From Dolev-Yao to Strong Adaptive Corruption: Analyzing Security in the Presence of Compromising Adversaries

February, 2009

David Basin and Cas Cremers
Department of Computer Science
ETH Zurich
Zurich, Switzerland
Email: {david.basin, cas.cremers}@inf.ethz.ch

Abstract

We formalize a hierarchy of adversary models for security protocol analysis, ranging from a Dolev-Yao style adversary to more powerful adversaries who can reveal different parts of principals' states during protocol execution. We define our hierarchy by a modular operational semantics describing adversarial capabilities. We use this to formalize various, practically-relevant notions of key and state compromise. We also use our semantics as a basis to extend an existing symbolic protocol-verification tool with our adversary models. This tool is the first that supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of so-called strong corruptions and state-reveal queries. As applications, we use our model hierarchy to relate different adversarial notions, gaining new insights on their relative strengths, and we use our tool find new attacks on protocols.

1. Introduction

Problem context. Many cryptographic protocols are designed to work in the face of limited forms of corruption. For example, a Diffie-Hellman key agreement protocol, where digital signatures are used to authenticate the exchanged half-keys, has the property of perfect-forward secrecy [24,41]. Namely, the resulting shared key remains secret even when the signature keys are later compromised by the adversary. Designing protocols that work even in presence of different forms of adversarial compromise has considerable practical relevance. It reflects the multifaceted computing reality with different rings of protection (user-space, kernel space, hardware security modules) offering different levels of assurance with respect to the computation of cryptographic functions (e. g., the quality of the pseudo-random numbers generated) and the storage of keys and intermediate results.

Symbolic and computational approaches have addressed this problem to different degrees. Most symbolic formalisms are based on the Dolev-Yao model. These offer, with few exceptions, a limited view of honesty and conversely corruption: either principals are honest from the start and always keep their secrets to themselves or they are completely malicious and always under adversarial control. Thus, correctness proofs in such models with respect to a security property like message secrecy, must be interpreted as "if my communication partners have not been compromised in any way before, and will never be compromised in the future, then the property will hold." In this limited view, it is impossible to distinguish between the security guarantees provided by early key exchange protocols such as the Bilateral key-exchange [18] and current state-of-the art protocols such as (H)MQV [32,36]. It is also impossible to discern any benefit from storing the long-term private keys in a tamper-proof module, or from performing part of a computation in a cryptographic coprocessor. Despite these limitations, symbolic methods have the advantage that there are numerous effective tools for symbolic protocol analysis [2,4,9,19,38,43].

In contrast to the above, researchers in the computational setting, e. g. [12,15,29,31,34,45], have explored stronger adversarial notions, whereby principals may be selectively corrupted during protocol execution. For example, their short-term or long-term secrets may be revealed (at different times) to the adversary, as well as other parts of their state, including the results of intermediate computations. By reasoning about protocols in the presence of such adversaries, one can establish stronger properties, such as perfect-forward secrecy. There are, however, a number of drawbacks to these computational models. First, in general, they are only defined for key-agreement protocols, whereas one may expect similar definitions to exist for any security protocol. Second, they are generally defined in a monolithic fashion. The security model simultaneously captures the execution model of the protocol, the adversary capabilities, and the desired security property. This complicates the comparison of the individual models, as it is non-trivial to factor out these elements from the monolithic whole. Third, contrary

to the simpler security models used in symbolic security-protocol analysis, there is no tool support available for any of these more advanced computational models.

Contributions. We define a modular operational semantics for security protocols that includes a set of rules specifying different adversarial compromise capabilities. These rules act as building blocks for constructing a hierarchy of adversary models, ranging from a Dolev-Yao style adversary to more powerful adversaries with capabilities for dynamically compromising both long-term and short-term data. We use our modular semantics to formalize different, practically-relevant notions of key and state compromise, which can be tailored to different computing scenarios. For example, we can model attacks against implementations of cryptographic protocols involving the mixed 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.[1]

We also use our modular semantics to show how the analogs of different adversarial models studied in the computational setting can be constructed from combinations of our rules. This provides a more uniform view of the different adversary models and allows us to easily establish relationships between them. Our adversary model is orthogonal to the execution model and security properties. Although presented in a symbolic and possibilistic context, our modular adversary model can also be used with computational execution models, or with probabilistic (e. g. indistinguishability-based) security properties.

Finally, our semantics directly lends itself to protocol analysis and we extend the Scyther tool [19] for symbolic protocol verification to reason about security properties of protocols in the presence of adversaries in our hierarchy. The Scyther tool is thus the first automated tool that supports notions such as: weak perfect forward secrecy, key compromise impersonation, adversaries that are able to learn (parts of) the local state of agents, and malicious random number generators. We use the resulting tool to analyze a number of protocols, such as Naxos, KEA+, and protocols in the MQV family, thereby finding new attacks. We also define a hierarchy on protocols, using our tool to classify their relative strengths against different forms of adversarial compromise.

Organization. We define the modular operational semantics and associated security properties in Section 2. In Section 3, we define a hierarchy of adversary models, relate models in the hierarchy to existing adversarial notions, and define the notion of a protocol-security hierarchy. In Section 4, we use the Scyther tool to verify different protocols with respect to models in our hierarchy. We discuss related work in Section 5 and draw conclusions in Section 6. In the Appendix we further explain the design choices behind our adversary models, describe our tool and provide an example of its use, and provide details of attacks found.

1. For example, it is folklore in the PKCS#11 community that if the primary goal is high throughput, one only uses the private-key acceleration capabilities of the hardware and carries out all other cryptographic operations (e. g., symmetric cryptography and hashing) on the host computer [27].

2. The Compromising Adversary Model

We define an operational semantics that is modular with respect to the adversary's capabilities. After notational preliminaries, we define a general symbolic protocol-execution model, where agents execute threads that correspond to instances of protocol roles. Afterwards we give a set of rules, defining different adversary capabilities. A protocol and a subset of adversarial rules define a transition system, which has a trace semantics, as is standard.

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. For example, one that contains flow-control commands rather than just straight-line protocols.

2.1. Notational preliminaries

Let f be a function. We write $dom(f)$ and $ran(f)$ to denote f's domain and range, respectively. We write $f[b \hookleftarrow a]$ to denote f's update, that is 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 mapping some elements from X to elements from Y.

For any set S, $\mathcal{P}(S)$ denotes the power set of S and S^* denotes the set of finite sequence of elements from S. We write $\langle s_0, \ldots, s_n \rangle$ (sometimes omitting brackets when no confusion can result) to denote the sequence consisting of elements s_0 through s_n. For s a sequence of length $|s|$ and $i < |s|$, we write s_i to denote the i-th element. We write $s[..i]$ to denote the sequence $\langle s_0, \ldots, s_i \rangle$ and $s[i..]$ to denote the sequence $\langle s_i, \ldots, s_{|s|-1} \rangle$. We write $s \hat{} s'$ for the concatenation of sequences s and s'. Finally, abusing set notation, we write $e \in s$ iff $\exists i. s_i = e$.

We use standard notions for manipulating terms.[2] 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 $dom(\sigma) \cap dom(\sigma') = \emptyset$. We write $\sigma(t)$ for the application of the substitution σ to t and $FV(t)$ to denote the free variables occurring in t.

For $\rightarrow$ a binary relation, $\rightarrow^*$ denotes its reflexive transitive closure.

2. We will later introduce the relevant syntactic categories of terms, events, and sequences of events. Functions like substitution are defined as standard (e. g., [3]) over elements of these categories.

2.2. Terms and events

We assume given the infinite sets $Agent$, $Role$, $Fresh$, Var, and TID of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, and thread identifiers. In order to bind local terms, such as freshly generated terms or local variables, to a protocol role instance (thread), we write $T\sharp tid$, which denotes that the term T is local to the protocol role instance identified by tid.

Definition 1: Basic terms

$$\begin{aligned} BasicTerm ::= {} & Agent \mid Role \mid Fresh \mid Var \\ & \mid Fresh\sharp TID \mid Var\sharp TID \end{aligned}$$

Definition 2: Terms

$$\begin{aligned} Term ::= {} & BasicTerm \mid Func(Term^*) \\ & \mid \{\!|\, Term \,|\!\}^a_{Term} \mid \{\!|\, Term \,|\!\}^s_{Term} \end{aligned}$$

Here $\{\!|\, t_1 \,|\!\}^a_{t_2}$ denotes asymmetric encryption of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^s_{t_2}$ denotes symmetric encryption. The $Func$ notation can be used to model other cryptographic functions, such as hash functions and key infrastructures that associate keys with agents. Freshly generated terms and variables are assumed to be local to a thread. We model constants as functions of arity 0.

Depending on the protocol analyzed, we assume that symmetric or asymmetric long-term keys have been distributed prior to protocol execution. We write $k(A, B)$ for the symmetric long-term key shared between agents A and B, $sk(A)$ for A's long-term private key, and $pk(A)$ for A's public key. We assume the existence of an inverse function on terms, where t^{-1} denotes the inverse key of t. In particular, $pk(X)^{-1} = sk(X)$, $sk(X)^{-1} = pk(X)$ for all X, and $t^{-1} = t$ for all other terms t.

For simplicity of presentation, we work with a free term algebra, whereby term equality is just syntactic equality. Note that our model can be straightforwardly extended to capture algebraic properties.

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

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

The syntactic subterm relation $\sqsubseteq$ is defined as the smallest relation satisfying:

$$\begin{array}{lll} t \sqsubseteq t & t \sqsubseteq f(\ldots, t, \ldots) & \\ t_1 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2} & t_1 \sqsubseteq \{\!|\, t_1 \,|\!\}^a_{t_2} & t_1 \sqsubseteq (t_1, t_2) \\ t_2 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2} & t_2 \sqsubseteq \{\!|\, t_1 \,|\!\}^a_{t_2} & t_2 \sqsubseteq (t_1, t_2) \end{array}$$

An agent can engage in the following events when running a protocol.

