Session-state Reveal **is stronger than** Ephemeral Key Reveal**: Attacking the NAXOS Authenticated Key Exchange protocol**

(extended version)

Cas J.F. Cremers[*]

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

Abstract. In the paper "Stronger Security of Authenticated Key Exchange" [1, 2], 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. The model includes a new notion of an Ephemeral Key Reveal adversary query, which is claimed in e. g. [2–4] to be at least as strong as the Session-state Reveal query. We show that Session-state Reveal is stronger than Ephemeral Key Reveal, implying that the eCK security model is incomparable to the CK model [5, 6]. In particular we show that the proposed NAXOS protocol from [1, 2] does not meet its security requirements if the Session-state Reveal query is allowed in the eCK model. We discuss the implications of our result for some related protocols proven correct in the eCK model, and discuss the interaction between Session-state Reveal and protocol transformations.

Keywords. Provably-secure, Authenticated Key Exchange, Session-state reveal, Ephemeral Key reveal.

1 Introduction

In the area of secure key agreement protocols many security models [1, 5, 7–10] and protocols have been proposed. Many of the proposed protocols have been shown to be correct in some particular security model, but have also shown to be incorrect 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 [8], 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. In particular, we focus on the ability of the adversary to learn

[*] This work was supported by the Hasler Foundation within the ComposeSec project.

the local state of an agent. For example, when an agent chooses a random value, or computes the hash function of a 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. This corresponds to the situation in which the long-term private keys reside in e. g. a tamper-proof module (TPM) or cryptographic coprocessor, while the remainder of the protocol computations are done in regular (unprotected) memory. The corresponding adversary ability is captured in security models for key agreement protocols by the Session-state Reveal query.

A drawback of the Session-state Reveal query in current security models is that the query is often underspecified. For example, in the Canetti-Krawczyk (CK) model [5], Session-state Reveal 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.

In [1,2] a security model is proposed which is said to be stronger than existing AKE (Authenticated Key Exchange) security models. The model is based on the CK model, and is referred to in [1] as the Extended Canetti-Krawczyk (eCK) model. The eCK model differs in a number of aspects from the CK model, where the main difference seems to be that the adversary is allowed to reveal part of the local state of participants even during a normal protocol session. A more subtle aspect in which the eCK model differs from the CK model is that it replaces the Session-state Reveal query by a new Ephemeral Key Reveal query. In this paper we focus on this aspect.

In order to provide a definition of the local state within the security model, the eCK model (re)defines the notion of ephemeral key and introduces a corresponding Ephemeral Key Reveal query that reveals this key. The ephemeral key is defined to contain all secret session-specific information. The authors argue for the new Ephemeral Key Reveal query that "by setting the ephemeral secret key equal to all session-specific secret information, we seem to cover all definitions of Session-state Reveal queries which exist in literature" [2, p. 2]. Similar arguments can be found in [3, 4, 11, 12]. Within the resulting eCK model, the NAXOS protocol is proposed and proven correct in [1].

Contrary to the above, it is argued in [13] that strictly speaking the eCK and CK models are incomparable. Regarding the difference between Session-state Reveal and Ephemeral Key Reveal, 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" [13, Section 3.1].

Contributions In this paper we show that contrary to the claims in [2–4, 11, 12], the Ephemeral Key Reveal query is weaker than the Session-state Reveal query.

Consequently, it follows that the CK and eCK models are incomparable, as the CK model does not allow compromise of the ephemeral key of the tested session. We show that the difference between the queries not only has theoretical but also practical implications, by providing two attacks on the NAXOS protocol, which can be performed using Session-state Reveal, but cannot be performed by using Ephemeral Key Reveal. The security model we use is nearly identical to the eCK model: we only replace Ephemeral Key Reveal by Session-state Reveal. Our attacks are also valid in the CK model, which implies that there is a meaningful difference between CK and eCK, as NAXOS was proven correct in the eCK model. We show how our attacks can be extended to the KEA, KEA+, and KEA+C protocols, and we discuss the interaction between Session-state Reveal and protocol transformations in e. g. the CK model.

The attacks presented here were found automatically by the Scyther tool [14]. For our attacks we use the NAXOS protocol exactly as specified in [1, 2]. 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 local state just before the computation. As a result, performing a Session-state Reveal query just before the computation of $H_2(x)$ reveals x. This assumption does not require changing to the protocol. Rather, we make the contents of the session state explicit, as would be required for a proof in the CK model.

