Degrees of Security: Protocol Guarantees in the Face of Compromising Adversaries

David Basin and Cas Cremers[⋆]

Department of Computer Science, ETH Zurich

Abstract. We present a symbolic framework, based on a modular operational semantics, for formalizing different notions of compromise relevant for the analysis of cryptographic protocols. The framework's rules can be combined in different ways to specify different adversary capabilities, capturing different practically-relevant notions of key and state compromise. We have extended an existing security-protocol analysis tool, Scyther, with our adversary models. This is the first tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of state-reveal queries. We also introduce the concept of a protocol-security hierarchy, which classifies the relative strength of protocols against different forms of compromise. In case studies, we use Scyther to automatically construct protocol-security hierarchies that refine and correct relationships between protocols previously reported in the cryptographic literature.

1 Introduction

Compromise is a fact of life! Keys are leaked or broken. Memory and disks may be read or subject to side-channel attacks. Hardware protection may fail. Security-protocol designers are well aware of this and many protocols are designed to work in the face of various forms of corruption. For example, Diffie-Hellman key agreement, which uses digital signatures to authenticate the exchanged half-keys, provides perfect-forward secrecy [1,2]: the resulting session key remains secret even when the long-term signature keys are later compromised by an adversary.

In this paper, we survey and extend recent results of ours [3] on a symbolic framework for modeling and reasoning about security protocols in the presence of adversaries with a wide range of compromise capabilities. Our framework is inspired by the models developed in the computational setting, e. g. [4–8], where principals may be selectively corrupted during protocol execution. We reinterpret these computational models in a uniform way, build tool support for analyzing protocols with respect to these models, and gain new insights on the relationships between different models and protocols.

Our starting point is an operational semantics for security protocols. We parameterize this semantics by a set of rules that formalize adversarial capabilities.

[⋆] This work was supported by ETH Research Grant ETH-30 09-3 and the Hasler Foundation, ManCom, Grant 2071.

These rules capture three dimensions of compromise: *whose* data is compromised, *which* kind of data it is, and *when* the compromise occurs. These dimensions are fundamental and different rule combinations formalize symbolic analogs of different practically-relevant notions of key and state compromise from the computational setting. The operational semantics gives rise, in the standard way, to a notion of correctness with respect to state and trace-based security properties.

Symbolic and computational approaches have addressed the problem of formalizing adversary compromise to different degrees. Most symbolic formalisms are based on the Dolev-Yao model and offer only a limited view of compromise: either principals are honest from the start and always keep their secrets to themselves or they are completely malicious and always under adversarial control. In contrast, numerous computational models have been proposed that formalize different notions of adversary compromise in the setting of key-exchange protocols, e.g., the models of Canetti and Krawczyk [4,9], Shoup [6], Bellare et al. [10–12], Katz and Yung [13], LaMacchia et al. [5], and Bresson and Manulis [7], to name but a few. These models are usually incomparable due to (often minor) differences in their adversarial notions, the execution models, and security property specifics. Moreover, they are generally presented in a monolithic way, where all parts are intertwined and it is difficult to separate these notions.

Our framework has a number of advantages over alternative approaches, both symbolic and computational. First, it cleanly separates the basic operational semantics of protocols, the adversary rules, and the security properties. This makes it simple to tailor a model to the scenario at hand by selecting appropriate rules. For example, we can reason about the security guarantees provided when cryptographic protocol implementations mix the use of cryptographic co-processors for the secure storage of long-term secrets with the computation of intermediate results in less-secure main memory for efficiency reasons. Moreover, as we will see, it is easy to define new security properties in a modular way.

Second, our framework directly lends itself to protocol analysis and we have extended the Scyther tool [14] to support our framework. This is the first tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries that can reveal agents' local state.

Finally, we introduce the concept of a protocol-security hierarchy, in which protocols are classified by their relative strength against different forms of adversary compromise. Protocol-security hierarchies can be automatically constructed by Scyther. As case studies, we construct protocol-security hierarchies that refine and correct relationships reported in the cryptographic literature. This shows that symbolic methods can be effectively used for analyses that were previously possible only using a manual computational analysis.

Organization. We present our framework in Section 2. In Section 3, we use it to construct protocol-security hierarchies. Afterwards, in Section 4, we prove general results relating models and properties, which aid the construction of such hierarchies. Finally, we draw conclusions in Section 5.

2 Compromising Adversary Model

We define an operational semantics that is modular with respect to the adversary's capabilities. Our framework is compatible with the majority of existing semantics for security protocols, including trace and strand-space semantics. We have kept our execution model minimal to focus on the adversary rules. However, it would be straightforward to incorporate a more elaborate execution model, e. g., with control-flow commands.

Notational preliminaries.

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

We write $[t_0, \ldots, t_n \ / \ x_0, \ldots, x_n] \in \mathcal{S}ub$ to denote the substitution of t_i for x_i, for $0 \leq i \leq n$. We extend the functions dom and ran to substitutions. We write $\sigma \cup \sigma'$ to denote the union of two substitutions, which is defined when $dom(\sigma) \cap dom(\sigma') = \emptyset$, and write $\sigma(t)$ for the application of the substitution σ to t. Finally, for R a binary relation, R^* denotes its reflexive transitive closure.

2.1 Terms and events

We assume given the infinite sets *Agent*, *Role*, *Fresh*, *Var*, *Func*, and *TID* of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, function names, and thread identifiers. We assume that *TID* contains two distinguished thread identifiers, *Test* and $tid_{\mathcal{A}}$. These identifiers single out a distinguished "point of view" thread of an arbitrary agent and an adversary thread, respectively.

To bind local terms, such as freshly generated terms or local variables, to a protocol role instance (thread), we write $T\sharp tid$. This denotes that the term T is local to the protocol role instance identified by tid.

Definition 1. *Terms*

$$\begin{aligned} Term ::= & Agent \mid Role \mid Fresh \mid Var \mid Fresh\sharp TID \mid Var\sharp TID \\ & \mid (Term, Term) \mid pk(Term) \mid sk(Term) \mid k(Term, Term) \\ & \mid \{\!| Term |\!\}^a_{Term} \mid \{\!| Term |\!\}^s_{Term} \mid Func(Term^*) \end{aligned}$$

For each $X, Y \in Agent$, $sk(X)$ denotes the long-term private key, $pk(X)$ denotes the long-term public key, and $k(X, Y)$ denotes the long-term symmetric key shared between X and Y. Moreover, $\{\!| t_1 |\!\}^a_{t_2}$ denotes the asymmetric encryption (for public keys) or the digital signature (for signing keys) of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^s_{t_2}$ denotes symmetric encryption. The set *Func* is used to model other cryptographic functions, such as hash functions. Freshly generated terms and variables are assumed to be local to a thread (an instance of a role).

Depending on the protocol analyzed, we assume that symmetric or asymmetric long-term keys have been distributed prior to protocol execution. We assume the existence of an inverse function on terms, where t^{-1} denotes the inverse key of t. We have that $pk(X)^{-1} = sk(X)$ and $sk(X)^{-1} = pk(X)$ for all $X \in \mathit{Agent}$, and $t^{-1} = t$ for all other terms t.