Definition 3: Agent events

$$\begin{aligned} AgentEvent ::= {} & \mathsf{create}(Role, Agent) \\ & \mid \mathsf{send}(Term) \mid \mathsf{recv}(Term) \\ & \mid \mathsf{generate}(\mathcal{P}(Fresh)) \\ & \mid \mathsf{state}(\mathcal{P}(Term)) \\ & \mid \mathsf{sessionkey}(\mathcal{P}(Term)) \end{aligned}$$

In contrast to normal agents, the adversary can engage in the following events.

Definition 4: Adversary events

$$\begin{aligned} AdvEvent ::= {} & \mathsf{LongtermKeyReveal}(Agent) \\ & \mid \mathsf{SessionKeyReveal}(Agent, TID) \\ & \mid \mathsf{StateReveal}(Agent, TID) \\ & \mid \mathsf{RandomReveal}(Agent, TID) \end{aligned}$$

We assume all adversary events are executed in a single adversary thread, identified by the distinguished thread identifier $\mathrm{tid}_{\mathcal{A}} \in TID$.

We will explain the interpretation of agent and adversarial events shortly. Here we simply note that the first three honest agent events are conventional: starting a thread, sending a message, and receiving a message. The last three events tag state information, which can possibly be compromised later by the adversary. The four adversary events specify which information the adversary compromises. These events can occur at any time during protocol execution and correspond to different kinds of *adversary queries* from computational models.

Finally, system events are events originating from agents or the adversary.

$$Event = AgentEvent \cup AdvEvent$$

2.3. Protocols and threads

A protocol is a mapping from role names from the set $Role$ to sequences of events, i.e. $Protocol : Role \to AgentEvent^*$. We require that no thread identifiers occur as subterms of events in a protocol definition.

Example 1 (Simple protocol): Let $n \in Fresh$. Let the protocol P be defined as follows.

$$\begin{aligned} P(\mathrm{Init}) = \langle & \mathsf{generate}(\{n\}), \\ & \mathsf{send}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}) \rangle \\ P(\mathrm{Resp}) = \langle & \mathsf{recv}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}) \rangle \end{aligned}$$

Protocols are executed by agents that execute roles, thereby instantiating role names with agent names.

We now define a function $thread : (Protocol \times Role \times TID \times \mathcal{S}ub) \to AgentEvent^*$. Given a protocol, a role, a thread identifier, and a substitution mapping role names to agents, $thread$ yields a sequence of agent events that may occur in a thread.

Definition 5 (Thread): Let P be a protocol, $tid \in TID$, $R \in dom(P)$, and let σ be a substitution such that $dom(\sigma) = dom(P)$. Then

$$thread(P, R, tid, \sigma) = \sigma(\sigma'(P(R))),$$

where σ' is defined as follows. For a sequence of events l, let $CV(l)$ denote the finite set of variables and freshly generated terms occurring in the events in l. To bind the variables and fresh terms to their thread, we define $\sigma' = \bigcup_{cv \in CV(P(R))} [cv \sharp tid / cv]$.

Example 2: Let P be the protocol from Example 1, $t_1 \in TID$, and $\{A, B\} \subseteq Agent$. Then

$$\begin{aligned} thread(P, \mathrm{Init}, t_1, [A, B/\mathrm{Init}, \mathrm{Resp}]) = \\ \langle \mathsf{generate}(\{n \sharp t_1\}), \\ \mathsf{send}(A, B, \{\!|\, B, n \sharp t_1 \,|\!\}^a_{sk(A)}) \rangle \,. \end{aligned}$$

2.4. Execution model

We define the set $Trace$ as $(TID \times Event)^*$, which represents possible execution histories. The state of our system is a triple $(tr, IK, th) \in Trace \times \mathcal{P}(Term) \times (TID \nrightarrow Event^*)$. The three components are (1) the trace tr, (2) the adversary's knowledge IK, and (3) the partial function th mapping thread identifiers of initiated threads (executing or completed) to event traces. Note that IK and th can actually be computed directly from tr, but we explicitly include them in the state for clarity of exposition as they improve the readability of our operational semantics.

The initial system state is $(\langle\rangle, IK_0, \emptyset)$, where IK_0 is the public knowledge associated with the protocol. For example, this includes the names and public keys of all agents. Note that, in contrast to Dolev-Yao models, IK_0 does not include any long-term private keys. The adversary may learn these by performing LongtermKeyReveal events.

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

Execution-model rules. The execution-model rules are given in Figure 1. The create rule starts a new instance of a protocol role (a *thread*), executed by an agent. A fresh thread identifier tid is assigned to the thread, thereby distinguishing it from existing threads and the adversary thread. Note that tid is neither visible to the agents nor the adversary. The role names $dom(P)$, which can occur in events associated with the role, are replaced by the agents names by the substitution ρ. In case the thread is the test thread, discussed below, R and ρ are set to R_{Test} and ρ_{Test}, respectively.

The send rule sends a message m to the network. The message m does not include explicit sender or recipient fields although, if desired, they can be given as subterms of m. As is standard, the adversary receives all messages sent, independent of the intended recipient.

The receive rule accepts messages from the network. The pattern pt is a term that may contain free variables. In our model, recipients accept all messages that match the pattern pt, and block on any other messages. The resulting substitution σ is applied to the remaining protocol steps l.

We introduce three additional annotation rules. These rules simply store information about generated fresh terms, the local state, and session keys in the trace. The generate rule marks the fresh terms that have been generated,[3] the state rule marks the current local state, and the sessionkey rule marks a set of terms as session keys.

Test thread. For verification of security properties we will focus on a particular thread. In the computational setting this is the thread in which the adversary performs a so-called test query. In the same spirit, we refer to the thread under consideration as the *test thread*, with the corresponding thread identifier Test. For the test thread, the substitution of role names by agent names is given by ρ_{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 $R_{\mathrm{Test}} = \mathrm{Init}$ and $\rho_{\mathrm{Test}} = [\mathrm{Alice}, \mathrm{Bob}/\mathrm{Init}, \mathrm{Resp}]$.

Auxiliary functions. Prior to giving the adversary-compromise rules, we define some auxiliary functions.

Given a trace tr in which a thread identifier tid occurs, we define the partial function $role : Trace \times TID \nrightarrow Role$ by $role(tr, tid) = R$ if $(tid, \mathsf{create}(\mathsf{R})) \in tr$, and $role(tr, tid)$ is undefined otherwise. This function is well-defined, i. e. there is at most one such R, due to the third premise $tid \notin dom(th)$ of the create rule.

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

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

During execution of a protocol, the test thread may share its secrets with other threads. Hence some adversary rules require distinguishing between intended *partner threads* and other threads. We define the partner threads using a generalization of the notion of matching histories from [7] (sometimes also called matching conversations). We only give here the definition for two-party protocols. Within our model, however, we use the general definition for arbitrary protocols with an arbitrary number of participants from [22], where it is called non-injective message agreement. We assume protocols consist of matching send-receive pairs and

3. Note that this rule need not ensure that m is unique. This is handled by the mapping of freshly generated terms c, occurring in a protocol, to $c \sharp tid$ in a thread tid, by applying the $thread$ function.

$$\frac{R \in dom(P) \quad \rho \in dom(P) \to Agent \quad tid \notin (dom(th) \cup \{\mathrm{tid}_{\mathcal{A}}\}) \quad tid = \mathrm{Test} \Rightarrow (R,\rho) = (R_{\mathrm{Test}}, \rho_{\mathrm{Test}})}{(tr, IK, th) \longrightarrow (tr^\frown\langle(tid, \mathsf{create}(R, \rho(R)))\rangle, IK, th[thread(P, R, tid, \rho) \hookleftarrow tid])}[\mathsf{create}]$$

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

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

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

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

$$\frac{th(tid) = \langle \mathsf{sessionkey}(M)\rangle^\frown l}{(tr, IK, th) \longrightarrow (tr^\frown\langle(tid, \mathsf{sessionkey}(M))\rangle, IK, th[l \hookleftarrow tid])}[\mathsf{sessionkey}]$$

Figure 1. Execution-model rules

that each pair is labeled with a unique label. We subscript the matching send and receive events with the label ℓ as in $\mathsf{send}_\ell(m)$ and $\mathsf{recv}_\ell(m)$.

Definition 6 (Partnering for two-party protocols): Let tr be a trace and let tid be a thread identifier occurring in tr. We distinguish two cases. (1) If tid has no receive events in tr, we define $Partners(tr, tid)$ as the set of all thread identifiers occurring in $tr[..k]$, where k is the index of the last event associated with tid in tr. (2) If tid is associated with one or more receive events in tr, we define

$$\begin{aligned}\mathrm{MH}(tr, tid, tid') = \forall k, m, \ell\,.\\ tr_k = (tid, \mathsf{recv}_\ell(m)) \Rightarrow \big(\exists j.j < k \wedge\\ tr_j = (tid', \mathsf{send}_\ell(m)) \wedge \mathrm{MH}(tr[..j], tid', tid)\big)\end{aligned}$$

and

$$Partners(tr, tid) = \{tid\} \cup \big\{tid' \;\big|\; \mathrm{MH}(tr, tid, tid')\big\}.$$

Intuitively, this definition ensures that two threads are partners if and only if all their received messages have been previously sent by the other thread.

