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

Version 1.1, November 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. Our semantics can be used as a basis for protocol analysis tools. As an example, we extend an existing symbolic protocol-verification tool with our adversary models. The result is the first tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of so-called strong corruptions and state-reveal queries. As further applications, we use our model hierarchy to relate different adversarial notions, gaining new insights on their relative strengths, and we use our tool to find new attacks on protocols.**

I. 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 [21,37]. 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." Under this limited view, it is impossible to distinguish between the security guarantees provided by early key-exchange protocols such as the Bilateral key-exchange [16] and state-of-the art protocols such as (H)MQV [28,33]. It is also impossible to discern any benefit from storing the long-term keys in a tamper-proof module or from performing part of a computation in a cryptographic coprocessor. Despite this, symbolic methods have the advantage that there are numerous effective tools for symbolic protocol analysis, such as [2,8,17].

In contrast to the above, researchers in the computational setting, e. g. [11,13,26,31,40], 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, drawbacks to the above computational models. First, these models have been defined just for key-agreement protocols, whereas one may expect similar definitions to exist for any security protocol. Second, their definitions are given in a monolithic fashion, where the security model simultaneously formalizes the execution model of the protocol, the adversary capabilities, and the desired security property. This complicates the comparison of models as it is non-trivial to factor out these elements from the monolithic whole. Third, contrary to the security models used in symbolic approaches, there is no automated tool support available for the stronger adversarial notions from the 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 serve 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 short-term and long-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 analogs of the 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 models are orthogonal to the execution model and security properties. Although presented in a symbolic and possibilistic context, our modular adversary models 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. As an example, we extend an existing symbolic protocol analysis tool [17] to reason about security properties of protocols in the presence of adversaries in our hierarchy. The resulting tool is the first automated tool that systematically 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 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 the tool to classify their relative strengths against different forms of adversarial compromise.

Organization. We define our operational semantics and associated security properties in Section II. In Section III, we define a hierarchy of adversary models, relate models in the hierarchy to existing adversarial notions, and define our protocol-security hierarchy. In Section IV, we use an extended version of the Scyther tool to verify different protocols with respect to models in our hierarchy. We discuss related work in Section V and draw conclusions in Section VI. In the Appendix, we further explain the design choices behind our adversary models, describe our tool and its use, and provide details on 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 [24].

II. 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.

Our framework is based on a simple operational semantics that is compatible with the majority of existing semantics for security protocols, including semantics based on traces and strand-spaces. 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.

A. 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, which 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 sequences 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 $s\hat{\ }s'$ for the concatenation of the sequences s and s'. Finally, abusing set notation, we write $e \in s$ iff $\exists i. s_i = e$, and write $set(s)$ for $\{x \mid x \in s\}$.

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.

B. 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. For technical reasons,

[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.

we assume that TID contains two distinguished thread identifiers, Test and $\mathrm{tid}_{\mathcal{A}}$. The thread Test represents an agent thread: we will formulate both the security properties and the adversary capabilities from the local view of an arbitrary agent thread.[3] The thread $\mathrm{tid}_{\mathcal{A}}$ represents the adversary thread. In this thread, the adversary events, introduced later, are executed.

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

Definition 1: Basic terms

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

Definition 2: Terms

$$\begin{aligned} \mathit{Term} ::= {} & \mathit{BasicTerm} \mid (\mathit{Term}, \mathit{Term}) \\ & \mid pk(\mathit{Actor}) \mid sk(\mathit{Actor}) \mid k(\mathit{Actor}, \mathit{Actor}) \\ & \mid \{\!|\, \mathit{Term} \,|\!\}^{a}_{\mathit{Term}} \mid \{\!|\, \mathit{Term} \,|\!\}^{s}_{\mathit{Term}} \mid \mathit{Func}(\mathit{Term}^{*}) \end{aligned}$$

For each $X, Y \in \mathit{Agent}$, $sk(X)$ 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 of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^{s}_{t_2}$ denotes symmetric encryption. Elements of the set Func can be used to model other cryptographic functions, such as hash functions. Freshly generated terms and variables are assumed to be local to a thread. We model constants as 0-ary functions.

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 $pk(X)^{-1} = sk(X)$, $sk(X)^{-1} = pk(X)$ for all $X \in \mathit{Agent}$, 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 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:

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

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 $t_2 \sqsubseteq \{\!|\, t_1 \,|\!\}^{s}_{t_2}$.

An agent can engage in the following events.

Definition 3: Agent events

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

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. As is standard, the adversary receives all messages sent, independent of the intended recipient.

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

Definition 4: Adversary events

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

We assume all adversary events are executed in the single adversary thread $\mathrm{tid}_{\mathcal{A}}$.

We will explain the interpretation of the 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.

$$\mathit{Event} = \mathit{AgentEvent} \cup \mathit{AdvEvent}$$

C. Protocols and threads

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

[3] Fixing the name of this "point of view" thread to Test is standard in computational approaches, and simplifies many of the formulas.

Example 1 (Simple protocol): Let $key \in \mathit{Fresh}$ and $x \in \mathit{Var}$. Let P be the protocol defined as follows.

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

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 protocol steps in each party mark session keys and state information.

Protocols are executed by agents that execute roles, thereby instantiating role names with agent names. We define a function $\mathit{thread} : (\mathit{Protocol} \times \mathit{Role} \times \mathit{TID} \times \mathcal{S}ub) \rightarrow \mathit{AgentEvent}^*$. Given a protocol, a role, a thread identifier, and a substitution mapping role names to agents, *thread* yields the sequence of agent events that may occur in a thread.

Definition 5 (Thread): Let P be a protocol, let $R \in dom(P)$, $tid \in \mathit{TID}$, and let σ be a substitution such that $dom(\sigma) = dom(P)$. Then $\mathit{thread}(P, R, tid, \sigma) = \sigma'(\sigma(P(R)))$, 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 $\sigma' = \bigcup_{cv \in CV(\sigma(P(R)))} [cv \sharp tid / cv]$.

Example 2: Let P be the protocol from Example 1, $t_1 \in \mathit{TID}$, and $\{A, B\} \subseteq \mathit{Agent}$. For a thread t_1 performing the Init role we have $\sigma' = [key\sharp t_1 / key]$ and

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

D. Execution model

We define the set *Trace* as $(\mathit{TID} \times \mathit{Event})^*$, which represents possible execution histories. The state of our system is a triple $(tr, IK, th) \in \mathit{Trace} \times \mathcal{P}(\mathit{Term}) \times (\mathit{TID} \rightharpoonup \mathit{Event}^*)$, whose components are (1) a trace tr, (2) the adversary's knowledge IK, and (3) a partial function th mapping thread identifiers of initiated threads (executing or completed) to event traces. We include the trace as part of the state to facilitate defining the partner function later.

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 keys. The adversary may learn these by performing LongtermKeyReveal events.