We proceed as follows. In Section 2 we explain some notation, and present the NAXOS protocol. Then, in Section 3 we show two attacks on this protocol that use Session-state Reveal. Further issues are discussed in Section 4, and we conclude in Section 5.

2 The NAXOS key exchange protocol

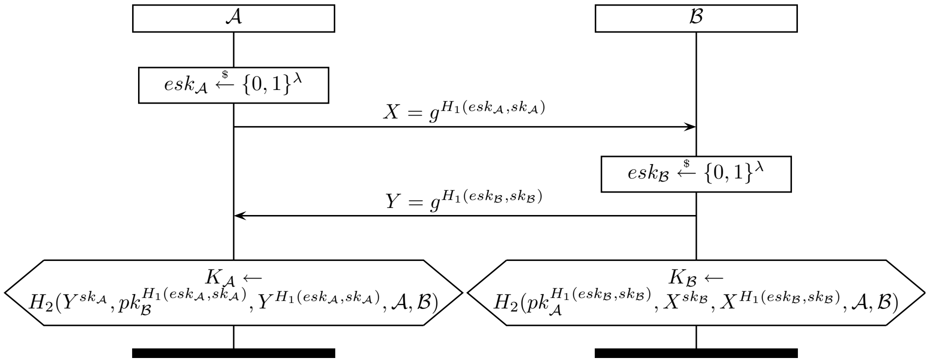
$\mathcal{A}$
$\mathcal{B}$
$esk_{\mathcal{A}} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$
$X = g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$
$esk_{\mathcal{B}} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$
$Y = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}$
$K_{\mathcal{A}} \leftarrow H_2(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \mathcal{A}, \mathcal{B})$
$K_{\mathcal{B}} \leftarrow H_2(pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B})$

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

Table 1. Notation

$\mathcal{A}, \mathcal{B}$	The *initiator* and *responder* roles of the protocol.
a, b	Agents (participants) executing roles of the protocol.
G	A mathematical 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 $pk_{\mathsf{a}} = g^{sk_{\mathsf{a}}}$.
H_1, H_2	Hash functions, where $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \to \{0,1\}^\lambda$ (for some constant λ).
$esk_{\mathsf{a}}, esk'_{\mathsf{a}}$	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.
λ	A constant.
$x \xleftarrow{\$} S$	The variable x is drawn uniformly from the set S.
$x \leftarrow e$	The variable x is assigned the result of the expression e.

The NAXOS protocol, as defined in [1, 2], is shown in Figure 1. NAXOS builds on earlier ideas from the KEA and KEA+ protocols [15,16]. 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}. In Table 1 we give an overview of the notation used in the protocol as well as the remainder of this paper. We follow the notation from [1] where possible.

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 combining the long-term private key with the short term ephemeral key inside the hash function, 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 a TPM, and computes other operations in unprotected memory. For full details we refer the reader to [1, 2].

In a normal execution, we have the following equivalences based on the properties of the modular exponentiation:

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

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

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

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

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

3 Attacking NAXOS using Session-state Reveal

3.1 Security model eCK'

We use a slightly modified security model from the one defined in [1]. The only change is that we replace the Ephemeral Key Reveal query by the Session-state Reveal query throughout the security definition. In particular, we require that whenever $H_2(x_1, \ldots, x_n)$ is computed, $x_1, \ldots, x_n$ are part of the local state just before the computation, and can therefore be revealed by a Session-state Reveal query. An example of an execution model where this condition holds, is a TPM 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 TPM.

In contrast, applying the Ephemeral Key Reveal query to a session of the agent a in the eCK model (and original NAXOS proof) from [1] reveals only the ephemeral key esk_{a}.

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*.

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.*

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 $L = comm_1, \ldots, comm_n$, such that the session identifier of sid is $(role, ID, ID', L)$ and the session identifier of sid$'$ is $(role', ID', ID, L)$.*

In the eCK model, the adversary does not have access to a Session-state Reveal query, but instead has Ephemeral Key Reveal. Below we redefine the notion of clean and the security experiment from the eCK model [1, p. 8-9], in which we replace Ephemeral Key Reveal with Session-state Reveal, to define our security model eCK'.

