Actor Key Compromise: Consequences and Countermeasures

David Basin
Institute of Information Security
Dept. of Computer Science, ETH Zurich
Zurich, Switzerland
basin@inf.ethz.ch

Cas Cremers
Dept. of Computer Science
University of Oxford
Oxford, UK
cas.cremers@cs.ox.ac.uk

Marko Horvat
Dept. of Computer Science
University of Oxford
Oxford, UK
marko.horvat@cs.ox.ac.uk

***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 to communicate securely with other parties, depending 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, adversary models, TLS, SSH

I. Introduction

If a government agency obtains the long-term secret key service provider [1,2], it is clear that the agency can impersonate the service provider to its users. 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. For example, Alice is guaranteed the secrecy of the nonce na and agreement on its value when sending it encrypted to Bob and receiving a hash of it as follows:

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

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. As a standard example, consider the Needham-Schroeder-Lowe protocol [3]:

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

Such bilateral protocols can be viewed as combining 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? Since Bob's key is not compromised, Alice might expect 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 Alice's secret key is compromised in the Needham-Schroeder-Lowe protocol, she then no longer obtains secrecy of the nonces nor achieves agreement on nb. Consider the following attack, in which $\mathcal{A}_{Alice}$ denotes that the adversary $\mathcal{A}$ sends or receives a message as Alice:

$$\begin{array}{lll} 1. & Alice \rightarrow Bob: & \{na, Alice\}_{\mathrm{pk}(Bob)} \\ 2. & Bob \rightarrow \mathcal{A}_{Alice}: & \{na, nb, Bob\}_{\mathrm{pk}(Alice)} \\ 3. & \multicolumn{2}{l}{\mathcal{A} \text{ decrypts message using } \mathrm{sk}(Alice) \text{ and learns } na, nb} \\ 4. & \multicolumn{2}{l}{\mathcal{A} \text{ chooses } nb' \text{ and constructs } \{na, nb', Bob\}_{\mathrm{pk}(Alice)}} \\ 5. & \mathcal{A}_{Bob} \rightarrow Alice: & \{na, nb', Bob\}_{\mathrm{pk}(Alice)} \\ 6. & Alice \rightarrow \mathcal{A}_{Bob}: & \{nb'\}_{\mathrm{pk}(Bob)} \end{array}$$

We say that such protocols 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 by the security protocol community. A notable exception is in the research literature on authenticated key exchange protocols, where a limited instance of this problem has been studied. Namely, there are so-called *Key Compromise Impersonation* (KCI) attacks [4,5], where the actor's key is revealed and used by the adversary to impersonate another party communicating 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 any security properties of that party in any type of security protocol.

Contributions. We provide the first systematic analysis of the consequences of compromising the actor's secret key, and propose countermeasures. First, we introduce actor key compromise and define 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 showing how some actor key compromise vulnerabilities can be avoided in protocol design by using asymmetric encryption and signatures. Third, we prove impossibility results showing 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 look at actor key compromise in practice. 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 implement and carry out 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 and use it 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 impossibility results 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

We first give an informal definition of an AKC attack to provide the context for our model. In the next section, we formally define this concept.

Definition 1 (Actor key compromise attack, informal): We say that an attack on a security property of an agent A is an *actor key compromise attack* if the attack requires that the adversary obtains and uses a long-term private key of A.

Before we describe our formal framework, we give the notation we use for functions and sequences. For f a partial function from X to Y, we write $f : X \nrightarrow Y$. For every partial function f, we denote its domain by $\mathrm{dom}(f)$ and its range by $\mathrm{ran}(f)$. We write $f \cup f'$ for the union of two partial functions with disjoint domains. 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 $x \in \mathrm{dom}(f) \setminus \{a\}$, $f'(x) = f(x)$. If f is also a (total) function, i.e. it is defined for each element of X, we write $f : X \rightarrow Y$. 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 $i \in \{1, \ldots, n\}$ 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 [6,7], where the main building blocks for protocol specifications 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 contain role and run terms, respectively. We assume, prior to protocol execution, that every agent has generated or securely received a long-term asymmetric key pair consisting of a public and a secret key, and has authentic copies of his long-term symmetric keys and the public keys of all other agents.

We assume given the pairwise disjoint, infinite sets Role, Agent, Fresh, RID, Var, and Func of roles, agent names, freshly generated terms (nonces, coin flips, etc.), run identifiers, variables, and function names (hash functions, constants, etc.). We also assume that for all $n \in \mathbb{N}_0$, 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. 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 under attack. We refer to it to define the adversary's capabilities to perform different types of key reveal queries. Such queries can be executed by the adversary run.

Definition 2 (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 $n \in \mathbb{N}_0$ depends on the function name. We omit the brackets if $n = 0$. By $\mathrm{k}(X, X')$ we denote 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 the corresponding secret key. By $\{t_1\}_{t_2}$ we denote either signature, asymmetric encryption, or symmetric encryption, depending on whether t_2 is an asymmetric secret key, public key, or any other term, respectively. Pairing is not associative, and we write (a, b, c) to denote $((a, b), c)$.

