Actor Key Compromise: Consequences and Countermeasures

David Basin
Institute of Information Security
ETH Zurich

Cas Cremers
Department of Computer Science
University of Oxford

Marko Horvat
Department of Computer Science
University of Oxford

***Abstract*—Despite Alice's best efforts, her long-term secret keys may be revealed to an adversary. Possible reasons include weakly generated keys, compromised key storage, subpoena, and coercion. However, Alice may still be able afterwards to communicate securely with other parties. Whether this is possible depends on the protocol used. We call the associated property resilience against *Actor Key Compromise* (AKC). We formalise this property in a symbolic model and identify conditions under which it can and cannot be achieved. In case studies that include TLS and SSH, we find that many protocols are not resilient against AKC. We implement a concrete AKC attack on the mutually authenticated TLS protocol.**

Key words: Security protocols, security properties, Key Compromise Impersonation, authentication, secrecy, adversary models, TLS, SSH

I. Introduction

If a service provider reveals its long-term secret key to a government agency, it is clear that the agency can impersonate the service provider to its users [1]. But can the agency also impersonate an arbitrary user to the service provider? Whether this is possible depends on the security protocol in question. In this paper we study the property that formalises this behaviour, which we call resilience against *Actor Key Compromise* (AKC).

To illustrate AKC, consider a setting with a public-key infrastructure: each party X has a long-term key pair for asymmetric encryption or signing, where $\mathrm{pk}(X)$ denotes the public key and $\mathrm{sk}(X)$ denotes the corresponding secret key. We write $\{m\}_k$ to denote the encryption of m with k and $h(m)$ for the hash of m. In this setting, Alice can use certain protocols to establish unilateral security guarantees: any such protocol is called *unilateral*. For example, Alice is guaranteed secrecy of the nonce na and agreement on its value by sending it encrypted to Bob and receiving a hash of it as follows:

1. $A \to B: \quad \{na, A\}_{\mathrm{pk}(B)}$
2. $B \to A: \quad h(na, A, B)$

Here, as in many unilateral protocols, the protocol's security relies only on the secrecy of Bob's long-term secret key.

Most modern protocols offer bilateral guarantees, established through mutual authentication protocols or authenticated key exchange protocols. For example, consider the Needham-Schroeder-Lowe protocol [2]:

1. $A \to B: \quad \{na, A\}_{\mathrm{pk}(B)}$
2. $B \to A: \quad \{na, nb, B\}_{\mathrm{pk}(A)}$
3. $A \to B: \quad \{nb\}_{\mathrm{pk}(B)}$

Such bilateral protocols can be viewed as the combination of two unilateral protocols: if Alice's long-term secret key is compromised, Bob's half of the bilateral guarantees is lost because the adversary can impersonate Alice. But what about Alice's half? Bob's key is not compromised, so Alice should be able to obtain the guarantees she would have when using an appropriate unilateral protocol.

It turns out that not every bilateral protocol has this property. For example, if $\mathrm{sk}(\mathrm{Alice})$ is compromised in the Needham-Schroeder-Lowe protocol, Alice no longer obtains secrecy of the nonces nor achieves agreement on nb, as witnessed by the following attack, in which $\mathcal{A}_{\mathrm{Alice}}$ denotes that the adversary sends or receives a message as Alice:

1. $\mathrm{Alice} \to \mathrm{Bob}: \quad \{na, \mathrm{Alice}\}_{\mathrm{pk(Bob)}}$
2. $\mathrm{Bob} \to \mathcal{A}_{\mathrm{Alice}}: \quad \{na, nb, \mathrm{Bob}\}_{\mathrm{pk(Alice)}}$
3. $\mathcal{A}$ decrypts message using $\mathrm{sk}(\mathrm{Alice})$ and learns na, nb
4. $\mathcal{A}$ chooses nb'
5. $\mathcal{A}_{\mathrm{Bob}} \to \mathrm{Alice}: \quad \{na, nb', \mathrm{Bob}\}_{\mathrm{pk(Alice)}}$
6. $\mathrm{Alice} \to \mathcal{A}_{\mathrm{Bob}}: \quad \{nb'\}_{\mathrm{pk(Bob)}}$

We say that protocols such as Needham-Schroeder-Lowe are vulnerable to AKC attacks: if the long-term secret key of a party (the actor) is compromised, the party can no longer obtain unilateral guarantees when communicating with another party (the peer) even when the peer's key is still secret. From the actor's local perspective, protocols that are vulnerable to AKC attacks offer weaker security guarantees than many unilateral protocols, because the vulnerable protocols only achieve the unilateral guarantees if *both* the long-term keys of the actor and the peer are secret.

This phenomenon has been largely ignored in the protocol literature. A notable exception is the research area of authenticated key exchange protocols, where a limited instance of this problem has been studied. In this area, there are so-called *Key Compromise Impersonation* (KCI) attacks [3,4]: the key of the actor is revealed and used by the adversary to impersonate another party in communication with the actor. Of course, one could also consider such an adversary when ensuring the non-repudiation of online payments, or for secrecy of votes in an e-voting protocol. We conclude that the core issue is neither limited to key exchange protocols nor to authentication: the loss of a party's long-term secret key may impact that party's other security properties in any type of security protocol.

Contributions. We provide the first systematic analysis of the consequences of the compromise of the actor's secret key. First, we introduce and formally define actor key compromise and

the related notions of actor key compromise security, actor key compromise resilience and actor key compromise attack. Our definitions are independent of the choice of protocol, adversary, and property. We show that key compromise impersonation is a specific instance of actor key compromise.

Second, we provide constructive results: we demonstrate how certain actor key compromise attacks can be prevented in general by using asymmetric encryption and signatures.

Third, we provide impossibility results: we prove that a large class of authentication properties cannot be achieved under actor key compromise by protocols that only use symmetric cryptography and hashing.

Finally, we analyse a set of protocols, including TLS and SSH, for their resilience against actor key compromise. We find attacks on several protocols, including AKC attacks on mutually authenticated TLS-RSA as well as the combinations of unilateral TLS-RSA with authorisation protocols such as Apache's mod_auth_basic, OAUTH, and SAML. For mutually authenticated TLS, we provide a concrete implementation of our AKC attack against a fully-patched Apache webserver running TLS v1.2. We provide and verify concrete fixes for the vulnerable protocols.

Organisation. We describe our modelling framework in Section II to formalise actor key compromise in Section III. We show how to achieve actor key compromise security by protocol transformation in Section IV and prove an impossibility result in Section V. We present case studies in Section VI, examine related work in Section VII, and draw conclusions in Section VIII. We provide full proofs in the appendix.

II. Modelling framework

Before specifying protocols, we give the notation we use for functions and sequences. For f a partial function from X to Y, we write $f : X \nrightarrow Y$. If f is also a function, we write $f : X \to Y$. For every partial function f, we denote its domain by $\mathrm{dom}(f)$ and its range by $\mathrm{ran}(f)$. If $S \subseteq \mathrm{dom}(f)$, we write $f|_S$ for the restriction of f to S. We write $f[a \mapsto b]$ to denote f's update, i. e., f' where $f'(a) = b$ and for all $x \in \mathrm{dom}(f) \setminus \{a\}$, $f'(x) = f(x)$.

Let S^* denote the set of all finite sequences of elements from S. We write $\langle s_1, \ldots, s_n \rangle$ to denote the sequence of s_1 to s_n and define $last(\langle s_1, \ldots, s_n \rangle) = s_n$. We write $e \in s$ if there exists an $i \in \mathbb{N}$ such that $e = s_i$. Finally, we write $s.s'$ for the concatenation of the sequences s and s'.

A. Protocol specification

Our framework is based on [5,6], where the main building blocks for protocol specification are roles. A protocol can have any finite number of roles, and is run by agents who execute those roles. Agents may execute each role multiple times, and every role can be executed by any agent. Concrete role instances that occur during protocol execution are called runs. While roles are built out of role events, runs consist of their instantiated counterpart, run events. Role and run events are comprised of role and run terms, respectively.

We assume given the pairwise disjoint, infinite sets Agent, Role, Fresh, Var, Func, and RID of agent names, roles, freshly generated terms (nonces, coin flips, etc.), variables, function names (hash functions, constants, etc.), and run identifiers. We also assume that for all $n \in \mathbb{N}$, there is an infinite number of function names of arity n. We denote by $\mathrm{Const} \subseteq \mathrm{Func}$ the set of all constants, i.e., the function names of arity 0. Additionally, we assume that RID contains two distinguished run identifiers, $Test$ and $rid_{\mathcal{A}}$, which are respectively used to identify the *test run* and the *adversary run*. In the computational setting, the test run is the run where the adversary performs a so-called test query. We use the test run to define the adversary's capabilities to perform different types of long-term key reveal, and advance the adversary run when any such key reveal is executed.

Definition 1 (Terms):

$$\begin{aligned}\mathrm{Term} ::= {} & \mathrm{Role} \mid \mathrm{Agent} \mid \mathrm{Fresh} \mid \mathrm{Fresh}^{\#\mathrm{RID}} \mid \mathrm{Var} \\ & \mid (\mathrm{Term}, \mathrm{Term}) \mid \{\mathrm{Term}\}_{\mathrm{Term}} \mid \mathrm{Func}(\mathrm{Term}^n) \\ & \mid \mathrm{k}(\mathrm{Role}, \mathrm{Role}) \mid \mathrm{pk}(\mathrm{Role}) \mid \mathrm{sk}(\mathrm{Role}) \\ & \mid \mathrm{k}(\mathrm{Agent}, \mathrm{Agent}) \mid \mathrm{pk}(\mathrm{Agent}) \mid \mathrm{sk}(\mathrm{Agent})\end{aligned}$$

The superscript n in $\mathrm{Func}(\mathrm{Term}^n)$ denotes that the arity $n \in \mathbb{N}$ depends on the function name. We omit the brackets if $n = 0$. For all $X, X' \in \mathrm{Agent} \cup \mathrm{Role}$, $\mathrm{k}(X, X')$ denotes the symmetric long-term secret key shared between X and X', $\mathrm{pk}(X)$ denotes X's asymmetric long-term public key, and $\mathrm{sk}(X)$ denotes X's asymmetric long-term secret key. By $\{t_1\}_{t_2}$ we denote either asymmetric encryption, signature, or symmetric encryption, depending on whether t_2 is a public key, secret key, or any other term, respectively. Pairing is not associative, and we write (a, b, c) to denote $((a, b), c)$.

We define substitutions in the usual way except that we consider both Var and Role to be variables. That is, a substitution σ is written as $\sigma = [y_1, \ldots, y_n / x_1, \ldots, x_n]$, where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$, $\mathrm{ran}(\sigma) = \{y_1, \ldots, y_n\}$, $\mathrm{dom}(\sigma) \subseteq \mathrm{Role} \cup \mathrm{Var}$, and $\mathrm{ran}(\sigma) \subseteq \mathrm{Term}$. When applying a substitution as a function to terms, we extend it to an endomorphism on Term in the standard way. We write $\sigma \cup \sigma'$ for the union of two substitutions with disjoint domains. Similarly, we write $\{t/t'\}$ to denote the function that replaces every occurrence of the subterm t' with the term t. We call such functions *replacements*.

We define two subterm relations: the *syntactic* subterm relation, which does not take into account the position of a subterm within a term, and the *accessible* subterm relation, which only identifies potentially retrievable subterms, given knowledge of appropriate keys.

Definition 2 (Syntactic subterm relation): We define the *syntactic subterm relation* as the reflexive, transitive closure of the smallest relation $\sqsubseteq$ such that for all $n \in \mathbb{N}$ and $f \in \mathrm{Func}$ where f is of arity n:

$$\begin{array}{llll} t_j \sqsubseteq (t_1, t_2) & t_j \sqsubseteq \{t_1\}_{t_2} & t_j \sqsubseteq \mathrm{k}(t_1, t_2) & (j \in \{1, 2\}) \\ t \sqsubseteq \mathrm{pk}(t) & t \sqsubseteq \mathrm{sk}(t) & t_i \sqsubseteq f(t_1, \ldots, t_n) & (i \in \{1, \ldots, n\}) \end{array}$$

Definition 3 (Accessible subterm relation): We define the *accessible subterm relation* as the reflexive, transitive closure of the smallest relation $\sqsubseteq_{\mathrm{acc}}$ such that for all $t_1, t_2 \in \mathrm{Term}$, $t_1 \sqsubseteq_{\mathrm{acc}} (t_1, t_2)$, $t_2 \sqsubseteq_{\mathrm{acc}} (t_1, t_2)$, and $t_1 \sqsubseteq_{\mathrm{acc}} \{t_1\}_{t_2}$.

We extend both subterm relations to sets in the following way: for a term t and set S, $t \sqsubseteq S$ ($t \sqsubseteq_{\mathrm{acc}} S$) means that there

exists a term $t' \in S$ such that $t \sqsubseteq t'$ ($t \sqsubseteq_{\text{acc}} t'$). We write $\text{vars}(t)$ for $\{x \in \text{Var} : x \sqsubseteq t\}$.

We assume the existence of an inverse function on terms, such that for all $t \in \text{Term}$, if $t = \text{pk}(t')$ or $t = \text{sk}(t')$ for some $t' \in \text{Term}$, then $t^{-1} = \text{sk}(t')$ or $t^{-1} = \text{pk}(t')$, respectively; otherwise, $t^{-1} = t$.

We now define what terms an adversary or agent can infer. Specifically, by $\vdash$ we denote the smallest relation such that for all $n \in \mathbb{N}$, $f \in \text{Func}$ where f is of arity n, and $S \cup \{t_1, \ldots, t_n\} \subseteq \text{Term}$, $\vdash$ respects the following rules:

$$\frac{}{S \vdash t_1}\ (t_1 \in S)$$

$$\frac{S \vdash t_1 \qquad S \vdash t_2}{S \vdash (t_1, t_2)} \qquad \frac{S \vdash t_1 \qquad S \vdash t_2}{S \vdash \{t_1\}_{t_2}} \qquad \frac{S \vdash t_1 \qquad \ldots \qquad S \vdash t_n}{S \vdash f(t_1, \ldots, t_n)}$$

$$\frac{S \vdash (t_1, t_2)}{S \vdash t_1} \qquad \frac{S \vdash (t_1, t_2)}{S \vdash t_2} \qquad \frac{S \vdash \{t_1\}_{t_2} \qquad S \vdash t_2^{-1}}{S \vdash t_1}$$

We call the rules in the second row *composition rules* and the ones in the third row *decomposition rules*. If a term t can be derived from a set S in finitely many applications of the above rules so that every branch of the derivation is closed, we write $S \vdash t$ and say that S *infers* t. We call every corresponding derivation a $\vdash$*-derivation tree for* t *from* S. If there exists a $\vdash$-derivation tree for t from S of height at most n, we write $S \vdash_n t$.