We define a binary relation $\vdash$, where $M \vdash t$ denotes that the term t can be inferred from the set of terms M. Let $t_0, \ldots, t_n \in \mathit{Term}$ and let $f \in \mathit{Func}$. We define $\vdash$ as the smallest relation satisfying:

$$\begin{array}{rl}
t \in M \Rightarrow M \vdash t & M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash (t_1, t_2) \\
M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2} \wedge M \vdash t_2 \Rightarrow M \vdash t_1 & M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2} \\
M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2} \wedge M \vdash (t_2)^{-1} \Rightarrow M \vdash t_1 & M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2} \\
 & \bigwedge_{0 \le i \le n} M \vdash t_i \Rightarrow M \vdash f(t_0, \ldots, t_n)
\end{array}$$

Subterms t of a term t', written $t \sqsubseteq t'$, are defined as the syntactic subterms of t', e. g., $t_1 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2}$ and $t_2 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2}$. We write $FV(t)$ for the free variables of t, where $FV(t) = \{t' \mid t' \sqsubseteq t\} \cap \big(\mathit{Var} \cup \{v \sharp tid \mid v \in \mathit{Var} \wedge tid \in \mathit{TID}\}\big)$.

Definition 2. *Events*

$$\begin{array}{rl}
\mathit{AgentEvent} ::= & \mathsf{create}(\mathit{Role}, \mathit{Agent}) \mid \mathsf{send}(\mathit{Term}) \mid \mathsf{recv}(\mathit{Term}) \\
 & \mid \mathsf{generate}(\mathcal{P}(\mathit{Fresh})) \mid \mathsf{state}(\mathcal{P}(\mathit{Term})) \mid \mathsf{sessionkeys}(\mathcal{P}(\mathit{Term})) \\
\mathit{AdversaryEvent} ::= & \mathsf{LKR}(\mathit{Agent}) \mid \mathsf{SKR}(\mathit{TID}) \mid \mathsf{SR}(\mathit{TID}) \mid \mathsf{RNR}(\mathit{TID}) \\
\mathit{Event} ::= & \mathit{AgentEvent} \mid \mathit{AdversaryEvent}
\end{array}$$

We explain the interpretation of the agent and adversary events shortly. Here we simply note that the first three agent events are standard: starting a thread, sending a message, and receiving a message. The message in the send and receive events does not include explicit sender or recipient fields although, if desired, they can be given as subterms of the message. The last three agent events tag state information, which can possibly be compromised by the adversary. The four adversary events specify which information the adversary compromises. These events can occur any time during protocol execution and correspond to different kinds of *adversary queries* from computational models. All adversary events are executed in the single adversary thread $tid_{\mathcal{A}}$.

2.2 Protocols and threads

A protocol is a partial function from role names to event sequences, i. e., $\mathit{Protocol} : \mathit{Role} \nrightarrow \mathit{AgentEvent}^*$. We require that no thread identifiers occur as subterms of events in a protocol definition.

Example 1 (Simple protocol). Let $\{\mathrm{Init}, \mathrm{Resp}\} \subseteq \mathit{Role}$, $\mathit{key} \in \mathit{Fresh}$, and $x \in \mathit{Var}$. We define the simple protocol SP as follows.

$$\begin{aligned}
\mathrm{SP}(\mathrm{Init}) = \langle &\mathsf{generate}(\{\mathit{key}\}), \mathsf{state}(\{\mathit{key}, \{\!|\, \mathrm{Resp}, \mathit{key}\,|\!\}^a_{sk(\mathrm{Init})}\}),\\
&\mathsf{send}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \{\!|\, \mathrm{Resp}, \mathit{key}\,|\!\}^a_{sk(\mathrm{Init})}\,|\!\}^a_{pk(\mathrm{Resp})}), \mathsf{sessionkeys}(\{\mathit{key}\})\rangle\\
\mathrm{SP}(\mathrm{Resp}) = \langle &\mathsf{recv}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \{\!|\, \mathrm{Resp}, x\,|\!\}^a_{sk(\mathrm{Init})}\,|\!\}^a_{pk(\mathrm{Resp})}),\\
&\mathsf{state}(\{x, \{\!|\, \mathrm{Resp}, x\,|\!\}^a_{sk(\mathrm{Init})}\}), \mathsf{sessionkeys}(\{x\})\rangle
\end{aligned}$$

Here, the initiator generates a key and sends it (together with the responder name) signed and encrypted, along with the initiator and responder names. The recipient expects to receive a message of this form. The additional events mark session keys and state information. The state information is implementation-dependent and marks which parts of the state are stored at a lower protection level than the long-term private keys. The state information in SP corresponds to, e. g., implementations that use a hardware security module for encryption and signing and perform all other computations in ordinary memory.

Protocols are executed by agents who execute roles, thereby instantiating role names with agent names. Agents may execute each role multiple times. Each instance of a role is called a *thread*. We distinguish between the fresh terms and variables of each thread by assigning them unique names, using the function $\mathit{localize} : \mathit{TID} \to \mathcal{S}\mathit{ub}$, defined as $\mathit{localize}(\mathit{tid}) = \bigcup_{cv \in \mathit{Fresh} \cup \mathit{Var}} [cv \sharp \mathit{tid} \,/\, cv]$. Using $\mathit{localize}$, we define a function $\mathit{thread} : (\mathit{AgentEvent}^* \times \mathit{TID} \times \mathcal{S}\mathit{ub}) \to \mathit{AgentEvent}^*$ that yields the sequence of agent events that may occur in a thread.

Definition 3 (Thread). *Let l be a sequence of events, $\mathit{tid} \in \mathit{TID}$, and let σ be a substitution. Then $\mathit{thread}(l, \mathit{tid}, \sigma) = \sigma(\mathit{localize}(\mathit{tid})(l))$.*

Example 2. Let $\{A, B\} \subseteq \mathit{Agent}$. For a thread $t_1 \in \mathit{TID}$ performing the Init role from Example 1, we have $\mathit{localize}(t_1)(\mathit{key}) = \mathit{key}\sharp t_1$ and

$$\begin{aligned}
&\mathit{thread}(\mathrm{SP}(\mathrm{Init}), t_1, [A, B \,/\, \mathrm{Init}, \mathrm{Resp}]) =\\
&\quad\langle \mathsf{generate}(\{\mathit{key}\sharp t_1\}), \mathsf{state}(\{\mathit{key}\sharp t_1, \{\!|\, B, \mathit{key}\sharp t_1\,|\!\}^a_{sk(A)}\}),\\
&\quad\ \mathsf{send}(A, B, \{\!|\, \{\!|\, B, \mathit{key}\sharp t_1\,|\!\}^a_{sk(A)}\,|\!\}^a_{pk(B)}), \mathsf{sessionkeys}(\{\mathit{key}\sharp t_1\})\rangle\,.
\end{aligned}$$

Test thread. When verifying security properties, we will focus on a particular thread. In the computational setting, this is the thread where the adversary performs a so-called *test query*. In the same spirit, we call the thread under consideration the *test thread*, with the corresponding thread identifier Test. For the test thread, the substitution of role names by agent names, and all free variables by terms, is given by σ_{Test} and the role is given by R_{Test}. For example, if the test thread is performed by Alice in the role of the initiator, trying to talk to Bob, we have that $R_{\mathit{Test}} = \mathrm{Init}$ and $\sigma_{\mathit{Test}} = [\mathrm{Alice}, \mathrm{Bob} \,/\, \mathrm{Init}, \mathrm{Resp}]$.