We define *substitutions* as partial functions in the usual way, except that they are defined on $\mathrm{Role} \cup \mathrm{Var}$. A substitution $\sigma : \mathrm{Role} \cup \mathrm{Var} \nrightarrow \mathrm{Term}$ such that $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$,

$\mathrm{ran}(\sigma) = \{t_1, \ldots, t_n\}$, and $\sigma(x_i) = t_i$ for all $i \in \{1 \ldots, n\}$ is written as $\sigma = [t_1, \ldots, t_n/x_1, \ldots, x_n]$. When applying a substitution to terms, we extend the substitution to an endomorphism on Term in the standard way. By $\{t/t'\}$ we 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 3 (Syntactic subterm relation): The *syntactic subterm relation* $\sqsubseteq$ is the reflexive, transitive closure of the smallest relation $\sqsubseteq$ on terms such that for all $n \in \mathbb{N}$, $t_1, \ldots, t_n \in$ Term, and $f \in$ Func of arity n:

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

Definition 4 (Accessible subterm relation): The *accessible subterm relation* $\sqsubseteq_{\mathrm{acc}}$ is the reflexive, transitive closure of the smallest relation on terms where for all $t_1, t_2 \in$ 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: 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 $t \sqsubseteq t'$ ($t \sqsubseteq_{\mathrm{acc}} t'$). We write $\mathrm{vars}(t)$ for $\{x \in \mathrm{Var} : x \sqsubseteq t\}$. We assume the existence of an inverse function on terms such that for all $t \in$ Term, if $t = \mathrm{pk}(t')$ or $t = \mathrm{sk}(t')$ for some t', then $t^{-1} = \mathrm{sk}(t')$ or $t^{-1} = \mathrm{pk}(t')$, respectively; otherwise, $t^{-1} = t$.

We now define which terms can be inferred from a given set of terms. We denote by $\vdash$ the smallest relation such that for all $n \in \mathbb{N}_0$, $f \in$ Func of arity n, and $S \cup \{t_1, \ldots, t_n\} \subseteq$ Term, $\vdash$ respects the following rules:

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

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

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

We call the rules in the second row *composition rules* and those 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 with every branch of the derivation closed, we write $S \vdash t$ and say that S *infers* t. We call such 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 terms that have no subterms in Agent $\cup \{n^{\#rid} : n \in \mathrm{Fresh}, rid \in \mathrm{RID}\}$. We define RunTerm as the set of terms that have no subterms in Role $\cup$ Fresh. We call Role $\cup$ Agent $\cup \{x, x^{\#rid} : x \in \mathrm{Fresh}, rid \in \mathrm{RID}\} \cup \mathrm{Var} \cup \mathrm{Const}$ the set of *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. They are Claim, the set of all claim names, which we use to specify claims of various security properties, and Label, which we use to label events. We additionally assume that $\{\mathrm{alive}, \mathrm{commit}, \mathrm{running}, \mathrm{secret}\} \subseteq$ Claim.

Definition 5 (Role and run events): For all $R \in$ Role and $a \in$ Agent, we define:

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

Additionally, we define $\mathrm{RoleEvent} = \bigcup_{R \in \mathrm{Role}} \mathrm{RoleEvent}_R$ and $\mathrm{RunEvent} = \bigcup_{a \in \mathrm{Agent}} \mathrm{RunEvent}_a$.
For example, the event $\mathtt{send}_{l_0}(Alice, Bob, \{n^{\#rid}\}_{\mathrm{pk}(Bob)})$ signifies that Alice sends Bob a nonce $n^{\#rid}$, which is generated in the run rid and encrypted with Bob's public key.

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

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

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

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

Definition 6 (Well-formed sequence of role events for R): Let $R \in$ Role. A sequence $s \in \mathrm{RoleEvent}_R^*$ is *well-formed for* R if:

- all event labels in s are unique,
- for all $x \in$ Var, if $e \in s$ is the first event in s such that $x \sqsubseteq \mathrm{cont}(e)$, then $\mathrm{evtype}(e) = \mathtt{recv}$ and $x \sqsubseteq_{\mathrm{acc}} \mathrm{cont}(e)$ *(every variable occurring in an event must be initialised in an accessible position in a* $\mathtt{recv}$*)*, and
- for all $l \in$ Label, $R' \in$ Role, and $t \in$ RoleTerm such that $\mathtt{send}_l(R, R', t) \in s$, we have that $\mathrm{LTK}(R) \cup \{S, \mathrm{pk}(S) : S \in \mathrm{Role}\} \cup \{n \in \mathrm{Fresh} : n \sqsubseteq (\mathrm{upto}(s, l) \downharpoonright \mathtt{send})\} \cup$

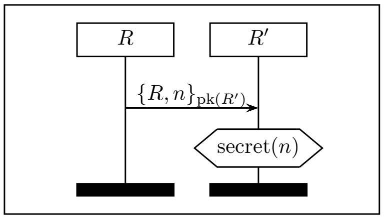


Fig. 1. Example protocol 1.

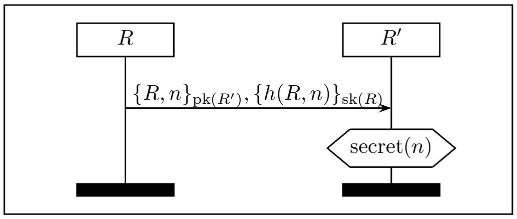


Fig. 2. Example protocol 2.

$(\mathrm{upto}(s,l) \lfloor \mathtt{recv}) \vdash t$ *(role R must be able to construct the contents of each of its* `send` *events).*

When a message is received during protocol execution (defined in the next section), some terms may be stored in variables. A *type function* formalises which terms can be stored in which variable. A `recv` step can be executed only if each variable stores a term of its type.

Definition 7 (Protocol): Let $\Pi : \mathrm{Role} \nrightarrow \mathrm{RoleEvent}^*$ be a partial function and $\mathrm{type}_\Pi : \mathrm{Var} \rightarrow \mathcal{P}(\mathrm{RunTerm})$ a function. If for all $R \in \mathrm{dom}(\Pi)$, $\Pi(R)$ is well-formed for R, 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, depicted as message sequence charts. In the first one, R sends to R' its identity and a freshly generated nonce n, encrypted with the public key of R'. In the second protocol, a signed hash of the payload is additionally transmitted. In both protocols, role R' claims the secrecy of n upon successful completion, i.e. that the adversary cannot infer it.

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.

Let us assume that the protocol from Figure 1 is a key transport protocol, and that n is a fresh session key. If n is secret, R' has the following guarantee: any messages symetrically encrypted with n that R' later receives are secret and are sent by R. If the adversary, however, knows $\mathrm{sk}(R')$, he can learn n and use it to encrypt any message for R'.