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 *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* $\mathsf{Reveal}(sid)$*, i. e. the session key of session sid is not revealed.*
3. *The experiment does not include both* $\mathsf{Long\text{-}term\ Key\ Reveal}(\mathsf{a})$ *and* $\mathsf{Session\text{-}state\ Reveal}(sid)$*.*
4. *If no session exists that matches sid, then the experiment does not include* $\mathsf{Long\text{-}term\ Key\ Reveal}(\mathsf{b})$*.*
5. *If a session sid∗ exists*[1] *that matches sid, then*
   (a) *the experiment does not include* $\mathsf{Reveal}(sid*)$*, i. e. the session key of session sid∗ is not revealed, and*
   (b) *the experiment does not include both* $\mathsf{Long\text{-}term\ Key\ Reveal}(\mathsf{b})$ *and* $\mathsf{Session\text{-}state\ Reveal}(sid*)$*.*

In the eCK' security model, queries such as Session-state Reveal may not be performed on *clean* sessions or their matching sessions as in [1, p. 7-8]. This is meant to exclude the cases in which an Session-state Reveal query trivially reveals the session key, such that no protocol could satisfy the security definition.

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

1. *The adversary may perform* $\mathsf{Send}(\mathsf{a}, \mathsf{b}, comm)$*,* $\mathsf{Long\text{-}term\ Key\ Reveal}(\mathsf{a})$*, and* $\mathsf{Reveal}(sid)$ *queries as in [1].*
2. *The adversary may perform a* $\mathsf{Session\text{-}state\ Reveal}(sid)$ *query. (This query replaces* $\mathsf{Ephemeral\ Key\ Reveal}(sid)$ *in the definition from [1].)*
3. *The adversary performs a* $\mathsf{Test}(sid)$ *query on a single* clean *session sid. A coin is flipped:* $b \overset{\$}{\leftarrow} \{0, 1\}$*. 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.*
4. *The adversary outputs a* $\mathsf{Guess}(b')$ *bit* b'*, after which the experiment ends.*

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

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

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

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 show two attacks on NAXOS in the eCK' model: One using test queries on sessions of the initiator type $\mathcal{A}$ and one using the responder type $\mathcal{B}$.

[1] There may not be a unique matching session $sid*$ for all executions of all protocols, but in the case of NAXOS, where each sent message contains randomness from the sending session, the matching session is unique if sid is a completed session.

3.2 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 that can reveal the local state of an agent.

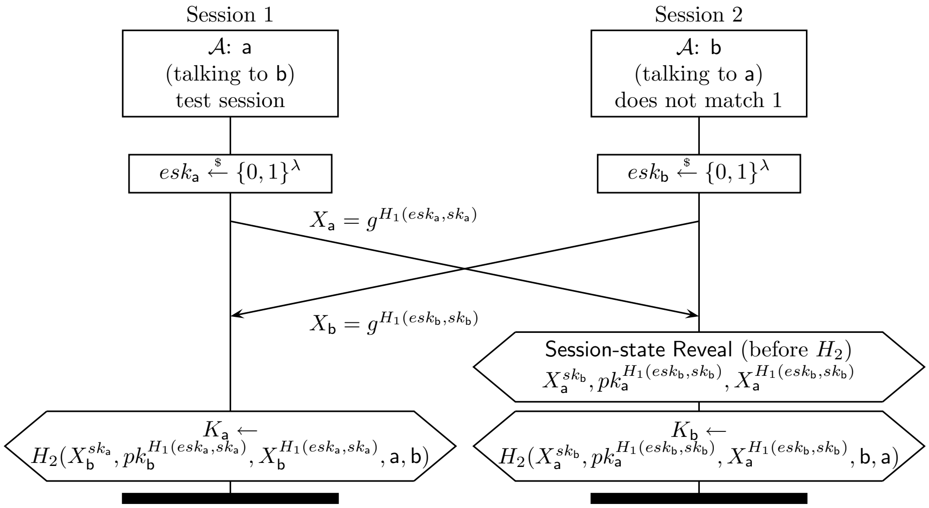
Session 1
$\mathcal{A}$: a
(talking to b)
test session
$esk_{\mathsf{a}} \xleftarrow{\$} \{0,1\}^{\lambda}$
Session 2
$\mathcal{A}$: b
(talking to a)
does not match 1
$esk_{\mathsf{b}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$
$X_{\mathsf{b}} = g^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$
Session-state Reveal (before H_2)
$X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$
$K_{\mathsf{a}} \leftarrow$
$H_2(X_{\mathsf{b}}^{sk_{\mathsf{a}}}, pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{a}, \mathsf{b})$
$K_{\mathsf{b}} \leftarrow$
$H_2(X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, \mathsf{b}, \mathsf{a})$

Fig. 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.