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

Execution-model rules. The create rule starts a new instance of a protocol role R (a *thread*), executed by an agent $\rho(R)$. 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 $dom(P)$, which can occur in events associated with the role, are replaced by agent names by the substitution ρ.

The createTest rule starts the test thread. It is similar to the create rule, but instead of choosing arbitrary role and agent assignments, the rule takes as additional parameters R_{Test} and σ_{Test}. These parameters represent the test role, and the agent to role and variable assignments of the test thread, respectively. These parameters will be instantiated in the definition of traces in Definition 8.

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. 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.

The last three rules support our subsequent adversary rules. These rules simply store information about freshly generated terms, the local state, and session keys in the trace. The generate rule marks the fresh terms that have been generated,[4] the state rule marks the current local state, and the sessionkey rule marks a set of terms as session keys.

Test thread. When verifying 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, 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 $R_{\mathrm{Test}} = \mathrm{Init}$ and $\sigma_{\mathrm{Test}} = [\mathrm{Alice}, \mathrm{Bob}/\mathrm{Init}, \mathrm{Resp}]$.

Auxiliary functions. Prior to giving the compromise rules, we define several auxiliary functions.

Given a trace tr and a thread identifier tid, we define the function $\mathit{role} : \mathit{Trace} \times \mathit{TID} \rightarrow (\mathit{Role} \cup \{\bot\})$ by

[4]Note that this rule need not ensure that m is unique. The function *thread* maps freshly generated terms c to $c\sharp tid$ in a thread tid, ensuring uniqueness.

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

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

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

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

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

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

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

Figure 1. Execution-model rules

$role(tr, tid) = R$ if $\exists a.(tid, \mathsf{create}(R, a)) \in tr$, and $role(tr, tid) = \perp$ otherwise. This function is well-defined. In particular, there is at most one such R, due to the third premise $tid \notin dom(th)$ of the create rule.

The long-term secret keys of an agent a are defined as $LongTermKeys(a) = \{sk(a)\} \cup \bigcup_{b \in Agent}\{k(a, b), k(b, a)\}$.

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

$$\langle\rangle \downarrow tid = \langle\rangle$$

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

Similarly, we define for sequences of events an operator $\lfloor$ that selects the contents of events of a particular type. For all $\mathsf{evtype} \in \{\mathsf{create}, \mathsf{send}, \mathsf{recv}, \mathsf{generate}, \mathsf{state}, \mathsf{sessionkey}\}$:

$$\langle\rangle \lfloor \mathsf{evtype} = \langle\rangle$$

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

During protocol execution, the test thread may share its secrets with other threads. Hence some adversary rules require distinguishing between intended *partner threads* and other threads. There exist many notions of partnering in the literature. In general, we use partnering based on matching histories as defined below. For full details we refer to Appendix A-D.

Definition 6 (Matching histories): Let l and l' be sequences of events. We define matching histories (MH) as

$$\mathrm{MH}(l, l') \equiv \big(l \lfloor \mathsf{recv} = l' \lfloor \mathsf{send} \ \wedge \ l \lfloor \mathsf{send} = l' \lfloor \mathsf{recv}\big).$$

Our partnering definition is parameterized over the protocol P, the test role R_{Test}, and the instantiation of variables in the test thread σ_{Test}. These parameters are later instantiated in the definition of the transition system.

Definition 7 (Partnering): Let tr be a trace. Then,

$$\begin{aligned} Partner(tr) = \big\{tid \ \big| \ & tid \neq \mathrm{Test} \\ & \wedge \exists l \ . \ \mathrm{MH}(\sigma_{\mathrm{Test}}(P(R_{\mathrm{Test}})), (tr \downarrow tid)\hat{}l) \\ & \wedge role(tr, tid) \neq R_{\mathrm{Test}}\big\}. \end{aligned}$$

Intuitively, if the partner thread completes, its history will match with the Test thread (for $l = \langle\rangle$), and the partner and Test execute distinct roles. The above definition also ensures that incomplete threads that may complete with matching histories in the future, are also considered partners.

E. Adversary-compromise rules

We define the adversary-compromise rules in Figure 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. Not all combinations of capabilities have been used for analyzing protocols. Some combinations are not covered because of impossibility results (for example [28]), whereas other combinations appear to have been previously overlooked. The combinations

$$\frac{a \notin \{\sigma_{\text{Test}}(R) \mid R \in dom(P)\}}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\textsf{LongtermKeyReveal}_{\textsf{others}}]$$

$$\frac{a = \sigma_{\text{Test}}(R_{\text{Test}}) \qquad a \notin \{\sigma_{\text{Test}}(R) \mid R \in dom(P) \setminus R_{\text{Test}}\}}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\textsf{LongtermKeyReveal}_{\textsf{actor}}]$$

$$\frac{th(\text{Test}) = \langle\rangle}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\textsf{LongtermKeyReveal}_{\textsf{after}}]$$

$$\frac{th(\text{Test}) = \langle\rangle \qquad Partner(tr) \neq \emptyset}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th)}[\textsf{LongtermKeyReveal}_{\textsf{aftercorrect}}]$$

$$\frac{tid \neq \text{Test} \qquad tid \notin Partner(tr)}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{SessionKeyReveal}(tid))\rangle, IK \cup set((tr\!\downarrow\! tid)\lfloor\textsf{sessionkey}), th)}[\textsf{SessionKeyReveal}]$$

$$\frac{tid \neq \text{Test} \qquad tid \notin Partner(tr)}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{StateReveal}(tid))\rangle, IK \cup set((tr\!\downarrow\! tid)\lfloor\textsf{state}), th)}[\textsf{StateReveal}]$$

$$\frac{}{(tr, IK, th) \longrightarrow (tr\hat{}\langle(\text{tid}_{\mathcal{A}}, \textsf{RandomReveal}(tid))\rangle, IK \cup set((tr\!\downarrow\! tid)\lfloor\textsf{generate}), th)}[\textsf{RandomReveal}]$$

Figure 2. Adversary-compromise rules

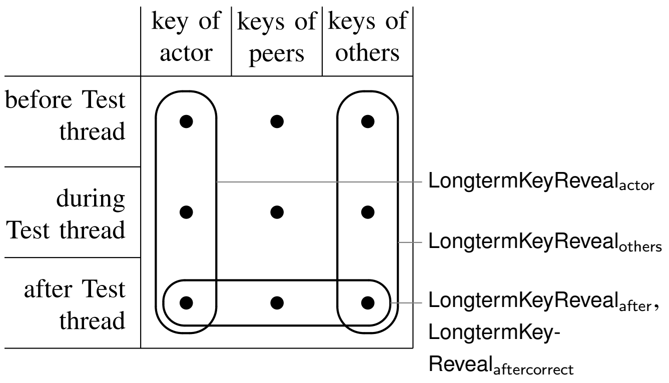

key of actor
keys of peers
keys of others
before Test thread
during Test thread
after Test thread
LongtermKeyReveal$_{\text{actor}}$
LongtermKeyReveal$_{\text{others}}$
LongtermKeyReveal$_{\text{after}}$,
LongtermKeyReveal$_{\text{aftercorrect}}$

Figure 3. Relating long-term key reveal rules

we present here include those previously studied in the literature.