We define RoleTerm as the set of all terms that have no subterms in $\text{Agent} \cup \{n^{\#rid} : n \in \text{Fresh}, rid \in \text{RID}\}$. We define RunTerm as the set of all terms that have no subterms in $\text{Role} \cup \text{Fresh}$. We call $\text{Role} \cup \text{Agent} \cup \{x, x^{\#rid} : x \in \text{Fresh}, rid \in \text{RID}\} \cup \text{Var} \cup \text{Const}$ the set of all *atomic terms*.

Next we define role and run events. We assume two infinite, disjoint sets are given, both pairwise disjoint from Term, Func and RID: Claim, the set of all claim names, which we will use to specify claims of various security properties, and Label, which will be used to label role events. We additionally assume that $\{\text{alive}, \text{commit}, \text{running}, \text{secret}\} \subseteq \text{Claim}$.

Definition 4 (Role and run events): For all $R \in \text{Role}$ and $a \in \text{Agent}$, we define:

$$\begin{aligned}
\text{RoleEvent}_R ::= {} & \texttt{send}_{\text{Label}}(R, \text{Role}, \text{RoleTerm}) \\
\mid {} & \texttt{recv}_{\text{Label}}(\text{Role}, R, \text{RoleTerm}) \\
\mid {} & \texttt{claim}_{\text{Label}}(R, \text{Claim}[\,, \text{Role}][\,, \text{RoleTerm}]) \\
\text{RunEvent}_a ::= {} & \texttt{send}_{\text{Label}}(a, \text{Agent}, \text{RunTerm}) \\
\mid {} & \texttt{recv}_{\text{Label}}(\text{Agent}, a, \text{RunTerm}) \\
\mid {} & \texttt{claim}_{\text{Label}}(a, \text{Claim}[\,, \text{Agent}][\,, \text{RunTerm}])
\end{aligned}$$

Additionally, we define $\text{RoleEvent} = \bigcup_{R \in \text{Role}} \text{RoleEvent}_R$ and $\text{RunEvent} = \bigcup_{a \in \text{Agent}} \text{RunEvent}_a$.

For example, the event $\texttt{send}_{l_0}(Alice, Bob, \{n^{\#rid}\}_{\text{pk}(Bob)})$ is labelled l_0 and signifies that Alice sends to Bob a nonce $n^{\#rid}$, which is generated in the run rid and encrypted for Bob with his public key.

Two more events can occur during execution: `create` and LKR. They respectively denote the creation of runs of roles and the revealing of long-term keys to the adversary. We denote the set $\text{RunEvent} \cup \{\texttt{create}(R) : R \in \text{Role}\} \cup \{\mathsf{LKR}(a) : a \in \text{Agent}\}$ by TraceEvent, and the set $\text{RoleEvent} \cup \text{TraceEvent}$ of all *events* by Event. For all $\texttt{ev} \in \{\texttt{send}, \texttt{recv}, \texttt{claim}\}$, $l \in \text{Label}$ and $e = \texttt{ev}_l(\cdot)$, we let $\text{evtype}(e) = \texttt{ev}$ and $\text{label}(e) = l$. We call $\texttt{ev}$ the *event type of* e, and l the *label of* e. Additionally, if e is of the form $\texttt{send}_l(\cdot, \cdot, m)$, $\texttt{recv}_l(\cdot, \cdot, m)$, $\texttt{claim}_l(\cdot, \cdot, m)$, or $\texttt{claim}_l(\cdot, \cdot, \cdot, m)$, where $m \in \text{Term}$, we write $\text{cont}(e) = m$ and call m the *contents of* e. We homomorphically extend all replacements and substitutions to events and sequences of terms and events.

We require every sequence of role events that occurs in a protocol specification to satisfy some well-formedness conditions. Before specifying them, for all $S \in \{\text{Role}, \text{Agent}\}$, we define the set $\text{LTK}(x)$ of all long-term secret keys of $x \in S$ as $\{\text{sk}(x)\} \cup \bigcup_{x' \in S} \{\text{k}(x, x'), \text{k}(x', x)\}$. We also define the operator $\lfloor$ that selects the set of terms that the form contents of events of a particular type: for all $e \in \text{Event}$, $s \in \text{Event}^*$, and $\texttt{ev} \in \{\texttt{send}, \texttt{recv}, \texttt{claim}\}$, let $\langle\rangle \lfloor \texttt{ev} = \emptyset$ and

$$(\langle e \rangle . s) \lfloor \texttt{ev} = \begin{cases} \{\text{cont}(e)\} \cup (s \lfloor \texttt{ev}), & \text{if } \text{evtype}(e) = \texttt{ev}, \\ s \lfloor \texttt{ev}, & \text{otherwise.} \end{cases}$$

For all $s \in \text{Event}^*$ and $l \in \text{Label}$ such that l occurs in s, we define $\text{upto}(s, l)$ as the prefix of s, up to and including the first event labelled l.

Definition 5 (Well-formed sequence of role events): Let $R \in \text{Role}$. A sequence $s \in \text{RoleEvent}_R^*$ is *well-formed* if:

- all event labels in s are unique,
- for all $x \in \text{Var}$, if $e \in s$ is the first event in s such that $x \sqsubseteq \text{cont}(e)$, then $\text{evtype}(e) = \texttt{recv}$ and $x \sqsubseteq_{\text{acc}} \text{cont}(e)$ *(every occurring variable must be initialised in an accessible position in a* `recv`*)*,
- for all $t_1, t_2 \in \text{RoleTerm}$ and $e \in s$ such that $\text{evtype}(e) = \texttt{send}$, $\text{k}(t_1, t_2) \not\sqsubseteq_{\text{acc}} \text{cont}(e)$ and $\text{sk}(t_1) \not\sqsubseteq_{\text{acc}} \text{cont}(e)$ *(no long-term secret keys can occur in an accessible position in a* `send`*)*, and
- for all $l \in \text{Label}$, $R' \in \text{Role}$, and $t \in \text{RoleTerm}$ such that $\texttt{send}_l(R, R', t) \in s$, $\text{LTK}(R) \cup \{S, \text{pk}(S) : S \in \text{Role}\} \cup \{n \in \text{Fresh} : n \sqsubseteq (\text{upto}(s, l) \lfloor \texttt{send})\} \cup (\text{upto}(s, l) \lfloor \texttt{recv}) \vdash t$ *(every role must able to construct the contents of each of its* `send` *events)*.

When a message is received during protocol execution, which we define in the next section, some terms may be stored in variables. We therefore define the notion of a *type function* that encodes exactly which terms can be stored in which variable. A `recv` step can be executed only if all variable types in its contents are satisfied.

Definition 6 (Protocol): Let $\Pi : \text{Role} \nrightarrow \text{RoleEvent}^*$ be a partial function and $\text{type}_\Pi : \text{Var} \to \mathcal{P}(\text{RunTerm})$ a function. If for all $R \in \text{dom}(\Pi)$, $\Pi(R) \in \text{RoleEvent}_R^*$ and $\Pi(R)$ is well-formed, we say that (Π, type_Π) is a *protocol*.

We introduce two protocols that we use as running examples in Figure 1 and 2. Both protocols are two-role protocols, represented as message sequence charts. In the first one, R sends to R' its identity and a freshly generated nonce n, which are encrypted with the public key of R'. Upon receiving the encrypted message, R' claims that n is secret, i.e. that the

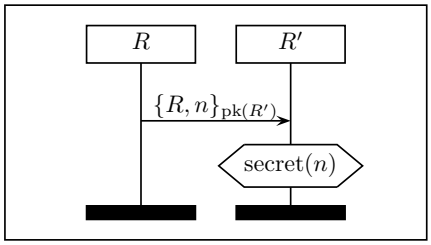

Fig. 1. Example protocol 1.

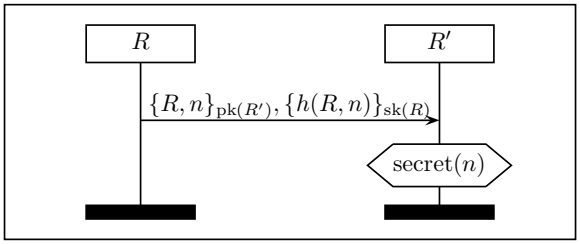

Fig. 2. Example protocol 2.

adversary does not know it. In the second protocol, a signed hash of the payload is additionally transmitted. In both protocols, role R' claims secrecy of n upon successful completion.

We now consider the protocol from Figure 1 and assume that it is a key transport protocol and the nonce n is a fresh session key that, if secret, provides R' with the following guarantee: any messages symetrically encrypted with n that R' later receives are secret and come from R. If the adversary, however, knows $\mathrm{sk}(R')$, he can learn n and send any message he wants to R'.

Example 1: One possible formal specification of the protocol in Figure 1 is as follows: let $x_n \in \mathrm{Var}$, define

$$\Pi(x) = \begin{cases} \langle \mathtt{send}_1(R, R', \{R, n\}_{\mathrm{pk}(R')})\rangle, & \text{if } x = R, \\ \langle \mathtt{recv}_2(R, R', \{R, x_n\}_{\mathrm{pk}(R')}), & \\ \mathtt{claim}_3(R', \mathrm{secret}, x_n)\rangle, & \text{if } x = R', \end{cases}$$

and let type_Π be the function that assigns RunTerm to every variable.

We can extend the domain of any type_Π function to Term by assigning to each term the set of all its possible instantiations: for all $R \in \mathrm{Role}$, $n \in \mathrm{Fresh}$, and $y \in \mathrm{Agent} \cup \mathrm{Fresh}^{\#\mathrm{RID}}$, we define $\mathrm{type}_\Pi(R) = \mathrm{Agent}$, $\mathrm{type}_\Pi(n) = \{n^{\#rid} : rid \in \mathrm{RID}\}$, and $\mathrm{type}_\Pi(y) = \{y\}$. Then we homomorphically extend type_Π to Term.

B. Execution model and security properties

We model protocol execution as a transition system. The set of all states of our system is $\mathrm{State} = \mathrm{Trace} \times \mathcal{P}(\mathrm{RunTerm}) \times (\mathrm{RID} \nrightarrow \mathrm{RunEvent}^*) \times (\mathrm{RID} \nrightarrow (\mathrm{Role} \cup \mathrm{Var}) \nrightarrow \mathrm{RunTerm})$, where $\mathrm{Trace} = (\mathrm{RID} \times \mathrm{TraceEvent})^*$ represents all possible execution histories or *traces*. Every execution state $s = (tr_s, AK_s, th_s, \sigma_s)$ consists of (1) a trace tr_s, (2) the adversary's knowledge AK_s, (3) a partial function th_s mapping the run identifiers of initiated runs to sequences of run events, and (4) the role-to-agent and variable assignments σ_s of all runs. To keep the notation compact, we write the argument of σ_s as a subscript, e.g. $\sigma_{s,rid}$.

To define initial states, for each $rid \in \mathrm{RID}$, we define a replacement $(\cdot)^{\sharp rid}$ to distinguish between local freshly generated terms of each run by assigning unique names to the terms: $(\cdot)^{\sharp rid} = \bigcup_{n \in \mathrm{Fresh}} \{n^{\sharp rid} / n\}$. We define the set of all *test substitutions* $TS(\Pi, \mathrm{type}_\Pi)$ as the set of all substitutions $\sigma : \mathrm{Role} \cup \mathrm{Var} \to \mathrm{RunTerm}$ such that $\mathrm{vars}(\mathrm{ran}(\sigma)) = \emptyset$ and for all $x \in \mathrm{dom}(\sigma)$, $\sigma(x) \in \mathrm{type}_\Pi(x)$.

Definition 7 (Initial states): Let (Π, type_Π) be a protocol. For all $R \in \mathrm{dom}(\Pi)$, the set of *initial states* $IS(\Pi, \mathrm{type}_\Pi, R)$ is defined as

$$\bigcup_{\sigma \in TS(\Pi, \mathrm{type}_\Pi)} \big\{(\langle\rangle, AK_0, Test \mapsto \sigma(\Pi(R)^{\#Test}), Test \mapsto \sigma)\big\},$$

where $AK_0 = \{a, \mathrm{pk}(a) : a \in \mathrm{Agent}\} \cup \{n^{\#rid_{\mathcal{A}}} : n \in \mathrm{Fresh}\}$ is the *initial adversary knowledge*.

Example 2: We provide two initial states for the protocol in Example 1:

$$(\langle\rangle, AK_0, Test \mapsto \langle \mathtt{recv}_2(Alice, Bob, \{Alice, n^{\#rid}\}_{pk(Bob)}),$$
$$\mathtt{claim}_3(Bob, secret, n^{\#rid})\rangle,$$
$$Test \mapsto [Alice, Bob, n^{\#rid}, \ldots / R, R', x_n, \ldots])$$

$$(\langle\rangle, AK_0, Test \mapsto \langle \mathtt{recv}_2(Bob, Joe, \{Bob, n^{\#rid_{\mathcal{A}}}\}_{\mathrm{pk}(Joe)}),$$
$$\mathtt{claim}_3(Joe, secret, n^{\#rid_{\mathcal{A}}})\rangle,$$
$$Test \mapsto [Bob, Joe, n^{\#rid_{\mathcal{A}}}, \ldots / R, R', x_n, \ldots]).$$

We assume that for all $a \in \mathrm{Agent}$, a has generated or securely received all keys in $\mathrm{LTK}(a)$ and public keys of all agents prior to protocol execution. We also assume that agents do not initially have any previously established session keys. We identify adversaries with sets of their *capabilities* [5], i. e., the adversary-compromise rules that they can use. The $\mathsf{LKR}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term keys of the agent executing the test run (also called *the actor*). The rule takes Π and R as parameters. The rule's second premise is required because our model allows agents to communicate with themselves. Since $\mathsf{LKR}_{\mathsf{actor}}$ is the core component of AKC, we discuss it in detail in Section III. The $\mathsf{LKR}_{\mathsf{others}}$ rule, which is parametrised by Π, formalises the standard Dolev-Yao adversary's capability to learn the long-term secret keys of any agent a that is not an intended partner (or *peer*) of the test run.

