Session-state Reveal is stronger than Ephemeral Key Reveal: Breaking the NAXOS key exchange protocol

MANUSCRIPT

Cas J.F. Cremers

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

September 18, 2008

Abstract

In the papers "Stronger Security of Authenticated Key Exchange" [LLM07, LLM06], a new security model for key exchange protocols is proposed. The new model is suggested to be at least as strong as previous models for key exchange protocols. In particular, the model includes a new notion of an Ephemeral Key Reveal adversary query, which is claimed in [LLM06, Oka07, Ust08] to be at least as strong as existing definitions of the Session-state Reveal query. We show that for some protocols, Session-state Reveal is strictly stronger than Ephemeral Key Reveal. In particular, we show that the NAXOS protocol from [LLM07, LLM06] does not meet its security requirements if the Session-state Reveal query is allowed in the security model.

1 Introduction

In the area of secure key agreement protocols many security models [BCPQ01, MU08, LLM07, CBH05, CK01, BPR00] 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 very recent works such as [MU08] on security models for key agreement protocols, 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 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. This 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 such as the Canetti-Krawczyk (CK) model [CK01], is that the query is underspecified. The security model does not define what is exactly revealed in such a query for a given protocol. Effectively, this decision is postponed to the proof of a particular protocol.

In [LLM06,LLM07] 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 [LLM07] 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.

The Session-state Reveal query from the CK model leaves underspecified what exactly is revealed. To address this, 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" [LLM06, p. 2]. Similar arguments can be found in [Ust08,Oka07]. Within the resulting eCK model, the NAXOS protocol is proposed and proven correct in [LLM07].

In [BCNP08] it is argued 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" [BCNP08, Section 3.1]. However, the authors also remark that "it is arguable whether the differences between the two models are meaningful in reality" [BCNP08, Section 3.1].

In this paper we show that contrary to the claims in [LLM06, Ust08, Oka07], Ephemeral Key Reveal is weaker than Session-state Reveal. We show this 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. Furthermore, 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.

The assumption needed for our attacks is 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.

We base our assumption on two generic observations and two observations specific to NAXOS. First, our requirements on Session-state Reveal fit the definition[1] of Session-state Reveal given in the CK model [CK01, p. 457]. Second, given that the difference between long-term private keys and local state seems to be inspired by TPM-based scenarios in which only the long-term keys are protected, it seems reasonable to assume that intermediate computations that do not involve the long-term keys are performed in local memory. Third, for protocols such as NAXOS, it seems that such a value x needs to be stored in memory for a

[1] A subtlety of the CK model is that in general, a receive-send combination is considered atomic. However, the CK model explicitly allows for modeling any computations as sub-protocols (or procedure calls). Modeling H_2 within NAXOS as a procedure call implies that the inputs to H_2 are in the local state, which exactly captures our requirements.

similar time duration as the ephemeral key. Fourth, in the case of NAXOS, if we assume H_2 is computed entirely in a TPM, there is no reason to store the ephemeral key in unprotected memory, in which case the local state effectively becomes empty.

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. We conclude in Section 4.

Acknowledgements The author would like to thank David Basin, Ran Canetti, Mark Manulis, and Björn Tackmann for useful discussions.