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

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

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

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

During the computation of K_{b}, the adversary uses Session-state Reveal 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

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

$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})}, \tag{8}$$

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

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 3 on page 5 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-state Reveal, such as the CK model [5].

3.3 Attacking the responder

Second, we show an attack for a test query on a responder session in Figure 3. It seems this attack is more easily exploited than the previous one.

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.

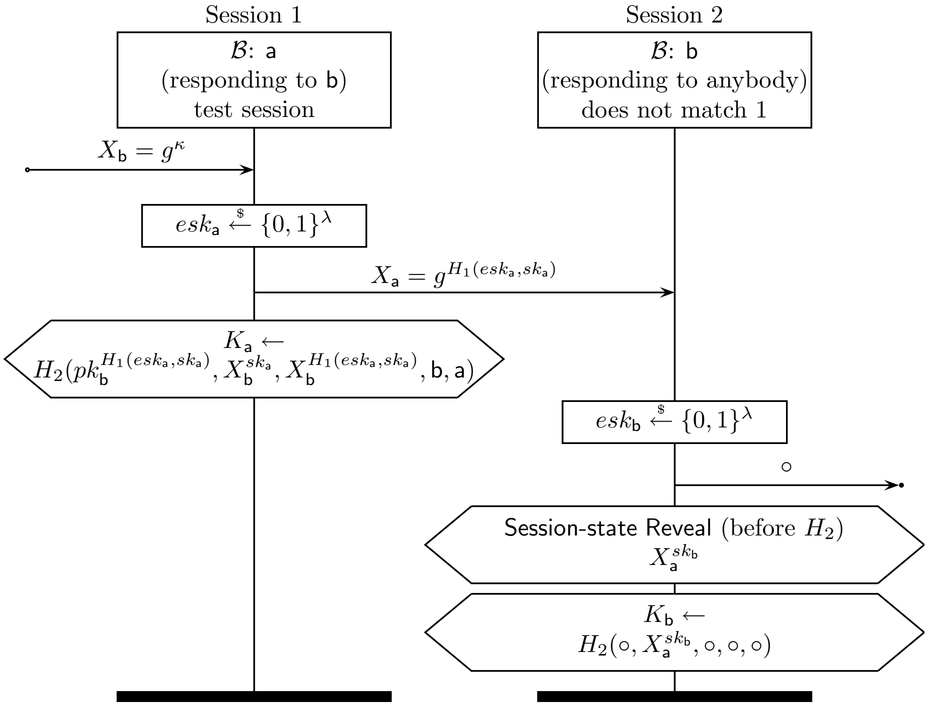
Session 1
$\mathcal{B}$: a
(responding to b)
test session
Session 2
$\mathcal{B}$: b
(responding to anybody)
does not match 1
$X_{\mathsf{b}} = g^{\kappa}$
$esk_{\mathsf{a}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$
$K_{\mathsf{a}} \leftarrow$
$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{sk_{\mathsf{a}}}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a})$
$esk_{\mathsf{b}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$\circ$
Session-state Reveal (before H_2)
$X_{\mathsf{a}}^{sk_{\mathsf{b}}}$
$K_{\mathsf{b}} \leftarrow$
$H_2(\circ, X_{\mathsf{a}}^{sk_{\mathsf{b}}}, \circ, \circ, \circ)$

Fig. 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.

5. a computes the session key

$$K_{\mathsf{a}} = H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{sk_{\mathsf{a}}}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}) \tag{10}$$

which is equal to

$$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, g^{\kappa\ sk_{\mathsf{a}}}, g^{\kappa\ H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}). \tag{11}$$

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.
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-state Reveal 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:

$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})}, \tag{12}$$

$$(pk_{\mathsf{a}})^{\kappa} = g^{sk_{\mathsf{a}}\kappa} = X_{\mathsf{b}}^{sk_{\mathsf{a}}}, \tag{13}$$

$$(X_{\mathsf{a}})^{\kappa} = g^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})\kappa} = X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}},sk_{\mathsf{a}})}. \tag{14}$$

The three terms on the right-hand side are the first three components of the session key K_{a} from Formula 10.

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 3 on page 5. In practical terms, this attack even allows the adversary to determine a part of the session key of a.