Definition 8 (Transition relation and reachable states): Let (Π, type_Π) be a protocol, $R \in \mathrm{dom}(\Pi)$ a role and A an adversary. We define a *transition relation* $\to_{\Pi, \mathrm{type}_\Pi, R, A}$ from the execution-model rules in Figure 3 and the rules in A. For states s and s', $s \to_{\Pi, \mathrm{type}_\Pi, R, A} s'$ iff there exists a rule in either A or the execution-model rules with the conclusion $s \to s'$ such that all of the premises hold. We define the set of *reachable states* $\mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$ by $\{s : (\exists s_0 \in IS(\Pi, \mathrm{type}_\Pi, R))(s_0 \to^*_{\Pi, \mathrm{type}_\Pi, R, A} s)\}$.

Example 3: Continuing from Example 2, the following state is reached from the first of the two initial states by any adversary A in a single transition:

$$(\langle(rid, \mathtt{create}(R))\rangle, AK_0, th, \sigma)$$

where

$$\frac{R \in \mathrm{dom}(\Pi) \quad rid \notin (\mathrm{dom}(th) \cup \{rid_{\mathcal{A}}, Test\}) \quad \sigma' : \mathrm{Role} \to \mathrm{Agent}}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid, \mathtt{create}(R)) \rangle, AK, th[rid \mapsto \sigma'(\Pi(R)^{\#rid})], \sigma[rid \mapsto \sigma'])}[\mathtt{create}_{\Pi}]$$

$$\frac{th(rid) = \langle \mathtt{send}_l(a, b, m) \rangle.seq}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid, \mathtt{send}_l(a, b, m)) \rangle, AK \cup \{m\}, th[rid \mapsto seq], \sigma)}[\mathtt{send}]$$

$$\frac{th(rid) = \langle \mathtt{recv}_l(a, b, pt) \rangle.seq \quad \mathrm{dom}(\sigma') = \mathrm{vars}(pt) \quad (\forall x \in \mathrm{dom}(\sigma'))(\sigma'(x) \in \mathrm{type}_{\Pi}(x)) \quad AK \vdash \sigma'(pt)}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid, \mathtt{recv}_l(a, b, \sigma'(pt))) \rangle, AK, th[rid \mapsto \sigma'(seq)], \sigma[rid \mapsto \sigma_{rid} \cup \sigma'])}[\mathtt{recv}_{\mathrm{type}_{\Pi}}]$$

$$\frac{th(rid) = \langle e \rangle.seq \quad \mathrm{evtype}(e) = \mathtt{claim}}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid, e) \rangle, AK, th[rid \mapsto seq], \sigma)}[\mathtt{claim}]$$

Fig. 3. Execution-model rules

$$\frac{a = \sigma_{Test}(R) \qquad a \notin \{\sigma_{Test}(R') : R' \in \mathrm{dom}(\Pi) \setminus \{R\}\}}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle, AK \cup \mathrm{LTK}(a), th, \sigma)}[\mathsf{LKR}_{\mathsf{actor}\,\Pi,R}]$$

$$\frac{a \notin \{\sigma_{Test}(R) : R \in \mathrm{dom}(\Pi)\}}{(tr, AK, th, \sigma) \longrightarrow (tr.\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle, AK \cup \mathrm{LTK}(a), th, \sigma)}[\mathsf{LKR}_{\mathsf{others}\,\Pi}]$$

Fig. 4. Adversary-compromise rules

$$th(x) = \begin{cases} \langle \mathtt{send}_1(Alice, Bob, \{Alice, n^{\#rid}\}_{\mathrm{pk}(Bob)}) \rangle, & \text{if } x = rid, \\ \langle \mathtt{recv}_2(Alice, Bob, \{Alice, n^{\#rid}\}_{\mathrm{pk}(Bob)}), & \\ \mathtt{claim}_3(Bob, \mathrm{secret}, n^{\#rid}) \rangle, & \text{if } x = Test \end{cases}$$

and

$$\sigma(x) = \begin{cases} [Alice, Bob, \ldots / R, R', \ldots], & \text{if } x = rid, \\ [Alice, Bob, n^{\#rid}, \ldots / R, R', x_n, \ldots], & \text{if } x = Test. \end{cases}$$

In this state, Alice is running the newly created run rid with who she believes to be Bob. Alice has not yet executed any protocol steps in run rid.

We model security properties as reachability properties. To keep our definitions independent of the protocol, we use the claim events to declare that a protocol is meant to satisfy a certain property. We will define three different security properties for role R: secrecy, aliveness, and non-injective data agreement.

First we need some auxiliary functions. For all $rid \in \mathrm{RID}$, $R' \in \mathrm{Role}$, and reachable states s (for any instantiation of $\Pi, \mathrm{type}_{\Pi}, R, A$) such that $(rid, \mathtt{create}(R')) \in tr_s$, we define $\mathrm{actor}_s(rid) = \sigma_{s,rid}(R')$ and $\mathrm{role}_s(rid) = R'$. We also let $\mathrm{actor}_s(Test) = \sigma_{s,Test}(R)$ and $\mathrm{role}_s(Test) = R$.

Definition 9 (Security claims, $\models$): Let $l \in \mathrm{Label}$, $R, R' \in \mathrm{Role}$, $t \in \mathrm{RoleTerm}$, and

$$\gamma \in \{\mathtt{claim}_l(R, \mathrm{secret}, t), \\ \mathtt{claim}_l(R, \mathrm{alive}, R'), \\ \mathtt{claim}_l(R, \mathrm{commit}, R', t)\}.$$

We call γ a *security claim for* R. For all $s \in \mathrm{State}$, by $s \models \gamma$ we denote that the following implication is true: if $(Test, \sigma_{s,Test}(\gamma^{\#Test})) \in tr_s$ and:

- $\gamma = \mathtt{claim}_l(R, \mathrm{secret}, t)$, then
$$AK_s \nvdash \sigma_{s,Test}(t^{\#Test})$$
(secrecy of $t \in \mathrm{RoleTerm}$ for R)
- $\gamma = \mathtt{claim}_l(R, \mathrm{alive}, R')$, then
$$(\exists rid \in \mathrm{RID})(\mathrm{actor}_s(rid) = \sigma_{s,Test}(R'))$$
(aliveness of R' for R)
- $\gamma = \mathtt{claim}_l(R, \mathrm{commit}, R', t)$, then
$$(\exists rid \in \mathrm{RID})(\mathrm{role}_s(rid) = R' \wedge \\ (rid, \sigma_{s,Test}(\mathtt{claim}_l(R', \mathrm{running}, R, t^{\#Test}))) \in tr_s)$$
(non-injective agreement for R with R' on $t \in \mathrm{RoleTerm}$)

Let $(\Pi, \mathrm{type}_{\Pi})$ be a protocol, $R \in \mathrm{dom}(\Pi)$ a role, $\gamma \in \Pi(R)$ a claim, and A an adversary. By $(\Pi, \mathrm{type}_{\Pi}) \models_A \gamma$ we denote the fact that for all $s \in \mathrm{RS}(\Pi, \mathrm{type}_{\Pi}, R, A)$, $s \models \gamma$.

Example 4: If we let P be the protocol in Example 1, then $P \not\models \mathrm{claim}_3(R', \mathrm{secret}, x_n)$. If we change P to P' by adding $\mathrm{claim}_4(R, \mathrm{secret}, n)$ in any position to the R role, we have $P' \models \mathrm{claim}_4(R, \mathrm{secret}, n)$. We will prove the generalisation of this fact in Section IV.

Note that while we are usually most interested in one or two roles relevant to the security property we are considering, and only draw those roles in message sequence charts, there is no finite bound on the number of roles in our protocols.

III. Formalising actor key compromise

In this section we formalize the concept of actor key compromise and related notions. These concepts enable reasoning about the security guarantees of agents whose long-term secret keys are compromised. Our notions are defined independently of the choice of protocol, adversary, and property.

Definition 10 (Actor key compromise security): Let $(\Pi, \mathrm{type}_{\Pi})$ be a protocol, $R \in \mathrm{dom}(\Pi)$, A an adversary such that $\mathsf{LKR}_{\mathsf{actor}} \in A$, and $\gamma \in \Pi(R)$ a security claim. We say that γ is *actor key compromise secure in* $(\Pi, \mathrm{type}_{\Pi})$ *with respect to* A if $(\Pi, \mathrm{type}_{\Pi}) \models_A \gamma$.

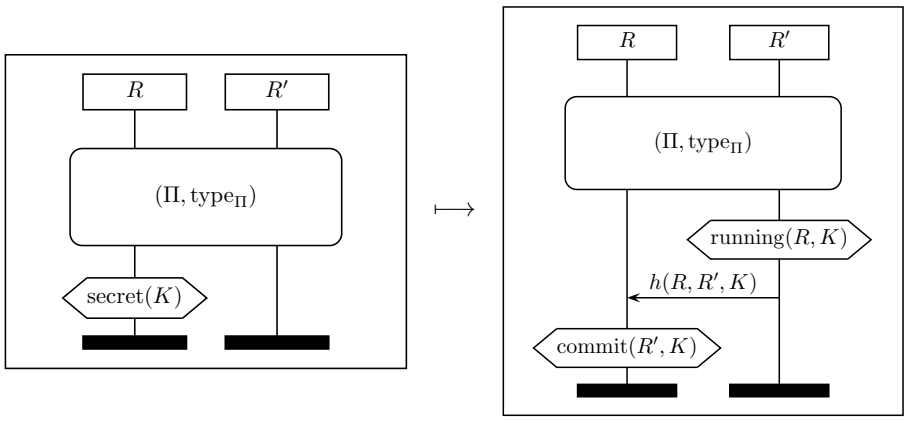

Fig. 5. KCIR as AKCS of secrecy and non-injective agreement

For example, secrecy of the nonce n in the protocols from Figure 1 and 2 is not AKC secure. Consider the trace of a regular execution of either one of the protocols, in which Alice performs R and Bob performs R'. We can extend this trace using the $\mathsf{LKR}_{\mathsf{actor}}$ rule on Bob. This is allowed since Bob is not an element of the set $\{\sigma_{Test}(R)\} = \{\text{Alice}\}$. The adversary can subsequently use $\mathrm{sk}(Bob)$ to decrypt the nonce, violating the claim.

In contrast, the informal notion of KCI attack suggests that the secret keys of the actor are not only available to the adversary, but integral in performing the attack. We formalise this in the following way: if an adversary without $\mathsf{LKR}_{\mathsf{actor}}$ cannot violate a property, but can violate it with $\mathsf{LKR}_{\mathsf{actor}}$, only then do we call any attacks that arise AKC attacks. We term the absence of such attacks AKC resilience.

Definition 11 (AKC resilience and AKC attack): Let (Π, type_Π) be a protocol, $R \in \mathrm{dom}(\Pi)$, A an adversary such that $\mathsf{LKR}_{\mathsf{actor}} \in A$, and $\gamma \in \Pi(R)$ a security claim. We say that γ is *actor key compromise resilient in* (Π, type_Π) *with respect to* A if the following implication holds: if $(\Pi, \mathrm{type}_\Pi) \models_{A \setminus \{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma$, then $(\Pi, \mathrm{type}_\Pi) \models_A \gamma$. Otherwise, every $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$ where $s \not\models \gamma$ is an *actor key compromise attack by* A *on* γ *in* (Π, type_Π).

We say that a protocol is *AKC resilient* or *AKC secure* with respect to an adversary if all security claims in it are. If there is an AKC attack by an adversary on a security claim, we say the protocol is *vulnerable to AKC attacks*.

We now revisit the examples. The KCI attack on the protocol in Figure 1, which is described just before Example 1, can be formalised as an attack on secrecy by $\{\mathsf{LKR}_{\mathsf{actor}}\}$. It does not however represent an AKC attack on secrecy by $\{\mathsf{LKR}_{\mathsf{actor}}\}$ because there already exist attacks on secrecy by the empty adversary, i.e. one with no capabilities beyond the execution-model rules. The adversary can generate and encrypt the nonce himself. Hence, the protocol is trivially AKC resilient with respect to $\{\mathsf{LKR}_{\mathsf{actor}}\}$. We can see that, in general, AKC resilience does not imply AKC security, which is the absence of *all* attacks on secrecy by an adversary who has the $\mathsf{LKR}_{\mathsf{actor}}$ capability.

The protocol from Figure 2, however, is not AKC resilient with respect to $\{\mathsf{LKR}_{\mathsf{actor}}\}$. The empty adversary cannot generate the nonce himself, because he cannot forge the signature. In fact, the nonce is secret with respect to the empty adversary. Therefore, decrypting a sent, encrypted nonce as in the KCI attack described just before Example 1 gives rise to an AKC attack by $\{\mathsf{LKR}_{\mathsf{actor}}\}$ on the secrecy of the nonce.

When AKC resilience is non-trivially satisfied, it coincides with AKC security. We can see one such example if we replace the asymmetric key in the first protocol in Figure 1 with a symmetric one. This is one of the reasons we will mostly be focusing on AKC security: after all, we want to use protocols with no vulnerabilities, regardless if they are AKC-related. Another reason is the fact that we can represent KCIR as an instance of AKCS.

KCIR as an instance of AKCS. KCIR key establishment protocols provide an important security guarantee to their users: even if their own long-term keys are compromised, the users should still be able to authenticate messages encrypted with previously established session keys. Although the notion of KCI *attack* has remained informal and subjective, KCI *resilience* has been incorporated into different formal models expressing the same underlying idea: if an adversary capable of getting the actor's keys cannot perform *any* attack on session key secrecy in a key establishment protocol, then he cannot perform a KCI attack.

Concluding subsequent message authentication just from session key secrecy during a key establishment handshake, however, might fail if the handshake protocol is implicitly authenticated. Even if the adversary does not know the established key, perhaps the test run does not agree with its peers on the value of the key. If so, then the handshake's purpose of establishing a secure communication channel has been defeated.

In the following proposition, we prove that session key secrecy after an implicitly authenticated key establishment handshake coincides with key agreement when a key confirmation step where the key is inaccessible is appended to the handshake. This shows that the above definition of KCIR, which we can reformulate in our terms as AKCS of session key secrecy, matches the requirement of secrecy and authenticity of session keys under actor key compromise.