We define next the predicate $PartnersExist$, which specifies that partner threads exist for each role of the protocol.

$$\begin{aligned}PartnersExist(tr, \mathrm{Test}, P) = \forall R \in dom(P).\\ \exists tid \in Partners(tr, \mathrm{Test}).role(tr, tid) = R\end{aligned}$$

Finally, we define the function $FindLast$, which identifies the contents of the last event in a trace of a given type associated with a given thread.

$$FindLast(tr, tid, evtype) = \begin{cases}\emptyset & \text{if } \neg\exists M.(tid, evtype(M)) \in tr, \text{ and}\\ M & \text{if } \exists i.tr_{i-1} = (tid, evtype(M)) \wedge\\ & FindLast(tr[i..], tid, evtype) = \emptyset.\end{cases}$$

Adversary-compromise rules. We now define the adversary-compromise rules in Figure 2. The first five rules model the compromise of the long-term private keys of agents. This is represented by the event $\mathsf{LongtermKeyReveal}(a)$, which models the adversary learning the long-term private keys of the agent a. After this event occurs, the adversary can emulate new threads of the agent a. In traditional Dolev-Yao models, this event occurs implicitly for dishonest agents before the honest agents start their threads.

The $\mathsf{LongtermKeyReveal}_{\mathsf{notgroup}}$ rule formalizes the adversary capability typically used in the symbolic analysis of security protocols since Lowe's Needham-Schroeder attack [37]: the adversary can learn all long-term private keys of agents that are not in the group, i. e., not the intended partners of the test thread. Hence, when Alice starts a session with Bob, the adversary can learn the private key of, for example, Charlie and Eve.

The $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$ rule allows the adversary to learn the private key of the agent executing the test thread. The intuition is that a protocol may still function as long as the long-term keys of the partners are not revealed: the key of the agent executing the test thread (also called the *actor*) may be compromised. This rule allows the adversary to perform so-called Key Compromise Impersonation attacks [29].

The $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, and $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$ rules each have restrictions on when

$$\frac{a \notin ran(\rho_{\mathrm{Test}})}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\mathsf{LongtermKeyReveal}_{\mathsf{notgroup}}]$$

$$\frac{a = \rho_{\mathrm{Test}}(R_{\mathrm{Test}}) \qquad \forall r \in dom(\rho_{\mathrm{Test}}) \setminus \{R_{\mathrm{Test}}\}.\rho_{\mathrm{Test}}(r) \neq a}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\mathsf{LongtermKeyReveal}_{\mathsf{actor}}]$$

$$\frac{th(\mathrm{Test}) = \langle\rangle}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\mathsf{LongtermKeyReveal}_{\mathsf{after}}]$$

$$\frac{th(\mathrm{Test}) = \langle\rangle \qquad PartnersExist(tr, \mathrm{Test}, P)}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}]$$

$$\frac{th(\mathrm{Test}) = \langle\rangle \quad PartnersExist(tr, \mathrm{Test}, P) \quad \forall tid \in Partners(tr, \mathrm{Test}).(\mathrm{tid}_{\mathcal{A}}, \mathsf{RandomReveal}(a, tid)) \notin tr}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}]$$

$$\frac{(tid, \mathsf{create}(R, a)) \in tr \qquad th(tid) \neq \langle\rangle \qquad tid \notin Partners(tr, \mathrm{Test})}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{SessionKeyReveal}(a, tid))\rangle, IK \cup FindLast(tr, tid, \mathsf{sessionkey}), th)}[\mathsf{SessionKeyReveal}]$$

$$\frac{(tid, \mathsf{create}(R, a)) \in tr \qquad th(tid) \neq \langle\rangle \qquad tid \notin Partners(tr, \mathrm{Test})}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{StateReveal}(a, tid))\rangle, IK \cup FindLast(tr, tid, \mathsf{state}), th)}[\mathsf{StateReveal}]$$

$$\frac{(tid, \mathsf{create}(R, a)) \in tr \quad th(tid) \neq \langle\rangle \quad tid \notin Partners(tr, \mathrm{Test}) \vee (\mathrm{tid}_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a)) \notin tr}{(tr, IK, th) \longrightarrow (tr^\frown\langle(\mathrm{tid}_{\mathcal{A}}, \mathsf{RandomReveal}(a, tid))\rangle, IK \cup \{t \mid t \in M \wedge (tid, \mathsf{generate}(M)) \in tr\}, th)}[\mathsf{RandomReveal}]$$

Figure 2. Adversary-compromise rules

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 $th(\mathrm{Test}) = \langle\rangle$. This is the sole premise of $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, it is said to satisfy Perfect Forward Secrecy [24,41].

For $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, we have the additional premise that partner threads must exist for the test thread. This condition stems from [32], 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 $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, it is said to satisfy weak Perfect Forward Secrecy [32].

For $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$, we have an additional premise that no RandomReveal event has previously occurred in the agent's thread. This event corresponds to the adversary learning the random data generated by the thread tid executed by the agent a (for example due to a malicious or badly designed [30,46] random number generator). This condition stems from [34] and excludes the adversary from both learning the random values generated by an agent a as well as the long-term private keys of a.

The three remaining adversary rules correspond to the compromise of short-term data, that is, data local to a specific thread. The events $\mathsf{SessionKeyReveal}(a, tid)$ and $\mathsf{StateReveal}(a, tid)$ indicate that the adversary gains access to the session key or, respectively, the local state of the agent a executing the thread tid. These are respectively marked by the last sessionkey and state events. The rules SessionKeyReveal and StateReveal allow such compromise. Their premises are that the thread is not a partner thread and has not finished. The premise of the RandomReveal rule allows for the compromise of the partner threads, but only if the long-term key of the agent executing the thread was not previously revealed. This condition stems from [34].

2.5. Traces for adversary-compromise models

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

Definition 7 (Traces): Let P be a protocol, IK_0 a set of terms, Adv an adversary-compromise model, R_{Test} a role, and ρ_{Test} a mapping from $Role$ to $Agent$. We define a transition relation $\rightarrow_{P, Adv, R_{\mathrm{Test}}, \rho_{\mathrm{Test}}}$ from the execution-model rules from Figure 1 and the rules in Adv. For states s and s', $s \rightarrow_{P, Adv, R_{\mathrm{Test}}, \rho_{\mathrm{Test}}} s'$ if and only if there exists a rule in Adv or the execution-model rules with premises $Q_1(s), \ldots, Q_n(s)$ and conclusion $s \rightarrow s'$ such that all of the premises hold. We then define the function $traces$ as

$$traces(P, IK_0, Adv, R_{\mathrm{Test}}, \rho_{\mathrm{Test}}) = \{tr \mid \exists IK, th. \\ (\langle\rangle, IK_0, \emptyset) \rightarrow^*_{P, Adv, R_{\mathrm{Test}}, \rho_{\mathrm{Test}}} (tr, IK, th)\}.$$

2.6. Security properties

We provide a basic symbolic definition of session-key secrecy which, when combined with different adversary

models, gives rise to different notions of secrecy found in the literature. Other security properties, such as secrecy of general terms, symbolic indistinguishability, or different variants of authentication (e. g., from [39]), can be defined analogously in our model.

Definition 8 (Session-key secrecy): Let P be a protocol. Let R_{Test} be the role under investigation. Let IK_0 be the initial knowledge of the adversary[4]. Let Adv be an adversary model, and let the predicate $completed(tr, R, tid)$ hold iff the thread tid contains all events of role R in the trace tr. Let $RoleSub$ be the set of all substitutions of roles by agents. We then define secrecy of the session keys of R_{Test} as

$$\begin{aligned}&\forall \rho_{\mathrm{Test}} \in RoleSub.\\ &\quad \forall tr \in traces(P, IK_0, Adv, R_{\mathrm{Test}}, \rho_{\mathrm{Test}}).\forall k.\\ &(\mathrm{Test}, \mathsf{sessionkey}(k)) \in tr \wedge completed(tr, R_{\mathrm{Test}}, \mathrm{Test})\\ &\qquad \Rightarrow IK(tr) \nvdash k\,.\end{aligned}$$

Within our operational semantics, we can reinterpret different security properties as classical secrecy and authentication properties, but with respect to different adversary models. This provides a uniform view of security properties.

For example, we reinterpret the *secrecy of a session key* as defined in symbolic models as secrecy (Definition 8) with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{notgroup}}$ rule. We reinterpret *Weak Perfect Forward Secrecy* as secrecy with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rule, but not the $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule. We reinterpret *Perfect Forward Secrecy* as secrecy with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule. Finally, we reinterpret *Key Compromise Impersonation* as a form of authentication (e. g. agreement from [39]) with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$ rule.

3. Adversary and protocol hierarchies

In this section, we first define an *adversary-model hierarchy* and relate models in this hierarchy to adversarial notions from the literature. Our hierarchy enables the comparison of existing adversarial notions. Moreover, it has implications for security protocol verification. A protocol that is verified with respect to a model implies correctness with respect to all weaker models. Similarly, falsification with respect to a model implies falsification for all stronger models.

We also define a *protocol-security hierarchy* in which protocols can be compared with respect to the adversarial models in which they satisfy their security properties. This hierarchy can be used to select protocols based on the implementation requirements and the worst-case expectations for adversaries in the application domain.

4. The adversary initially knows information shared by all agents that is not secret. For example, IK_0 includes public keys and hash functions but excludes any master secret shared by all agents.

Hierarchy of adversary-compromise models. We define a partial order $\leq_{\mathcal{A}}$ on the adversary-compromise models. For all adversary-compromise models Adv and Adv', we define

$$\begin{aligned}&Adv \leq_{\mathcal{A}} Adv' \equiv \forall P.\forall M.\forall R.\forall \rho.\\ &\quad traces(P, M, Adv, R, \rho) \subseteq traces(P, M, Adv', R, \rho).\end{aligned}$$

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

Each short-term compromise rule, i. e. the last three rules in Figure 2, is incomparable to the other adversary rules with respect to $\leq_{\mathcal{A}}$, since each short-term rule introduces an event into the trace that is unique to the rule. For most realistic application scenarios, the local state will at least include the generated random numbers. In this case, the effect of the RandomReveal on the adversary knowledge will be subsumed by the effect of the StateReveal rule. However, if some random numbers are generated within a cryptographic coprocessor, this is not the case.