2 The NAXOS key exchange protocol

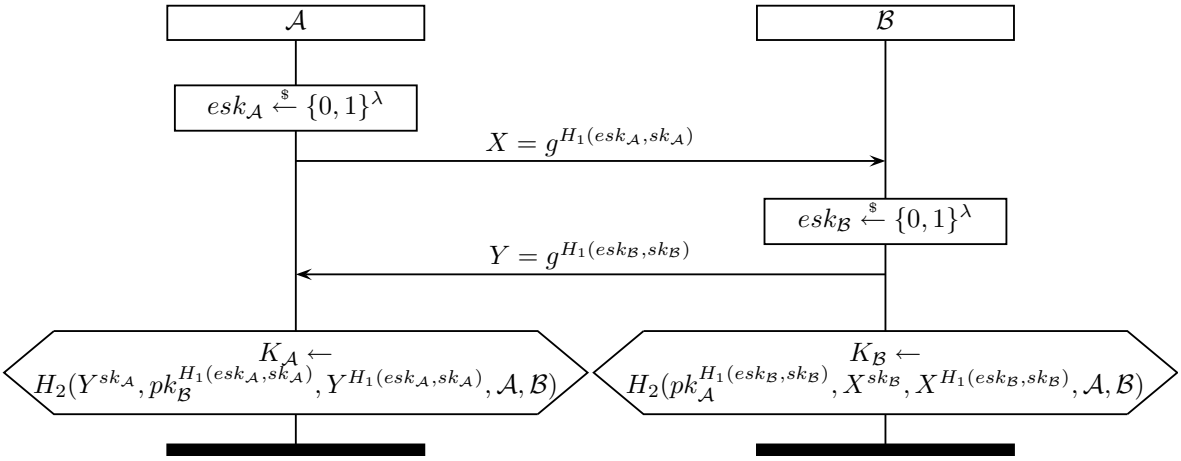

Figure 1: The NAXOS protocol. At the end of a normal execution we have that $K_{\mathcal{A}} = K_{\mathcal{B}}$ ($pk_x = g^{sk_x}$).

The NAXOS protocol, as defined in [LLM06, LLM07], is shown in Figure 1. NAXOS builds on earlier ideas from the KEA and KEA+ protocols [NIS98, LM06]. 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 [LLM07] 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 [LLM07, LLM06].

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.
clean	A notion from [LLM07]: reveal queries can only be performed on *clean* sessions, see Definition 3 on the following page.

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

$$H_2(g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})sk_{\mathcal{A}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})sk_{\mathcal{B}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B}). \tag{1}$$

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

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

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

$$Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} = X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})} \tag{4}$$

3 Attacking NAXOS using Session-state Reveal

3.1 Security model eCK'

We use a slightly modified security model from the one defined in [LLM07]. 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.

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.

In the security model, queries such as Session-state Reveal may not be performed on *clean* sessions or their matching sessions as in [LLM07, 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 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 $role, role', ID, ID', comm_1, \ldots, comm_n$ such that $role \neq role'$, the session identifier of sid is $(role, ID, ID', comm_1, \ldots, comm_n)$ and the session identifier of sid' is $(role', ID', ID, comm_1, \ldots, comm_n)$.

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 [LLM07, 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 Reveal(sid), i. e. the session key of session sid is not revealed.
3. The experiment does not include both Long-term Key Reveal(a) and Session-state Reveal(sid).
4. If no session exists that matches sid:
   (a) The experiment does not include Long-term Key Reveal(b).
5. If a session $sid*$ exists[2] that matches sid:
   (a) The experiment does not include Reveal($sid*$), i. e. the session key of session $sid*$ is not revealed.
   (b) The experiment does not include both Long-term Key Reveal(b) and Session-state Reveal($sid*$).

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

[2]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

1. The adversary may perform Send(a, b, *comm*), Long-term Key Reveal(a), and Reveal(*sid*) queries as in [LLM07].

2. The adversary may perform a Session-state Reveal(*sid*) query (replacing Ephemeral Key Reveal(*sid*) in the definition from [LLM07]).

3. The adversary performs a Test(*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.

4. The adversary outputs a Guess(b') bit b', after which the experiment ends.

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

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

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 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 $\mathcal{B}$.

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.