2.3 Execution model

We define the set *Trace* as $(\mathit{TID} \times \mathit{Event})^*$, representing possible execution histories. The state of our system is a four-tuple $(\mathit{tr}, \mathit{IK}, \mathit{th}, \sigma_{\mathit{Test}}) \in \mathit{Trace} \times$

$\mathcal{P}(\mathit{Term}) \times (\mathit{TID} \rightharpoonup \mathit{Event}^*) \times \mathcal{S}ub$, whose components are (1) a trace tr, (2) the adversary's knowledge IK, (3) a partial function th mapping thread identifiers of initiated threads to sequences of events, and (4) the role to agent and variable assignments of the test thread. We include the trace as part of the state to facilitate defining the partner function later.

Definition 4 **($\mathcal{T}est\mathcal{S}ub_P$).** *Given a protocol P, we define the set of* test substitutions *$\mathcal{T}est\mathcal{S}ub_P$ as the set of ground substitutions σ_{Test} such that $dom(\sigma_{Test}) = dom(P) \cup \{v\sharp Test \mid v \in Var\}$ and $\forall r \in dom(P).\ \sigma_{Test}(r) \in Agent$.*

For P a protocol, the set of initial system states $IS(P)$ is defined as

$$IS(P) = \bigcup_{\sigma_{Test} \in \mathcal{T}est\mathcal{S}ub_P} \big\{ (\langle\rangle, Agent \cup \{pk(a) \mid a \in Agent\}, \emptyset, \sigma_{Test}) \big\}.$$

In contrast to Dolev-Yao models, the initial adversary knowledge does not include any long-term secret keys. The adversary may learn these from long-term key reveal (LKR) events.

The semantics of a protocol $P \in Protocol$ is defined by a transition system that combines the execution-model rules from Fig. 1 with a set of adversary rules from Fig. 2. We first present the execution-model rules.

The create rule starts a new instance of a protocol role R (a *thread*). A fresh thread identifier tid is assigned to the thread, thereby distinguishing it from existing threads, the adversary thread, and the test thread. The rule takes the protocol P as a parameter. The role names of P, which can occur in events associated with the role, are replaced by agent names by the substitution σ. Similarly, the createTest rule starts the test thread. However, instead of choosing an arbitrary role, it takes an additional parameter R_{Test}, which represents the test role and will be instantiated in the definition of the transition relation in Def. 7. Additionally, instead of choosing an arbitrary σ, the test substitution σ_{Test} is used.

The send rule sends a message m to the network. In contrast, the receive rule accepts messages from the network that match the pattern pt, where pt is a term that may contain free variables. The resulting substitution σ is applied to the remaining protocol steps l.

The last three rules support our adversary rules, given shortly. The generate rule marks the fresh terms that have been generated,[1] the state rule marks the current local state, and the sessionkeys rule marks a set of terms as session keys.

Auxiliary functions. We define the long-term secret keys of an agent a as

$$LongTermKeys(a) = \{sk(a)\} \cup \bigcup_{b \in Agent} \{k(a,b), k(b,a)\}\,.$$

[1] Note that this rule need not ensure that fresh terms are unique. The function *thread* maps freshly generated terms c to $c\sharp tid$ in the thread tid, ensuring uniqueness.

$$\frac{R \in dom(P) \quad dom(\sigma) = Role \quad ran(\sigma) \subseteq Agent \quad tid \notin (dom(th) \cup \{tid_{\mathcal{A}}, Test\})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{create}(R, \sigma(R)))\rangle, IK, th[thread(P(R), tid, \sigma) \hookleftarrow tid], \sigma_{Test})}[\mathsf{create}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad Test \notin dom(th)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(Test, \mathsf{create}(R_{Test}, a))\rangle, IK, th[thread(P(R_{Test}), Test, \sigma_{Test}) \hookleftarrow Test], \sigma_{Test})}[\mathsf{createTest}]$$

$$\frac{th(tid) = \langle\mathsf{send}(m)\rangle\hat{\ }l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{send}(m))\rangle, IK \cup \{m\}, th[l \hookleftarrow tid], \sigma_{Test})}[\mathsf{send}]$$

$$\frac{th(tid) = \langle\mathsf{recv}(pt)\rangle\hat{\ }l \qquad IK \vdash \sigma(pt) \qquad dom(\sigma) = FV(pt)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{recv}(\sigma(pt)))\rangle, IK, th[\sigma(l) \hookleftarrow tid], \sigma_{Test})}[\mathsf{recv}]$$

$$\frac{th(tid) = \langle\mathsf{generate}(M)\rangle\hat{\ }l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{generate}(M))\rangle, IK, th[l \hookleftarrow tid], \sigma_{Test})}[\mathsf{generate}]$$

$$\frac{th(tid) = \langle\mathsf{state}(M)\rangle\hat{\ }l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{state}(M))\rangle, IK, th[l \hookleftarrow tid], \sigma_{Test})}[\mathsf{state}]$$

$$\frac{th(tid) = \langle\mathsf{sessionkeys}(M)\rangle\hat{\ }l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid, \mathsf{sessionkeys}(M))\rangle, IK, th[l \hookleftarrow tid], \sigma_{Test})}[\mathsf{sessionkeys}]$$

Fig. 1. Execution-model rules

For traces, we define an operator $\downarrow$ that projects traces on events belonging to a particular thread identifier. For all tid, tid', and tr, we define $\langle\rangle \downarrow tid = \langle\rangle$ and

$$(\langle(tid', e)\rangle\hat{\ }tr) \downarrow tid = \begin{cases} \langle e\rangle\hat{\ }(tr \downarrow tid) & \text{if } tid = tid', \text{ and} \\ tr \downarrow tid & \text{otherwise.} \end{cases}$$

Similarly, for event sequences, the operator $\downharpoonright$ selects the contents of events of a particular type. For all $\mathsf{evtype} \in \{\mathsf{create}, \mathsf{send}, \mathsf{recv}, \mathsf{generate}, \mathsf{state}, \mathsf{sessionkeys}\}$, we define $\langle\rangle \downharpoonright \mathsf{evtype} = \langle\rangle$ and

$$(\langle e\rangle\hat{\ }l) \downharpoonright \mathsf{evtype} = \begin{cases} \langle m\rangle\hat{\ }(l \downharpoonright \mathsf{evtype}) & \text{if } e = \mathsf{evtype}(m), \text{ and} \\ l \downharpoonright \mathsf{evtype} & \text{otherwise.} \end{cases}$$

During protocol execution, the test thread may intentionally share some of its short-term secrets with other threads, such as a session key. Hence some adversary rules require distinguishing between the intended *partner threads* and other threads. There exist many notions of partnering in the literature. In general, we use partnering based on matching histories for protocols with two roles, as defined below.