Compromise of long-term keys. The first four rules model the compromise of agents' long-term keys, represented by the event LongtermKeyReveal(a). This event models the adversary learning the long-term 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.

In Figure 3, we clarify the relationships between our long-term key compromise rules in the remaining two dimensions: the vertical axis specifies *when* the compromise occurs, and the horizontal axis specifies *whose* long-term keys are compromised. With respect to *when* a compromise occurs, we differentiate between before, during, and after the test thread. With respect to *whose* keys are compromised, we differentiate between agents not involved in the communication (others), the agent performing the test thread (actor), and the other partner (peer). In Figure 3, the ovals specify the effects of each of the long-term key reveal rules from Figure 2.

The LongtermKeyReveal$_{\text{others}}$ rule formalizes the adversary capability typically used in the symbolic analysis of security protocols since Lowe's Needham-Schroeder attack [34]: the adversary can learn all 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 who is communicating with Bob, the adversary can learn, for example, Charlie's long-term key.

The LongtermKeyReveal$_{\text{actor}}$ 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 [26]. The rule's second premise is required because our model allows agents to communicate with themselves.[5]

The LongtermKeyReveal$_{\text{after}}$ and LongtermKeyReveal$_{\text{aftercorrect}}$ rules have restrictions on when the compromise may occur. In particular, they allow the compromise of long-term keys

[5]In our model, an initiator Alice can start a thread communicating with a responder Alice. In this case, revealing the actor's long-term key also reveals the key of the partner, and as a result no protocol that establishes a shared secret can be correct. The second premise disallows revealing the actor's key for threads that communicate with themselves.

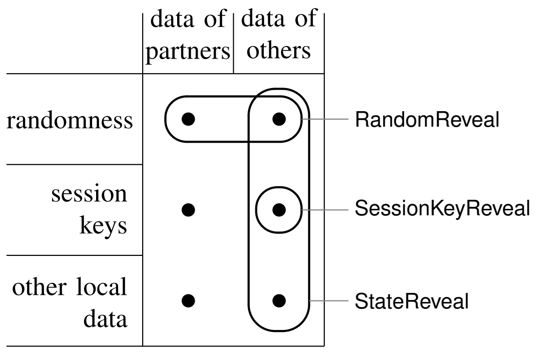


Figure 4. Relating short-term data reveal rules

only after the test thread has finished, captured by the premise $th(\text{Test}) = \langle\rangle$. 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 (PFS) [21,37].

For $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, we have the additional premise that partner threads must exist for the test thread. This condition stems from [28] 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 (wPFS). This property is motivated by a class of protocols given in [28] 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. In our adversary-compromise models, the $\mathsf{SessionKeyReveal}(tid)$ and $\mathsf{StateReveal}(tid)$ events indicate that the adversary gains access to the session key or, respectively, the local state of the thread tid. These are respectively marked by the last $\mathsf{sessionkey}$ and state events. The $\mathsf{RandomReveal}(tid)$ event indicates that the adversary learns the random numbers generated in the thread tid.

In Figure 4, we show the dimensions considered for the compromise of short-term data: *whose* data, on the horizontal axis, and *which* kind of data, on the vertical axis. 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. Furthermore, because we assume that local data does not exist before or after a session, we can ignore the temporal dimension.

For protocols that establish a session key, we assume the session key is shared by all partners and should be secret: revealing it through a compromise will trivially break the security of the protocol. Hence the rules disallow the compromise of the session keys of the test or partner threads. 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 currently unclear whether any protocol is correct with respect to such an adversary.

The rules $\mathsf{SessionKeyReveal}$ and $\mathsf{StateReveal}$ allow for the compromise of session keys and the contents of the local state of a thread. Their premise is that the compromised thread is not a partner thread. In contrast, the premise of the $\mathsf{RandomReveal}$ rule allows for the compromise of all threads, including the partner threads. This rule stems from [31], where it is shown that it is possible to construct protocols that are correct in the presence of an adversary capable of $\mathsf{RandomReveal}$.

F. Traces for adversary-compromise models

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

Definition 8 (Traces): Let P be a protocol, IK_0 a set of terms, Adv an adversary-compromise model, R_{Test} a role, and σ_{Test} a mapping from roles and free variables to $Agent$ and $Term$, respectively. We define a transition relation $\rightarrow_{P,Adv,R_{\text{Test}},\sigma_{\text{Test}}}$ from the execution-model rules from Figure 1 and the rules in Adv. For states s and s', $s \rightarrow_{P,Adv,R_{\text{Test}},\sigma_{\text{Test}}} s'$ iff there exists a rule in Adv or the execution-model rules with the premises $Q_1(s), \dots, Q_n(s)$ and the conclusion $s \rightarrow s'$ such that all of the premises hold. We then define the function *traces* as

$$\begin{aligned} &traces(P, IK_0, Adv, R_{\text{Test}}, \sigma_{\text{Test}}) = \{tr \mid \exists IK, th. \\ &\qquad (\langle\rangle, IK_0, \emptyset) \rightarrow^*_{P,Adv,R_{\text{Test}},\sigma_{\text{Test}}} (tr, IK, th)\}\,. \end{aligned}$$

G. Security properties

We provide a 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, can be defined analogously in our model.

Definition 9 (Session-key secrecy): Let P be a protocol. Let R_{Test} be a role of P that contains $\mathsf{sessionkey}$ events marking the session keys. Let IK_0 be the initial knowledge of the adversary. Let $IK(tr)$ denote the adversary knowledge after the events in tr. 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 $\mathcal{T}estSub$ be the set of all substitutions of roles by agents, and all free variables of R_{Test} by terms. We then define the *secrecy of the session keys of* R_{Test} as

$$\begin{aligned} &\forall \sigma_{\text{Test}} \in \mathcal{T}estSub. \forall tr \in traces(P, IK_0, Adv, R_{\text{Test}}, \sigma_{\text{Test}}). \\ &\forall k.(\text{Test}, \mathsf{sessionkey}(k)) \in tr \wedge completed(tr, R_{\text{Test}}, \text{Test}) \\ &\qquad\qquad \Rightarrow IK(tr) \nvdash k\,. \end{aligned}$$

Table I
Mapping adversary-compromise models from key-agreement literature

	Adversary rules							
	Long-term data				Short-term data			
	Owner		Timing		Type			
Name	others	actor	after	aftercorrect	SessionKey	State	Random	Origin of model
Adv_{EXT}								Dolev-Yao (external)
Adv_{INT}	✓							Dolev-Yao (internal) [34]
Adv_{CA}		✓						Key Compromise Impersonation [26]
Adv_{AFC}				✓				Weak Perfect Forward Secrecy [28]
Adv_{AF}			✓	✓				Perfect Forward Secrecy [21,37]
Adv_{BPR}					✓			BPR2000 [5]
Adv_{BR}	✓				✓			BR93 [6], BR95 [7]
Adv_{CKw}	✓	✓		✓	✓	✓		CK2001-wPFS [28]
Adv_{CK}	✓		✓	✓	✓	✓		CK2001 [13]
$Adv_{\text{eCK-1}}$	✓				✓		✓	eCK [31]
$Adv_{\text{eCK-2}}$	✓	✓		✓	✓			

Within our operational semantics, we can reinterpret different security properties as classical secrecy and authentication properties, with respect to different adversary models. This provides a uniform view of these security properties. It also gives us an account of these properties where the execution model, adversary model, and properties are cleanly separated.