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

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

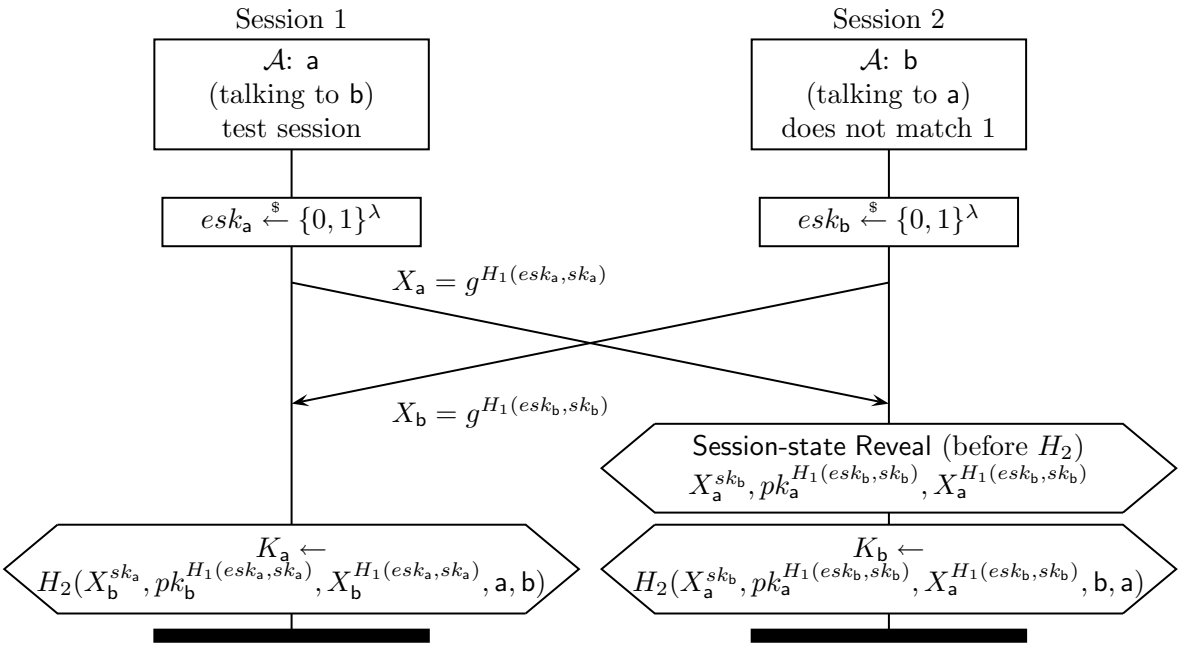

Figure 2: Attacking an initiator session. Note that $K_a \neq K_b$. The adversary can compute K_a after compromising the local state of b.

7. The adversary sends the message X_a to b.

8. b computes the session key

$$K_b = H_2(X_a^{sk_b}, pk_a^{H_1(esk_b, sk_b)}, X_a^{H_1(esk_b, sk_b)}, b, a). \tag{6}$$

   During the computation of K_b, the adversary uses Session-state Reveal to learn the input to H_2. In particular, the adversary learns $X_a^{sk_b}$, $pk_a^{H_1(esk_b, sk_b)}$, and $X_a^{H_1(esk_b, sk_b)}$.

9. The adversary now knows

$$pk_a^{H_1(esk_b, sk_b)} = g^{sk_a H_1(esk_b, sk_b)} = X_b^{sk_a}, \tag{7}$$

$$X_a^{sk_b} = g^{H_1(esk_a, sk_a) sk_b} = pk_b^{H_1(esk_a, sk_a)}, \tag{8}$$

$$X_a^{H_1(esk_b, sk_b)} = g^{H_1(esk_a, sk_a) H_1(esk_b, sk_b)} = X_b^{H_1(esk_a, sk_a)}. \tag{9}$$

   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 [CK01].

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.