One of the lemmas we use in the proof depends on terms t that have no proper accessible subterms. Provided that a set

S is given and some conditions are met, t is unnecessary for inferring from S terms that do not contain t as a subterm. The other lemma ensures that any term inferred from S in as few steps as possible that is not accessible in S, must be composed in the last inference step.

Lemma 5 (inference-irrelevant sets): Let $S \cup T \subseteq \mathrm{Term}$ where T is finite and for all $t \in T$, $t \not\sqsubseteq S \setminus T$, $t^{-1} = t$ and t has no proper accessible subterms. Let $t' \in \mathrm{Term}$ such that for all $t \in T$, $t \not\sqsubseteq t'$. Then $S \vdash t'$ if and only if $S \setminus T \vdash t'$.

Lemma 6 (composition lemma): Let $S \cup \{t\} \subseteq \mathrm{Term}$ and $S \vdash t$. If $t \not\sqsubseteq_{\mathrm{acc}} S$, then every minimal-height derivation of t from S ends in a composition rule.

Finally, we define a generic transformation that adds a single key confirmation step where the key is inaccessible, and prove that the adversary can violate agreement on a session key in the transformed protocol if and only if he knows the key in the initial one. The transformation adds a single message flow from R' to R, consisting of a hash $h(R, R', K)$ that contains the established session key K. We will instantiate all the parameters just before using the transformation.

$$KC(\Pi)(x) = \begin{cases} \Pi(x).\langle \mathtt{claim}_{l_1}(R', \mathrm{running}, R, K'), \\ \mathtt{send}_{l_2}(R', R, h(R, R', K'))\rangle, & \text{if } x = R', \\ \Pi(x).\langle \mathtt{recv}_{l_2}(R', R, h(R, R', K)), \\ \mathtt{claim}_{l_1}(R, \mathrm{commit}, R', K)\rangle, & \text{if } x = R, \\ \Pi(x), & \text{otherwise.} \end{cases}$$

The term K' corresponds to the view from R' on the key K; usually, one of them is a variable and the other a fresh value, but some protocols use compound terms. We want the transformation to work on any key establishment protocol, so we choose a symbol h that does not occur in the protocol specification. This prevents messages from the confirmation step from being accepted in the receive events of the handshake.

Proposition 7 (KCIR as an instance of AKCS): Let (Π, type_Π) be a protocol, $R, R' \in \mathrm{dom}(\Pi)$, $K \in \mathrm{RoleTerm}$ and $l_0 \in \mathrm{Label}$ such that $R \neq R'$ and $last(\Pi(R)) = \mathtt{claim}_{l_0}(R, \mathrm{secret}, K)$. Let A be an adversary. Let $l_1, l_2 \in \mathrm{Label}$ and $h \in \mathrm{Func}$ all be unused in Π, where $l_1 \neq l_2$ and h does not occur in the set $\mathrm{ran}(\mathrm{type}_\Pi) \cup \mathrm{ran}(\mathrm{type}_{KC(\Pi)}(K'))$. If $\mathrm{type}_{KC(\Pi)} = \mathrm{type}_\Pi$ and $(KC(\Pi), \mathrm{type}_{KC(\Pi)})$ is a protocol, then

$$(\Pi, \mathrm{type}_\Pi) \models_A \mathtt{claim}_{l_0}(R, \mathrm{secret}, K) \iff (KC(\Pi), \mathrm{type}_{KC(\Pi)}) \models_A \mathtt{claim}_{l_1}(R, \mathrm{commit}, R', K).$$

Proof (sketch). $\boxed{\Rightarrow}$ We prove the contrapositive. Let s be a reachable state where a commit claim with label l_1 was executed, but no run of role R' executed the corresponding running claim. We inductively construct a state s' where $AK_{s'} \vdash \sigma_{s',Test}(K^{\#Test})$, by following a sequence obtained by deleting all events labelled l_1 and l_2 from tr_s. The auxiliary statements used in the induction exploit that the sequence does not contain h, and Lemma 5.

To prove $AK_{s'} \vdash \sigma_{s',Test}(K^{\#Test})$, let $\tau = \sigma_{s,Test}(h(R, R', K)^{\#Test})$. From the executed running claim and the typing, we prove that τ is not a syntactic subterm of any $\mathtt{send}$ in tr_s, so $\tau \not\sqsubseteq AK_s$ holds. From $AK_s \vdash \tau$ and Lemma 6 we have that the adversary constructed τ, so he knows the instance of K in s. Since $AK_s \setminus AK_{s'}$ only contains finitely many hashes, Lemma 5 concludes the proof.

$\boxed{\Leftarrow}$ We prove the contrapositive. If the adversary knows the instance of K in a state s, he can compose the needed hash so that $Test$ receives it. Hence, $Test$ executes a commit claim without a corresponding running claim. ∎

IV. Achieving AKCS by transformation

In this section, we show how to achieve AKCS by exploiting unilateral protocols, whose security only depends on the long-term secret key of the peer.

A. Achieving AKCS of secrecy

We first present a transformation that ensures AKCS of secrecy. The transformation, shown in Figure 6, adds a single message flow containing an asymmetrically encrypted message. The message stays secret due to encryption and tagging, which prevent the message from being rerouted to the old part of the protocol.

Definition 12 (tagging function τ_c): Let $c \in \mathrm{Const}$. We define $\tau_c : \mathrm{Term} \to \mathrm{Term}$ for all $t, t_1, \ldots t_n \in \mathrm{Term}$ by

$$\tau_c(t) = \begin{cases} t, & \text{if } t \text{ atomic or long-term key,} \\ (\tau_c(t_1), \tau_c(t_2)), & \text{if } t = (t_1, t_2), \\ \{\tau_c(t_1), c\}_{\tau_c(t_2)}, & \text{if } t = \{t_1\}_{t_2}, \\ f(\tau_c(t_1), \ldots, \tau_c(t_n), c), & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

We usually restrict the domain of τ_c to some set S of terms if tagging more than the terms in S is unnecessary.

A technical lemma tells us that if some term t that the adversary cannot construct only occurs in his knowledge within a term t', then every term the adversary can infer only contains t within t'. The proof is given in Appendix B.

Lemma 8: Let $S \cup \{t, t'\} \subseteq \mathrm{Term}$ and $\bot \in \mathrm{Const}$ such that $t \neq t'$, $\bot$ does not occur in $S \cup \{t, t'\}$, and $t \not\sqsubseteq_{\mathrm{acc}} \{\bot \,/\, t'\}(S)$. Then for all t'' such that $S \vdash t''$, $t \not\sqsubseteq_{\mathrm{acc}} \{\bot \,/\, t'\}(t'')$.

We also note here that the adversary cannot infer the key of the peer and give the full proof in the appendix. The essential ingredient in the proof is that our adversaries can never reveal the peers' keys. The lemma must be restricted if more powerful adversaries such as the ones in [5] are allowed, e.g. those that can reveal peers' keys after the test run ends.

Lemma 9 (obtaining asymmetric secret keys of peers): Let (Π, type_Π) be a protocol, $R \in \mathrm{dom}(\Pi)$ a role, A an adversary, $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$ a state, and $a \in \{\sigma_{s,Test}(R') : R' \in \mathrm{dom}(\Pi) \setminus \{R\}\}$ an agent. Then $AK_s \nvdash \mathrm{sk}(a)$ holds. ∎

For the transformation in Figure 6, we use the following function:

$$TS(\Pi)(x) = \begin{cases} \tau_{c_1}|_S(\Pi(R)). \\ \langle \mathtt{send}_l(R, R', \{m, c_2\}_{\mathrm{pk}(R')}), \\ \mathtt{claim}_{l'}(R, \mathrm{secret}, m)\rangle, & \text{if } x = R, \\ \tau_{c_1}|_S(\Pi(x)), & \text{otherwise.} \end{cases}$$

The transformation TS will only be useful if the protocol designer ensures that R can indeed send the added message.

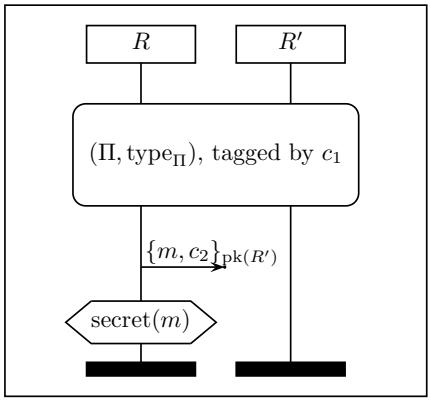

Fig. 6. Transforming Π for secrecy of m

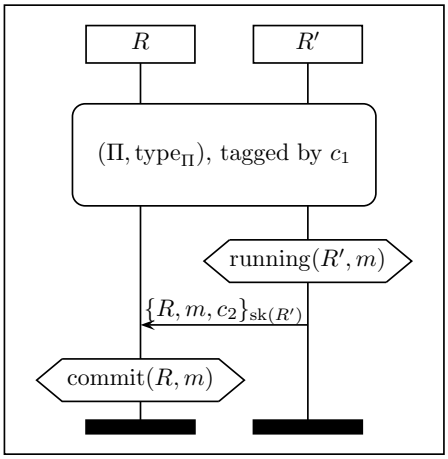

Fig. 7. Transforming Π for non-injective agreement on m

In other words, $(TS(\Pi), \mathrm{type}_{TS(\Pi)})$ must be a protocol, i.e. $TS(\Pi)(R) \in \mathrm{RoleEvent}^*_R$ must be a well-formed sequence of role events.

Proposition 10 (secrecy by asymmetric encryption): Let (Π, type_Π) be a protocol, $R, R' \in \mathrm{dom}(\Pi)$ where $R \neq R'$, and A an adversary. Let $c_1, c_2 \in \mathrm{Const}$, $l, l' \in \mathrm{Label}$ and $n \in \mathrm{Fresh}$ all be unused in Π such that $c_1 \neq c_2$ and $l \neq l'$. Let $m, m' \in \mathrm{RoleTerm}$ such that $n \sqsubseteq_{\mathrm{acc}} m$. Let $S = \{\{t\}_{t'} : \mathrm{type}_\Pi(\{t\}_{t'}) \cap \mathrm{type}_{TS(\Pi)}(\{m, c_2\}_{\mathrm{pk}(R')}) \neq \emptyset\}$. If $(TS(\Pi), \mathrm{type}_{TS(\Pi)})$ is a protocol such that $\mathrm{type}_{TS(\Pi)} = \mathrm{type}_\Pi$, then $(TS(\Pi), \mathrm{type}_{TS(\Pi)}) \models_A \mathtt{claim}_{l'}(R, \mathrm{secret}, m)$.

Proof (sketch). If the secrecy claim is executed in state s by the test run, we need to prove that AK_s cannot infer $\sigma_{s,Test}(m^{\#Test})$. We can inductively prove, over the prefix length of tr_s, that $n^{\#Test}$ is only ever sent as a subterm of $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\mathrm{pk}(R')})$. There are two steps to the proof: Lemma 8 tells us that $n^{\#Test}$ was only received as a subterm of $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\mathrm{pk}(R')})$, and the tagging ensures that it was not decrypted by a run and sent out. Moreover, due to Lemma 9, $AK_s \nvdash n^{\#Test}$. But then Lemma 8 gives us $AK_s \nvdash \sigma_{s,Test}(m^{\#Test})$. ■

B. Achieving AKCS of agreement

We now define a function TA that transforms a protocol into one that achieves non-injective agreement, as depicted in Figure 7. A message signed by a peer convinces the test run that at least one of the peer's runs agrees with it on the message:

$$TA(\Pi)(x) = \begin{cases} \tau_{c_1}|_S(\Pi(x)).\langle \mathtt{claim}_l(R', \mathrm{running}, R, m'), \\ \mathtt{send}_{l'}(R', R, \{R, m', c_2\}_{\mathrm{sk}(R')})\rangle, & \text{if } x = R', \\ \tau_{c_1}|_S(\Pi(x)).\langle \mathtt{recv}_{l'}(R', R, \{R, m, c_2\}_{\mathrm{sk}(R')}), \\ \mathtt{claim}_l(R, \mathrm{commit}, R', m)\rangle, & \text{if } x = R, \\ \tau_{c_1}|_S(\Pi(x)), & \text{otherwise.} \end{cases}$$

As before, m' is an arbitrary term meant to unify with m at R', and the transformation TA assumes that R' can indeed send the required signature. Therefore, we require that $(TA(\Pi), \mathrm{type}_{TA(\Pi)})$ is a protocol, i.e. that $TA(\Pi)(R') \in \mathrm{RoleEvent}^*_{R'}$ is a well-formed sequence of role events.

The following proposition states that agreement on some data can be achieved in a single added message flow using signatures. The proof uses the fact that the adversary cannot forge the signature in $(TA(\Pi), \mathrm{type}_{TA(\Pi)})$, because by Lemma 9 he cannot get the required key. Through the signature, the peer confirms to the actor that he agrees on the identities and the data.

Proposition 11 (agreement by signing): Let (Π, type_Π) be a protocol and $R, R' \in \mathrm{dom}(\Pi)$ such that $R \neq R'$. Let $l, l' \in \mathrm{Label}$ and $c_1, c_2 \in \mathrm{Const}$ all be different and unused in Π, $m, m' \in \mathrm{RoleTerm}$, $S = \{\{t'\}_{t''} : \mathrm{type}_\Pi(\{t'\}_{t''}) \cap \mathrm{type}_{TA(\Pi)}(\{R, m, c_2\}_{\mathrm{sk}(R')}) \neq \emptyset\}$, and A an adversary. If it is the case that $(TA(\Pi), \mathrm{type}_{TA(\Pi)})$ is a protocol and $\mathrm{type}_{TA(\Pi)} = \mathrm{type}_\Pi$, then $(TA(\Pi), \mathrm{type}_{TA(\Pi)}) \models_A \mathtt{claim}_l(R, \mathrm{commit}, R', m)$.