For example, we reinterpret the *secrecy of a session key* as defined in symbolic models as secrecy (Definition 9) with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{others}}$ 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 resilience against *Key Compromise Impersonation* as a form of authentication (e. g. agreement from [35]) with respect to an adversary model that includes the $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$ rule.

III. 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 or design protocols based on the implementation requirements and the worst-case expectations for adversaries in the application domain.

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':

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

We write $Adv =_{\mathcal{A}} Adv'$ when $Adv \leq_{\mathcal{A}} Adv'$ and $Adv' \leq_{\mathcal{A}} Adv$. We say that two adversary-compromise rules r_1 and r_2 are incomparable if and only if

$$\begin{aligned} \forall Adv. \neg\big((Adv \cup \{r_1\}) \leq_{\mathcal{A}} (Adv \cup \{r_2\}) \vee \\ (Adv \cup \{r_2\}) \leq_{\mathcal{A}} (Adv \cup \{r_1\})\big) \end{aligned}$$

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 $\mathsf{RandomReveal}$ on the adversary knowledge will be subsumed by the effect of the $\mathsf{StateReveal}$ 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 two rules reveal the keys only after the Test thread has ended. In particular, we have the following relation for all adversary compromise models Adv:

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

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 [5,7,13,28,31]. 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 consider traces where the partner's long-term key 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 [13] has an execution model that restricts the agents' choice of thread identifiers. This leads to a notion of partner threads that differs from that of other computational models. Here we define partnering uniformly by matching histories. We refer the reader to [12,14,15,36] for further details.

Table I 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 [13], 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 timing-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 $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$, because $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ can simulate their effects.

In Figure 5, we show the hierarchy of adversary models defined in Table I. An arrow $m_1 \rightarrow m_2$ denotes that the model m_1 is weaker than the model m_2. Note 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 model is the 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 10 (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.

$$\begin{tikzcd}
 & Adv_{\text{CKw}} & \\
Adv_{\text{eCK-1}} & Adv_{\text{eCK-2}} \arrow[u] & Adv_{\text{CK}} \\
Adv_{\text{CA}} \arrow[ur] & Adv_{\text{BR}} \arrow[ul] \arrow[u] \arrow[ur] & Adv_{\text{AF}} \arrow[u] \\
Adv_{\text{BPR}} \arrow[ur] & Adv_{\text{INT}} \arrow[u] & Adv_{\text{AFC}} \arrow[u] \arrow[uul] \\
 & Adv_{\text{EXT}} \arrow[ul] \arrow[u] \arrow[ur] \arrow[uul] &
\end{tikzcd}$$

Figure 5. Hierarchy of the adversary-compromise models from Table I

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 11 (Protocol-security hierarchy): We say a *protocol P is stronger than a protocol P' with respect to a set of adversary rules S* iff $security(P, S) \supset security(P', S)$. Given a set of protocols and a set of adversary rules, this ordering gives rise to a protocol-security hierarchy.

IV. Analyzing existing protocols

We have integrated our adversary-model hierarchy in an existing symbolic protocol verification tool [17] and analyzed a number of protocols. We also used the tool to compare the relative strengths of protocols from the perspective of their resilience to a class of adversaries, as in Definition 11.

MQV and HMQV. The MQV protocol family [29,33,41] is a class of authenticated key-exchange protocols designed to provide strong security guarantees.[6] The HMQV protocol was proven secure with respect to the adversary model in [28]. This model is the analog of 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 finds that the HMQV protocol is, depending

[6]The elliptic curve variant of MQV is part of the NSA-B suite of cryptographic protocols [4] and is recommended for use in secret and top-secret contexts.

on the definition of the state, incorrect in adversary models that contain StateReveal rules, such as the CK model [13].

The attack implies that MQV and HMQV are not secure in, e. g., 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. An attack trace is given in Appendix C-C.

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

Signed Diffie-Hellman and variants. The original Diffie-Hellman key-exchange protocol only satisfies its security properties in the presence of a passive adversary, since the messages 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 signature key. The resulting family of protocols is referred to as *signed Diffie-Hellman*. We analyzed several variants, including the ISO, ISO-C (key confirmation), and DHKE-1 variants from [22], as well as the variant from [13]. The tool finds attacks on the Diffie-Hellman signed protocols for all models that contain the RandomReveal rule. This is consistent with the proofs in [13,22], which do not consider this rule, as well as with the observation in [31] that allowing RandomReveal introduces an attack on a signed Diffie-Hellman protocol.

KEA+ and Naxos. KEA+ [32] and Naxos [31] belong to the same class of protocols. In [32], KEA+ is proven correct with respect to a variant of the adversary model of [28], 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 State-Reveal attacks and therefore are incorrect in, for example, the Adv_{CK} model. 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 cannot be modified to work for Naxos.

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

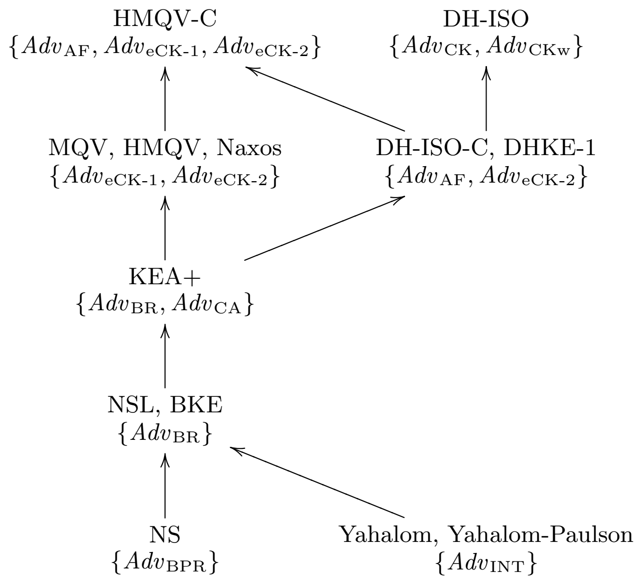

HMQV-C
$\{Adv_{\mathrm{AF}}, Adv_{\mathrm{eCK\text{-}1}}, Adv_{\mathrm{eCK\text{-}2}}\}$
DH-ISO
$\{Adv_{\mathrm{CK}}, Adv_{\mathrm{CKw}}\}$
MQV, HMQV, Naxos
$\{Adv_{\mathrm{eCK\text{-}1}}, Adv_{\mathrm{eCK\text{-}2}}\}$
DH-ISO-C, DHKE-1
$\{Adv_{\mathrm{AF}}, Adv_{\mathrm{eCK\text{-}2}}\}$
KEA+
$\{Adv_{\mathrm{BR}}, Adv_{\mathrm{CA}}\}$
NSL, BKE
$\{Adv_{\mathrm{BR}}\}$
NS
$\{Adv_{\mathrm{BPR}}\}$
Yahalom, Yahalom-Paulson
$\{Adv_{\mathrm{INT}}\}$