Definition 5 (Matching histories). *For sequences of events l and l', we define* $\mathrm{MH}(l, l') \equiv \big(l \downharpoonright \mathsf{recv} = l' \downharpoonright \mathsf{send}\big) \ \wedge \ \big(l \downharpoonright \mathsf{send} = l' \downharpoonright \mathsf{recv}\big)$.

Our partnering definition is parameterized over the protocol P and the test role R_{Test}. These parameters are later instantiated in the transition-system definition.

Definition 6 (Partnering). *Let R be the non-test role, i. e., $R \in dom(P)$ and $R \neq R_{Test}$. For tr a trace,* $Partner(tr, \sigma_{Test}) = \big\{tid \ \big| \ tid \neq Test \wedge \big(\exists a.\mathsf{create}(R, a) \in tr \downarrow tid\big) \wedge \exists l \,.\, \mathrm{MH}(\sigma_{Test}(P(R_{Test})), (tr \downarrow tid)\hat{\ }l)\big\}$.

A thread *tid* is a partner iff (1) *tid* is not *Test*, (2) *tid* performs the role different from *Test*'s role, and (3) *tid*'s history matches the *Test* thread (for $l = \langle\rangle$) or the thread may be completed to a matching one (for $l \neq \langle\rangle$).

2.4 Adversary-compromise rules

We define the adversary-compromise rules in Fig. 2. They factor the security definitions from the cryptographic protocol literature along three dimensions of adversarial compromise: *which* kind of data is compromised, *whose* data it is, and *when* the compromise occurs.

Compromise of long-term keys. The first four rules model the compromise of an agent a's long-term keys, represented by the long-term key reveal event $\mathsf{LKR}(a)$. In traditional Dolev-Yao models, this event occurs implicitly for dishonest agents before the honest agents start their threads.

The $\mathsf{LKR}_{\mathsf{others}}$ rule formalizes the adversary capability used in the symbolic analysis of security protocols since Lowe's Needham-Schroeder attack [15]: the adversary can learn the long-term keys of any agent a that is not an intended partner of the test thread. Hence, if the test thread is performed by Alice, communicating with Bob, the adversary can learn, e. g., Charlie's long-term key.

The $\mathsf{LKR}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term key of the agent executing the test thread (also called the *actor*). The intuition is that a protocol may still function as long as the long-term keys of the other partners are not revealed. This rule allows the adversary to perform so-called Key Compromise Impersonation attacks [8]. The rule's second premise is required because our model allows agents to communicate with themselves.

The $\mathsf{LKR}_{\mathsf{after}}$ and $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rules restrict when the compromise may occur. In particular, they allow the compromise of long-term keys only after the test thread has finished, captured by the premise $th(Test) = \langle\rangle$. This is the sole premise of $\mathsf{LKR}_{\mathsf{after}}$. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{after}}$, it is said to satisfy Perfect Forward Secrecy (PFS) [1,2]. $\mathsf{LKR}_{\mathsf{aftercorrect}}$ has the additional premise that a finished partner thread must exist for the test thread. This condition stems from [9] and excludes the adversary from both inserting fake messages during protocol execution and learning the key of the involved agents later. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{aftercorrect}}$, it is said to satisfy weak Perfect Forward Secrecy (wPFS). This property is motivated by a class of protocols given in [9] whose members fail to satisfy PFS, although some satisfy this weaker property.

Compromise of short-term data. The three remaining adversary rules correspond to the compromise of short-term data, that is, data local to a specific thread. Whereas we assumed a long-term key compromise reveals *all* long-term keys of an agent, we differentiate here between the different kinds of local data. Because we assume that local data does not exist before or after a session, we can ignore the temporal dimension. We differentiate between three kinds of local data: *randomness*, *session keys*, and *other local data* such as the results of intermediate

$$\frac{a \notin \{\sigma_{Test}(R) \mid R \in dom(P)\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR_{others}}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad a \notin \{\sigma_{Test}(R) \mid R \in dom(P) \setminus \{R_{Test}\}\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR_{actor}}]$$

$$\frac{th(Test) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR_{after}}]$$

$$\frac{th(Test) = \langle\rangle \qquad tid \in Partner(tr, \sigma_{Test}) \qquad th(tid) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR_{aftercorrect}}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{SKR}(tid))\rangle, IK \cup union((tr \downarrow tid) \downharpoonleft \mathsf{sessionkeys}), th, \sigma_{Test})}[\mathsf{SKR}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test}) \qquad th(tid) \neq \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{SR}(tid))\rangle, IK \cup last((tr \downarrow tid) \downharpoonleft \mathsf{state}), th, \sigma_{Test})}[\mathsf{SR}]$$

$$\frac{}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr\hat{\ }\langle(tid_{\mathcal{A}}, \mathsf{RNR}(tid))\rangle, IK \cup union((tr \downarrow tid) \downharpoonleft \mathsf{generate}), th, \sigma_{Test})}[\mathsf{RNR}]$$

Fig. 2. Adversary-compromise rules

computations. The notion that the adversary may learn the randomness used in a protocol stems from [5]. Considering adversaries that can reveal session keys, e. g., by cryptanalysis, is found in many works, such as [12]. An adversary capable of revealing the local state was described in [4].

In our adversary-compromise models, the session-key reveal event $\mathsf{SKR}(tid)$ and state reveal event $\mathsf{SR}(tid)$ indicate that the adversary gains access to the session key or, respectively, the local state of the thread *tid*. These are marked respectively by the sessionkeys and state events.

The contents of the state change over time and are erased when the thread ends. This is reflected in the SR rule by the *last* state marker for the state contents and the third premise requiring that the thread *tid* has not ended. The random number reveal event $\mathsf{RNR}(tid)$ indicates that the adversary learns the random numbers generated in the thread *tid*.

The rules SKR and SR allow for the compromise of session keys and the contents of a thread's local state. Their premise is that the compromised thread is not a partner thread. In contrast, the premise of the RNR rule allows for the compromise of all threads, including the partner threads. This rule stems from [5], where it is shown that it is possible to construct protocols that are correct in the presence of an adversary capable of RNR.

For protocols that establish a session key, we assume the session key is shared by all partners and should be secret: revealing it trivially violates the protocols' security. Hence the rules disallow the compromise of session keys of the test or partner threads. Similarly, our basic rule set does not contain a rule for the compromise of other local data of the partners. Including such a rule is straightforward. However it is unclear whether any protocol would be correct with respect to such an adversary.