Proof (sketch). Let s be a state where a commit claim was executed by $Test$. We prove that there was a corresponding running claim. Before the commit claim occurred, in some state s', $Test$ executed a `recv` containing the instance t of the signature in $TA(\Pi)(R)$ that appears in the claim. By Lemma 9, we have $AK_{s'} \nvdash \mathrm{sk}(\sigma_{s,Test}(R'))$, so no derivation of t from $AK_{s'}$ can end in a composition step. Lemma 6 then implies $t \sqsubseteq_{\mathrm{acc}} AK_{s'}$. If (rid, e) is the first send of t in an accessible position, because of the tagging we have $\mathrm{label}(e) = l'$. The running claim we needed was then executed prior to e by rid. ■

V. An impossibility result

It is conjectured in [7] that KCIR requires the use of asymmetric cryptography. We give a partial confirmation of this conjecture: under weak assumptions on the protocol specification and the adversary model, the use of only symmetric cryptography and hashing cannot ensure AKCR for a large class of security properties. That class includes, for example, all authentication properties in Lowe's hierarchy [8].

Proposition 12 (impossibility of aliveness): Let (Π, type_Π) be a protocol, $R, R' \in \mathrm{dom}(\Pi)$ and $l \in \mathrm{Label}$ such that $R \neq R'$ and $\mathtt{claim}_l(R, \mathrm{alive}, R') \in \Pi(R)$. If for all $S, T \in \mathrm{Role}$, $t \in \mathrm{RoleTerm}$, $x \in \mathrm{Var}$, $n \in \mathrm{Fresh}$ and $rid \in \mathrm{RID}$,

- $\mathrm{pk}(t) \not\sqsubseteq \Pi(R)$ and $\mathrm{sk}(t) \not\sqsubseteq \Pi(R)$,

- if $\mathrm{k}(S,T) \sqsubseteq \Pi(R)$, then $S = R$ or $T = R$,
- there exists $n_x \in \mathrm{Fresh}$ such that $n_x^{\#rid_{\mathcal{A}}} \in \mathrm{type}(x)$, and
- if $n \sqsubseteq \Pi(R)$, n in $\Pi(R)$ appears first in a `send`, in accessible positions only,

then $(\Pi, \mathrm{type}_\Pi) \not\models_{\{\mathsf{LKR}_{\mathsf{actor}}\}} \mathtt{claim}_l(R, \mathrm{alive}, R')$.

Proof (sketch). Let two different agents, a and b, respectively execute R and R'. Instantiate all other roles in $\Pi(R)$ arbitrarily, and variables by nonces local to $\mathrm{rid}_{\mathcal{A}}$. When $\mathsf{LKR}(a)$ is prepended to the trace, we get a trace leading to a reachable state s. The proof by induction on the prefix length of tr_s establishes two statements: (1) all accessible atomic subterms of `send` events can be inferred after their corresponding `send`, and (2) all `recv` steps are enabled, i.e. the adversary knows their contents just before they occur. By construction, b is not alive in s. ∎

Definition 13 (stronger security claim): Let γ and γ' be security claims. We say that γ is *stronger than* γ' if for all protocols (Π, type_Π), roles $R \in \mathrm{dom}(\Pi)$ such that $\gamma, \gamma' \in \Pi(R)$, adversaries A, and reachable states $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$, whenever $s \models \gamma$, then $s \models \gamma'$.

Corollary 13 (impossibility of authentication): Under the assumptions of Proposition 12, for all security claims γ such that γ is stronger than $\mathrm{claim}(R, \mathrm{alive}, R')$ and $(\Pi, \mathrm{type}_\Pi) \models_\emptyset \gamma$, the claim γ in (Π, type_Π) is not AKC resilient with respect to the $\{\mathsf{LKR}_{\mathsf{actor}}\}$ adversary.

Proof: The statement follows directly from the definitions of stronger security claim and AKC resilience. ∎

VI. Case studies

In this section, we use the Scyther tool [9] to analyse the NSL protocol, the CCITT X.509 family of protocols, the SSH Transport Layer protocol, and the TLS protocol. Our findings include an AKC attack on mutually authenticated TLS-RSA, for which we provide a concrete implementation against an Apache web server running TLS v1.2. All of the protocol specifications needed to reproduce the tests in this section, as well as the scripts to execute the concrete attack, are available at [10].

There are several ways to fix the vulnerable protocols. While we could simply use the generic Proposition 10 and Proposition 11, they are best suited for the incremental design of AKC secure protocols. Their use may not be the most practical way to prevent AKC attacks in widely deployed, special-purpose protocols. For example, adding new message flows to one-pass key transport protocols may not be an option for the particular application. In contrast, Proposition 12 tells us that we will not be able to achieve authentication under actor key compromise by employing only hashing and symmetric keys. However, adding one or both to a protocol that utilises other mechanisms may suffice. We also note that if the vulnerable protocol is just one mode of a protocol suite, there may be another mode that is already secure against AKC. With this in mind, we propose practical fixes for each of the vulnerable protocols and verify their correctness.

A. Needham-Schroeder-Lowe

In the presence of a Dolev-Yao adversary, the Needham-Schroeder-Lowe (NSL) protocol [2] achieves mutual authentication and secrecy of both nonces. However, as explained in the introduction, it is vulnerable to AKC attacks on non-injective agreement on the nonces for the initiator, and secrecy of both nonces for both roles.

We can fix the AKC vulnerabilities by hashing the nonces in the response messages, which prevents leakage of the actor's nonces after learning the actor's key. However, this is insufficient to achieve agreement on both nonces, because the claim of an actor in the A role could be violated by the adversary replacing nb by nb'. We remedy this problem by linking the nonces in the hash of the second message, resulting in the following protocol, which we call NSL-AKC:

$$\begin{array}{lll} 1. & A \to B: & \{na, A\}_{\mathrm{pk}(B)} \\ 2. & B \to A: & \{h(na, nb), nb, B\}_{\mathrm{pk}(A)} \\ 3. & A \to B: & \{h(nb)\}_{\mathrm{pk}(B)} \end{array}$$

This protocol is not vulnerable to the attack from the introduction because the adversary does not learn na from decrypting the second message. Therefore, he cannot produce a hash of $h(na, nb')$, and agreement on both nonces is achieved even under AKC. In NSL-AKC, AKC leads to learning the peer's nonce nb. However, the combination of both nonces under a different hash, e.g. $h'(na, nb)$, can still act as a shared secret. We used Scyther to verify that NSL-AKC achieves *synchronisation* (a strong authentication property that implies agreement, cf. [6]) and secrecy of $h'(na, nb)$, with respect to $\{\mathsf{LKR}_{\mathsf{actor}}\}$ and for both roles.

B. CCITT X.509 family

We consider a family of protocols from the CCITT X.509 standard as modelled in the SPORE library [11]. The protocols are meant to enable secure and (mutually) authenticated access to a certificate directory. There are AKC attacks on secrecy of yb for the A role (likewise, ya for the B role) in the BAN modified version of the CCITT X.509 three-message protocol:

$$\begin{array}{lll} 1. & A \to B: & \{na, B, xa, \{ya\}_{\mathrm{pk}(B)}\}_{\mathrm{sk}(A)} \\ 2. & B \to A: & \{nb, A, na, xb, \{yb\}_{\mathrm{pk}(A)}\}_{\mathrm{sk}(B)} \\ 3. & A \to B: & \{B, nb\}_{\mathrm{sk}(A)} \end{array}$$

To obtain secrecy of ya for B under AKC, we need to encrypt it with something other than $\mathrm{pk}(B)$. We normally have two sources of secrecy to choose from (long-term secret keys and freshly generated values), but under AKC we cannot depend on the actor's long-term secret keys. Therefore, we postpone the transmission of ya to the third message and encrypt it with yb, which is generated in the second message. The encryption with $\mathrm{pk}(A)$ in message 2 makes yb secret for B, analogous to the transformation in Proposition 10.

We make xa secret for A under AKC by encrypting it with $\mathrm{pk}(B)$. We also encrypt yb by xa so that it is secret for A. We put xb inside of the encryption with $\mathrm{pk}(A)$ to achieve agreement on xb for B, which then leads to synchronisation [6] under AKC for both roles. The repaired protocol is successfully verified by Scyther:

$$\begin{array}{lll} 1. & A \to B: & \{na, B, \{xa\}_{\mathrm{pk}(B)}\}_{\mathrm{sk}(A)} \\ 2. & B \to A: & \{nb, A, na, \{xb, \{yb\}_{xa}\}_{\mathrm{pk}(A)}\}_{\mathrm{sk}(B)} \\ 3. & A \to B: & \{B, nb, \{ya\}_{yb}\}_{\mathrm{sk}(A)} \end{array}$$

TABLE I. AUTOMATIC ANALYSIS RESULTS

	Protocol	AKC attack
1.	Needham-Schroeder-Lowe	Yes
2.	CCITT X.509 three-message BAN	Yes
3.	CCITT X.509 one-message	Yes
4.	SSH	No
5.	Mutual TLS-RSA	Yes
6.	Mutual TLS-DHE	No
7.	Unilateral TLS-RSA with mod_auth_basic	Yes
8.	NSL-AKC	No
9.	CCITT X.509 three-message BAN fixed	No
10.	CCITT X.509 one-message fixed	No

The CCITT X.509 one-message protocol is also vulnerable to an AKC attack. Even though it is unilateral, its properties depend on the keys of both parties.

$$1. \quad A \to B: \quad \{ta, na, B, xa, \{ya\}_{\mathrm{pk}(B)}\}_{\mathrm{sk}(A)}$$

There is an AKC attack on the secrecy of ya for the responder, similar to the three-message protocol. The protocol can be made AKC resilient by prepending a message $\{nb\}_{\mathrm{pk}(A)}$ from B to A, and replacing ya by $\{ya\}_{nb}$ in message 1.

C. SSH

The Secure Shell (SSH) protocol [12] is used to establish a secure channel between two endpoints, mainly for remote login and command execution purposes. The mutually authenticated public key version of the protocol is essentially a signed Diffie-Hellman key exchange. We used Scyther to verify AKC security of session key secrecy and synchronisation in the SSH2 Transport Layer protocol with respect to $\{\mathsf{LKR}_{\mathsf{actor}}, \mathsf{LKR}_{\mathsf{others}}\}$.

D. Mutually authenticated TLS

The TLS protocol [13] is the most widely deployed protocol for secure communications on the Internet. It can be used to unilaterally authenticate a server to a client and also supports mutually authenticated modes. We first analyse the mutually authenticated mode before returning to the unilateral authentication modes in the next section.

The mutually authenticated modes of TLS are typically used in, e.g., VPN access and specialised banking applications (for examples of the use of the mutually authenticated modes in banking, see [14]). The most commonly deployed mode of TLS is the RSA mode. Abstractly (omitting, e.g., the explicit certificate exchange), TLS-RSA proceeds as follows:

1. Client C and server S exchange nonces nc, ns and parameters; C picks a random pre-master secret PMS.
2. $C \to S: \quad \{PMS\}_{\mathrm{pk}(S)}$
3. $C \to S: \quad \{h(\texttt{msgs\_so\_far})\}_{\mathrm{sk}(C)}$
4. Both C and S compute $CLIENTMK, SERVERMK, CLIENTK, SERVERK$, and F, using only PMS and public information.
5. $C \to S: \quad CFIN = \{F\}_{CLIENTK}$
6. S computes F' from the keys derived from PMS and public information.
7. $S \to C: \quad SFIN = \{F'\}_{SERVERK}$

Afterwards, the four computed session keys are used to encrypt and authenticate the subsequent communications that contain application data.

We observe that the only secret information involved in the computation of all session keys is the pre-master secret PMS. Therefore, the secrecy of the session keys critically depends on the secrecy of PMS. In turn, the secrecy of PMS is based only on the secrecy of the server's long-term secret key $\mathrm{sk}(S)$ (see message labelled 2).

Using Scyther, we find a server-side AKC attack by $\{\mathsf{LKR}_{\mathsf{actor}}\}$ on session key secrecy. In the attack, the adversary essentially eavesdrops on a regular handshake. Then, by using $\mathrm{sk}(S)$, he decrypts $\{PMS\}_{\mathrm{pk}(S)}$ to get PMS, enabling him to compute the session keys. Hence, he can intercept any subsequently transmitted messages, or inject his own, thereby rendering the established communication channel completely insecure.

We implemented this attack against an Apache web server running TLS v1.2 [13], using a man-in-the-middle attack script written in Python. The script connects to an OpenSSL integrated client program (s_client) on one end, and the Apache web server on the other. We start the AKC attack by providing our script with the long-term secret key of the web server. Next, the attack script eavesdrops on a regular handshake between the server and the client, and uses the messages and the long-term key to compute the session keys. The script can then decrypt all sent application data and is able to insert or modify all received application data, both to and from the web server. While the attack does not depend on the concrete hash algorithms, ciphers, and their modes of operation, we used SHA–256 and AES–256 in CBC mode in our experiments. The required files and instructions on running the attack can be downloaded from [10].

The simplest way to prevent the AKC attack is to switch to the mutually authenticated DHE mode of TLS. This mode is not vulnerable to AKC attacks, because it uses temporary Diffie-Hellman public keys of the form g^x, where x is a freshly generated value. The client's temporary key g^x is combined with the server's temporary secret key y, and vice versa, to obtain g^{xy}. The adversary learns both temporary public keys g^x and g^y, but he does not learn x or y, and therefore cannot construct the session key. In this case, we use the Tamarin prover [15] (because of its more precise modelling of Diffie-Hellman exponentiation) to successfully verify that session key secrecy is AKC secure in the DHE mode of our TLS model.

Until now, the reason for using TLS-DHE (instead of TLS-RSA) has been that it offers *perfect forward secrecy*: even if all long-term keys are compromised at some point, previously sent application data is still secure. Our analysis reveals that there is an additional advantage to TLS-DHE over TLS-RSA: the AKC resilience implies that a server running TLS-DHE can still securely communicate with clients even if the server's key is compromised. Our attack proves that this is not the case for the RSA mode.

E. Unilateral TLS combined with authorization protocols

The most common use of TLS involves the unilateral modes, in which only the server has a certificate. In these

modes, the client authenticates the server, but the server does not authenticate the client. The message flow is similar to mutually authenticated TLS, except that the client does not send a certificate and does not send the so-called *client_verify* message (no. 3 in the earlier TLS description). Because the only security requirements stem from the client, who uses no secret key or has no secret key to reveal, the unilateral modes are resilient against AKC attacks.