Figure 6. Protocol-security hierarchy (secrecy)

Yahalom. [39] presents two versions of the Yahalom protocol. The original version of this protocol allows the adversary to reuse old keys. As a result, the 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. We find attacks on both protocols for adversaries capable of revealing session keys (SessionKeyReveal). At first sight, this appears to contradict Paulson's result. This discrepancy is due to the difference in the property considered. In [39], the adversary may compromise *other* session keys, whereas our SessionKeyReveal rule, following the definitions from key-agreement literature, allows the adversary to compromise keys of threads that are not partners of the test thread.

A protocol-security hierarchy. In Figure 6 we show the protocol-security hierarchy (cf. Definition 11) of a set of protocols from the literature, with respect to the adversary models from Table I for secrecy. 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 $n_1 \rightarrow n_2$ means that the protocols in n_2 are stronger than those in n_1, with respect to Definition 11. 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.

V. Related work

Related work in computational analysis. Most research on adversary compromise has been performed in the context of key-exchange protocols in the computational setting, e. g. Canetti and Krawczyk [13,28], Shoup [40], Bellare et al. [5]–[7], Katz and Yung [27], LaMacchia et al. [31], and Bresson and Manulis [11].

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 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, e. g., Choo et al. [14,15], Bresson et al. [12], LaMacchia et al. [31], and Menezes and Ustaoglu [36].

The eCK model from [31] requires protocols to be correct with respect to $Adv_{\text{eCK-1}}$ and $Adv_{\text{eCK-2}}$. In [31,38,41] 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 $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ from $Adv_{\text{eCK-2}}$. However, the $Adv_{\text{eCK-1}}$ model allows for RandomReveal which is not allowed in Adv_{CK}. Therefore we have that the Adv_{CK} and eCK adversary models are incomparable, which is in line with the analysis of CK and eCK in [10].

The CryptoVerif tool by Blanchet [9] is an automated tool for computational analysis. Its adversary model covers Adv_{INT} (corresponding to static corruption, i. e., 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 [25] 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 use a compromising adversary model, but are specialized 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 the analog of our $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule in their setting.

As noted in Section IV, 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 [22,23] link a symbolic adversary model that includes dynamic corruptions to an adversarial model used in the computational analysis of key-agreement protocols. They describe in [23] 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 keys of agents dynamically.

VI. Conclusions

We have provided the first symbolic framework capable of systematically 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 short-term and long-term data. We used this framework to extend an automated protocol-analysis tool, resulting in the first such tool that is capable of systematically 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. We would like 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 defined a protocol-security hierarchy that characterizes the security provided by different protocols. As future work, this protocol hierarchy could be extended to a larger class of protocols and to compare systematically the merits of protocols that provide similar security guarantees. We would also like to explore more fine-grained versions of the whose/which/when dimensions considered here. For example, by introducing time (for key expiration), or splitting the "during the test thread" notion into multiple phases (for long-running protocols with clearly separated phases.)

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 *Proc. CAV 2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

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

[4] E. Barker, D. Johnson, and M. Smid. NIST SP 800-56: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised), 2007.

[5] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, LNCS, pages 139–155. Springer, 2000.

[6] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249. Springer, 1993.

[7] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proc. STOC '95*, pages 57–66. ACM, 1995.

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

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

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

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

[12] E. Bresson, M. Manulis, and J. Schwenk. On security models and compilers for group key exchange protocols. In *IWSEC*, volume 4752 of *LNCS*, pages 292–307. Springer, 2007.

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

[14] K.-K. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment proofs. In *ASIACRYPT*, volume 3788 of *LNCS*, pages 624–643. Springer, 2005.

[15] K.-K. Choo, C. Boyd, Y. Hitchcock, and G. Maitland. On session identifiers in provably secure protocols. In *SCN'05*, volume 3352 of *LNCS*, pages 351–366. Springer, 2005.

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

[17] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. CAV 2008*, volume 5123/2008 of *LNCS*, pages 414–418. Springer, 2008.

[18] C. Cremers. 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*, pages 119–128. ACM, 2008.

[19] C. Cremers. Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange. Cryptology ePrint Archive, Report 2009/253, 2009. http://eprint.iacr.org/.

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

[21] C. Günther. An identity-based key-exchange protocol. In *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, 1990.

[22] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proc. FMSE 2005*, pages 23–32. ACM, 2005.

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

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

[25] J. D. Guttman. Key compromise, strand spaces, and the authentication tests. *ENTCS*, 45, 2001. Invited lecture, 17th Annual Conference on Mathematical Foundations of Programming Semantics.

[26] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT 1996*, volume 1163 of *LNCS*, pages 36–49, 1996.

[27] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO*, volume 2729 of *LNCS*, pages 110–125. Springer, 2003.

[28] H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. Cryptology ePrint Archive, Report 2005/176, 2005. http://eprint.iacr.org/, retrieved on April 14, 2009.

[29] H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *CRYPTO*, volume 3621 of *LNCS*, pages 546–566. Springer, 2005.

[30] R. Küsters and T. Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *Proc. of the 2008 ACM Conference on Computer and Communications Security (CCS)*, pages 129–138. ACM, 2008.

[31] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, volume 4784 of *LNCS*, pages 1–16. Springer, 2007.

[32] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *LNCS*, pages 378–394, 2006.

[33] 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.

[34] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

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

[36] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In *Proc. of ACISP 2008*, volume 5107 of *LNCS*, pages 53–68, 2008.

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

[38] T. Okamoto. Authenticated key exchange and key encapsulation in the standard model. In *ASIACRYPT*, volume 4833 of *LNCS*, pages 474–484, 2007.

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

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

[41] 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 explain the relation between each of our adversary rules from Section II and security definitions from the computational setting.

A. 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 definitions.

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

In symbolic models, a protocol's possible execution history is often referred to as a trace; 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.