For this attack there are also some observations to be made:

- 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.
- In this attack, the adversary does not need to learn any long term private keys or ephemeral keys.
- 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 $z \neq \mathsf{a}$.

4 Discussion

The structure of our attacks can be used to attack some protocols that were proven correct in the CK model [5]. We first briefly discuss these other protocols, and afterwards discuss the implications for existing proofs and protocol transformation theorems in the CK model.

4.1 The KEA, KEA+ and KEA+C protocols

In [16], the KEA+ protocol is proven correct 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 [16] and KEA from [15]. 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 [16] 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 exact same scenarios and exploit the same Session-state Reveal definition, in which the inputs to the final hash function are part of the local state.

4.2 Session-state Reveal and protocol transformations in the CK model

In the CK model [5] the Session-state Reveal query is defined as revealing the full internal state of the Turing machine executing the protocol to the adversary. This internal state is not defined within the CK model, and can be viewed as a parameter of the correctness proof of the protocol. As a result, proving that a protocol is correct in the CK model requires one to define this internal state. Technically this implies that the resulting proof holds only for execution models corresponding to this definition of the internal state. This puts restrictions on the implementation details of the protocol code as well as on the platform executing the code.

In existing protocol proofs in the CK model the internal state is not explicitly stated as a parameter of the correctness proof. In most cases, the internal state is simply defined as the ephemeral secret, i. e. the private exponent of a participant in a Diffie-Hellman style key exchange. As a consequence, any implementation in which the local state (as revealed to an adversary) contains more information, falls outside of the scope of the proof.

However, making the definition of the internal state an explicit parameter of the correctness proofs has implications for the methodology underlying the CK model. Central to the methodology underlying the model is the notion of security preserving (or security enhancing) protocol transformations. An example of a central result is Theorem 6 from [5, p. 16]. This theorem involves the notion of (MT-)authenticators, which are protocol transformations that satisfy certain security preserving/enhancing properties. The theorem aims to establish that a protocol that is secure in one security model (AM) can be transformed by an authenticator into another protocol that satisfies the same security property in a stronger adversarial model (UM). In this case the security property is SK-security, which involves an adversary that has access to the Session-state Reveal query.

Theorem 6 from [5, p. 16] can be rephrased in the following way. Let P be a protocol and let f be an (MT-) authenticator. If the protocol P satisfies SK-security in the AM model, then the protocol $f(P)$ satisfies SK-security in the UM model. The proof of this theorem is generic and applies to any authenticator.

In the precondition of this theorem and the definition of authenticators, there is no constraint that prevents authenticators from changing the local state, i. e., there is no requirement on f that ensures that Session-state Reveal for P is equal to Session-state Reveal for $f(P)$. Note that from a practical point of view, including such a requirement may be unrealistic: transforming the protocol in any non-trivial way implies that the local state of the protocol $f(P)$ is different from that of the protocol P. Even with the MT-authenticator proposed in paper the state is underspecified, such that a straightforward implementation leads to a state-reveal attack [17].

If we assume that that applying the authenticator does change the local state, i. e. Session-state Reveal for $f(P)$ is not equal to Session-state Reveal for P, then it is not immediately clear how to prove the correctness of the theorem, as it would involve proving that the transformation of the local state does not introduce new attacks, possibly along the lines of the attacks presented here, that exploit Session-state Reveal for $f(P)$. Recall that the attacks on NAXOS do not require revealing the ephemeral key (which would already be in the local state of P), nor the long-term private keys (which would be excluded from Session-state Reveal for $f(P)$), but rather some intermediate computations. We expect that a generic proof of this theorem requires a significant restriction on the class of allowed authenticators.

The existence of these attacks shows the importance of explicitly specifying the definition of local 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 local state. It would be more precise to say that in [16] KEA+ is proven secure with respect to the CK model if the local state only includes the ephemeral keys.

5 Conclusion

In common definitions of AKE security the Session-state Reveal query is underspecified. The definition of Session-state Reveal is only made explicit in particular protocol proofs. This approach turns the exact definition of Session-state Reveal 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 [1,2] the Session-state Reveal query is replaced by the Ephemeral Key Reveal query, which is claimed to be at least as strong as Session-state Reveal. Thus, the notion of Session-state Reveal is reduced to Ephemeral Key Reveal. Reducing Session-state Reveal to Ephemeral Key Reveal 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 [1, 3, 4]. However, the validity of this reduction has not been proven.