However, many applications such as e-banking and online e-mail services require mutual authentication even if the client has no certificate. For such applications, mutual authentication is usually achieved by combining (unilateral) TLS with an authorisation protocol in the following way. First, the client establishes a unilateral TLS session with the server, authenticating the server on the basis of the server's certificate. Then, the server performs an authorisation protocol inside the TLS connection. Typical examples of such protocols are Apache's password-based mod_auth_basic [16], or single sign-on protocols such as OAUTH [17] or SAML [18]. Abstractly, these examples all take the following approach, where the last three communications are encrypted using the previously established TLS session keys:

1. $C \longleftrightarrow S$: C authenticates S by means of unilateral TLS and they establish $CLIENTK$ and $SERVERK$.
2. $C \underset{\text{TLS}}{\longleftrightarrow} S$: S authenticates C using an authorisation protocol.
3. $C \underset{\text{TLS}}{\longrightarrow} S$: Communicate or request resource.
4. $S \underset{\text{TLS}}{\longrightarrow} C$: Communicate or provide resource.

We modelled the above setup for unilateral TLS-RSA followed by the default Apache password authentication, mod_auth_basic. Scyther finds an AKC attack on the server that, upon inspection, is straightforward: the adversary eavesdrops on the TLS handshake and the following authentication, which correspond to step 1 and 2 above. As in the mutually authenticated case, he can decrypt PMS and compute the session keys. He can then arbitrarily eavesdrop, modify, or inject messages in steps 3 and 4, regardless of the particular authorisation protocol. Thus, the mutual authentication protocols obtained by combining unilateral TLS-RSA with either mod_auth_basic, OAUTH, or SAML, are all vulnerable to AKC attacks on the server.

Because Apache's mod_auth_basic relies on a secret that is known to both the server and the client, compromising the server would lead to compromise of the secret, which would always enable an AKC attack against the above setup. The only solution, according to our impossibility result, is to introduce further asymmetric cryptography, for example by switching to mutually authenticated TLS-DHE, in order to ensure that the secrecy of the session keys does not only rely on the server's secret key.

We can also consider a weaker threat model, in which the adversary learns the long-term secret key of the server through cryptanalysis but does not have access to the server's password store. For this threat model, similar to mutually authenticated TLS, we can select the Diffie-Hellman mode of unilateral TLS to prevent the adversary from computing the session keys even if he has the secret key of the server. As a result, combining an authorisation protocol with TLS-DHE as above, yields a mutually authenticated protocol that is resilient against AKC attacks that only use the server's long-term secret key. Whether the resulting protocol is resilient against AKC attacks on the client depends on the specific authorisation protocol.

VII. Related work

The vast majority of related work has been on Key Compromise Impersonation, and has appeared solely in the domain of key establishment protocols. The first key compromise impersonation attack was described by Just and Vaudenay [4] in 1996. The first explicit, but informal definition of the notion, due to Blake-Wilson, Johnson and Menezes [3], dates back to 1997. Both of these papers consider an adversary who obtains long-term secret keys of a party, usually referred to as *the actor*, running a key establishment protocol. The adversary uses the keys to establish a session key as another protocol participant with the actor, without being detected. This results in a session key known to the adversary, which is therefore useless for securing subsequent communication.

Following [3,4], researchers have examined concrete protocols or small classes of protocols, and, in some cases, classified KCI attacks [19,20,21,22,23,24]. This has lead to a partial understanding of KCI. However, determining if an attack is a KCI attack is still done on a per-case basis, guided by minor variations on the early informal definitions.

Starting with [25], researchers have allowed the compromise of the actor's keys in computational [23,25,26] models and proved KCI resilience in concrete protocols. The underlying idea is that resilience to KCI attacks can be proved without formally stating what constitutes a KCI attack, as long as it is informally argued that KCI resilience is implied by the monolithic security model. Conversely, attacks are informally argued to be KCI attacks on a case-by-case basis.

Examples of KCI attacks on one–pass key establishment protocols can be found in [20,21], where they are classified as one of two types, depending on whether the responder authenticates the initiator. In [23], KCI attacks are either "insider" or "outsider", depending on whether the adversary actively participates in the execution of a protocol on behalf of a party whose long-term keys have been revealed. We assume that every AKC-capable adversary can actively use any keys he reveals and make no distinction between the two types of KCI attacks. A third classification is outlined in [24], where KCI attacks by adversaries with a session-key reveal capability are classified according to the way that capability is used in the attack. These attacks would be captured in our model by adding the session-key reveal capability from [5].

In [27], the fact that derivability of session keys from the secret keys and public values of just one party can be a source of insecurity is demonstrated by KCI attacks on four protocols. It is argued that the session keys should be derived from the secret keys and, ideally, run-specific data of another party.

A KCI attack by an adversary with a randomness reveal capability against the 3-pass HMQV protocol is shown in [22]. The protocol is fixed by adding a confirmation message consisting of a signed hash of both ephemeral and long-term public values. However, since the adversary can get the

actor's randomness and his long-term secret key, the adversary can compute all session keys. Therefore, their fixed protocol does not provide any security guarantees for the subsequent communications with respect to their adversary model.

In [28], it is proven that the DHE mode of TLS satisfies a monolithic security notion that implies KCI resilience. This is in line with our findings. In contrast, it is proven in [29] that all modes of TLS are secure in a weaker security model. This weaker security model does not capture AKC attacks, and these proofs therefore do not contradict our AKC attacks on TLS. Note that Paulson's simplified model of TLS [30] is vulnerable to an additional AKC attack on authentication where the adversary can replace the client's nonce by an arbitrary value to make the client's and server's views of all session keys diverge. The reason for this additional attack is that in Paulson's simplified version, the hash in the *client_verify* message does not contain all previously transmitted data. However, this is not an actual attack on the TLS protocol.

VIII. Conclusions

One of the guiding principles of modern information security is containment: given that security mechanisms may be compromised, it is prudent to design systems that limit the resulting damage as much as possible. In the domain of security protocols, AKC resilience and security are desirable features because they improve the containment of the effects of key compromise. We have provided the first systematic analysis of this phenomenon, and have given conditions under which it can and cannot be achieved.

Our transformations show how to construct protocols that are resilient against AKC. Our work thereby facilitates incremental protocol design, and enables protocol designers to provide strong security guarantees to the users of their protocols, even under actor key compromise.

For already widely deployed protocols, we have introduced fixes that make use of the underlying structure of the protocols at hand. In comparison with applying the generic transformations developed in this paper, this approach allowed us to do less intrusive fixes and get more efficient results. For TLS-RSA, the most efficient fix is to use the Diffie-Hellman mode. We showed that asymmetric cryptography is needed to obtain authentication guarantees, which has direct consequences for the improvement of existing protocols and the development of new protocols.

Our AKC attacks on protocols such as mutually authenticated and unilateral TLS-RSA show that there is still room for improvement in practice, and reveal that perfect forward secrecy is not the only advantage of using TLS-DHE.

References

[1] K. Poulsen, "Edward Snowdens E-mail provider defied FBI demands to turn over crypto keys, documents show," http://www.wired.com/threatlevel/2013/10/lavabit_unsealed/ (Retrieved 11 October 2013).

[2] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *TACAS'96*, ser. LNCS. Springer, 1996, vol. 1055, pp. 147–166.

[3] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key agreement protocols and their security analysis," in *IMA Int. Conf.*, 1997, pp. 30–45.

[4] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *ASIACRYPT*, 1996, pp. 36–49.

[5] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *ESORICS*, 2010, pp. 340–356.

[6] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*, ser. Information Security and Cryptography. Springer, 2012.

[7] C. Boyd and A. Mathuria, "Protocols for authentication and key establishment." Springer, 2003.

[8] G. Lowe, "A hierarchy of authentication specifications." IEEE Computer Society Press, 1997, pp. 31–43.

[9] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols," in *Proc. CAV*, ser. LNCS, vol. 5123. Springer, 2008, pp. 414–418.

[10] "AKC protocol models," http://www.cs.ox.ac.uk/people/cas.cremers/scyther/AKC/.

[11] "Security Protocols Open Repository," http://www.lsv.ens-cachan.fr/Software/spore/index.html.

[12] T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol," IETF RFC 4253, January 2006.

[13] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.2," IETF RFC 5246, August 2008.

[14] T. Weigold and A. Hiltgen, "Secure confirmation of sensitive transaction data in modern Internet banking services," in *WorldCIS 2011*, 2011, pp. 125–132.

[15] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification (CAV 2013)*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

[16] "Apache Module: mod auth basic," http://httpd.apache.org/docs/2.2/mod/mod_auth_basic.html (Retrieved 1 February 2014).

[17] "RFC 6749: The OAuth 2.0 Authorization Framework," http://tools.ietf.org/html/rfc6749 (Retrieved 1 February 2014).

[18] OASIS, "Security Assertion Markup Language (SAML) v2.0," https://www.oasis-open.org/standards#samlv2.0 (Retrieved 1 February 2014).

[19] G. Meng and Z. Futai, "Key-compromise impersonation attacks on some certificateless key agreement protocols and two improved protocols," in *ETCS '09. First International Workshop on*, vol. 2, 2009, pp. 62–66.

[20] K. Chalkias, F. Mpaldimtsi, D. Hristu-Varsakelis, and G. Stephanides, "On the key-compromise impersonation vulnerability of one-pass key establishment protocols," in *SECRYPT*, 2007, pp. 222–228.

[21] ——, "Two types of key-compromise impersonation attacks against one-pass key establishment protocols," in *SECRYPT*. Springer, 2008, pp. 227–238.

[22] Q. Tang and L. Chen, "Extended KCI attack against two-party key establishment protocols," *Inf. Process. Lett.*, vol. 111, no. 15, pp. 744–747, 2011.

[23] M. Gorantla, C. Boyd, and J. G. Nieto, "Modeling key compromise impersonation attacks on group key exchange protocols," in *Public Key Cryptography*, 2009, pp. 105–123.

[24] K. Shim, "The risks of compromising secret information," in *ICICS*, 2002, pp. 122–133.

[25] R. Zhu, X. Tian, and D. Wong, "Enhancing CK-model for key compromise impersonation resilience and identity-based key exchange," Cryptology ePrint Archive, Report 2005/455, 2005, http://eprint.iacr.org/.

[26] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *ProvSec*, 2007, pp. 1–16.

[27] M. Strangio, "On the resilience of key agreement protocols to key compromise impersonation," in *EuroPKI*, 2006, pp. 233–247.

[28] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Advances in Cryptology CRYPTO 2012*, ser. LNCS. Springer, 2012, vol. 7417, pp. 273–293. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32009-5_17

[29] H. Krawczyk, K. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," in *Advances in Cryptology CRYPTO 2013*, ser. LNCS. Springer, 2013, vol. 8042, pp. 429–448. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40041-4_24

[30] L. C. Paulson, "Inductive analysis of the Internet Protocol TLS," *ACM*

Trans. Inf. Syst. Secur., vol. 2, no. 3, pp. 332–351, Aug. 1999. [Online]. Available: http://doi.acm.org/10.1145/322510.322530

APPENDIX A
AKCR IS NOT MONOTONIC

One might expect AKCR to be monotonic in the following sense: if an adversary cannot perform AKC attacks, then a weaker one also cannot perform AKC attacks. We show that this is not the case.

Proposition 14: There is a protocol P, role $R \in \mathrm{dom}(\Pi)$, adversaries A and B where $A \subseteq B$, and a security claim $\gamma \in \Pi(R)$ such that AKCR with respect to the $B \cup \{\mathsf{LKR}_{\mathsf{actor}}\}$ adversary does not imply AKCR with respect to $A \cup \{\mathsf{LKR}_{\mathsf{actor}}\}$.

Proof: We need to find P, R, A, B, and γ such that the implication

$$\text{if } P \models_B, \text{ then } P \models_{B \cup \{\mathsf{LKR}_{\mathsf{actor}}\}}$$

is true and

$$\text{if } P \models_A, \text{ then } P \models_{A \cup \{\mathsf{LKR}_{\mathsf{actor}}\}}$$

is false. Let $A = \emptyset$ and $B = \{\mathsf{LKR}_{\mathsf{others}}\}$. We are done if we can give a $P = (\Pi, \mathrm{type}_\Pi)$ and γ such that $P \not\models_{\{\mathsf{LKR}_{\mathsf{others}}\}} \gamma$, $P \models_\emptyset \gamma$, and $P \not\models_{\{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma$. A well-known example of such a protocol is the Needham-Schroeder protocol where the type function assigns $\mathrm{RunTerm}$ to every variable and γ is the claim of secrecy of one of the exchanged nonces from the point of view of the responder. While an adversary cannot learn any of the two nonces without revealing any long-term secret keys, he can use either $\mathsf{LKR}_{\mathsf{actor}}$ to get the keys to decrypt them or use $\mathsf{LKR}_{\mathsf{others}}$ and perform the well-known man-in-the-middle attack. ∎

APPENDIX B
PROOFS

A. Proof of Lemma 5

We first prove an additional lemma:

Lemma 15 (inference-irrelevant terms): Let $S \cup \{t\} \subseteq \mathrm{Term}$ where $t \not\sqsubseteq S \setminus \{t\}$, $t^{-1} = t$ and t has no proper accessible subterms. Let $t' \in \mathrm{Term}$ such that $t \not\sqsubseteq t'$. Then $S \vdash t'$ if and only if $S \setminus \{t\} \vdash t'$.

Lemma 15: $\boxed{\Leftarrow}$ Trivial.

$\boxed{\Rightarrow}$ Let $t' \in \mathrm{Term}$ such that $t \not\sqsubseteq t'$. We prove by induction on n that for all $n \in \mathbb{N}$, if $S \vdash_n t'$, then $S \setminus \{t\} \vdash_n t'$. First we consider the case $n = 0$, so we have $t' \in S$. From $t \not\sqsubseteq t'$ we get $t \neq t'$, so $t' \in S \setminus \{t\}$, i.e. $S \setminus \{t\} \vdash_0 t'$. Now we assume that for all natural numbers up to some $n \in \mathbb{N}$, the statement holds and that a $\vdash$-derivation tree for t' from S of height at most $n+1$ is given.

We proceed by doing a case split on the last rule applied in that tree. If the last rule infers t' from $t_1, \ldots, t_m$ by composition, the proof is trivial: assuming $t \sqsubseteq t_i$ for some $i \in \{1, \ldots, m\}$, we would have $t \sqsubseteq t'$, which is a contradiction. Therefore, we can apply the inductive hypothesis and infer each of $t_1, \ldots, t_m$ from $S \setminus \{t\}$ with trees of height at most n each. Note that we needed $t \not\sqsubseteq t'$ precisely for this case and the base case.