Security definitions for key-agreement protocols use 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 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 queries, e. g. *session-state reveal* and *session-key reveal* [13]. 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" [13, p. 13–14]. These three definitions together restrict the set of allowed experiments 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".

B. Origins of adversary rules

The $\mathsf{LongtermKeyReveal}_{\mathsf{notgroup}}$ rule is implicitly used 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 [20]. 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. [26]. In models that capture these attacks, it is assumed that the adversary can learn the long-term key of the agent executing the test thread. The complexity of the rule's premise 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. The corresponding experiment in the eCK model [31] disallows learning the long-term private key of the actor if there is also an $\mathsf{RandomReveal}$ query in the same experiment.

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 [21,37]. 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 [29] 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 [29] depends on the construction of a generic attack involving the $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ rule. If the adversary model instead includes the weaker $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rule, the generic attack cannot be constructed. Hence, the corresponding notion of weak Perfect Forward Secrecy is defined in [29]. It is intended to serve as a slightly weaker security property that is satisfiable by two-party key-agreement protocols that are based only on public-key cryptography.

For $\mathsf{LongtermKeyReveal}_{\mathsf{rnsafe}}$ of an agent a, we have a further premise that no $\mathsf{RandomReveal}$ event has previously occurred in a thread of a. This additional condition stems from [31] where the adversary is excluded from both learning random values generated by an agent a as well as a's long-term private keys in a given trace (experiment).

The SessionKeyReveal rule corresponds to revealing a session key. The underlying assumption 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 [5].

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 [13]. However, in [13] 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.

[31] defines 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.

C. 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 [14,36] for a more detailed comparison of some of these execution models. For example some models, such as [13], 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.

D. Partner function variations

There are a number of different definitions of partner threads in the literature. For example, [7] defines partnering using an existentially-guaranteed partner function. Others define partnering by an explicit session identifier specified in the protocol, or using (some notion of) matching sessions.

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, by allowing SessionKeyReveal on all threads *not* in matching sessions, therefore provides the strongest possible adversary capability with respect to SessionKeyReveal.

In Section II-D, we defined the most common case of partnering, which is based on matching histories. This definition is appropriate for most protocols. However, as pointed out in [19], matching histories is an inappropriate partnering function for protocols that were designed for *role-symmetry*.[7] An example of a role-symmetric protocol is (H)MQV [29]: the protocol allows two instances of the initiator role of the protocol to exchange messages and establish a shared key.

Recall that our partnering definition is parameterized over the protocol P, the test role R_{Test}, and the instantiation of variables of the test thread σ_{Test}. These parameters are instantiated in the definition of the transition system.

Definition 12 (Partnering for role-symmetric protocols): Let tr be a trace. Then,

$$Partner_{\mathsf{sym}}(tr) = \big\{ tid \;\big|\; tid \neq \mathrm{Test} \\ \wedge \exists l \,.\, \mathrm{MH}(\sigma_{\mathrm{Test}}(P(R_{\mathrm{Test}})), (tr{\downarrow} tid)\hat{\ } l) \big\}.$$

In contrast to Definition 7 we have omitted the requirement that the partners' roles are distinct from the Test role.

However, most protocols are not symmetric in the sense that a regular execution involves an instance of each role. We call these protocols *standard protocols* and for these protocols we use the definition of partnering based on matching histories in Section II-D.

Throughout this paper, we have written $Partner$ to denote the partnering function that is appropriate for the protocol under consideration, i. e., either Definition 7 or 12. In the analysis tool, protocols are assumed to be standard (and Definition 7 is applied) unless explicitly marked as being role-symmetric by the user.

E. 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.

F. 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.

[7]For details on role-symmetry in security protocols we refer the reader to [19]. Note that the vast majority of protocols were not designed for role-symmetry.

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 [17,18] with our adversarial rules from Figure 2.

We describe below the two main technical hurdles that we faced. First, we must 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.

A. Adapting existing tools

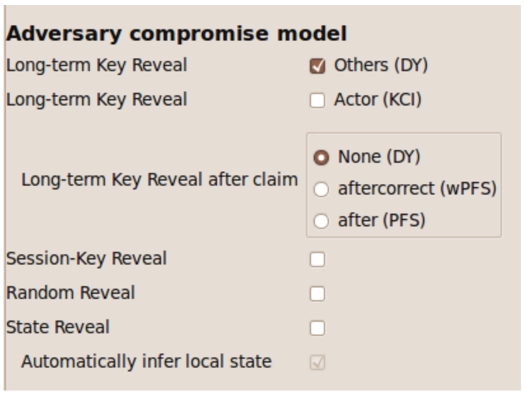
In standard symbolic verification tools and the corresponding protocol descriptions, agents are considered to be either fully honest or fully dishonest. The first change is to lift this restriction. Second, we must incorporate new protocol events for marking local state and local secrets, as well as 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.

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.

B. 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 abstract and functional nature of such specifications, as well as to the complexity and diversity of actual implementations.

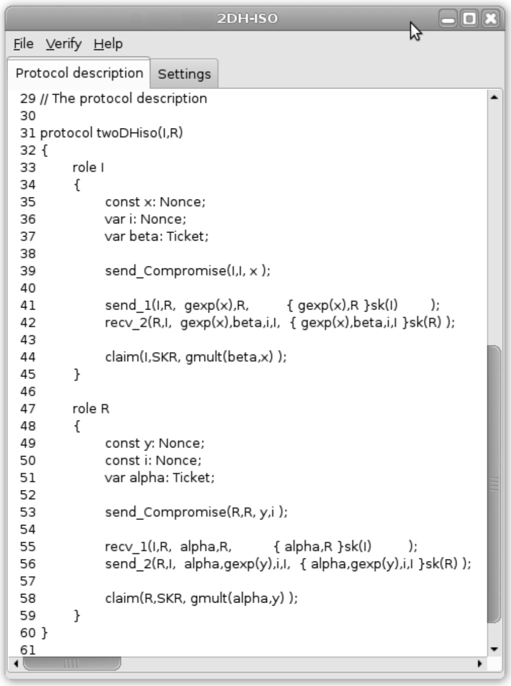
As a result, defining the local state of an abstract protocol is at best an underapproximation of the local state of an actual implementation. Our approach is to infer which data *must* be present and, although still underspecified, this allows us to find attacks that should function for any implementation.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



Adversary compromise model

Long-term Key Reveal	[✓] Others (DY)
Long-term Key Reveal	[ ] Actor (KCI)
Long-term Key Reveal after claim	[● None (DY)] ○ aftercorrect (wPFS) ○ after (PFS)
Session-Key Reveal	[ ]
Random Reveal	[ ]
State Reveal	[ ]
Automatically infer local state	[✓]