Example 1: One possible formal specification of the protocol in Figure 1 is as follows: let $R, R' \in \mathrm{Role}$, $\{1,2,3\} \subseteq \mathrm{Label}$, $n \in \mathrm{Fresh}$, $x_n \in \mathrm{Var}$, and define

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

Moreover, let type_Π assign RunTerm to every variable.

We can extend 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 $\mathrm{type}_\Pi(y) = \{y\}$. 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 and variable instantiations σ_s of all runs. To keep the notation compact, we write $\sigma_s(rid)$ as $\sigma_{s,rid}$.

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

Definition 8 (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

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

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

Example 2: We consider the following initial state for the protocol in Example 1:

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

The operational semantics of a protocol (Π, type_Π) is defined by a transition system that combines the execution-model rules from Figure 3 with a set of adversary-compromise rules or *capabilities* [6], chosen from those in Figure 4. We identify adversaries with the set of their capabilities. We normally omit the subscripted parameters from the rule names.

The `create` rule starts a new run of a protocol role R. The rule is parametrised by the function Π. A fresh run identifier rid is assigned to the run, thereby distinguishing it from previously created runs, the adversary run, and the test run. The role names of $\Pi(R)$ are replaced with agent names by a substitution σ', which is saved in the state. The `send` rule sends a message m to the network, thereby adding it to the adversary knowledge. In contrast, the `recv` rule, which is parametrised by type_Π, accepts messages from the network that match the pattern pt, where pt is a term that may contain variables. Each variable must be instantiated with an element of its type. The resulting

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

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

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

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

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

substitution σ' is applied to the remaining steps of rid and saved in the state. The `claim` rule is used simply to log the statements that runs make about the security properties they expect to hold. We explain the connection between claims and security properties in detail later in this section.

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 needed since we allow 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 reveal the keys of any agent a that is not an intended partner (or *peer*) of the test run.

Definition 9 (Transition relation and reachable states): Let $(\Pi, \mathrm{type}_{\Pi})$ 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: The state $(\langle(rid, \mathtt{create}(R))\rangle, AK_0, th, \sigma)$ where

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

and

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

is reached from the initial state in Example 2 by a single application of the `create` rule, regardless of the adversary A. In this state, Alice is running the newly created run rid of the role R with whom she believes to be Bob in role R'. 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 security properties for role R: secrecy, aliveness, and non-injective data agreement [8]. First we introduce an auxiliary function: for all $rid \in \mathrm{RID}$, $R' \in \mathrm{Role}$, and reachable states s (for any instantiation of Π, type_{Π}, R and A) such that $(rid, \mathtt{create}(R')) \in tr_s$, we define $\mathrm{role}_s(rid) = R'$, and we let $\mathrm{role}_s(Test) = R$.

Definition 10 (Security claims, $\models$): Given a label $l \in \mathrm{Label}$, roles $R, R' \in \mathrm{Role}$, and $t \in \mathrm{RoleTerm}$, we call every

$$\begin{aligned} \gamma \in \{&\mathtt{claim}_l(R, \mathrm{secret}, t), \\ &\mathtt{claim}_l(R, \mathrm{alive}, R'), \\ &\mathtt{claim}_l(R, \mathrm{commit}, R', t)\} \end{aligned}$$

a *security claim for R*. For all states $s \in \mathrm{State}$, $s \models \gamma$ denotes that the following implication is true: if $(Test, \sigma_{s,Test}(\gamma^{\#Test})) \in tr_s$, then

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

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

Note that we do not consider a running claim to be a security claim and simply use it to define non-injective agreement. Namely, if in a particular state a commit claim executed by

$Test$ matches a running claim previously executed by rid, and rid was run in the correct role, then $Test$ and rid agree on the claims' contents.

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

III. Formalising actor key compromise

In this section we formalise actor key compromise and related notions. This enables us to reason about the security guarantees of agents whose long-term secret keys are compromised. We define these notions independently of the choice of protocol, adversary, and property.

Definition 11 (Actor key compromise security): 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 secure in* (Π, type_Π) *with respect to* A if $(\Pi, \mathrm{type}_\Pi) \models_A \gamma$.

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

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

Definition 12 (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

$$(\Pi, \mathrm{type}_\Pi) \models_{A \setminus \{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma \implies (\Pi, \mathrm{type}_\Pi) \models_A \gamma.$$

Otherwise, each $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$ where $s \not\models \gamma$ is an *actor key compromise attack by* A *on* γ *in* (Π, type_Π). Note that a claim γ is trivially AKC resilient if there is an attack on γ without the $\mathsf{LKR}_{\mathsf{actor}}$ capability. 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 protocol's security claim, we say the protocol is *vulnerable to AKC attacks.*

We now revisit our examples. The KCI attack on protocol in Figure 1, which is described just before Example 1, represents 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 a nonce himself. Hence the protocol is trivially AKC resilient with respect to $\{\mathsf{LKR}_{\mathsf{actor}}\}$. We see that, in general, AKC resilience does not imply AKC security, which is the *absence* of 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 (AKCS). This is the reason we will primarily focus on AKCS: after all, we want to use protocols with no vulnerabilities, regardless if they are AKC-related. Another reason is that we can represent KCI resilience (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 the users' own long-term keys are compromised, they can still authenticate messages encrypted with established session keys. Although the notion of a KCI *attack* has remained informal and subjective, KCI *resilience* has been incorporated into different formal models based on a recurring 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.

In the following proposition, we prove that session-key secrecy during a key establishment handshake coincides with key agreement after a particular key confirmation step, even under AKC. The step requires a peer to apply any hash function unused in the handshake to the key and the relevant identities, and send the hash value to the actor as depicted in Figure 5. This shows how the standard notion of KCIR can be recast in our terms as both AKCS of secrecy and authentication.