We call each subset of the set of adversary rules from Fig. 2 an *adversary-compromise model.*

2.5 Transition relation and security properties

Given a protocol and an adversary-compromise model, we define the possible protocol behaviors as a set of reachable states.

Definition 7 (Transition relation and reachable states). *Let P be a protocol, Adv an adversary-compromise model, and R_{Test} a role. We define a* transition relation $\rightarrow_{P,Adv,R_{Test}}$ *from the execution-model rules from Fig. 1 and the rules in Adv. The variables P, Adv, and R_{Test} in the adversary rules are instantiated by the corresponding parameters of the transition relation. For states s and s', $s \rightarrow_{P,Adv,R_{Test}} s'$ iff there exists a rule in either Adv or the execution-model rules with the premises $Q_1(s), \ldots, Q_n(s)$ and the conclusion $s \rightarrow s'$ such that all of the premises hold. We define the set of* reachable states RS *as*

$$\mathrm{RS}(P, Adv, R_{Test}) = \left\{ s \;\middle|\; \exists s_0.\, s_0 \in IS(P) \wedge s_0 \rightarrow^*_{P,Adv,R_{Test}} s \right\}.$$

We now provide two examples of security property definitions. We give a symbolic definition of *session-key secrecy* which, when combined with different adversary models, gives rise to different notions of secrecy from the literature. We also define *aliveness*, which is one of the many forms of authentication [16,17]. Other security properties, such as secrecy of general terms, symbolic indistinguishability, or other variants of authentication, can be defined analogously.

Definition 8 (Session-key secrecy). *Let $(tr, IK, th, \sigma_{Test})$ be a state. We define the* secrecy of the session keys in $(tr, IK, th, \sigma_{Test})$ *as*

$$th(Test) = \langle\rangle \Rightarrow \forall k \in union((tr \downarrow Test) \downharpoonright \mathsf{sessionkeys}).\, IK \nvdash k\,.$$

Definition 9 (Aliveness for two-party protocols). *Let $(tr, IK, th, \sigma_{Test})$ be a state. We say that $(tr, IK, th, \sigma_{Test})$ satisfies* aliveness *if and only if*

$$\begin{aligned} th(Test) = \langle\rangle \Rightarrow \exists R_{Test}, R, tid, a.\, (Test, \mathsf{create}(R_{Test}, a)) \in tr \\ \wedge R \neq R_{Test} \wedge (tid, \mathsf{create}(R, \sigma_{Test}(R))) \in tr. \end{aligned}$$

We denote the set of all state properties by Φ. For all protocols P, adversary models Adv, and state properties $\phi \in \Phi$, we write $sat(P, Adv, \phi)$ iff $\forall R. \forall s.\, s \in \mathrm{RS}(P, Adv, R) \Rightarrow \phi(s)$. In the context of a state property ϕ, we say a protocol is *resilient against* an adversary capability AC if and only if $sat(P, \{AC\}, \phi)$.

Finally, we define a partial order $\leq_{\mathcal{A}}$ on adversary-compromise models based on inclusion of reachable states. For all adversary models Adv and Adv':

$$Adv \leq_{\mathcal{A}} Adv' \equiv \forall P, R.\, \mathrm{RS}(P, Adv, R) \subseteq \mathrm{RS}(P, Adv', R).$$

We write $Adv =_{\mathcal{A}} Adv'$ if and only if $Adv \leq_{\mathcal{A}} Adv'$ and $Adv' \leq_{\mathcal{A}} Adv$.

3 Protocol-security hierarchies

We introduce the notion of a *protocol-security hierarchy.* Such a hierarchy orders sets of security protocols with respect to the adversary models in which they satisfy their security properties. Protocol-security hierarchies can be used to select or design protocols based on implementation requirements and the worst-case expectations for adversaries in the application domain.

Because each combination of adversary rules from Figure 2 represents an adversary model, determining for which models a protocol satisfies its properties involves analyzing the protocol with respect to $2^7 = 128$ models. This is infeasible to do by hand and we therefore aim to use automatic analysis methods.

Automated analysis methods have the limitation that, in our models, even simple properties such as secrecy are undecidable. Fortunately, there exist semi-decision procedures that are successful in practice in establishing the existence of attacks. Moreover, some of these procedures can also successfully verify some protocols and properties. When analyzing the security properties of protocols with respect to an adversary model, we deal with undecidability by allowing the outcome of the analysis to be undefined, which we denote by $\perp$. The two other possible outcomes are F (falsified) or V (verified).

Definition 10 (Recursive approximation of *sat***).** *We say that a function* $f \in \mathit{Protocol} \times A \times \Phi \to \{F, \perp, V\}$ recursively approximates *sat if and only if f is recursive and for all protocols P, adversary models Adv, and state properties* ϕ*, we have* $f(P, \mathit{Adv}, \phi) \neq \perp \Rightarrow \big(f(P, \mathit{Adv}, \phi) = V \Leftrightarrow \mathit{sat}(P, \mathit{Adv}, \phi)\big)$.

Given such a function f, we can define a protocol-security hierarchy.

Definition 11 (Protocol-security hierarchy). *Let* Π *be a set of protocols,* ϕ *a state property, A be a set of adversary models, and let f recursively approximate sat. The* protocol-security hierarchy *with respect to* Π*, A,* ϕ*, and f is a directed graph* $H = (N, \to)$ *that satisfies the following properties:*

1. *N is a partition of* Π*, i. e.,* $\bigcup_{\pi \in N} \pi = \Pi$ *and for all* $\pi, \pi' \in N$ *we have that* $\pi \neq \emptyset$ *and* $\pi \neq \pi' \Rightarrow \pi \cap \pi' = \emptyset$.
2. *The function f respects the partitions N in that for all* $P, P' \in \Pi$ *we have*
$$\big(\exists \pi \in N. \{P, P'\} \subseteq \pi\big) \Leftrightarrow \forall \mathit{Adv} \in A. f(P, \mathit{Adv}, \phi) = f(P', \mathit{Adv}, \phi).$$
3. $\pi \to \pi'$ *if and only if*
$$\forall P \in \pi. \forall P' \in \pi'. \forall \mathit{Adv} \in A.\, f(P, \mathit{Adv}, \phi) = V \Rightarrow f(P', \mathit{Adv}, \phi) = V \wedge \\ f(P', \mathit{Adv}, \phi) = F \Rightarrow f(P, \mathit{Adv}, \phi) = F.$$

Lemma 1. *Let* $H = (N, \to)$ *be a protocol-security hierarchy with respect to* Π*,* ϕ*, A, and f. Let* $\leq_H$ *be defined as follows: for all* $\pi, \pi' \in N$*,* $\pi \leq_H \pi'$ *iff* $\pi \to \pi'$*. Then* $\leq_H$ *is a partial order.*

Proof. First, $\rightarrow$ is reflexive by Property 3 and hence $\leq_H$ is also reflexive. Second, since $\rightarrow$ is transitive by Property 3, so is $\leq_H$. Finally, assume $\pi \leq_H \pi'$ and $\pi' \leq_H \pi$. Then $\pi \rightarrow \pi'$ and $\pi' \rightarrow \pi$. Hence, by Property 3, for all adversary models $Adv \in A$ and all protocols $P \in \pi, P' \in \pi'$, we have $f(P, Adv, \phi) = f(P', Adv, \phi)$. By Property 2, this implies that $\pi = \pi'$ and therefore $\leq_H$ is antisymmetric. Hence $\leq_H$ is a partial order.

