=1 \
 *   --max-runs=4 \
 *   naxos.spdl
 */

// Hash functions
```

```
hashfunction h1,h2;

/*
 * Here we use the construct 'p(t,t')' to model
 * the notation t^{t'} from the abstract language
 * in the paper.
 *
 * The construct is modeled using a wrapper
 * function p (a one-way function), and we use a
 * helper protocol (@DHapproximation, specified
 * later) to model a strict subset of the
 * corresponding equational theories.
 */
hashfunction p;

// Generator constant
const g;

/*
 * Simulate public knowledge of public keys.
 *
 * The '@' prefix of the protocol name denotes
 * that it is a helper protocol, which is used by
 * Scyther for displaying, and such protocols are
 * ignored in auto-generation of protocol
 * modifiers.
 */
protocol @publickeys(PK) {
  role PK {
    send_!1(PK,PK, p(g,sk(PK)));
  }
}

/*
 * Approximation for the equational theory g^ab =
 * g^ba in subterms of the Naxos protocol.
 */
protocol @DHapproximation(RA,RB,RC,RD) {
  role RA {
    var X,Y, T1,T2: Ticket;
    recv_!1(RA,RA,h2( p(p(g,X),Y),T1,T2, RA,RB));
    send_!2(RA,RA,h2( p(p(g,Y),X),T1,T2, RA,RB));
  }
  role RB {
    var X,Y, T1,T2: Ticket;
    recv_!3(RB,RB,h2( T1,p(p(g,X),Y),T2, RA,RB));
    send_!4(RB,RB,h2( T1,p(p(g,Y),X),T2, RA,RB));
  }
  role RC {
    var X,Y, T1,T2: Ticket;
```

```
    recv_!5(RC,RC,h2( T1,T2, p(p(g,X),Y), RA,RB));
    send_!6(RC,RC,h2( T1,T2, p(p(g,Y),X), RA,RB));
  }
  role RD {
    var X,Y: Ticket;
    recv_!7(RD,RD, p(p(g,X),Y));
    send_!8(RD,RD, p(p(g,Y),X));
  }
}

/*
 * The main part of the Naxos protocol description
 *
 * No session-state is specified in the protocol
 * description because it is inferred
 * automatically by the Scyther tool.
 *
 * The SKR claims both identify the session keys
 * and claim that these session keys are secret.
 */
protocol naxos(I,R) {
  role I {
    fresh eskI: Nonce;
    var Y: Ticket;

    /* 'send' and 'recv' match their counterparts
     * in the abstract specification.
     */
    send_1(I,R, p(g,h1(eskI,sk(I))) );
    recv_2(R,I, Y );

    /* 'claim(role,SKR,t)' corresponds to
     * 'sesionkey(t)' in the abstract
     * specification.
     */
    claim(I,SKR,h2(
      p(Y,sk(I)),
      p(p(g,sk(R)),h1(eskI,sk(I))),
      p(Y,h1(eskI,sk(I))),
      I,R));
  }
  role R {
    fresh eskR: Nonce;
    var X: Ticket;

    recv_1(I,R, X );
    send_2(R,I, p(g,h1(eskR,sk(R))) );

    claim(R,SKR,h2(
      p(p(g,sk(I)),h1(eskR,sk(R))),
```

```
      p(X,sk(R)),
      p(X,h1(eskR,sk(R))),
      I,R));
  }
}
```