One of the lemmas we use in the proof concerns terms t that have no proper accessible subterms. For certain sets S, t is unnecessary for inferring terms from S that do not contain t as a subterm. The second lemma helps us understand the options available to an adversary for inferring a given term: the term has been sent as an accessible subterm and the adversary managed to retrieve it, or he composed it himself. The proofs of both lemmas can be found in the appendix.

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 $S \vdash t$. Then $t \sqsubseteq_{\mathrm{acc}} S$, or every minimal-height derivation of t from S ends in a composition rule. ■

We now define the transformation in Figure 5, and prove that the adversary can violate agreement on the session key in the transformed protocol if and only if he knows the key in the initial one. We will instantiate all the parameters just before using the transformation.

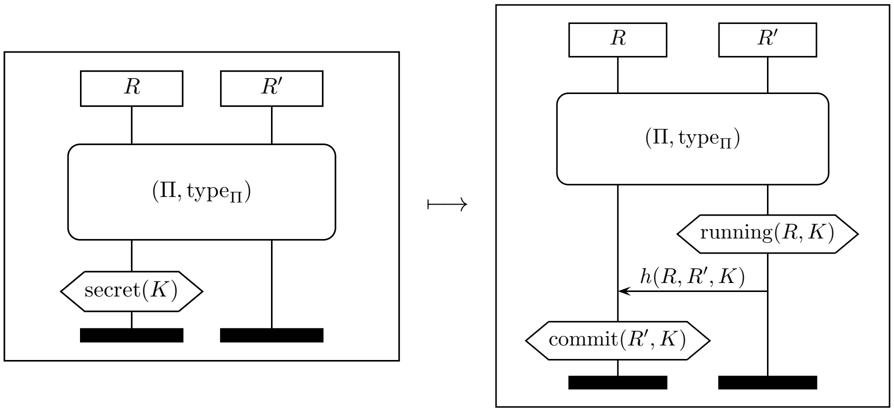


Fig. 5. KCIR as AKCS of secrecy and non-injective agreement

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

The term K' corresponds to the view from R' on the key K; usually, one of them is a variable and the other is 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. With the help of some typing assumptions, 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, $R, R' \in \text{dom}(\Pi)$ such that $R \neq R'$, and $\mathit{last}(\Pi(R)) = \texttt{claim}_{l_0}(R, \text{secret}, K)$ where $l_0 \in \text{Label}$ and $K \in \text{RoleTerm}$. Let $l_1, l_2 \in \text{Label}$ and $h \in \text{Func}$ all be unused in Π, where $l_1 \neq l_2$ and h does not occur in the set $\text{ran}(\text{type}_\Pi) \cup \text{ran}(\text{type}_{KC(\Pi)}(K'))$. Let A be an adversary. If $\text{type}_{KC(\Pi)} = \text{type}_\Pi$ and $(KC(\Pi), \text{type}_{KC(\Pi)})$ is a protocol,

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

IV. Achieving AKCS by Transformation

In this section, we show how to avoid AKC vulnerabilities during the protocol design stage. We achieve this by exploiting unilateral protocols whose security only depends on the long-term secret keys of the peers. To ensure that such keys are unavailable to the adversary, we restrict ourselves to protocols where no role sends out accessible asymmetric long-term secret keys.

A. Achieving AKCS of secrecy

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

Definition 13 (Tagging function τ_c): Let $c \in \text{Const}$. We define $\tau_c : \text{Term} \to \text{Term}$ for all $t, t_1, \ldots, t_n \in \text{Term}$ 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.

The following 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'. Its proof is given in Appendix A.

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

We require that no role is instructed to send its asymmetric long-term secret key in an accessible position. Therefore, since no adversary can *reveal* asymmetric long-term secret keys of peers, they can also never *infer* those keys. The lemma in which we prove this statement must be restricted if more powerful adversaries such as the ones in [6] are allowed.

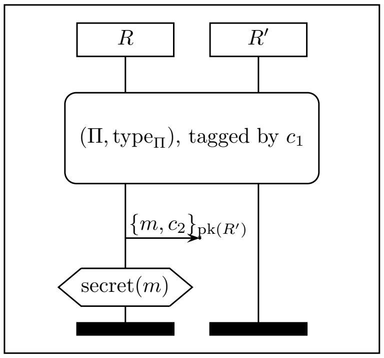
Lemma 9 (Peers' asymmetric secret keys not inferable): Let (Π, type_Π) be a protocol where no asymmetric long-term secret keys appear in accessible positions in send events. Let $R \in \text{dom}(\Pi)$, A an adversary, $s \in \text{RS}(\Pi, \text{type}_\Pi, R, A)$ a state, and $a \in \{\sigma_{s,Test}(R') : R' \in \text{dom}(\Pi) \setminus \{R\}\}$ an agent. Then $AK_s \nvdash \text{sk}(a)$ holds. ■

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

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

The transformation TS will only be useful if the protocol designer ensures that R can indeed send the added message. In other words, $(TS(\Pi), \text{type}_{TS(\Pi)})$ must be a protocol, i.e. $TS(\Pi)(R)$ must be well-formed for R.

Proposition 10 (Secrecy by asymmetric encryption): Let (Π, type_Π) be a protocol where no asymmetric long-term secret keys appear in accessible positions in send events. Let $R, R' \in \text{dom}(\Pi)$ where $R \neq R'$. Let $c_1, c_2 \in \text{Const}$,



R
R'
(Π, type_Π), tagged by c_1
$\{m, c_2\}_{\mathrm{pk}(R')}$
$\mathrm{secret}(m)$

Fig. 6. Transforming Π for secrecy of m

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

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 on the message:

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

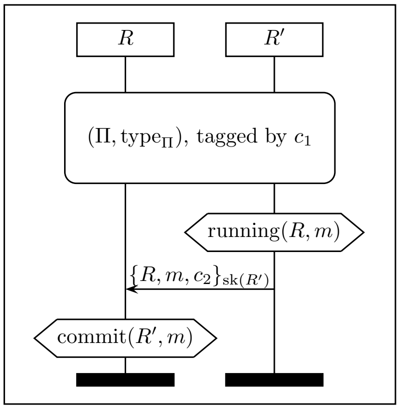
As before, m' is an arbitrary term meant to unify with the term m, and the transformation TA assumes that R' can indeed send the required signature. Therefore, we require that $(TA(\Pi), \mathrm{type}_{TA(\Pi)})$ is a protocol, i.e. that $TA(\Pi)(R')$ is a well-formed sequence of role events for R'.