3.1 Examples of protocol-security hierarchies

In Fig. 3, we show the protocol-security hierarchy for the secrecy property of a set of protocols with respect to all possible sets of adversary rules from Fig. 2. In Fig. 4, we show a protocol-security hierarchy for authentication properties. We discuss the protocol sets in detail in Section 3.2.

Each node π in Fig. 3 and 4 corresponds to a set of protocols and is annotated with a set of adversary models. For each adversary model in the set, we require that no attacks are found in this or any weaker model, and also that attacks are found in all stronger models. Formally, each node π is annotated with all adversary models $a \in A$ for which

$$\forall a' \in A, P \in \pi.(a <_{\mathcal{A}} a' \Rightarrow f(P, a', \phi) = F) \wedge (a' \leq_{\mathcal{A}} a \Rightarrow f(P, a', \phi) \neq F).$$

The protocol-security hierarchies in Fig. 3 and 4 are automatically generated. We extended the symbolic security-protocol verification tool Scyther [14,18] with our adversary rules from Fig. 2. This tool recursively approximates *sat* and enables us to automatically analyze protocols with respect to any combination of adversary rules. Scyther produces an output from $\{F, \perp, V\}$, where F denotes that an attack was found, thereby falsifying the property, V denotes that the property was verified, and $\perp$ denotes that a timeout occurred. Using Scyther, the properties of each protocol are analyzed with respect to all adversary models. The graph is computed automatically by combining this data for each of the protocols with the order $\leq_{\mathcal{A}}$ on the adversary models. The protocol description files, analysis tools, and graph generation scripts can all be downloaded from [19].

Ideally we would like to establish hierarchies based on *sat*. However, only the recursive approximation f is available, which may return $\perp$, thereby providing only partial information about *sat*. Consequently, some edges in the hierarchies (involving nodes where f yields $\perp$) are also based on this partial information. Roughly speaking, we say an edge is *strict* if an edge also occurs between the protocols when given complete information about *sat*. More formally:

Definition 12 (strictness of edges in a protocol security hierarchy). *We say an edge $\pi \rightarrow \pi'$ in a protocol-security hierarchy is* strict *if the following two properties hold.*

1. *The protocols in π' are at least as strong as those in π.*

$$\forall P \in \pi, P' \in \pi'. \forall Adv \in A. f(P, Adv, \phi) \neq F \Rightarrow f(P', Adv, \phi) = V$$

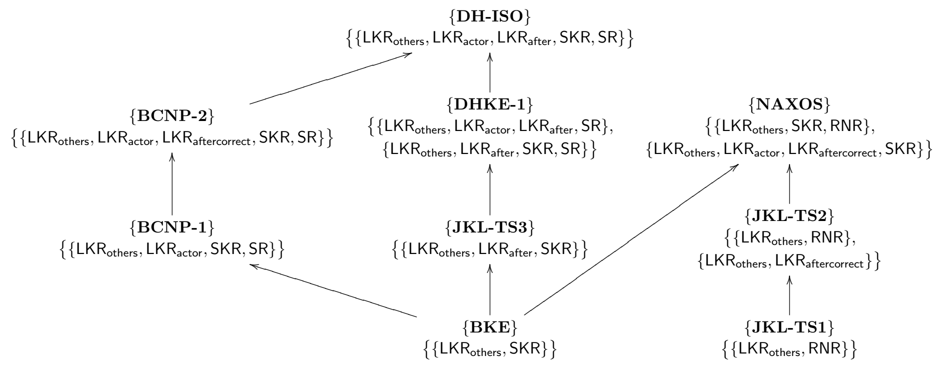
{DH-ISO}
{{LKR_others, LKR_actor, LKR_after, SKR, SR}}
{BCNP-2}
{{LKR_others, LKR_actor, LKR_aftercorrect, SKR, SR}}
{DHKE-1}
{{LKR_others, LKR_actor, LKR_after, SR},
{LKR_others, LKR_after, SKR, SR}}
{NAXOS}
{{LKR_others, SKR, RNR},
{LKR_others, LKR_actor, LKR_aftercorrect, SKR}}
{BCNP-1}
{{LKR_others, LKR_actor, SKR, SR}}
{JKL-TS3}
{{LKR_others, LKR_after, SKR}}
{JKL-TS2}
{{LKR_others, RNR},
{LKR_others, LKR_aftercorrect}}
{BKE}
{{LKR_others, SKR}}
{JKL-TS1}
{{LKR_others, RNR}}

Fig. 3. Protocol-security hierarchy for secrecy

2. The protocols in π are not equally strong as those in π'.

$$\forall P \in \pi, P' \in \pi'. \exists Adv \in A. f(P, Adv, \phi) = F \land f(P', Adv, \phi) = V$$

All edges in the authentication hierarchy in Fig. 4 are strict. This reflects Scyther's success in either verifying or falsifying these protocols. In contrast, for the secrecy hierarchy in Fig. 3, most protocols contain Diffie-Hellman exponentiation, for which Scyther currently cannot provide verification. Therefore, the edges in Fig. 3 are only based on attacks. Because they are not strict, they might not occur in the corresponding hierarchy based on *sat*.

3.2 Analyzing protocols using protocol-security hierarchies

Protocol-security hierarchies provide a novel way for choosing an optimal protocol for a given application domain, for example, exchanging a secret as illustrated here. We discuss below the protocols included in our two protocol-security hierarchies. We show how the resulting hierarchies facilitate fine-grained protocol comparisons that often refine or even contradict comparisons made in the literature. We start by discussing the hierarchy for secrecy in Fig. 3.

DH-ISO and DHKE-1. The original Diffie-Hellman protocol is only secure in the presence of a passive adversary since the messages sent are not authenticated. A simple fix is for agents to sign each message sent, along with the intended recipient, using the sender's long-term signature key. The resulting protocol family is referred to as *signed Diffie-Hellman.* We have analyzed the ISO variant of signed Diffie-Hellman as well as the DHKE-1 variant by Gupta and Shmatikov [20]. Scyther finds attacks on the Diffie-Hellman signed protocols for all models containing the RNR rule. This is consistent with the proof in [20], which does not consider this rule, as well as with the observation in [5] that RNR allows an attack on a signed Diffie-Hellman protocol.

JKL-TS1, JKL-TS2, and JKL-TS3. Jeong, Katz and Lee propose the protocols TS1, TS2, and TS3 [21]. TS1 is designed to satisfy *key independence* (keys of non-matching sessions may be revealed), whereas TS2 and TS3 should additionally satisfy *forward secrecy* (long-term keys of the agents may be revealed after the test session ends). They prove TS1 and TS2 correct in the random oracle model and TS3 in the standard model.