In contrast, each of the long-term key-reveal rules in Figure 2 introduces the same event in the trace and only the premises differ. The long-term rules can be divided into two classes: the premises of the first two rules consider the agents involved in the test session, and the other three rules have increasingly weaker premises.

As a result, we have the following relations for all adversary compromise models Adv:

$$\begin{aligned}&Adv \cup \{\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}\}\\ &\quad \leq_{\mathcal{A}} Adv \cup \{\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}\}\\ &\quad\quad \leq_{\mathcal{A}} Adv \cup \{\mathsf{LongtermKeyReveal}_{\mathsf{after}}\}. \quad (1)\end{aligned}$$

The rule $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$ only differs from $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ if the traces can contain RandomReveal events. Hence we have for all Adv:

$$\begin{aligned}&\mathsf{RandomReveal} \notin Adv \Rightarrow\\ &\quad \big(Adv \cup \{\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}\}\\ &\quad =_{\mathcal{A}} Adv \cup \{\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}\}\big). \quad (2)\end{aligned}$$

Relations between adversarial notions from the literature. We use our modular semantics to provide a uniform formalization of different adversary models, including a number of established adversary models from the computational setting [6,8,15,32,34]. This is challenging because the existing models are individually defined and therefore are not instances of a general framework. Moreover, the existing adversary models are entangled with their associated execution models and security properties.

In the computational setting, adversaries may compromise agents at any time. This capability is constrained only in the definition of the security property (commonly known as the *security experiment*), which restricts the sequences of actions considered. For example, the experiment may only

Table 1. Mapping adversary-compromise models from key-agreement literature

	Adversary rules								
	LongtermKeyReveal					Short-term reveal			
	Involvement-based		Ordering-based						
Name	notgroup	actor	after	aftercorrect	rnsafe	SessionKey	State	Random	Origin of model
Adv_{EXT}									Dolev-Yao (external) [14]
Adv_{INT}	✓								Dolev-Yao (internal) [37]
Adv_{CA}		✓							Key Compromise Impersonation [29]
Adv_{AFC}				✓	✓				Weak Perfect Forward secrecy [32]
Adv_{AF}			✓	✓	✓				Perfect Forward secrecy [24,41]
Adv_{BPR}						✓			BPR2000 [6]
Adv_{BR}	✓					✓			BR93 [7], BR95 [8]
Adv_{CKw}	✓	✓		✓	✓	✓	✓		CK2001-wPFS [32]
Adv_{CK}	✓		✓	✓	✓	✓	✓		CK2001 [15]
Adv_{eCK}	✓	✓			✓	✓		✓	eCK [34]

consider traces where the long-term key of the partner is not revealed. This effectively serves as a precondition: one may reveal the long-term key of an agent that is not the partner. In contrast, our rules explicitly formalize the preconditions as premises, thereby giving an operational interpretation of the restrictions in the security experiments. In this way, our rules capture the essence of the intruder capabilities used by the models in the computational setting.

We focus on the adversarial capabilities only. We thereby abstract from some subtle differences between the computational execution models and their interaction with the security properties. For example, the model in [15] has an execution model that restricts the agents' choice of thread identifiers. This leads to a notion of partner threads that differs from the one found in other computational models. Here we define partnering uniformly by matching histories, which provides the strongest security definition [16,17]. We refer the reader to [13,16,40] for further details.

Table 1 provides an overview of different adversary models, interpreted as instances of our modular operational semantics. As a naming convention, we write Adv_{CK} to denote the adversary model extracted from the CK model [15], and similarly for other models. In the table, a check mark (✓) denotes that the rule labelling the column is included in the adversary model named in the row. For the three ordering-based long-term key reveal queries from (1), we have also checked all weaker rules (those with additional premises). For example, since the Adv_{CK} model allows for $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, we also check $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ and $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$, because $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ can simulate their effects. We explain these interpretations in Appendix A.

In Figure 3 we show the hierarchy of adversary models defined in Table 1. An arrow $m_1 \rightarrow m_2$ denotes that the model m_1 is weaker than model m_2. We observe that in the formal methods community, the Dolev-Yao model (here Adv_{INT}) is often regarded as a very strong adversary model. In contrast, within our hierarchy, the only weaker adversary

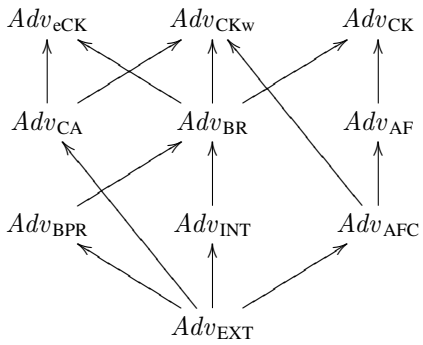

Figure 3. Hierarchy of the models from Table 1

model is the adversary model with no rules (Adv_{EXT}). All other models are either incomparable to, or stronger than, Adv_{INT}.

Protocol-security hierarchy. We now define a *protocol-security hierarchy*, which characterizes the relative strengths of different protocols.

Definition 9 (Security provided by a protocol): Let S be a set of adversary models, that is each model in S is a subset of the rules from Figure 2, and let P be a protocol. Furthermore, let *correct* be a predicate that corresponds to a security property, such as the secrecy of the session keys, with respect to a protocol P and an adversarial model Adv. We define the security of P with respect to S as

$$security(P, S) = \{Adv \in S \mid correct(P, Adv)\}.$$

Constructing the set $security(P, S)$ involves verifying the security properties of P in all adversary models Adv in S. We use the function *security* to construct a hierarchy of security protocols. We give a concrete example of such a hierarchy at the end of the next section.

Definition 10 (Protocol-security hierarchy): We say a protocol P is stronger than a protocol P' with respect to a set of adversary rules S, if and only if $security(P, S) \supset$

$security(P', S)$. For a set of protocols and a set of adversary rules, this ordering gives rise to a protocol-security hierarchy.

4. Verification of existing protocols using the Scyther tool

We have integrated our adversary-model hierarchy in a state-of-the-art symbolic protocol verification tool called Scyther [19]. We provide details on the integration and an example of using Scyther in Appendix B.

We describe here the results of analyzing a number of protocols. We also use our tool to compare the relative strengths of protocols from the perspective of their resilience to a class of adversaries, as in Definition 10.

4.1. MQV and HMQV

The MQV protocol family [32,36,47] is a class of authenticated key-exchange protocols designed to provide strong security guarantees. The elliptic curve variant of MQV is part of the NSA-B suite of cryptographic protocols [5] and is recommended for use in secret and top-secret contexts.

The HMQV protocol was proven secure with respect to the adversary model in [32], analogous to our Adv_{CKw} model, where the local state of HMQV is defined as the random values generated for the Diffie-Hellman key-exchange. Surprisingly, our tool reveals that the HMQV protocol is among a number of protocols that are, strictly speaking, incorrect in adversary models that contain either StateReveal or SessionKeyReveal rules, such as the CK model [15], if partnering is based on matching histories.

The first attack, which depends on the SessionKeyReveal rule, is a generic one that can be applied to many protocols that exchange Diffie-Hellman half-keys of the form g^x, where these half-keys are used in a symmetric way by both parties, i. e. where the session key does not include role names. In particular, the following trace can occur: Alice starts an instance of the initiator role communicating with Bob. She sends the first message, which contains g^a. Bob then starts another instance of the initiator role communicating with Alice, sending g^b. The adversary intercepts both messages and redirects Alice's message to the receive in Bob's thread, and vice-versa. We give the complete attack trace for the HMQV protocol in Appendix C.2. This mix-up is possible for many DH-style protocols. The result is that both threads (even though they are both instances of the initiator role) compute the same session key. However, the threads are *not* partners: the exchanged messages occur in the reverse order in the other thread. Hence, according to the adversary model, the adversary can use a StateReveal or SessionKeyReveal query to reveal the session key in the other (non-test) thread, violating the secrecy of the computed key. It is debatable whether this mix-up poses a practical threat, but it is clear that an adversary can force two agents into a state in which they share a session key under different assumptions.

This attack shows that a number of protocols that were proven secure in, for example, the model of [15], are strictly speaking not correct in the Adv_{CK} model. However, the protocols can easily be fixed by including in the session key a bit that links an agent to the role he is assumed to be performing, which breaks the symmetry.

The second attack involves the StateReveal rule, and implies that MQV and HMQV are not secure for Adv_{CKw}, if the final exponentiation in the computation of the session key is performed in the local state. Given the design of (H)MQV, it is possible for an adversary to reuse the inputs to this exponentiation to impersonate an agent in future sessions. A full attack trace is given in Appendix C.3.

We assume that in mission-critical implementations, (H)MQV will be fully implemented in a tamper-proof module or cryptographic coprocessor, so that the local state is empty, which would prevent this attack. At the other end of the spectrum, if (H)MQV is completely implemented in local state, the state will also include the long-term private keys, which also enables an attack where the adversary compromises these long-term keys through a StateReveal.

4.2. Signed Diffie-Hellman and variants

The original Diffie-Hellman key-exchange protocol establishes a shared key between two parties that do not share keys in advance. This protocol only satisfies its security properties in the presence of a passive adversary. When the adversary can insert messages, he can defeat the protocol by impersonating agents since the messages sent are not authenticated. A straightforward fix is for agents to sign each message sent, along with the intended recipient, using the sender's long-term private (signature) key. The resulting family of protocols is referred to as *signed Diffie-Hellman*. We have analyzed several variants, including the ISO, ISO-C (key confirmation), and DHKE-1 variants from [25], as well as the variant from [15].