The following proposition states that data agreement can be achieved by exchanging an additional signed message. 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 where no asymmetric long-term secret keys appear in accessible positions in $\mathtt{send}$ events. Let $R, R' \in \mathrm{dom}(\Pi)$ 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 $(TA(\Pi), \mathrm{type}_{TA(\Pi)})$ is a protocol and $\mathrm{type}_{TA(\Pi)} = \mathrm{type}_\Pi$, then we have $(TA(\Pi), \mathrm{type}_{TA(\Pi)}) \models_A \mathtt{claim}_l(R, \mathrm{commit}, R', m)$.

V. Impossibility results

It is conjectured in [9] that KCIR requires the use of asymmetric cryptography. We give a partial confirmation of



R
R'
(Π, type_Π), tagged by c_1
$\mathrm{running}(R, m)$
$\{R, m, c_2\}_{\mathrm{sk}(R')}$
$\mathrm{commit}(R', m)$

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

this conjecture: under weak assumptions on the protocol specification and the adversary model, the use of just symmetric cryptography and hashing cannot ensure AKCR for a large class of security properties. This class includes, for example, all authentication properties in Lowe's hierarchy [8]. We prove the result for aliveness and generalise it to all stronger claims.

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

- $\mathrm{pk}(S) \not\sqsubseteq \Pi(R)$ and $\mathrm{sk}(S) \not\sqsubseteq \Pi(R)$,
- if $\mathrm{k}(S, T) \sqsubseteq \Pi(R)$, then $S = R$ or $T = R$,
- there exists $n_x \in \mathrm{Fresh}$ such that $n_x^{\#rid_A} \in \mathrm{type}_\Pi(x)$, and
- if $n \sqsubseteq \Pi(R)$, n in $\Pi(R)$ appears first in a $\mathtt{send}$, in accessible positions only,

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

Definition 14 (At least as strong security claim): Let γ and γ' be security claims for $R \in \mathrm{Role}$. We say that γ is *at least as strong as* γ' if for all protocols (Π, type_Π) such that $R \in \mathrm{dom}(\Pi)$ and $\gamma, \gamma' \in \Pi(R)$, adversaries A, and reachable states $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, A)$, whenever instances of both γ and γ' are executed by $Test$ in tr_s and $s \models \gamma$, then $s \models \gamma'$.

For example, $\mathtt{claim}_l(R, \mathrm{commit}, R', t)$ is at least as strong as $\mathtt{claim}_{l'}(R, \mathrm{alive}, R')$.

Corollary 13 (Impossibility of authentication): Under the assumptions of Proposition 12, for every security claim γ that is at least as strong as $\mathrm{claim}_l(R, \mathrm{alive}, R')$ and occurs before it in $\Pi(R)$, if $(\Pi, \mathrm{type}_\Pi) \models_\emptyset \gamma$, then γ is not AKC resilient in (Π, type_Π) with respect to the $\{\mathsf{LKR}_{\mathsf{actor}}\}$ adversary.

Proof: Let $\gamma' = \mathrm{claim}_l(R, \mathrm{alive}, R')$. From Proposition 12, we have $(\Pi, \mathrm{type}_\Pi) \not\models_{\{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma'$. Therefore, we know there exists a state $s \in \mathrm{RS}(\Pi, \mathrm{type}_\Pi, R, \{\mathsf{LKR}_{\mathsf{actor}}\})$ such that $Test$ executed an instance of γ' in tr_s, but $s \not\models \gamma'$. Since γ appears before γ' in $\Pi(R)$, we know $Test$ also executed an instance of γ in tr_s. Since γ is at least as strong as γ', $s \not\models \gamma$. This implies $(\Pi, \mathrm{type}_\Pi) \not\models_{\{\mathsf{LKR}_{\mathsf{actor}}\}} \gamma$. Since $(\Pi, \mathrm{type}_\Pi) \models_\emptyset \gamma$ holds by assumption, we are done. ■

VI. Case studies

We have used the Scyther tool [10] 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 for the attack, are available at [11].

There are several ways to fix a vulnerable protocol, depending on its requirements and deployment status: (1) switch to a different mode of the protocol within its protocol suite, (2) use generic Propositions 10 and 11 verbatim, or (3) use a modified version of the transformations described in these propositions, perhaps to achieve slightly different security requirements, and prove the resulting protocol secure, e.g., with tool support. The approach listed as (2) is best suited for use in the protocol design stage. We give an application of (1) in the sections on TLS, and examples of (3) when discussing NSL and the CCITT X.509 protocols.

Note that Proposition 12 tells us what kind of changes are insufficient to ensure AKCS: we cannot achieve authentication under actor key compromise by employing just hashing and symmetric keys. However, adding one or both to a protocol that utilises other mechanisms may suffice. With this in mind, we propose practical fixes for each of the vulnerable protocols.

A. Needham-Schroeder-Lowe

In the presence of a Dolev-Yao adversary, the Needham-Schroeder-Lowe (NSL) protocol [3] 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 leaking 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 protocol 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 the adversary 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. [7]) and secrecy of $h'(na, nb)$, with respect to $\{\mathsf{LKR}_{\mathsf{actor}}\}$ and for both roles.