Our protocol-security hierarchy reveals the following. First, the TS3 protocol is incomparable to the other two. In contrast to TS2, TS3 additionally achieves resilience against $\mathsf{LKR}_{\mathsf{after}}$ and SKR, but it is not resilient against RNR. Second, the TS1 protocol is not resilient against SKR, which implies that the protocol does not satisfy key independence. Indeed, the missing identities in the session identifier of the protocol cause the protocol to be vulnerable to SKR. This is a flaw in the proof in [21]. Third, [21] suggests that the TS2 protocol satisfies forward secrecy. Our analysis shows that it only satisfies weak perfect forward secrecy, i. e., resilience against $\mathsf{LKR}_{\mathsf{aftercorrect}}$. The security model [21] requires the adversary to be passive when corrupting agents. This is in contrast to TS3, which does satisfy perfect forward secrecy. In this case, the authors have proven a weaker claim (weak perfect forward secrecy) whereas they might have been able to prove that TS3 satisfies a stronger property.

BKE. For the bilateral key-exchange (BKE) protocol [22], we find attacks in all models in our hierarchy except for adversaries capable of $\mathsf{LKR}_{\mathsf{others}}$ or SKR. BKE is therefore among the weakest protocols in our hierarchy. However, because it is resilient against SKR, it is not weaker than TS1 or TS2.

BCNP-1 and BCNP-2. Boyd, Cliff, Nieto, and Paterson propose two protocols [23], which we refer to as BCNP-1 and BCNP-2. When comparing their protocols to others, they focus on two properties, KCI resistance (resilience against $\mathsf{LKR}_{\mathsf{actor}}$) and weak forward secrecy (resilience against $\mathsf{LKR}_{\mathsf{aftercorrect}}$). Additionally, they claim that BCNP-2 provides more security that TS3 [21]. Our analysis allows for a more fine-grained comparison of the different protocols, confirming many remarks made in [23] but established by automatic instead of manual analysis. The hierarchy also explains why BCNP-2 does not provide more security than TS3. BCNP-2 is incomparable to TS3 because, unlike TS3, it is KCI-resilient (resilience against $\mathsf{LKR}_{\mathsf{actor}}$) but does not satisfy perfect forward secrecy (resilience against $\mathsf{LKR}_{\mathsf{after}}$). This disproves the claim in their paper.

NAXOS. LaMacchia, Lauter, and Mityagin propose the Naxos protocol [5] along with a new security model, claiming that this model is the strongest security model for AKE protocols. However, our hierarchy clearly reveals that NAXOS is not stronger than most other protocols in our set because it is vulnerable against SR and $\mathsf{LKR}_{\mathsf{after}}$. However, it is unique among the protocols we considered because it provides resilience against adversaries that are capable of both RNR and SKR.

Next, we discuss the hierarchy for authentication presented in Fig. 4. We verify the protocols with respect to a strong form of authentication called *synchronisation* [16]. Protocols that satisfy synchronisation also satisfy aliveness.

Needham-Schroeder. The Needham-Schroeder protocol [24] is resilient to adversaries capable of $\mathsf{LKR}_{\mathsf{after}}$ and SKR. As we will show in Theorem 1, all authentication properties of any protocol are resilient against $\mathsf{LKR}_{\mathsf{after}}$. The fact that the protocol is resilient against SKR is not surprising as the protocol does not contain any session keys.

$$\begin{tikzcd}
& \begin{array}{c}\{\textbf{CCITT X.509-1}\}\\ \{\{\mathsf{LKR}_{\mathsf{others}}, \mathsf{LKR}_{\mathsf{actor}}, \mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}, \mathsf{SR}, \mathsf{RNR}\}\}\end{array} & \\
\begin{array}{c}\{\textbf{CCITT X.509-3}\}\\ \{\{\mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}, \mathsf{SR}, \mathsf{RNR}\}\}\end{array} \arrow[ur] & & \begin{array}{c}\{\textbf{Needham-Schroeder-Lowe}\}\\ \{\{\mathsf{LKR}_{\mathsf{others}}, \mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}, \mathsf{SR}\}\}\end{array} \arrow[ul] \\
& \begin{array}{c}\{\textbf{Needham-Schroeder}\}\\ \{\{\mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}\}\}\end{array} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 4. Protocol-security hierarchy for authentication

Needham-Schroeder-Lowe. The original Needham-Schroeder protocol is vulnerable against a man-in-the-middle attack, which motivated Lowe's fix [15]. This attack requires precisely the $\mathsf{LKR}_{\mathsf{others}}$ capability. Our hierarchy reveals that the Needham-Schroeder-Lowe protocol is resilient to adversaries capable of $\mathsf{LKR}_{\mathsf{others}}$ and SR. The original Needham-Schroeder is not resilient against SR because the missing identity in the second message allows the adversary to exploit a non-matching session to decrypt this message, in which he uses SR to reveal the nonce of the first message.

CCITT X.509-1 and X.509-3. The CCITT X.509 standard [25] contains several protocol recommendations. Here we consider X.509-1 and X.509-3. X.509-1 satisfies its authentication properties with respect to the strongest possible adversary model, i. e., the adversary with all capabilities from Fig 2. The X.509-3 protocol is not resilient against $\mathsf{LKR}_{\mathsf{others}}$ or $\mathsf{LKR}_{\mathsf{actor}}$. However, unlike Needham-Schroeder(-Lowe), it is resilient against RNR.

4 Relations between Models and Properties

As previously noted, by classifying different basic adversarial capabilities from the literature, one quickly arrives at a large number of adversary models. Here we provide general results that aid in relating and reasoning with these models.

To begin with, our partial order on adversary models $\leq_{\mathcal{A}}$ has implications for security protocol verification. Given a state property ϕ like those from Section 2.5, a protocol that satisfies ϕ in a model also satisfies ϕ in all weaker models. Equivalently, falsification in a model entails falsification in all stronger models. Formally, if $Adv \leq_{\mathcal{A}} Adv'$, then for all protocols P and state properties ϕ, $sat(P, Adv', \phi) \Rightarrow sat(P, Adv, \phi)$ and, equivalently, $\neg sat(P, Adv, \phi) \Rightarrow \neg sat(P, Adv', \phi)$.

Since adding adversary rules only results in a larger transition relation and hence more reachable states, we have:

Lemma 2 (Adding rules only strengthens the adversary). *Let r be an adversary rule from Fig. 2 and Adv be an adversary model, i. e., a set of adversary rules. Then $Adv \leq_{\mathcal{A}} Adv \cup \{r\}$.*

Most of our rules are independent in that they provide adversary capabilities not given by other rules. The following lemma formalizes this.