Using our tool, we find attacks on the Diffie-Hellman signed protocols for all adversary models that contain the RandomReveal rule. This is consistent with the details of the various proofs in the papers [15,25], which do not consider this rule, as well as with the observation in [34] that allowing RandomReveal introduces an attack on a signed Diffie-Hellman protocol.

4.3. KEA+ and Naxos

KEA+ [35] and Naxos [34] belong to the same class of protocols. In [35], KEA+ is proven correct with respect to a variant of the adversary model [32], where the state is defined as containing only the ephemeral keys (the temporary private keys used in the Diffie-Hellman key-exchange).

We find that KEA+ and Naxos admit StateReveal attacks and therefore are incorrect in, for example, the Adv_{CK} model. The StateReveal attacks on Naxos found by the Scyther tool are described in detail in [20]. Additionally, for KEA+ we find an attack using the $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rule and hence KEA+ does not satisfy weak Perfect Forward Secrecy. This attack can not be modified to work for Naxos.

4.4. Needham-Schroeder(-Lowe) and Bilateral Key Exchange

We applied our model hierarchy to this well-studied protocol. For the Needham-Schroeder (NS) protocol, we find attacks in all models except for the external adversary model Adv_{EXT} and the session key-reveal model Adv_{BPR} (which is not surprising since the protocol does not specify a session key). The fixed Needham-Schroeder-Lowe protocol (NSL) prevents the attack by the internal adversary, modeled by Adv_{INT}. However, it is only correct in Adv_{EXT}, Adv_{INT}, and Adv_{BPR}. We find attacks in all other models. The Bilateral Key Exchange (BKE) protocol is correct in the same models as the Needham-Schroeder-Lowe protocol.

4.5. Yahalom

[44] presents two versions of the Yahalom protocol. The original version of the Yahalom protocol allows the adversary to reuse old keys. As a result, compromise of an old session key can lead to attacks on future sessions. Paulson uses the Isabelle theorem prover to prove that an improved version of the protocol does not suffer from this attack. He proves that the loss of one session key does not lead to attacks on other session keys.

Using our tool, we find attacks involving SessionKeyReveal queries on both of these protocols. At first sight, this appears to contradict Paulson's result. This discrepancy is due to the difference in the property considered. In [44], the adversary may compromise *other* session keys. Whereas our model, following the definitions from key-agreement literature, allows the adversary to compromise session keys of threads that are not partners of the test thread.

4.6. Example protocol-security hierarchy

In Figure 4 we show the protocol-security hierarchy (cf. Definition 10) of a set of protocols from the literature, with respect to the adversary models from Table 1 for secrecy. Details of the verification results can be found in Appendix C.1. Nodes correspond to (a set of) protocols that are correct in the same adversary models. The second line of each node lists in which models the protocols are correct, where we omit weaker adversary models based on our model hierarchy, for example, Needham-Schroeder-Lowe is also correct in Adv_{EXT}. In the hierarchy, an arrow $n_1 \rightarrow n_2$ implies that the protocols in n_2 are stronger than those in n_1, with respect to Definition 10. Using our tool, it is possible to automatically generate such hierarchies for a set of protocols. Protocol-security hierarchies provide a novel mechanism for choosing an optimal protocol for a given application domain, for example, exchanging a secret as illustrated here.

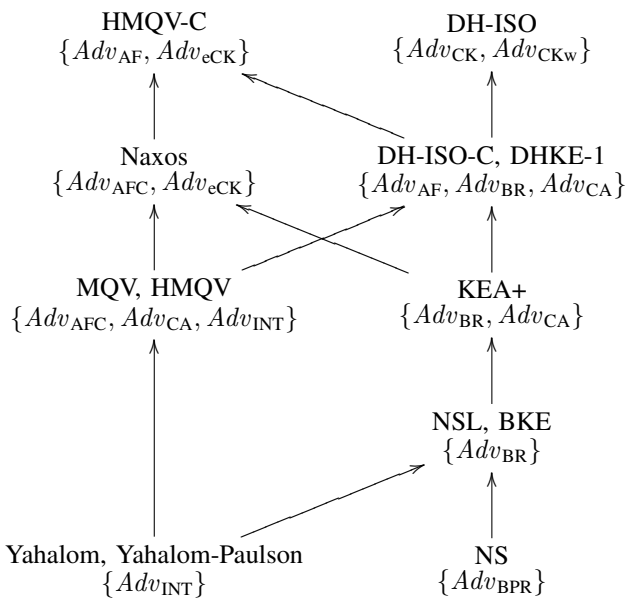

Figure 4. Protocol-security hierarchy (secrecy)

5. Related work

Related work in computational analysis. Most research on adversary compromise has been performed in the context of authenticated key-exchange protocols and extensions for group key agreement. The analysis has been with respect to the computational models of e. g. Canetti and Krawczyk [15,32], Shoup [45], Bellare et al. [6]–[8], Katz and Yung [31], LaMacchia et al. [34], and Bresson, Manulis et al. [12].

In general, any two computational models are incomparable due to (often minor) differences not only in the adversarial notions, but also in the definitions of partnership (variants of the notion of matching sessions used here), the execution models, and security property specifics. As these models are generally presented in a monolithic way, where all parts are intertwined, it is difficult to separate these notions. Details of some of these definitions and their relationships have been studied by, for example, Boyd, Choo, et al. [16,17], Bresson et al. [13], LaMacchia et al. [34], and Menezes and Ustaoglu [40].

In [34,42,47] it is argued that the eCK adversary model is stronger than the CK model. The (informal) reasoning used is that if the random numbers (ephemeral secrets) are

interpreted as containing the protocol's local state, revealing the random numbers means revealing the state. The Adv_{CK} model allows for $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, which is a stronger rule than $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$ from Adv_{eCK}. However, the Adv_{eCK} model allows for RandomReveal which is not allowed in Adv_{CK}. Therefore we have that the Adv_{CK} and Adv_{eCK} adversary models are incomparable, which is in line with the analysis of CK and eCK in [11].

The CryptoVerif tool by Blanchet [10] is an automated tool for computational analysis. Its adversary model covers Adv_{INT} (corresponding to static corruption, or the classical Dolev-Yao adversary) but none of the stronger notions present in our current work.

Related work in symbolic analysis. In the symbolic analysis setting, Guttman [28] has modeled a form of forward secrecy. With respect to verification, the only work we are aware of is where researchers have verified (or discovered attacks on) key-compromise related properties of particular protocols. These cases do not involve a compromising adversary model, but are ad-hoc constructions of key compromise, made for specific protocols, which can be verified in a Dolev-Yao style adversary model.

In [1], Abadi, Blanchet, and Fournet analyzed the key-establishment protocol JFK in the Pi Calculus and proved that it achieves perfect forward secrecy. This was proved by giving the long-term keys of all principals to the attacker at the end of the protocol run. This corresponds to manually instrumenting the analog of our $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule in their setting.

As noted in Section 4.5, Paulson used his inductive approach to reason about the compromise of short-term data. To model compromise, he adds a rule to the protocol, called *Oops*, that directly gives short-term data to the adversary. The *Oops* rule is roughly analogous to our SessionKeyReveal rule, omitting the partner check, which accounts for the difference we previously reported. Paulson did not explore compromise in general. Conversely, we have not used our rules for inductive theorem proving, although this should be possible along the lines of Paulson's work.

Gupta and Shmatikov [25,26] link a symbolic adversary model that includes dynamic corruptions to a particular adversarial model used in the computational analysis of key-agreement protocols. They describe in [26] a cryptographically-sound logic that can be used to prove security in the presence of adaptive corruptions, that is, the adversary is able to obtain the long-term private keys of agents dynamically.

6. Conclusions

We have provided the first symbolic framework capable of modeling a family of adversaries endowed with different compromise capabilities. Our adversaries extend the Dolev-Yao adversary with capabilities for the dynamic compromise of both long-term and short-term data. Based on this framework, we developed the first automated analysis tool capable of handling notions such as weak perfect forward secrecy, key compromise impersonation, compromise of (parts of) the state of a session, and malicious random number generators. We used the tool to discover attacks on a number of protocols for adversary models in which they were previously proven to be secure.

Our hierarchy includes many relevant adversarial notions for which (to the best of our knowledge) no efficient protocols have yet been proposed. For example, in the context of tamper-proof modules, it is reasonable to assume that the long-term secrets are never compromised, but that the local state may be. An interesting problem for future work is thus to design optimized protocols for these scenarios, which correspond to adversary compromise models that do not contain LongtermKeyReveal rules.

Based on the adversarial models, we define a protocol-security hierarchy that characterizes the security provided by different protocols. As future work, it would be interesting to extend this protocol hierarchy to a larger class of protocols and to compare systematically the merits of protocols that provide similar security guarantees.

Our adversarial models are independent of the security properties under consideration. Hence, another interesting research direction is to study how our stronger adversarial notions influence security properties such as, for example, anonymity or resistance to Denial-Of-Service attacks.

References

[1] M. Abadi, B. Blanchet, and C. Fournet. Just Fast Keying in the Pi calculus. *ACM Transactions on Information and System Security (TISSEC)*, 10(3):1–59, July 2007.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, L. Cuellar, P. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, 17th International Conference, CAV 2005, Proc.*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer-Verlag, 2005.

[3] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.

[4] M. Backes, S. Lorenz, M. Maffei, and K. Pecina. The CASPA tool: Causality-based abstraction for security protocol analysis. In A. Gupta and S. Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123 of *Lecture Notes in Computer Science*, pages 419–422. Springer-Verlag, 2008.