TABLE I
AUTOMATIC ANALYSIS RESULTS

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

B. CCITT X.509 family

We consider a family of protocols from the recommendations for the CCITT X.509 standard as modelled in the SPORE library [12]. The protocols are meant to enable secure and (mutually) authenticated access to a certificate directory. There are AKC attacks on the 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 must 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 leads to synchronisation [7] under AKC for both roles. The repaired protocol is successfully verified by Scyther:

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

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.

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

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 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 SSH Transport Layer protocol [13] is essentially a signed Diffie-Hellman key exchange. We used Scyther to verify the AKC security of session-key secrecy and synchronisation in the SSH Transport Layer protocol with respect to $\{\mathsf{LKR}_{\mathsf{actor}}, \mathsf{LKR}_{\mathsf{others}}\}$.

D. Mutually authenticated TLS

The TLS protocol [14] 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 mutual authentication before returning to unilateral authentication in the next section.

The mutually authenticated modes of TLS are typically used in, for example, specialised banking applications [15] and VPN access. 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 and 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. $S \to C: \quad SFIN = \{F'\}_{SERVERK}$

The four computed session keys are used to encrypt and authenticate the application data in subsequent communications.

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, which in turn is based on the secrecy of the server's long-term secret key $\mathrm{sk}(S)$ (see message 2).

Scyther finds 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 [14], 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. For the AKC attack we provide our script with the long-term secret key of the web server. 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 to run the attack can be downloaded from [11].

The simplest way to prevent the AKC attack is to switch to the mutually authenticated DHE_RSA 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 does not learn x or y, and therefore cannot construct the session key. In this case, we use the Tamarin prover [16] (for its more precise modelling of Diffie-Hellman exponentiation) to successfully verify that session key secrecy is AKC secure in the DHE_RSA mode of our TLS model.

Until now, the reason put forward for using the DHE modes of TLS instead of the RSA mode has been that they offer *Perfect Forward Secrecy* (PFS): 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_RSA over TLS-RSA. Namely, the AKC security implies that a server running TLS-DHE_RSA 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 authorisation 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 exchanges are similar to mutually authenticated TLS, except that the client does not send a certificate and does not send the so-called *CertificateVerify* message (message 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 when 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 based on 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 [17], or single sign-on protocols such as OAUTH [18] or SAML [19]. 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 authorisation protocol used. 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 compromises the secret, which enables AKC attacks against the above setup even when no clients are present. This is not the case in a weaker threat model, where the adversary learns the long-term secret key of the server through cryptanalysis but does not have access to the server's password store. In both of the above situations, switching to the mutually authenticated DHE_RSA mode of the TLS handshake provides the guarantee of AKCS of session-key secrecy to the server.

VII. Related work

The vast majority of related work has been on Key Compromise Impersonation in the domain of key establishment protocols. The first key compromise impersonation attack was described by Just and Vaudenay [5] in 1996. The first explicit, but informal definition of this notion, due to Blake-Wilson, Johnson and Menezes [4], 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 [4,5], researchers have examined concrete protocols or small classes of protocols, and classified KCI attacks [20,21,22,23,24,25]. 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 in the early informal definitions.

Starting with [26], researchers have considered the compromise of the actor's keys in computational [24,26,27] models and proved KCI resilience for 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 computational 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 [21,22], where they are classified as one of two types, depending on whether the responder authenticates the initiator. In [24], 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 [25], where KCI attacks by adversaries with a session-key reveal capability are classified according to how that capability is used in the attack. These attacks would be captured in our model by adding the session-key reveal capability from [6].

In [28], 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 [23]. 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 [29], it is proven that the DHE modes of TLS satisfy a monolithic security notion that implies KCI resilience. This is in line with our findings. In contrast, it is proven in [30] that all modes of TLS are secure in a weaker security model. This weaker security model does not capture AKC attacks; these proofs therefore do not contradict our AKC attacks on TLS. Note that Paulson's simplified model of TLS [31] 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 is that in Paulson's simplified version, the hash in the client's *CertificateVerify* message does not contain all previously transmitted data. However, this is not an actual attack on the TLS protocol.

In April 2014, a critical vulnerability [2] (CVE-2014-0160, also known as Heartbleed) was discovered in OpenSSL, versions 1.0.1–1.0.1f. The vulnerability allows an attacker to retrieve parts of an affected server's memory, potentially including the server's long-term secret keys. If the attacker obtains the keys, he can impersonate the compromised server. As we have shown, the attacker can additionally perform an

AKC attack on the server using TLS-RSA. Until the server changes its key pair and prevents further key leakage (by upgrading to a Heartbleed-unaffected implementation of TLS-RSA, e. g., OpenSSL 1.0.1g), it can avoid AKC attacks by switching to TLS-DHE_RSA.

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 help contain the effects of key compromise [1,2]. 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 existing, widely deployed protocols, we have introduced fixes that use the underlying structure of the protocols at hand. In comparison with using the generic transformations developed in this paper, this approach enables less intrusive fixes and 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 improving existing protocols and developing 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_RSA.

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] OpenSSL Security Advisory [07 Apr 2014], "TLS heartbeat read overrun (CVE-2014-0160)," https://www.openssl.org/news/secadv_20140407.txt.

[3] 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.

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

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

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

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

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

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

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

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

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

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

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

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

[16] 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.

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

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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/.

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

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

[29] 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.

[30] 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.

[31] L. C. Paulson, "Inductive analysis of the Internet Protocol TLS," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, pp. 332–351, Aug. 1999.

Appendix A
Proofs

A. Proof of Lemma 5

We first prove a lemma:

Lemma 14 (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 $S \setminus \{t\} \vdash t'$.

Proof: $\boxed{\Leftarrow}$ Trivial.