Lemma 3 (Rule independence). *Let Adv be an adversary model. Then we have for all adversary rules r from Fig. 2*

$$\big(r = \mathsf{LKR}_{\mathsf{aftercorrect}} \wedge \mathsf{LKR}_{\mathsf{after}} \in Adv\big) \Leftrightarrow \big(Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}\big).$$

Proof of ($\Rightarrow$): Let $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}} \in Adv$. Each transition using $\mathsf{LKR}_{\mathsf{aftercorrect}}$ can be simulated using $\mathsf{LKR}_{\mathsf{after}}$. Hence the sets of reachable states on both sides of the above equality are equal and thus $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. Proof of ($\Leftarrow$): Let $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. Suppose $r \neq \mathsf{LKR}_{\mathsf{aftercorrect}}$. Then there are transitions enabled by r that are not enabled by the other rules. In particular, even if $\mathsf{LKR}_{\mathsf{aftercorrect}} \in Adv$, there are protocols with roles that can be completed without matching sessions, whereby $\mathsf{LKR}_{\mathsf{after}}$ enables transitions not enabled by $\mathsf{LKR}_{\mathsf{aftercorrect}}$. Hence we have a contradiction and therefore $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$. Now suppose $\mathsf{LKR}_{\mathsf{after}} \notin Adv$. Then some transitions enabled by r are not enabled by $Adv \setminus \{r\}$, contradicting $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. Hence $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}} \in Adv$.

Corollary 1. *The rules in Fig. 2 give rise to* $2^5 \times 3 = 96$ *models with distinct sets of reachable states.*

This corollary follows from Lemmas 2 and 3.

Interestingly, to evaluate some properties it is only necessary to consider traces up to the end of the test session.

Definition 13 (post-test invariant properties). *We define the set Φ_{PTI} of* post-test invariant properties *as all state properties $\phi \in \Phi$ that satisfy*

$$\begin{aligned}&\forall P, R, Adv. \forall (tr, IK, th, \sigma_{Test}) \in \mathrm{RS}(P, Adv, R).\, th(Test) = \langle\rangle \Rightarrow \\ &\qquad \forall s.(tr, IK, th, \sigma_{Test}) \rightarrow^{*}_{P,Adv,R_{Test}} s \Rightarrow \big(\phi((tr, IK, th, \sigma_{Test})) \Leftrightarrow \phi(s)\big).\end{aligned}$$

Aliveness, as defined earlier, is a post-test invariant property. Other authentication goals such as various forms of agreement [17] or synchronisation [16] are also post-test invariant properties. Secrecy however is not such a property.

Theorem 1 (post-test invariant properties are resilient against future capabilities). *Let r be an adversary rule from Fig. 2 and $\phi \in \Phi_{PTI}$ be a post-test invariant property. Then for all protocols P and adversary models Adv,*

$$r \in \{\mathsf{LKR}_{\mathsf{aftercorrect}}, \mathsf{LKR}_{\mathsf{after}}\} \wedge sat(P, Adv, \phi) \Rightarrow sat(P, Adv \cup \{r\}, \phi)$$

This theorem follows as $\mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}}$ only enable new transitions in those states where the test thread has ended. By definition, post-test invariant properties are invariant with respect to such transitions. As a result, we need only consider 32 (out of 96) models when analyzing a protocol with respect to post-test invariant properties.

5 Conclusions

We see our work as a first step in providing models and tool support for systematically modeling and analyzing security protocols with respect to adversaries endowed with different compromise capabilities. We presented applications to protocol analysis and constructing protocol-security hierarchies.

Our adversary capabilities generalize those from the computational setting and combine them with a symbolic model. In doing so, we unify and generalize a wide range of models from both settings. Exploring the exact nature of this generalization as well as mappings between the two settings remains as future work. Also interesting would be to develop methods for designing protocols optimized for different adversarial scenarios or strengthening existing protocols.

Finally, the concept of a protocol-security hierarchy can be naturally extended to any domain where security properties of systems can be evaluated with respect to a set of adversary models. This leads to the more general notion of a *security hierarchy.* For example, in the domain of access control, attackers could have different capabilities with respect to how policies are enforced. A hierarchy in this setting could help distinguish the degrees of security provided by different access-control mechanisms.

References

1. Günther, C.: An identity-based key-exchange protocol. In: EUROCRYPT'89. Volume 434 of LNCS., Springer (1990) 29–37
2. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (October 1996)
3. Basin, D., Cremers, C.: From Dolev-Yao to strong adaptive corruption: Analyzing security in the presence of compromising adversaries. Cryptology ePrint Archive, Report 2009/079 (2009) `http://eprint.iacr.org/`.
4. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: EUROCRYPT. Volume 2045 of LNCS., Springer (2001) 453–474
5. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: ProvSec. Volume 4784 of LNCS., Springer (2007) 1–16
6. Shoup, V.: On formal models for secure key exchange (version 4) (November 1999) revision of IBM Research Report RZ 3120 (April 1999).
7. Bresson, E., Manulis, M.: Securing group key exchange against strong corruptions. In: ASIACCS, ACM (2008) 249–260
8. Just, M., Vaudenay, S.: Authenticated multi-party key agreement. In: ASIACRYPT 1996. Volume 1163 of LNCS. (1996) 36–49
9. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176 (2005) `http://eprint.iacr.org/`, retrieved on April 14, 2009.
10. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: Proc. STOC '95, ACM (1995) 57–66
11. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: EUROCRYPT. LNCS, Springer (2000) 139–155

12. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: CRYPTO, Springer (1993) 232–249
13. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: CRYPTO. Volume 2729 of LNCS., Springer (2003) 110–125
14. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Proc. CAV. Volume 5123 of LNCS., Springer (2008) 414–418
15. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: TACAS'96. Volume 1055 of LNCS. Springer (1996) 147–166
16. Cremers, C., Mauw, S., de Vink, E.: Injective synchronisation: an extension of the authentication hierarchy. Theoretical Computer Science (2006) 139–161
17. Lowe, G.: A hierarchy of authentication specifications. In: Proc. 10th IEEE Computer Security Foundations Workshop (CSFW), IEEE (1997) 31–44
18. Cremers, C.: Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In: CCS '08: Proc. of the 15th ACM conference on Computer and communications security, ACM (2008) 119–128
19. Cremers, C.: Scyther tool with compromising adversaries extension Includes protocol description files and test scripts. Available online at `http://people.inf.ethz.ch/cremersc/scyther/`.
20. Gupta, P., Shmatikov, V.: Towards computationally sound symbolic analysis of key exchange protocols. In: Proc. FMSE 2005, ACM (2005) 23–32
21. Jeong, I.R., Katz, J., Lee, D.H.: One-round protocols for two-party authenticated key exchange. In: Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings. Volume 3089 of LNCS., Springer (2004) 220–232
22. Clark, J., Jacob, J.: A survey of authentication protocol literature (1997) `http://citeseer.ist.psu.edu/clark97survey.html`.
23. Boyd, C., Cliff, Y., Nieto, J.M.G., Paterson, K.G.: One-round key exchange in the standard model. IJACT **1**(3) (2009) 181–199
24. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM **21**(12) (1978) 993–999
25. CCITT: The directory authentification framework (1987) Draft Recommendation X.509, Version 7.