[5] E. Barker, D. Johnson, and M. Smid. NIST SP 800-56: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised), March

2007. http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf.

[6] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, Lecture Notes in Computer Science, pages 139–155. Springer-Verlag, 2000.

[7] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag.

[8] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 1995. ACM.

[9] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society, 2001.

[10] B. Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, Oakland, California, May 2006.

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

[12] E. Bresson and M. Manulis. Securing group key exchange against strong corruptions. In *ASIACCS*, pages 249–260. ACM, 2008.

[13] E. Bresson, M. Manulis, and J. Schwenk. On security models and compilers for group key exchange protocols. In *IWSEC*, volume 4752 of *Lecture Notes in Computer Science*, pages 292–307. Springer-Verlag, 2007.

[14] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

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

[16] K.-K. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment proofs. In *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 624–643. Springer-Verlag, 2005.

[17] K.-K. Choo, C. Boyd, Y. Hitchcock, and G. Maitland. On session identifiers in provably secure protocols. In *Security in Communication Networks*, volume 3352 of *Lecture Notes in Computer Science*, pages 351–366. Springer-Verlag, 2005.

[18] J. Clark and J. Jacob. A survey of authentication protocol literature, 1997. http://citeseer.ist.psu.edu/clark97survey.html.

[19] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer-Verlag, 2008.

[20] C. Cremers. Session-state reveal is stronger than ephemeral key reveal: Breaking the NAXOS key exchange protocol. Cryptology ePrint Archive, Report 2008/376, 2008. http://eprint.iacr.org/.

[21] C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128, New York, NY, USA, 2008. ACM.

[22] C. Cremers, S. Mauw, and E. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, pages 139–161, 2006.

[23] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:667–721, 2003.

[24] C. Günther. An identity-based key-exchange protocol. In *Advances in Cryptology - EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 29–37. Springer-Verlag, 1990.

[25] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proc. of the 2005 ACM workshop on Formal methods in security engineering (FMSE 2005)*, pages 23–32. ACM, 2005.

[26] P. Gupta and V. Shmatikov. Key confirmation and adaptive corruptions in the protocol security logic. In *FCS-ARSPA'06*, 2006.

[27] P. Gutmann. Abstract performance characteristics of application-level security protocols. draft paper at www.cs.auckland.ac.nz/~pgut001/pubs/app_sec.pdf.

[28] J. D. Guttman. Key compromise, strand spaces, and the authentication tests. *Electr. Notes Theor. Comput. Sci.*, 45, 2001.

[29] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology-ASIACRYPT 1996*, volume 1163 of *Lecture Notes in Computer Science*, pages 36–49, 1996.

[30] S. Kamara and J. Katz. How to encrypt with a malicious random number generator. In *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 303–315. Springer-Verlag, 2008.

[31] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer-Verlag, 2003.

[32] H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer-Verlag, 2005.

[33] R. Küsters and T. Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31*, pages 129–138. ACM, 2008.

[34] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2007.

[35] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 378–394, 2006.

[36] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

[37] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[38] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 18–30. IEEE Computer Society, 1997.

[39] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society, 1997.

[40] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In *Proceedings of ACISP 2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 53–68, 2008.

[41] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[42] T. Okamoto. Authenticated key exchange and key encapsulation in the standard model. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 474–484, 2007.

[43] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[44] L. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[45] V. Shoup. On formal models for secure key exchange (version 4), Nov. 1999. revision of IBM Research Report RZ 3120 (April 1999).

[46] US-CERT. Debian/ubuntu openssl random number generator vulnerability, 2008. http://www.us-cert.gov/cas/techalerts/TA08-137A.html.

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

Appendix A.
Extracting adversary models from security definitions for Secure Key Agreement

Here we further explain each of our adversary rules from Section 2.

Security definitions for key-agreement protocols. Security definitions in computational settings differ substantially from the corresponding notions in symbolic models. Here we describe the relations between both types of definition.

In symbolic models, a single role instance of a protocol is referred to as thread, strand, or run. In computational models, the same notion is either called an oracle or a session.

In symbolic models, a possible execution history of a protocol is referred to as a trace or a run; the notion of bundle roughly corresponds to a set of traces. A protocol is said to satisfy a (trace) security property if the property holds for all protocol traces.

In security definitions for key-agreement protocols there is a notion called an experiment: it roughly corresponds to a set of traces, in which the adversary additionally performs a *Test* query and a *Guess* query. Intuitively, the *Test* query fixes the thread in the traces for which the security property is considered, similar to the test thread in our framework. Whereas in a symbolic model, secrecy is defined such that the adversary may not learn a term (e. g. the session key) of the test thread, in computational models we have that the adversary must not be able to distinguish it from a random bit string. To model this, the *Test* query is defined as flipping a coin, and returning either the session key of the thread under consideration, or a random key. The secrecy property is said to hold if the adversary has at most a negligible advantage in guessing the outcome of the coin flip over a random guess.

In computational models, the adversary capabilities are commonly modeled as so-called queries, e. g. *session-state reveal* and *session-key reveal* in [15]. Queries have no explicit preconditions, in contrast to the premises of our adversary rules. Instead, the restrictions on the adversary capabilities are enforced by placing restrictions on the experiments (corresponding to traces) that may be considered for a *Test* query. For example, one considers only *Test* queries for sessions (threads) that are "completed, unexpired and unexposed" [15, p. 13–14]. These three definitions together restrict the set of allowed experiments (traces) in a manner similar to the premises in our adversary rules for the corresponding Adv_{CK} model. In other computational models, similar predicates are known under the names "clean" or "fresh".

Origins of adversary rules. The $\mathsf{LongtermKeyReveal}_{\mathsf{notgroup}}$ rule can be found implicitly in many symbolic models. In protocol-verification tools, it usually corresponds to the fact that the initial adversary knowledge contains the long-term private key of one or more agents, e. g. $sk(\mathrm{Eve}) \in IK_0$. Correspondingly, security properties are verified for threads not executed by, or communicating with, the agent Eve. In symbolic protocol logics, the rule is often represented by an honesty predicate, as e. g. in [23]. In computational models, this rule corresponds to the notion of static corruption and is modeled by a constraint on the security experiment that the agents involved in the test session (test thread) are not corrupted.

The $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$ rule allows the adversary to learn the private key of the agent executing the test thread. This rule allows for so-called Key Compromise Impersonation attacks as defined in e. g. [29]. In models that capture these attack, it is assumed that the adversary can learn the long-term key of the agent executing the test thread. The complexity of the premise of the rule stems from the fact that a single agent may execute multiple roles within a session: if Alice decides to send a message to another thread of Alice (e. g. a different computer owned by Alice), we do not allow the long-term private key of Alice to be compromised, even though she is the actor. We note that the rule is not very common in security models. We also have that, strictly speaking, the corresponding experiment in the eCK model [34] disallows learning the long-term private key of the actor if there is also an $\mathsf{RandomReveal}$ query in the same experiment (trace).

The $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule corresponds to the adversary being able to learn any long-term private keys after the test thread ends. In the context of verifying secrecy, this corresponds to Perfect Forward Secrecy [24,41]. For the ad-hoc symbolic verification of Perfect Forward Secrecy, this can be modeled as sending all long-term private keys to the adversary at the end of the test thread. This property is therefore within the scope of most existing symbolic analysis tools.

For $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, we have the additional premise that partner threads must exist for the test thread. This additional condition was introduced in [32] in tandem with the negative result that no two-party key-agreement protocol based only on public-key cryptography can satisfy Perfect Forward Secrecy. The negative result in [32] depends on the construction of a generic attack involving the $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule. If the adversary model includes the weaker $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rule instead of the $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule, the generic attack cannot be constructed. Hence, the corresponding notion of weak Perfect Forward Secrecy is defined in [32]. It is intended to serve as a slightly weaker security property that is satisfiable by two-party key-agreement protocols that are based on public-key cryptography only.

For $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$, we have a further premise that no $\mathsf{RandomReveal}$ event has previously occurred in a thread of the agent. This additional condition stems from [34], where the adversary is excluded from both learning random values

generated by an agent a as well as the long-term private keys of a in a given trace (experiment).

The SessionKeyReveal rule corresponds to revealing a session key. The underlying observation is that the communication partners in a correct protocol session will share a key. Revealing this key to the adversary would break any key-agreement protocol. The idea is to allow the adversary to reveal the key of any non-partner thread. As a result, the effect of the SessionKeyReveal rule is tightly coupled with the definition of partner threads. (We will return to the notion of partner threads below.) The SessionKeyReveal rule used here stems from [6].

Similar to SessionKeyReveal, the StateReveal rule has a premise that no partner may have its local state revealed. This premise stems from the corresponding *session-state reveal* query in the CK model [15]. However, in [15] the contents of the local state are not defined inside the model, and are assumed to be known to the protocol prover. In proofs that use the CK model, the local state is defined as containing only the locally generated values, e. g. the ephemeral keys (private exponents) in a Diffie-Hellman key-exchange. For a simple signed Diffie-Hellman protocol, this seems to be the only relevant data that is computed privately and is not revealed during communication. However, it is clear that for many protocols this underapproximates the local state.

In [34] one can find a query called *ephemeral-key reveal*. This query is intended to resolve any ambiguities in the *session-state reveal* query. It strictly contains the ephemeral secrets generated in a thread, which matches our RandomReveal rule. The premise of the RandomReveal rule allows for compromise of the partner threads, but only if the long-term key of the agent executing the thread is not revealed.

Execution model variations. In the literature for secure (group) key-exchange protocols, it is common to specify the execution model within the security definition, making the security definitions self-contained. We refer the reader to [16,40] for a more detailed comparison. For example some models, such as [15], assume that unique session identifiers are externally provided, whereas most other models do not assume that such identifiers are a priori available. This influences both the allowed behaviours as well as the partner functions described below.