$\boxed{\Rightarrow}$ Let $t' \in \mathrm{Term}$ such that $t \not\sqsubseteq t'$. We prove by induction on n that for all $n \in \mathbb{N}_0$, 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 up to some $n \in \mathbb{N}_0$, the statement holds and that a $\vdash$-derivation tree for t' from S of height at most $n+1$ is given. We proceed with a case split on the last rule applied in that tree. If the last rule infers t' from $t_1, \dots, t_m$ by composition, assuming $t \sqsubseteq t_i$ for some $i \in \{1, \dots, m\}$, we would have $t \sqsubseteq t'$, which is a contradiction. Therefore, we can apply the inductive hypothesis and infer each of $t_1, \dots, t_m$ from $S \setminus \{t\}$ with trees of height at most n each. Note that we needed $t \not\sqsubseteq t'$ precisely for this and the base cases. 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.
  - $(t', t'') \sqsubseteq_{\text{acc}} S$: The term t has no proper accessible subterms, so from $(t', t'') \sqsubseteq_{\text{acc}} S$ we get $(t', t'') \sqsubseteq_{\text{acc}} S \setminus \{t\}$. Suppose $t \sqsubseteq (t', t'')$. By transitivity of $\sqsubseteq$, we get $t \sqsubseteq S \setminus \{t\}$, which contradicts our assumptions. We can now apply the inductive hypothesis to infer $S \setminus \{t\} \vdash_n (t, t')$. Hence $S \setminus \{t\} \vdash_{n+1} t'$.
  - $(t', t'') \not\sqsubseteq_{\text{acc}} S$: We can replace the derivation of (t', t'') with one of a minimal height, which is of height 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 inductive hypothesis implies $S \setminus \{t\} \vdash_{n-1} t'$.
- $S \vdash \{t'\}_{t''}$ and $S \vdash t''^{-1}$:
  - $\{t'\}_{t''} \sqsubseteq_{\text{acc}} S$: As above, we conclude $t \not\sqsubseteq \{t'\}_{t''}$. The inductive 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 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 $u \in \text{Term}$ such that $t'' = \text{pk}(u)$ and $t''^{-1} = \text{sk}(u)$. 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 inductive hypothesis to infer $S \setminus \{t\} \vdash_n t''^{-1}$.
  - $\{t'\}_{t''} \not\sqsubseteq_{\text{acc}} S$: 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 previous lemma, provided that it is done in the right order.

Proof of 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 14 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 15 (Inference of accessible subterms): Let $S \cup \{t\} \subseteq \text{Term}$ and $n \in \mathbb{N}_0$ such that $S \vdash_n t$. Then for all $t' \in \text{Term}$ such that $t' \sqsubseteq_{\text{acc}} t$, $S \vdash_n t'$ or $t' \sqsubseteq_{\text{acc}} S$.

Proof: We prove the lemma by induction on n. For $n = 0$, we have $t \in S$, so $t' \sqsubseteq_{\text{acc}} t$ implies $t' \sqsubseteq_{\text{acc}} S$. 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 inductive 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 $t' \sqsubseteq_{\text{acc}} S$. ■

Proof of Lemma 6: Assume $S \vdash t$. Then there is a derivation of t from S, so there is one of minimal height n. We are done if $n = 0$, because then $t \in S$, which implies $t \sqsubseteq_{\text{acc}} S$. Assume now that $n > 0$ and that some derivation of t from S of height n ends in a decomposition rule. The premises of that rule then imply that there is 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 15 we conclude $t \sqsubseteq_{\text{acc}} S$. ■

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 $\delta_L(\langle\rangle) = \langle\rangle$ and

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

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 $T = \{h(t_0, \dots, t_m) : t_0, \dots, t_m \in \text{Term}, m \in \mathbb{N}\}$, 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 $\texttt{recv}$ transitions are still enabled. Let s_n be a 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 $\texttt{send}$ 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 $AK_{s_n} \vdash \text{cont}(e)$.

Let $\tau = \sigma_{s,Test}(h(R, R', K)^{\#Test})$ for the rest of this proof. We still need to prove $AK_{s'} \vdash \sigma_{s',Test}(K^{\#Test})$, where the discussed instance of K, i.e. $\sigma_{s',Test}(K^{\#Test}) =$

$\sigma_{s,Test}(K^{\#Test})$, is a subterm of τ. To that end, 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 $\sqsubseteq$, the set $\{t : t \sqsubseteq \text{cont}(e')\}$ is the same as the set

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

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 get 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 occur 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 $tr_{s'}$. We know $th_s(Test) = \langle\rangle$, because $\texttt{claim}_{l_0}$ is the role's last step. For all $rid \in \text{dom}(th_s)$, we define $th_{s'}(rid)$:

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

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 $th_s(rid)$ 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 inductive 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, due to $S \nvdash t$ we have $t \neq t''$, which 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 $(rid, e) = (rid_{\mathcal{A}}, \textsf{LKR}(a))$, 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 our assumption on (Π, type_{Π}). 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 15 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 $AK_s \nvdash \sigma_{s,Test}(m^{\#Test})$. 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 inductive 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 $n^{\#Test}$ is accessible in $\texttt{recv}$ events in $tr_{s'}$ only inside $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$.

Assume that $n^{\#Test}$ occurs accessible outside the term $\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\text{pk}(R')})$ 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

$\sigma_{s,Test}(\{m^{\#Test}, c_2\}_{\mathrm{pk}(R')})$. However, from Lemma 9 we get $\mathrm{sk}(\sigma_{s,Test}(R')) \notin AK_s$. Therefore, $AK_s \nvdash n^{\#Test}$. From Lemma 8, we get $AK_s \nvdash \sigma_{s,Test}(m^{\#Test})$. ■

G. Proof of Proposition 11

Proof: Let $s \in \mathrm{RS}(TA(\Pi), \mathrm{type}_{TA(\Pi)}, R, A)$ and $(Test, \sigma_{s,Test}(\mathtt{claim}_l(R, \mathrm{commit}, R', m^{\#Test}))) \in tr_s$. We now prove that a run executed the corresponding running claim. Denote $t = \sigma_{s,Test}(\{R, m^{\#Test}, c_2\}_{\mathrm{sk}(R')})$. From the above, there exists $s' \in \mathrm{RS}(TA(\Pi), \mathrm{type}_{TA(\Pi)}, R, A)$ such that $s' \rightarrow^* s$ and