In the remaining cases, the last rule to derive t' is a decomposition rule:

- $S \vdash_n (t', t'')$ or $S \vdash_n (t'', t')$: Without loss of generality, we assume the former and do a case split on the accessibility of (t', t'') in S.
  - $(t', t'') \sqsubseteq_{\mathrm{acc}} S$: Since the term t has no proper accessible subterms, from $(t', t'') \sqsubseteq_{\mathrm{acc}} S$ we get $(t', t'') \sqsubseteq_{\mathrm{acc}} S \setminus \{t\}$. Suppose $t \sqsubseteq (t', t'')$. By transitivity of $\sqsubseteq$, we conclude $t \sqsubseteq S \setminus \{t\}$, which contradicts our assumptions. We can now apply the induction hypothesis to infer $S \setminus \{t\} \vdash_n (t, t')$. Hence, $S \setminus \{t\} \vdash_{n+1} t'$.
  - $(t', t'') \not\sqsubseteq_{\mathrm{acc}} S$: We can replace the derivation of (t', t'') with one of a minimal height, which is at most n. By Lemma 6, that derivation ends in a composition rule from t' and t''. Therefore, $S \vdash_{n-1} t'$, so the induction hypothesis implies $S \setminus \{t\} \vdash_{n-1} t'$.
- $S \vdash \{t'\}_{t''}$ and $S \vdash t''^{-1}$:
  - $\{t'\}_{t''} \sqsubseteq_{\mathrm{acc}} S$: As above, we conclude $t \not\sqsubseteq \{t'\}_{t''}$. The induction hypothesis then gives us $S \setminus \{t\} \vdash_n \{t'\}_{t''}$. From $t \not\sqsubseteq \{t'\}_{t''}$ we also get $t \not\sqsubseteq t''$.
    We now have two cases: if $t''^{-1} = t''$, then $t \not\sqsubseteq t''^{-1}$ is immediate. Otherwise, if $t''^{-1} \neq t''$, from $t^{-1} = t$ we get $t \neq t''^{-1}$. Without loss of generality, let $u \in \mathrm{Term}$ such that $t'' = \mathrm{pk}(u)$ and $t''^{-1} = \mathrm{sk}(u)$. From $t \not\sqsubseteq t''$ we get $t \not\sqsubseteq u$. Hence, $t \neq t''^{-1}$ implies $t \not\sqsubseteq t''^{-1}$.
    In both cases we get $t \not\sqsubseteq t''^{-1}$, so we can apply the induction hypothesis to infer $S \setminus \{t\} \vdash_n t''^{-1}$.
  - $\{t'\}_{t''} \not\sqsubseteq_{\mathrm{acc}} S$: As above.

∎

We can now prove Lemma 5, where we use that a finite number of terms can be removed from a set by applying the lemma above, provided that it is done in the right order.

Lemma 5: Assume $S \vdash t'$. Since T is finite and partially ordered by $\sqsubseteq$, it has a maximal element t. But then $t \not\sqsubseteq T \setminus \{t\}$, which with $t \not\sqsubseteq S \setminus T$ implies $t \not\sqsubseteq S \setminus \{t\}$. We can now apply Lemma 15 and get $S \setminus \{t\} \vdash t'$. The set T is finite, so by induction we get $S \setminus T \vdash t'$. ∎

B. Proof of Lemma 6

The next auxiliary lemma states that accessible subterms of terms inferable from a set are either inferable themselves or accessible in the set.

Lemma 16 (inference of accessible subterms): Let $S \cup \{t\} \subseteq \mathrm{Term}$ and $n \in \mathbb{N}$ such that $S \vdash_n t$. Then for all $t' \in \mathrm{Term}$ such that $t' \sqsubseteq_{\mathrm{acc}} t$, $S \vdash_n t'$ or $t' \sqsubseteq_{\mathrm{acc}} S$.

Lemma 16: We prove the lemma by induction on n. For $n = 0$, we have $t \in S$, so $t' \sqsubseteq_{\mathrm{acc}} t$ implies $t' \sqsubseteq_{\mathrm{acc}} S$. Assuming that the statement holds for all natural numbers less than some n, we prove the statement for n. All decomposition cases immediately follow from the induction hypothesis. The composition cases are similar, so we only provide a proof for one. If the last step of a derivation of t from S of height

n infers $S \vdash (t_1, t_2)$ from $S \vdash t_1$ and $S \vdash t_2$, then either $t' = t$ (and thus $S \vdash_n t'$), or it is the case that $t' \sqsubseteq_{\text{acc}} t_1$ or $t' \sqsubseteq_{\text{acc}} t_2$. Without loss of generality, assume the former. Then the inductive hypothesis gives us $S \vdash_{n-1} t'$ or $t' \sqsubseteq_{\text{acc}} S$. ■

We can apply Lemma 16 to prove the following: let t be a term that is inferable from, but not accessible in, the adversary knowledge. Then the adversary must be able to construct t himself.

Lemma 6: Assume $S \vdash t$. Then there is a derivation of t from S, so there is one of minimal height n. Since $t \not\sqsubseteq_{\text{acc}} S$, we know that $t \notin S$. Hence, $n \neq 0$.

Assume that there is a derivation of t from S of height n that ends in a decomposition rule. In every decomposition rule, the premises imply that there exists a term t' such that $S \vdash_{n-1} t'$ and $t \sqsubseteq_{\text{acc}} t'$. However, $S \nvdash_{n-1} t$ by the minimality of n, so from Lemma 16 we conclude $t \sqsubseteq_{\text{acc}} S$, contradiction. ■

C. Proof of Proposition 7

Proof: $\boxed{\Rightarrow}$ We prove the contrapositive. Suppose that $(KC(\Pi), \text{type}_{KC(\Pi)}) \not\models_A \texttt{claim}_{l_1}(R, \text{commit}, R', K)$. Then there is a state $s \in \text{RS}(KC(\Pi), \text{type}_{KC(\Pi)}, R, A)$ such that

$$(Test, \sigma_{s,Test}(\texttt{claim}_{l_1}(R, \text{commit}, R', K^{\#Test}))) \in tr_s$$

and there is no $rid \in \text{RID}$ where both $\text{role}_s(rid) = R'$ and

$$(rid, \sigma_{s,Test}(\texttt{claim}_{l_1}(R', \text{running}, R, K^{\#Test}))) \in tr_s.$$

For all $L \subseteq \text{Label}$, we define the function

$$\delta_L(\langle\rangle) = \langle\rangle$$
$$\delta_L(\langle(rid, e)\rangle.u) = \begin{cases} \delta(u), & \text{if } \text{label}(e) \in L, \\ \langle(rid, e)\rangle.\delta(u), & \text{otherwise.} \end{cases}$$

We want to construct an attack $s' \in \text{RS}(\Pi, \text{type}_\Pi, A, R)$ on the secrecy of K where $tr_{s'} = \delta_{\{l_1,l_2\}}(tr_s)$. First we prove that h does not appear in $\delta_{\{l_1,l_2\}}(tr_s)$. Assume that the opposite is true, i.e. that for $T = \{h(t_0, \ldots, t_m) : t_0, \ldots, t_m \in \text{Term}, m \in \mathbb{N}\}$, there exist $\tau \in T$, $rid \in \text{RID}$ and $e' \in \text{RunEvent}$ such that $(rid, e') \in tr_s$, $\tau \sqsubseteq \text{cont}(e')$ and $\text{label}(e') \notin \{l_1, l_2\}$. Then either there exists $e'' \in \Pi(\text{role}_s(rid))$ such that $x \sqsubseteq_{\text{acc}} \text{cont}(e'')$ and $\tau \sqsubseteq \sigma_{s',rid}(x)$, or there is a $\tau' \in T$ such that $\tau' \in \Pi(\text{role}_s(rid))$ and $\sigma_{s,rid}(\tau') = \tau$. Both cases contradict the assumptions on h.

The construction of s' proceeds inductively, by following $\delta_{\{l_1,l_2\}}(tr_s)$. For prefix length 0, we know the state

$$s_0 = (\langle\rangle, AK_0, Test \mapsto \sigma_{s,Test}(\Pi(R)^{\#Test}), Test \mapsto \sigma_{s,Test})$$

is reachable.

The only interesting case in the induction step involves checking if `recv` transitions are still enabled. Let s_n be the state reached after n transitions from s_0 and $e \in \text{RunEvent}$ with $\text{evtype}(e) = \texttt{recv}$ the next event. All we need to prove is $AK_{s_n} \vdash \text{cont}(e)$. Since e is also an event in tr_s, let s'_n be any state such that $s'_n \to^* s$, just after e is executed. The only `send` events deleted by $\delta_{\{l_1,l_2\}}$ are the ones labelled l_2. Hence, there exists a finite set $T' \subseteq T$ such that $AK_{s_n} = AK_{s'_n} \setminus T'$. We know that $\text{cont}(e)$ does not contain h. Since for all $\tau \in T'$, $\tau \not\sqsubseteq AK_{s_n}$, we can apply Lemma 5 to get $AK_{s_n} \vdash \text{cont}(e)$.

We still need to prove that the adversary knows the instance of K in s', i.e. $AK_{s'} \vdash \sigma_{s',Test}(K^{\#Test})$. Let $\tau = \sigma_{s,Test}(h(R, R', K)^{\#Test})$. We prove that no `send` event in tr_s contains τ as a syntactic subterm. Let $rid \in \text{RID}$ and $e' \in \text{RunEvent}$ such that $(rid, e') \in tr_s$, $\text{evtype}(e') = \texttt{send}$, and $\tau \sqsubseteq \text{cont}(e')$. Since $\tau \in T$, we know that $\text{label}(e') = l_2$ and $\text{role}_s(rid) = R'$. By the definition of $\sqsubseteq$, we have

$$\{t : t \sqsubseteq \text{cont}(e')\} = \{\text{cont}(e')\} \cup \{\sigma_{s,rid}(R), \sigma_{s,rid}(R')\} \cup \{t : t \sqsubseteq \sigma_{s,rid}(K'^{\#rid})\}.$$

Because of the running claim which precedes the $\texttt{send}_{l_2}$ role event in $KC(\Pi)$, $\tau = \text{cont}(e')$ would contradict our assumptions that no such claim occurs in tr_s. Since $\tau \notin \text{Agent}$, we get $\tau \sqsubseteq \sigma_{s,rid}(K'^{\#rid})$, which contradicts the typing assumptions on K'. Therefore, no `send` event in tr_s contains τ as a syntactic subterm, so $\tau \not\sqsubseteq AK_s$ holds.

From $AK_s \vdash \tau$ and Lemma 6, we now conclude that the adversary constructed the hash τ himself, i.e. $AK_s \vdash \sigma_{s',Test}(K^{\#Test})$. We have $AK_{s'} = AK_s \setminus T'$, for some finite $T' \subseteq T$. For all $\tau' \in T'$, we have $\tau' \not\sqsubseteq \sigma_{s',Test}(K^{\#Test})$ because h does not occurs in $tr_{s'}$, so Lemma 5 gives us $AK_{s'} \vdash \sigma_{s',Test}(K^{\#Test})$.

$\boxed{\Leftarrow}$ Let $s \in \text{RS}(\Pi, \text{type}_\Pi, R, A)$ such that

$$(Test, \sigma_{s,Test}(\texttt{claim}_{l_0}(R, \text{secret}, K^{\#Test}))) \in tr_s$$

and $AK_s \vdash \sigma_{s,Test}(K^{\#Test})$. We now construct $s' \in \text{RS}(KC(\Pi), \text{type}_{KC(\Pi)}, A, R)$ such that

$$(Test, \sigma_{s',Test}(\texttt{claim}_{l_1}(R, \text{commit}, R', K)^{\#Test})) \in tr_s$$

and there is no $rid \in \text{RID}$ such that $\text{role}_s(rid) = R'$ and

$$(rid, \sigma_{s,Test}(\texttt{claim}_{l_1}(R', \text{running}, R, K)^{\#Test}))$$

is in tr_s.

Since $\texttt{claim}_{l_0}$ is the last step of $\Pi(R)$, we know that $th_s(Test) = \langle\rangle$. For all $rid \in \text{dom}(th_s)$, we define $th_{s'}(rid)$ as

$$th_s(rid).\sigma_{s,rid}(\langle \texttt{recv}_{l_2}(R', R, h(R, R', K)), \texttt{claim}_{l_1}(R, \text{commit}, R', K)\rangle^{\#rid})$$

if $\text{role}_s(rid) = R$ and $rid \neq Test$,

$$th_s(rid).\sigma_{s,rid}(\langle \texttt{claim}_{l_1}(R', \text{running}, R, K'), \texttt{send}_{l_2}(R', R, h(R, R', K'))\rangle^{\#rid})$$

if $\text{role}_s(rid) = R'$ and $rid \neq Test$, and $th_s(rid)$ otherwise. Then for

$$s' = (tr_s.\sigma_{s,Test}(\langle(Test, \texttt{recv}_{l_2}(R', R, h(R, R', K))), (Test, \texttt{claim}_{l_1}(R, \text{commit}, R', K))\rangle^{\#Test}), AK_s, th_{s'}, \sigma_s),$$

we have $s' \in \text{RS}(KC(\Pi), \text{type}_{KC(\Pi)}, R, A)$. Therefore,

$$(KC(\Pi), \text{type}_{KC(\Pi)}) \not\models_A \texttt{claim}_{l_1}(R, \text{commit}, R', K).$$

■

D. Proof of Lemma 8

Proof: We prove the lemma by induction on the derivation height n for t'' from S. If $n = 0$, then $t'' \in S$, so the statement is trivial. Assume $n \neq 0$ and fix any derivation of height n. The induction hypothesis tells us that $t \not\sqsubseteq_{\text{acc}} \{\bot \,/\, t'\}(t_i)$ for all t_i appearing in the premises of the last rule in the derivation. If the rule is a decomposition rule, we are done. Otherwise, $t \neq t''$ implies $t \not\sqsubseteq_{\text{acc}} \{\bot \,/\, t'\}(t'')$. ∎

E. Proof of Lemma 9

Proof: We prove the statement by contradiction. Assume $AK_s \vdash \text{sk}(a)$. Then Lemma 6 implies $\text{sk}(a) \sqsubseteq_{\text{acc}} AK_s$, because no derivation of $\text{sk}(a)$ can end in a composition rule. Since $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_0$, there exist $s', s'' \in \text{RS}(\Pi, \text{type}_\Pi, R, A)$ where $s'' \to s' \to^* s$, $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_{s''}$ and $\text{sk}(a) \sqsubseteq_{\text{acc}} AK_{s'}$. Therefore, for $rid \in \text{RID}$ and $e \in \text{Event}$ such that $last(tr_{s'}) = (rid, e)$, either $(rid, e) = (rid_{\mathcal{A}}, \mathsf{LKR}(a))$, which contradicts the fact that adversaries cannot reveal peers' keys, or (rid, e) is the first $\texttt{send}$ of $\text{sk}(a)$ in an accessible position in tr_s.