The validity of the reduction is informally argued in [2], and similar arguments can be found in other works that use the eCK model [3, 4], e. g. in [4, p. 333]: "In general, by specifying that the session specific private information (the session state) is part of the ephemeral private key, the Session-state Reveal and Ephemeral Key Reveal queries can be made functionally equivalent".

In this paper we have shown that the reduction is invalid, that is, a security model with Ephemeral Key Reveal (eCK) is not as strong as a model with Session-state Reveal (eCK'). The attacks presented here on the NAXOS protocol, which was proven correct for Ephemeral Key Reveal in [1], strictly depend on the use of the Session-state Reveal query.

The attacks presented here fall just outside the eCK security model, and they therefore do not indicate a problem with the proofs in [1]. 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. [1]. 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 local state is revealed. Consequently, the CK and eCK models are not only theoretically, but also practically incomparable.

The structure of our attacks on NAXOS can be translated to attacks on the KEA, KEA+, and KEA+C protocols from [15, 16]. As a result, also these protocols are not CK-secure if the session state includes the inputs to the final hash function. We furthermore observed that it is non-trivial to combine protocol proofs that consider the Session-state Reveal query, such as those in the CK model, with protocol transformations.

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 local state. This effect is not captured by the definition of Ephemeral Key Reveal, which is the ultimate problem with the reduction from Session-state Reveal to Ephemeral Key Reveal, as was already noted in [13]. The attacks presented in this paper exploit exactly this difference.

A possible practical interpretation of the difference between the models is the following. The CK model considers a TPM 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 TPM. 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 a malicious (i. e. information-leaking) random number generator, which implies that the adversary learns all ephemeral keys.

Under the assumption that the session keys are part of the session state, a security model that allows Session-state Reveal on the test session or its partner would invalidate any AKE model. A possible alternative strengthening of eCK would be a hybrid model, i. e., a model in which Ephemeral Key Reveal is allowed on the test session and its partner, and in which Session-state Reveal is allowed on all other sessions. Note that the attacks on NAXOS presented here would be valid in such a hybrid model, as they exploit Session-state Reveal on non-matching sessions. The question remains whether it is possible to adapt NAXOS to be secure in such a hybrid model.

References

1. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: ProvSec. Volume 4784 of Lecture Notes in Computer Science., Springer-Verlag (2007) 1–16
2. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073 (2006) http://eprint.iacr.org/.
3. Okamoto, T.: Authenticated key exchange and key encapsulation in the standard model. In: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer-Verlag (2007) 474–484
4. Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Cryptography **46**(3) (2008) 329–342
5. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: EUROCRYPT. Volume 2045 of Lecture Notes in Computer Science., Springer-Verlag (2001) 453–474
6. Krawczyk, H.: HMQV: A high-performance secure diffie-hellman protocol. In: CRYPTO 2005. Volume 3621 of Lecture Notes in Computer Science., Springer-Verlag (2005) 546–566
7. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.: Provably authenticated group Diffie-Hellman key exchange. In: CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security, New York, USA, ACM (2001) 255–264
8. Menezes, A., Ustaoglu, B.: Comparing the pre- and post-specified peer models for key agreement. In: Proceedings of ACISP 2008. Volume 5107 of Lecture Notes in Computer Science., Springer-Verlag (2008) 53–68
9. Choo, K.K., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment proofs. In: ASIACRYPT. Volume 3788 of Lecture Notes in Computer Science., Springer-Verlag (2005) 624–643
10. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: EUROCRYPT. Lecture Notes in Computer Science, Springer-Verlag (2000) 139–155
11. Xia, J., Wang, J., Fang, L., Ren, Y., Bian, S.: 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.
12. Lee, J., Park, C.: 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.
13. Boyd, C., Cliff, Y., Nieto, J., Paterson, K.: Efficient one-round key exchange in the standard model. In: ACISP. Volume 5107 of Lecture Notes in Computer Science., Springer-Verlag (2008) 69–83
14. Cremers, C.: 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., Springer-Verlag (2008) 414–418
15. NIST: SKIPJACK and KEA algorithm specification (1998) http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.
16. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: PKC 2006. Volume 3958 of Lecture Notes in Computer Science., Springer-Verlag (2006) 378–394

17. Tian, X., Wong, D.: Session corruption attack and improvements on encryption based MT-authenticators. In: Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings. Volume 3860 of Lecture Notes in Computer Science., Springer-Verlag (2006) 34–51