Figure 7. Choosing an adversary model in the tool



```
// The protocol description

protocol twoDHiso(I,R)
{
	role I
	{
		const x: Nonce;
		var i: Nonce;
		var beta: Ticket;

		send_Compromise(I,I, x );

		send_1(I,R,  gexp(x),R,        { gexp(x),R }sk(I)        );
		recv_2(R,I,  gexp(x),beta,i,I,  { gexp(x),beta,i,I }sk(R) );

		claim(I,SKR, gmult(beta,x) );
	}

	role R
	{
		const y: Nonce;
		const i: Nonce;
		var alpha: Ticket;

		send_Compromise(R,R, y,i );

		recv_1(I,R,  alpha,R,         { alpha,R }sk(I)         );
		send_2(R,I,  alpha,gexp(y),i,I,  { alpha,gexp(y),i,I }sk(R) );

		claim(R,SKR, gmult(alpha,y) );
	}
}

```

Figure 8. Example input

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

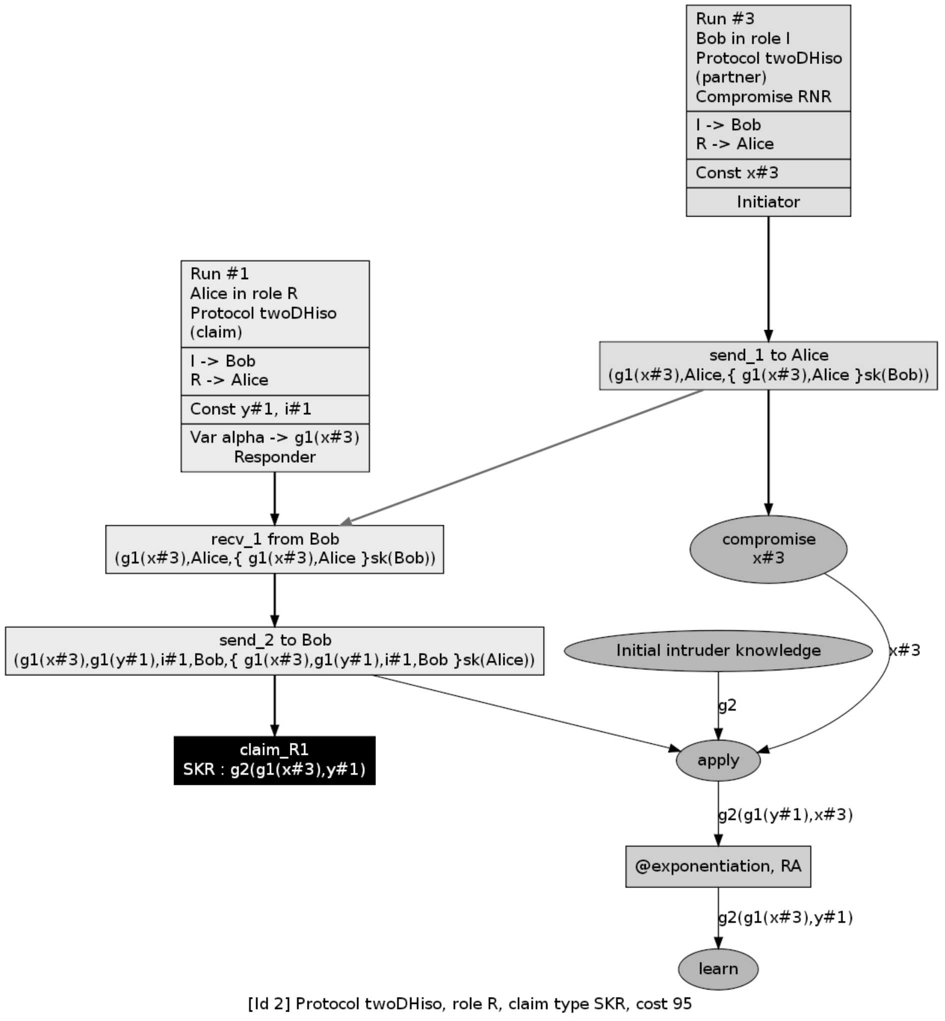


Figure 9. Example attack output generated by the tool

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 defined Φ so 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 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:

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

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.

Table II
SUMMARY OF ATTACKS FOUND

	Adv_{EXT}	Adv_{INT}	Adv_{CA}	Adv_{AFC}	Adv_{AF}	Adv_{BPR}	Adv_{BR}	Adv_{CKw}	Adv_{CK}	$Adv_{\text{eCK-1}}$	$Adv_{\text{eCK-2}}$
BKE			×	×	×			×	×	×	×
DH-ISO										×	
DH-ISO-C								×	×	×	
DHKE-1								×	×	×	
HMQV-C								×	×		
HMQV					×			×	×		
KEA+				×	×			×	×	×	×
MQV					×			×	×		
Naxos					×			×	×		
NS		×	×	×	×		×	×	×	×	×
NSL			×	×	×			×	×	×	×
Yahalom			×	×	×	×	×	×	×	×	×
Yahalom-Paulson			×	×	×	×	×	×	×	×	×

C. *Protocol analysis example*

In Figure 8, we provide an example of a protocol description as it appears in the tool. The protocol considered is the DH-ISO protocol.

Prior to protocol verification, the user selects an adversary model. The choices provided, as displayed in Figure 7, correspond exactly to the adversary rules presented in this paper. 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 I.

We provide an example of the output generated for an attack in Figure 9. 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 this 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 [30]. We extend the protocol specification with a set of rewrite rules that allow the adversary to replace $(g^a)^b$ with $(g^a)^b$ in subterms.

Analysis of the protocol from Figure 8 with respect to Adv_{EXT} takes 1.7 seconds on an Intel Centrino laptop (1.2 GHz) with 3 GB memory. The depth-first search requires a negligible amount of memory. Analyzing the same protocol with respect to Adv_{CK} takes 3.6 seconds. This increase in verification time is mainly due to the inclusion of short-term reveal rules. These rules extend the adversary knowledge with complex terms and checking their premises involves evaluating the $Partner$ function.

APPENDIX C.
ATTACK DETAILS

A. *Verification result summary*

In Table II, we summarize the attacks found using our tool on ten protocols with respect to the adversary models from Table I. A cross (×) in the table denotes that an attack was found. The protocol-security hierarchy in Figure 6 is derived from Tables I and II.

B. *Role symmetry of HMQV*

For protocols with symmetric roles, a partner definition based on matching histories leads to a spurious attack. In this section, we provide a concrete example of such an attack on the HMQV protocol [28], under the assumption that matching histories is used for the partner function, in Figure 10.

Because we use $Partner_{sym}$ for the partnering function of symmetric-role protocols (as described in detail in Appendix A-D) this attack is not considered in our model: the SessionKeyReveal query is not allowed because Thread 1 is considered a partner according to $Partner_{sym}$.