For $e' \in \Pi(\text{role}_{s'}(rid))$ such that $\text{label}(e') = \text{label}(e)$, the existence of $t \in \text{RoleTerm}$ such that $\text{sk}(t) \sqsubseteq_{\text{acc}} \text{cont}(e')$ would contradict the well-formedness of the sequence $\Pi(\text{role}_{s'}(rid))$. Hence, we conclude that there exists $x \sqsubseteq_{\text{acc}} \text{cont}(e')$ such that $\text{sk}(a) \sqsubseteq_{\text{acc}} \sigma_{s',rid}(x)$.

Assume e'' is the role event of the $\texttt{recv}$ where rid initialised the variable x, in s'' or a reachable state before s''. Let $t = \text{cont}(\sigma_{s'',rid}(e''^{\#rid}))$. We have $AK_{s''} \vdash t$, $\text{sk}(a) \sqsubseteq_{\text{acc}} t$ and $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_{s''}$. Lemma 16 then gives us $AK_{s''} \vdash \text{sk}(a)$. From Lemma 6 we deduce that there is a derivation of $\text{sk}(a)$ from $AK_{s''}$ that ends in a composition rule, contradiction. ∎

F. Proof of Proposition 10

Proof: Let $s \in \text{RS}(\,TS(\Pi), \text{type}_{TS(\Pi)}, R, A)$ such that

$$(Test, \sigma_{s,Test}(\texttt{claim}_{l'}(R, \text{secret}, m^{\#Test})) \in tr_s.$$

We want to prove that $AK_s \not\vdash \sigma_{s,Test}(m^{\#Test})$. First we use induction over the prefix length of tr_s to prove that $n^{\#Test}$ only appears accessible in $\texttt{send}$ events in tr_s as a subterm of $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$. Let $s' \in \text{RS}(\,TS(\Pi), \text{type}_{TS(\Pi)}, R, A)$, $rid \in \text{RID}$ and $e \in \text{Event}$ such that $s' \to^* s$, $(rid, e) = last(tr_{s'})$, $\text{evtype}(e) = \texttt{send}$ and $n^{\#Test} \sqsubseteq_{\text{acc}} \text{cont}(e)$. If $rid = Test$, the statement is true, because the only time $Test$ sends $n^{\#Test}$ is when $n^{\#Test}$ is generated in the claim labelled l.

Otherwise, if $rid \neq Test$, let $\bot \in \text{Const}$ be any constant unused in $TS(\Pi)$. From the induction hypothesis and the definition of AK_0, we know that $n^{\#Test}$ only appears accessible inside $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$ in $AK_{s'} \setminus \{\text{cont}(e)\}$, i.e.

$$n^{\#Test} \not\sqsubseteq_{\text{acc}} \{\bot / \sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})\}(AK_{s'} \setminus \{\text{cont}(e)\}).$$

For that reason, Lemma 8 tells us that for all $t \in \text{Term}$ such that $AK_{s'} \setminus \{\text{cont}(e)\} \vdash t$,

$$n^{\#Test} \not\sqsubseteq_{\text{acc}} \{\bot \,/\, \sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})\}(t).$$

Hence, $n^{\#Test}$ was only received in $tr_{s'}$ inside $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$.

Assume that $n^{\#Test}$ occurs accessible outside $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$ in $\text{cont}(e)$. But then there exist $\{t\}_{t'} \in S$, $x \in \text{Var}$ and $rid \in \text{RID}$ such that $\{t\}_{t'} \sqsubseteq TS(\Pi)(\text{role}_{s'}(rid))$, $x \sqsubseteq_{\text{acc}} t$ and $n^{\#Test} \sqsubseteq_{\text{acc}} \sigma_{s',rid}(x)$. We then have

$$\sigma_{s,rid}(\tau_{c_1}((\{t\}_{t'})^{\#rid})) = \sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')}),$$

which contradicts $c_1 \neq c_2$.

Hence, $n^{\#Test}$ is only accessible in the set AK_s as a subterm of the term $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$. However, from Lemma 9 we get $\text{sk}(\sigma_{s,Test}(R')) \notin AK_s$. Therefore, $AK_s \not\vdash n^{\#Test}$. From Lemma 8, we get $AK_s \not\vdash \sigma_{s,Test}(m^{\#Test})$. ∎

G. Proof of Proposition 11

Proof: Let $s \in \text{RS}(\,TA(\Pi), \text{type}_{TA(\Pi)}, R, A)$ and

$$(Test, \sigma_{s,Test}(\texttt{claim}_l(R, \text{commit}, R', m^{\#Test}))) \in tr_s.$$

We now prove that there is a run with the corresponding running claim.

Denote $t = \sigma_{s,Test}(\{R, m^{\#Test}, c_2\}_{\text{sk}(R')})$. From the above, there exists $s' \in \text{RS}(\,TA(\Pi), \text{type}_{TA(\Pi)}, R, A)$ such that $s' \to^* s$ and

$$(Test, \texttt{recv}_{l'}(\sigma_{s,Test}(R'), \sigma_{s,Test}(R), t)) = last(tr_{s'}).$$

From there, we have $AK_{s'} \vdash t$. We know from Lemma 9 that $AK_{s'} \not\vdash \text{sk}(\sigma_{s,Test}(R'))$ holds. That means no derivation of t from $AK_{s'}$ can end in a composition step, so Lemma 6 now implies $t \sqsubseteq_{\text{acc}} AK_{s'}$. Thus, there exist $rid \in \text{RID}$ and $e \in \text{RunEvent}$ such that $\text{evtype}(e) = \texttt{send}$, $(rid, e) \in tr_{s'}$ and $t \sqsubseteq_{\text{acc}} \text{cont}(e)$.

Without loss of generality, let (rid, e) be the first $\texttt{send}$ with the above properties in $tr_{s'}$. Assume first that $\text{label}(e) \neq l'$. Then e is an instance of a tagged step of Π, i.e. there is a unique $t' \in \text{RoleTerm}$ such that $\sigma_{s',rid}(t'^{\#rid}) = \text{cont}(e)$ and

$$\texttt{send}_{\text{label}(e)}(\cdot, \cdot, t') \in \tau_{c_1}|_S(\Pi(\text{role}_{s'}(rid))).$$

We know that t cannot occur in $\text{cont}(e)$ as a subterm of any instantiated variables of t', since Lemma 16 would contradict the minimality of (rid, e). That means there must be a $\{t_0\}_{t_1} \in S$ such that $\sigma_{s',rid}((\{t_0, c_1\}_{t_1})^{\#rid}) = t$. However, that implies $c_1 = c_2$, contradiction.

Therefore, $\text{label}(e) = l'$. Hence, $\text{role}_{s'}(rid) = R'$,

$$\sigma_{s',rid}(\texttt{claim}_l(R', \text{running}, R, m')^{\#rid}) \in tr_{s'}$$

and $\sigma_{s',rid}((R, R', m')^{\#rid}) = \sigma_{s,Test}((R, R', m)^{\#Test})$. ∎

H. Proof of Proposition 12

For the proof, we need two lemmas. The first lemma states that substitutions preserve inference.

Lemma 17 (substitutions preserve inference): Let σ be a substitution. For all $S \cup \{t\} \subseteq \text{dom}(\sigma)$, if $S \vdash t$, then $\sigma(S) \vdash \sigma(t)$.

Proof: By induction on height n of the $\vdash$-derivation tree for t. For $n = 0$, we have $t \in S$, so $\sigma(t) \in \sigma(S)$. Assuming that the statement holds for all trees of height less than some n, we

prove the statement for any tree of height n. If the premises of the last step of the derivation are $S \vdash t_1, \ldots, S \vdash t_n$, then the inductive hypothesis gives us $\sigma(S) \vdash \sigma(t_1), \ldots, \sigma(S) \vdash \sigma(t_n)$. Since σ is an endomorphism on Term, we can apply the appropriate derivation rule and conclude $\sigma(S) \vdash \sigma(t)$. ■

The second lemma we need states that inferring all atomic information and all long-term keys in a term is enough to infer the term.

Lemma 18 (composition from atomic subterms): Let $S \cup \{t\} \subseteq \mathrm{Term}$. If for all $x \sqsubseteq t$ where x is atomic or x is a long-term key, $S \vdash x$, then $S \vdash t$.

Proof: We prove the statement by structural induction on t. If t is atomic or a long-term key, since $t \sqsubseteq t$, the assumption gives us $S \vdash t$.

If for some $t_1, \ldots, t_n$, the term t is equal to (t_1, t_2), $\{t_1\}_{t_2}$ or $f(t_1, \ldots, t_n)$, then the induction hypothesis implies $S \vdash t_1, \ldots, t_n$. We can then use the corresponding composition rule to get t. ■

We can now prove Proposition 12.

Proof: Let $a, b \in \mathrm{Agent}$ such that $a \neq b$ and define $\tau(R) = a$ and $\tau(R') = b$. For all $x \in \mathrm{Var}$, let $\tau(x) = n_x^{\#rid_{\mathcal{A}}}$. Assume that τ is extended to $\mathrm{Role} \cup \mathrm{Var}$ in an arbitrary, but fixed way. Then $\tau \in TS(\Pi, \mathrm{type}_\Pi)$. If the length of $\Pi(R)$ is k for some $k \in \mathbb{N}$, we define seq to be

$$\langle (Test, \tau(\Pi(R)^{\#Test})_0), \ldots, (Test, \tau(\Pi(R)^{\#Test})_{k-1}) \rangle$$

and

$$\begin{aligned} s = (&\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle . seq, \\ &AK_0 \cup \mathrm{LTK}(a) \cup (\tau(\Pi(R)^{\#Test}) \lfloor \mathtt{send}), \\ &Test \to \langle \rangle, Test \mapsto \tau). \end{aligned}$$

The proof of reachability proceeds by induction on the prefix length of the sequence $\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle . seq$. More specifically, we prove two statements within the induction:

- the adversary can infer all atomic subterms of all `send` events after they occur, and
- the adversary can infer the contents of all `recv` events just before they occur.

For lengths 0 and 1, we know that the states

$s_0 = (\langle\rangle, AK_0, Test \mapsto \tau(\Pi(R)^{\#Test}), Test \mapsto \tau)$ and $s_1 =$
$(\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle, AK_0 \cup \mathrm{LTK}(a), Test \mapsto \tau(\Pi(R)^{\#Test}), Test \mapsto \tau)$

are reachable. Let $n \in \mathbb{N} \setminus \{0\}$, s_n the state reached after n transitions from s_0 and $e \in \mathrm{RunEvent}$ the next event.

Let K be any long-term secret key such that $K \sqsubseteq \mathrm{cont}(e)$. We want to prove that $K \in AK_{s_n}$. Since τ instantiated every variable in $\Pi(R)$ is by some nonce local to $\mathrm{rid}_{\mathcal{A}}$, we know that K does not occur inside a variable instance in $\mathrm{cont}(e)$. Therefore, there is a t in $\Pi(R)$ of the form $\mathrm{k}(\cdot,\cdot)$, $\mathrm{pk}(\cdot)$ or $\mathrm{sk}(\cdot)$ such that $K = \sigma_{s_n, Test}(t)$ and $\sigma_{s_n, Test} \sqsubseteq e$. The first assumption in the proposition statement now gives us that $K = \mathrm{k}(c, d)$, for some $c, d \in \mathrm{Agent}$, and the second one implies that $c = a$ or $d = a$. Hence, $K \in \mathrm{LTK}(a)$, which implies $K \in AK_{s_1}$. Since $AK_{s_1} \subseteq AK_{s_n}$, $K \in AK_{s_n}$.

For the first proof obligation, assume $\mathrm{evtype}(e) = \mathtt{send}$. Let $t \sqsubseteq \mathrm{cont}(e)$ such that t is atomic. If t is an agent, constant, or a nonce local to $rid_{\mathcal{A}}$, we have $AK_{s_n} \vdash t$. There is still the case $t = n^{\#Test}$ for some $n \in \mathrm{Fresh}$. By the fourth assumption, no freshly generated nonces in e are subterms of any keys that t is potentially encrypted with. Moreover, every long-term key in $\mathrm{cont}(e)$ is an element of AK_{s_n}. Therefore, $AK_{s_n} \cup \{\mathrm{cont}(e)\}$ infers any such key by Lemma 18. Hence, $AK_{s_n} \cup \{\mathrm{cont}(e)\} \vdash t$.

The second proof obligation involves checking if `recv` transitions are still enabled. Assume $\mathrm{evtype}(e) = \mathtt{recv}$. All we need to prove is $AK_{s_n} \vdash \mathrm{cont}(e)$. We know that AK_{s_n} contains all long-term keys in $\mathrm{cont}(e)$ and that every fresh value in tr_{s_n} is local to either $\mathrm{rid}_{\mathcal{A}}$ or $Test$. For all $n \in \mathrm{Fresh}$ such that $n^{\#Test} \sqsubseteq \mathrm{cont}(e)$, $n^{\#Test}$ was sent before. Since AK_{s_n} can infer all atomic subterms in earlier `send` events, $AK_{s_n} \vdash n^{\#Test}$. By Lemma 18, $AK_{s_n} \vdash \mathrm{cont}(e)$.

For the proof of the second part of the lemma statement, let γ be stronger than $\mathtt{claim}_l(R, \mathrm{alive}, R')$. Then $(\Pi, \mathrm{type}_\Pi) \not\models_{\{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma$. Due to $(\Pi, \mathrm{type}_\Pi) \models_\emptyset \gamma$, we know that the $\{\mathsf{LKR}_{\mathsf{actor}}\}$ adversary can perform AKC attacks on γ in (Π, type_Π). ■