Partner function variations. There are a number of possibilities for defining the partner threads of a particular thread. In literature one can find, for example, partnering defined using an existentially-guaranteed partner function (see [8]), partnering by an explicit session identifier specified in the protocol, or by means of (some definition of) matching sessions. Each of these choices has its own justification.

We observe that for protocols that function correctly in the *absence* of an adversary, threads in matching sessions will compute the same session keys. Hence, allowing SessionKeyReveal queries on these threads implies that no such protocol satisfies secrecy. Inverting this condition, i. e. allowing SessionKeyReveal on all threads *not* in matching sessions, therefore seems to provide the strongest possible adversary capability with respect to SessionKeyReveal.

Corrupt queries. Our model does not contain explicit corrupt events, contrary to most computational adversary models. The rationale for this design choice is that a corrupt event may simultaneously reveal all long-term and short-term secrets of a thread, and is therefore a combination of our other adversarial capabilities. In our model, we therefore represent corrupt queries as trace subsequences consisting of adversary events that involve LongtermKeyReveal events.

Erasure of state. In many computational protocol models, explicit actions are included to mark the erasure of state within a thread. Furthermore, most models assume that when a thread ends, its local state is erased.

In our model, we also assume that when a thread ends, its local state is erased. This is reflected in the premises of our state and key reveal rules by checking that the thread has not ended. However, we have no explicit erasure actions in our protocol language. This choice can be justified by observing that in most computational models, the preconditions for allowing a state or key compromise only depend on whether the compromised thread is a partner of the test thread. No distinction is made with respect to the progress within the step and hence the erasure commands specified in the protocols are ignored in computational proofs.

Appendix B. Tool support: the Scyther tool

We have extended the symbolic security-protocol verification tool Scyther [19,21] with our adversarial rules from Figure 2.

We describe below the two main technical hurdles that we faced. First, we explain how we adapt existing protocol-execution models and the corresponding security properties. Second, in general the internal state is not included the protocol specification. We therefore describe the procedure we use to infer the local state.

Adapting existing tools. In standard symbolic verification tools and the corresponding protocol descriptions, agents are considered to be either fully honest or fully dishonest. The first change is to lift this restriction. Second, we must incorporate new protocol events for marking local state and local secrets, as well as the new adversary rules. The addition of the new events and rules complicates the execution model, which makes verification more expensive. Depending on the verification algorithm, integrating the correct premises for each of the adversary rules may require significant changes. For protocol verification methods that dynamically generate threads over which the security properties are checked, care must be taken to define a particular Test thread.

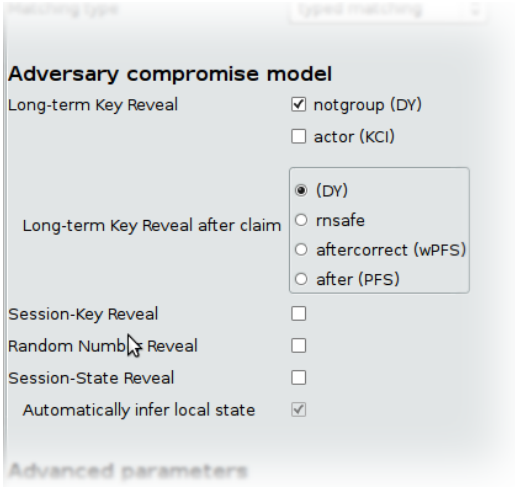

Figure 5. Choosing an adversary model in the tool

A further change is required to modify the security properties. In existing verification methods, these properties (or the scenarios that encode them) involve conditions of the form "if the agents are honest." In most cases, this condition can be dropped, as its role is now handled by the premises of the adversary rules. Depending on the tool, this requires either rewriting the tool or the protocol description files.

Automated inference of the local state from abstract protocol descriptions. In general, protocol specifications do not describe the local state of the protocol. This is due in part to the inherently abstract and functional nature of such specifications, as well as to the complexity and diversity of actual implementations.

As a result, defining the local state of an abstract protocol is at best an underapproximation of the actual local state. Our approach is to infer which data must be present and, although still underspecified, this allows us to find attacks that should function for any implementation. Conversely, particular implementations may still include additional data in their local state, which may enable additional StateReveal attacks.

Given a set of terms S from an event in a role, the function Φ determines the subterms that must be part of the local state.

$$\Phi(S) = \{t \sqsubseteq S \mid \exists t'.t' \sqsubseteq t \wedge t' \in (Fresh \cup Var)\}$$

Hence, all locally computed subterms are inferred from the set of terms S.

The idea behind the StateReveal rule is to capture the distinction between the data in local memory and other data (including long-term keys) that is inside more protected memory, such as a tamper-proof module. We therefore define Φ in such a way that long-term keys are excluded from the local state. The simplicity of our definition depends on the fact that in the model described here, all functions

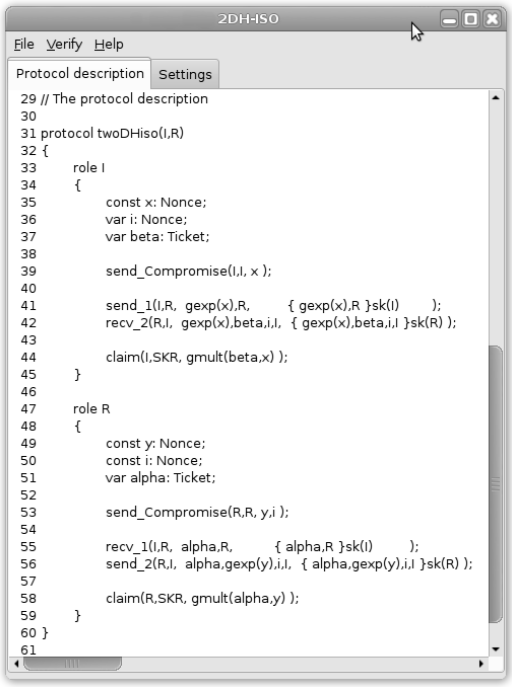

Figure 6. Example input

are assumed to be one-way. Hence the local state may include $f(sk(a), N)$ (N a fresh term) for some function f. If invertible functions are required in the model, then Φ can be suitably extended.

Given a sequence s of events in a role that does not include state events, we automatically insert state events by inference from the other events, by using the function Ψ:

$$\Psi(\langle ev(x)\rangle^\frown s) = \begin{cases} \langle ev(x), \mathsf{state}(\Phi(\{x\}))\rangle^\frown \Psi(s) & \text{if } x \in Term, \text{ and} \\ \langle ev(x), \mathsf{state}(\Phi(x))\rangle^\frown \Psi(s) & \text{if } x \in \mathcal{P}(Term). \end{cases}$$

Example 3 (Automated inference of state *events.):* Let P be the protocol from Example 1. We define the protocol P' as $P'(R) = \Phi(P(R))$ for all $R \in dom(P)$, resulting in:

$$\begin{aligned} P'(\mathrm{Init}) = \langle & \mathsf{generate}(\{n\}), \\ & \mathsf{state}(\{n\}), \\ & \mathsf{send}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}), \\ & \mathsf{state}(\{\mathrm{Resp}, \{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}, n\})\rangle \\ P'(\mathrm{Resp}) = \langle & \mathsf{recv}(\mathrm{Init}, \mathrm{Resp}, \{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}), \\ & \mathsf{state}(\{\{\!|\, \mathrm{Resp}, n \,|\!\}^a_{sk(\mathrm{Init})}, n\})\rangle. \end{aligned}$$

Users of the tool can choose between manual specification of the state (by manually inserting appropriate state events in the protocol specification) and automated inference of the state. Currently, our tool supports secrecy and several notions of authentication.