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 protocol's symmetry, 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,

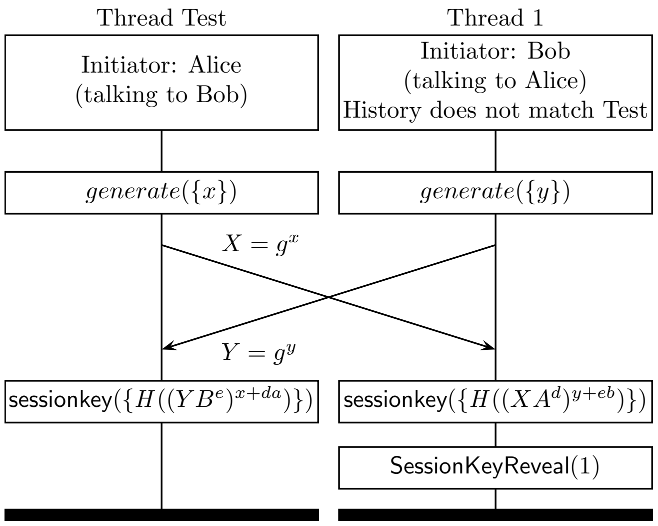

Thread Test
Thread 1
Initiator: Alice (talking to Bob)
Initiator: Bob (talking to Alice) History does not match Test
$generate(\{x\})$
$generate(\{y\})$
$X = g^x$
$Y = g^y$
sessionkey($\{H((YB^e)^{x+da})\}$)
sessionkey($\{H((XA^d)^{y+eb})\}$)
SessionKeyReveal(1)

Figure 10. (Spurious) HMQV attack for adversaries capable of SessionKeyReveal, e. g. Adv_{BPR}, if partnering would be defined by matching histories.

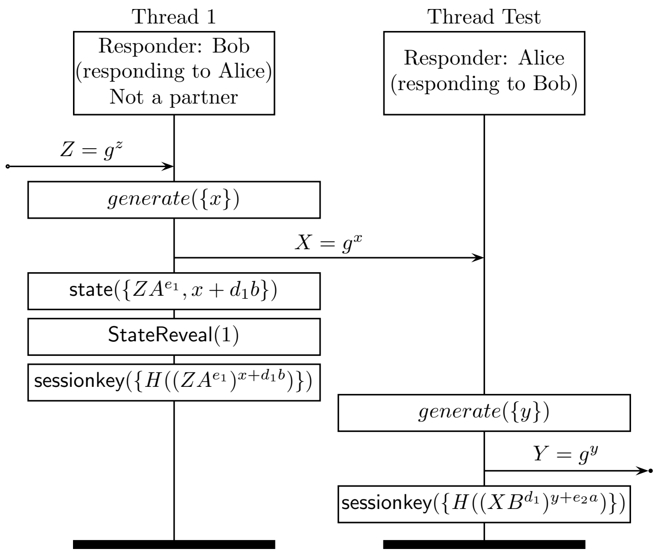

Thread 1
Thread Test
Responder: Bob (responding to Alice) Not a partner
Responder: Alice (responding to Bob)
$Z = g^z$
$generate(\{x\})$
$X = g^x$
state($\{ZA^{e_1}, x + d_1 b\}$)
StateReveal(1)
sessionkey($\{H((ZA^{e_1})^{x+d_1 b})\}$)
$generate(\{y\})$
$Y = g^y$
sessionkey($\{H((XB^{d_1})^{y+e_2 a})\}$)

Figure 11. HMQV attack for adversaries capable of StateReveal, e. g. Adv_{CK}.

as the send and receive labels do not match. Hence if partnering were based on matching histories, the adversary can use SessionKeyReveal to reveal the key of Bob's thread.

We consider the attack to be spurious because HMQV is a symmetric-role protocol, i. e., the protocol is designed such that two initiators can establish a session and compute the same session key. See [19] for a more detailed analysis of the difference between symmetric-role and "standard" protocols.

Note that this attack is possible in the original CK model from [13], but not in the adapted CK model used as the HMQV security model in [28]. In [28], 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 [28] disallows SessionKeyReveal for Bob's thread.

C. State-Reveal attack on HMQV

In Figure 11 we show an attack on HMQV using State-Reveal, 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 $d_1 = \bar{H}(X, \text{Bob})$, $e_1 = \bar{H}(Z, \text{Alice})$, and $e_2 = \bar{H}(Y, \text{Alice})$. The attack starts with 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 the test thread's. Hence the adversary can compromise the state of thread 1, gaining access to $x + d_1 b$.

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

Appendix D. Document revision history

Below we provide a revision history describing any major changes to this document. Small rewrites and bugfixes are omitted.

Version 1.0: February 2009

The initial version of this document was released in February 2009.

Version 1.1: November 2009

The second version was released in November 2009 and contains two major changes: (1) A simplified adversary rule set, and (2) an extension of the partnering function for symmetric-role protocols. We describe and motivate both these changes below.

In addition to these changes, the explanations of the adversary rules have been significantly expanded to provide more intuition.

A. Simplification of adversary rules

In the initial adversary rule set, many rules that revealed the long-term private keys of agents included a variant whose premise required that no RandomReveal events had occurred earlier. This condition was motivated by the Adv_{eCK} modeled after [31] in which an experiment is only allowed if either a long-term private key, or the ephemeral key, of a session is revealed.

In version 1.1, we have refactored the Adv_{eCK} model into two sub-models, $Adv_{\text{eCK-1}}$ and $Adv_{\text{eCK-2}}$. A protocol is considered eCK-secure if it is correct in both these sub-models. This refactoring has made it possible to remove the RandomReveal side condition in some rules and has resulted in removal of one rule altogether.

The mapping is not exact: in particular, the union of the traces (or experiments) in $Adv_{\text{eCK-1}}$ and $Adv_{\text{eCK-2}}$ does not entail traces in which the long-term key of a partner session is revealed as well as the ephemeral key of the test session (or vice versa.) Such behaviour is considered by the security notion in [31]. However, we are currently not aware of any attacks that exploit this particular combination. Overall, the cost of the model simplification is a slightly coarser approximation of the model from [31].

B. Extension of the partnering function for symmetric-role protocols

The version 1.0 of this document included only one partnering function, based on matching histories. This function was not well-suited for symmetric-role protocols such as MQV. In such protocols, two initiators, whose messages cross, may compute the same key. Whether or not to have role symmetry is a design decision that should be made by the protocol designer. The tool can help designers understand the consequences of this decision.

The application of a partnering function that is based on matching histories, to a symmetric-role protocol, can lead to the detection of a generic session-key reveal attack. For details we refer the reader to [19]. Strictly speaking this means a protocol such as MQV cannot be proven correct in the security model of [31]. Here we have chosen a more liberal approach as suggested in [19], in which the partnering function is adapted when analyzing symmetric-role protocols.

Because of this change in partnering function, the generic symmetric-role attacks reported in version 1.0 are no longer considered to be attacks. The attack table and protocol security hierarchy have been updated accordingly.