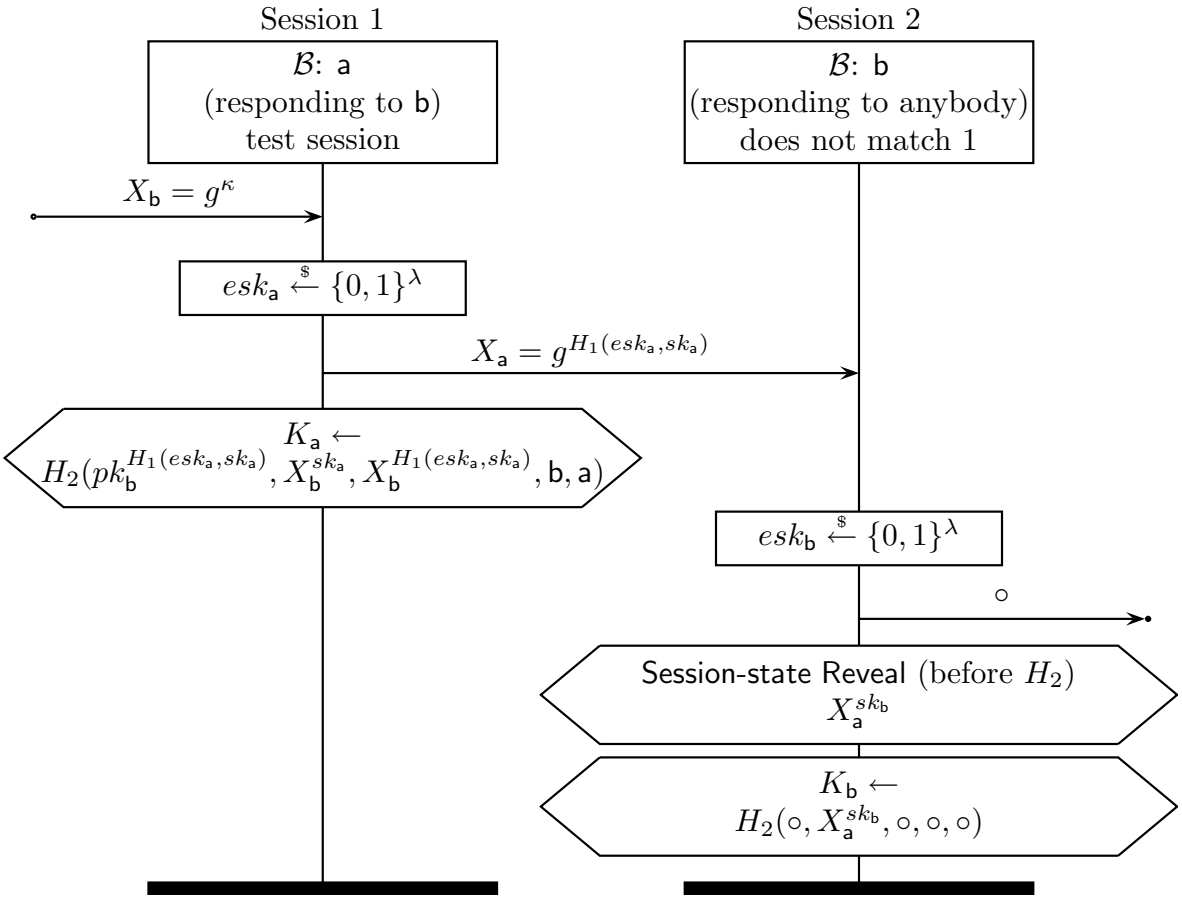

Figure 3: Attack on a responder session. We have $K_{\mathsf{a}} \neq K_{\mathsf{b}}$. The adversary can compute (and even contribute to) K_{a} after revealing the local state of b.

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

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

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 Conclusion

In common definitions of AKE security the Session-state Reveal query is underspecified. In many cases the definition 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 "AKE secure" does not mean they meet exactly the same property.

In [LLM07, LLM06] 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 [LLM07, Ust08, Oka07]. However, the validity of this reduction has not been proven.

The validity of the reduction is informally argued in [LLM06], and similar arguments can be found in other works that use the eCK model [Ust08, Oka07], e. g. in [Ust08, 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 [LLM07], 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 [LLM07]. 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. [LLM07]. Furthermore, the attacks are also valid in the CK model, which shows that contrary to the suggestion in [BCNP08], the difference between CK and eCK is in fact meaningful in reality. 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.

Note that the structure of our attacks on NAXOS can be translated to attacks on the KEA, KEA+, and KEA+C protocols from [NIS98, LM06]. Therefore, it seems that also these protocols cannot be considered to be 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. The way in which the ephemeral key (and hence Ephemeral Key Reveal) is defined and used in proofs, excludes the intermediate products of such subsequent computations. This is the ultimate problem with the reduction from Session-state Reveal to Ephemeral Key Reveal, as was already noted in [BCNP08]. Precisely this difference is exploited by the attacks shown here.

The question remains whether it is possible to adapt NAXOS to satisfy a security model similar to eCK that allows for Session-state Reveal queries.

References

[BCNP08] C. Boyd, Y. Cliff, J.G. Nieto, and K.G. Paterson. Efficient one-round key exchange in the standard model. In *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2008.

[BCPQ01] 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.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, Lecture Notes in Computer Science, pages 139–155. Springer, 2000.

[CBH05] K-K.R. 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, 2005.

[CK01] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[LLM06] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073, 2006. http://eprint.iacr.org/.

[LLM07] B.A. 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, 2007.

[LM06] 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, 2006.

[MU08] A. Menezes and B. Ustaoglu. 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, 2008.

[NIS98] NIST. SKIPJACK and KEA algorithm specification, 1998. http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.

[Oka07] 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, 2007.

[Ust08] B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Des. Codes Cryptography*, 46(3):329–342, 2008.