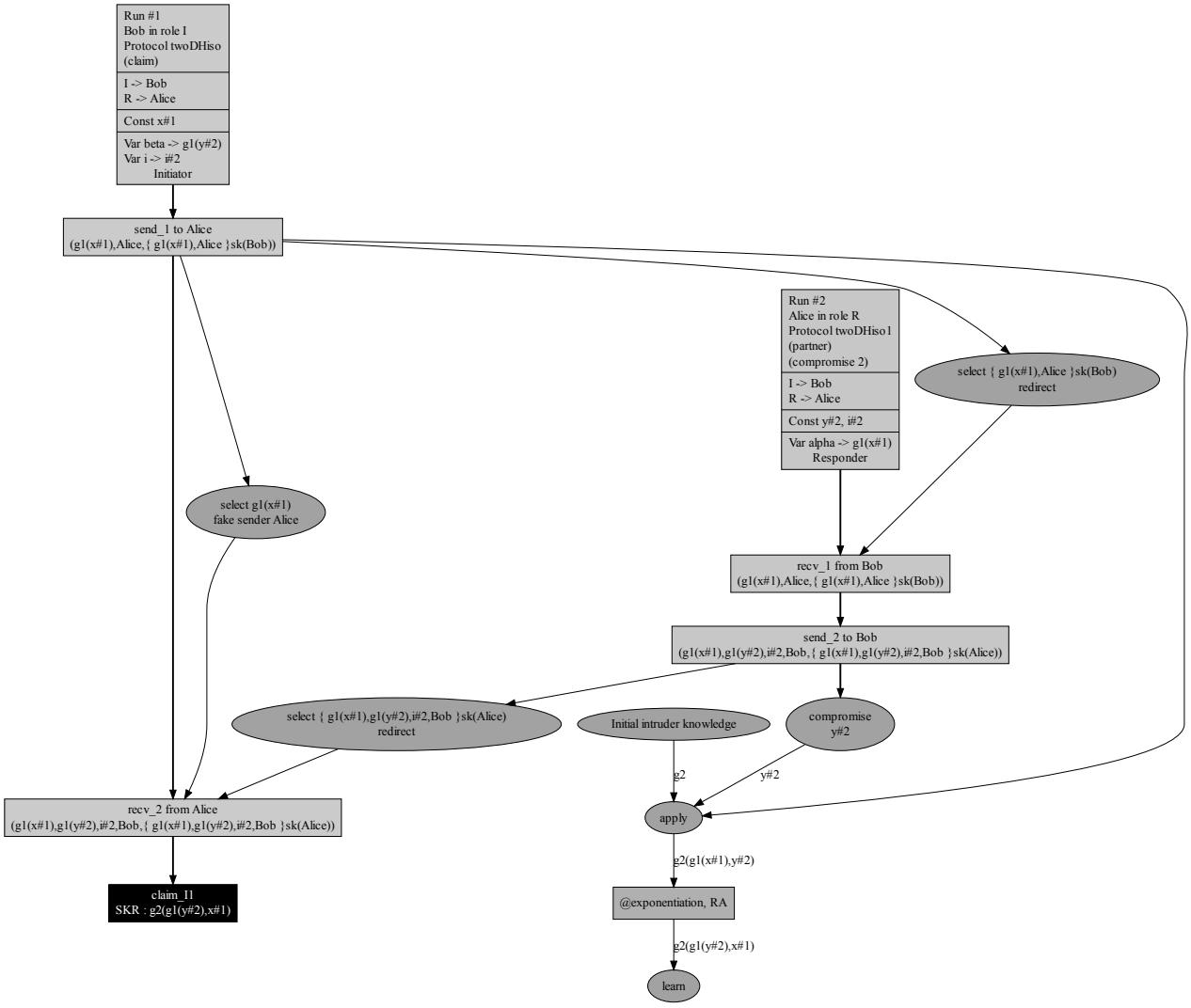

Figure 7. Example attack output

Protocol analysis example. In Figure 6 we provide an example of a protocol description as it appears in the tool. The protocol considered is the DH-ISO protocol.

Protocol verification depends on the selected adversary model. The choices provided correspond exactly to the adversary rules presented in this paper, as can be seen in Figure 5. Given a protocol with a specific local state (automatically inferred or manually specified), the Scyther tool allows one to verify or find attacks on the protocol with respect 112 different adversary models, corresponding to the different combinations of the adversary rules in Figure 2, taking into account the relations in Equation (1). This includes all the models in Table 1.

We provide an example attack output in Figure 7. The output graph shows the individual agent threads and adversary events. Thread events are depicted as rectangles, where each event is colored according to the role of the thread it occurs in. Adversary events are represented by ellipses. In the graph, a generated fresh term $x\sharp 1$ is denoted by "Const $x\sharp 1$". The example shown is an attack using the RandomReveal rule on the signed Diffie-Hellman protocol. Hence the protocol is not secure in any adversary model that includes the RandomReveal rule.

The current version of the Scyther tool does not directly support algebraic properties. For Diffie-Hellman style protocols, where the algebraic property $(g^a)^b = (g^a)^b$ is relevant, we adopt an approach similar to the approach taken for XOR in [33]. We extend the protocol specification with a set of

Table 2. Summary of attacks found

	Adv_{EXT}	Adv_{INT}	Adv_{CA}	Adv_{AFC}	Adv_{AF}	Adv_{BPR}	Adv_{BR}	Adv_{CKw}	Adv_{CK}	Adv_{eCK}
BKE			×	×	×			×	×	×
DH-ISO										×
DH-ISO-C								×	×	×
DHKE-1								×	×	×
HMQV-C								×	×	
HMQV					×	×*	×*	×	×	×*
KEA+				×	×			×	×	×
MQV					×	×*	×*	×	×	×*
Naxos					×			×	×	
NS		×	×	×	×		×	×	×	×
NSL			×	×	×			×	×	×
Yahalom			×	×	×	×	×	×	×	×
Yahalom-Paulson			×	×	×	×	×	×	×	×

rewrite rules that allow the adversary to replace $(g^a)^b$ with $(g^a)^b$ in subterms.

Analysis of the protocol from Figure 6 with respect to Adv_{EXT} takes 0.1 seconds on an Intel Centrino laptop (1.2 GHz) with 3 GB memory. Analyzing the same protocol with respect to Adv_{CK} takes 38.5 seconds. This significant increase in verification time is mainly due to the inclusion of short-term reveal rules, both because they extend the adversary knowledge with complex terms and because checking their premises involves evaluation of the $Partners$ function.

Appendix C. Attack details

C.1. Verification result summary

In Table 2 we summarize the attacks found using our tool on ten protocols with respect to the adversary models from Table 1. A cross (×) in the table denotes that an attack was found. A starred cross (×*) denotes a generic role mix-up attack as described in Section 4. The protocol-security hierarchy in Figure 4 is derived from Tables 1 and 2.

C.2. Attack on HMQV using SessionKeyReveal

In Figure 8 we show the attack on the HMQV protocol described in Section 4.1. For full details of the protocol we refer the reader to [32].

We use the following notation in describing the attack. The terms a and b refer to the long-term private keys of Alice and Bob, respectively. The corresponding public keys are $A = g^a$ and $B = g^b$. H and $\bar{H}$ are hash functions. We use the abbreviations $d = \bar{H}(X, \text{Bob})$ and $e = \bar{H}(Y, \text{Alice})$.

In the attack, both Alice and Bob start threads in the initiator role, trying to communicate with each other. Due to the symmetry of the protocol, both parties can complete the protocol even though they are executing the same role. Then, the symmetry of the session key computation results in both threads computing the same session key, based on the algebraic properties of the modular exponentiation. By the definition of matching histories, the threads are not partners, as the labels of the sends and receives do not match. Hence the adversary can use SessionKeyReveal to reveal the key of the thread of Bob.

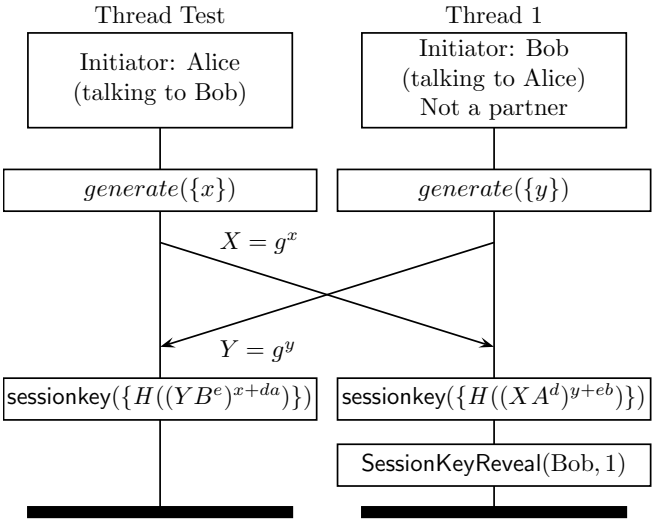

Figure 8. HMQV attack for adversaries capable of SessionKeyReveal, e. g. Adv_{BPR}.

Note that this attack is possible in the original CK model from [15], but not in the adapted CK model used as the HMQV security model in [32]. In the model from [32], the partnering function is based on session identifiers that are specifically constructed for DH-style protocols in such a way that both threads in the attack are considered partners. As a result, the model in [32] disallows SessionKeyReveal for Bob's thread.

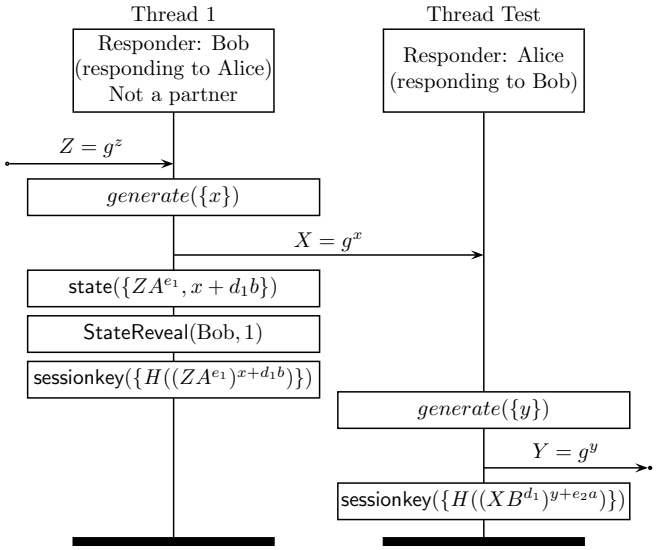

Figure 9. HMQV attack for adversaries capable of StateReveal, e. g. Adv_{CK}.

C.3. Attack on HMQV using StateReveal

In Figure 9 we show an attack on HMQV using StateReveal, where we assume that the inputs to the final exponentiation are part of the session-state of the protocol. This corresponds to performing the final exponentiation in unprotected memory.

We define $d_1 = \bar{H}(X, \mathrm{Bob})$, $e_1 = \bar{H}(Z, \mathrm{Alice})$ and $e_2 = \bar{H}(Y, \mathrm{Alice})$.

The attack starts by Bob receiving a message g^z that is apparently coming from Alice. This message may have been sent by an agent or have been generated by the adversary. Next, Bob generates x and sends $X = g^x$, which is intercepted by the adversary.

Thread 1 is not a partner of the test thread because its history does not match that of the test thread. Hence the adversary is able to compromise the state of thread 1, which gives him access to $x + d_1 b$.

At any desired time, the adversary sends X to the responder Test thread of Alice. Alice computes $Y = g^y$ and sends it. Now Alice computes the session key based on X and y. The adversary intercepts Y, after which he can compute $H((YA^{e_1})^{x+d_1 b})$, which yields the session key of the test thread.