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

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

Without loss of generality, let (rid, e) be the first $\mathtt{send}$ with the above properties in $tr_{s'}$. Assume first that $\mathrm{label}(e) \neq l'$. Then e is an instance of a tagged step of Π, i.e. there is a unique $t' \in \mathrm{RoleTerm}$ such that $\sigma_{s',rid}(t'^{\#rid}) = \mathrm{cont}(e)$ and $\mathtt{send}_{\mathrm{label}(e)}(\cdot, \cdot, t') \in \tau_{c_1}|_S(\Pi(\mathrm{role}_{s'}(rid)))$. We know that t cannot occur in $\mathrm{cont}(e)$ as a subterm of any instantiated variables of t', since Lemma 15 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, $\mathrm{label}(e) = l'$. Hence $\mathrm{role}_{s'}(rid) = R'$, $\sigma_{s',rid}(\mathtt{claim}_l(R', \mathrm{running}, R, m')^{\#rid}) \in tr_{s'}$ 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 a lemma that states that we can infer a term from all its subterms that are atomic or long-term keys.

Lemma 16 (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 inductive hypothesis implies $S \vdash t_1, \ldots, t_n$. We can then use the corresponding composition rule to get t. ■

Proof of Proposition 12: Let $a, b \in \mathrm{Agent}$ such that $a \neq b$ and define the substitution τ as follows:

$$\tau(x) = \begin{cases} a, & \text{if } x = R, \\ b, & \text{if } x \in \mathrm{Role} \setminus \{R\}, \\ n_x^{\#rid_{\mathcal{A}}}, & \text{if } x \in \mathrm{Var}. \end{cases}$$

By the third assumption in the proposition statement, $\tau \in TS(\Pi, \mathrm{type}_\Pi)$. If the length of $\Pi(R)$ is k for some $k \in \mathbb{N}$, we define:

$$\begin{aligned} seq &= \langle (Test, \tau(\Pi(R)^{\#Test})_1), \ldots, (Test, \tau(\Pi(R)^{\#Test})_k) \rangle \\ s &= (\langle (rid_{\mathcal{A}}, \mathsf{LKR}(a)) \rangle.seq, AK_0 \cup \mathrm{LTK}(a) \cup \\ &\quad (\tau(\Pi(R)^{\#Test}) \downharpoonleft \mathtt{send}), Test \rightarrow \langle\rangle, Test \mapsto \tau). \end{aligned}$$

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 sent nonces just after they are first sent, and
- the adversary can infer the contents of all $\mathtt{recv}$ 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 and that no nonces have yet been sent. Let $m \in \mathbb{N}$ such that $m > 1$, s_m a state reached after m transitions from s_0, and $e \in \mathrm{RunEvent}$ the next event. Let K be any long-term key such that $K \sqsubseteq \mathrm{cont}(e)$. We want to prove that $K \in AK_{s_m}$. Since τ assigned nonces local to $\mathrm{rid}_{\mathcal{A}}$ to variables in $\Pi(R)$, 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 $K = \sigma_{s_m,Test}(t^{\#Test}) = \tau(t^{\#Test})$. 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 $AK_{s_1} \subseteq AK_{s_m}$, we get $K \in AK_{s_m}$.

Suppose now that $(Test, e)$ is the first $\mathtt{send}$ for $n^{\#Test}$ where $n \in \mathrm{Fresh}$. By the inductive hypothesis, we know that for all $n'^{\#Test}$ sent strictly before $(Test, e)$ in tr_{s_m}, we have $AK_{s_m} \vdash n'^{\#Test}$. By the fourth assumption in the proposition statement, we know all other nonces local to $Test$, i.e. those that appear for the first time in $(Test, e)$, appear in $\mathrm{cont}(e)$ in accessible positions only. Therefore, AK_{s_m} infers all inaccessible atomic subterms and long-term keys in $\mathrm{cont}(e)$. By induction on $n^{\#Test} \sqsubseteq_{\mathrm{acc}} \mathrm{cont}(e)$, we prove $AK_{s_m} \vdash n^{\#Test}$. The basis, where $n^{\#Test} = \mathrm{cont}(e)$, is trivial. Assume now that $n^{\#Test} \neq \mathrm{cont}(e)$, and let $\mathrm{cont}(e) = \{t_1\}_{t_2}$ first. We want to conclude $AK_{s_m} \vdash t_2^{-1}$. We know that all atomic subterms and long-term keys in t_2 are inferable by AK_{s_m}, so by Lemma 16 we have $S \vdash t_2$. By construction of seq, we have $t_2 = t_2^{-1}$, so $S \vdash t_2^{-1}$. We can therefore apply the decryption rule to get $S \vdash t_1$. Finally, if $t = (t_1, t_2)$, without loss of generality we can assume that $n^{\#Test} \sqsubseteq_{\mathrm{acc}} t_1$ and use the unpairing rule to get $AK_{s_m} \vdash t_1$. In any case, the inductive hypothesis for $AK_{s_m} \vdash t_1$ gives us $AK_{s_m} \vdash n^{\#Test}$, which proves the first statement.

Now assume $\mathrm{evtype}(e) = \mathtt{recv}$. We need to check that e is enabled, i.e. that $AK_{s_m} \vdash \mathrm{cont}(e)$ holds. All nonces local to $Test$ to be received in $\mathrm{cont}(e)$ have been sent previously by $Test$, so we know they are inferable by AK_{s_m}. Additionally, AK_{s_m} already contains all long-term keys in $\mathrm{cont}(e)$ and AK_{s_0} contains all nonces local to $\mathrm{rid}_{\mathcal{A}}$ and the names of all agents, so we can apply Lemma 16 to get $AK_{s_m} \vdash \mathrm{cont}(e)$. ■