Know Your Enemy: Compromising Adversaries in Protocol Analysis

DAVID BASIN, ETH Zurich
CAS CREMERS, University of Oxford

We present a symbolic framework, based on a modular operational semantics, for formalizing different notions of compromise relevant for the design and analysis of cryptographic protocols. The framework's rules can be combined to specify different adversary capabilities, capturing different practically-relevant notions of key and state compromise. The resulting adversary models generalize the models currently used in different domains, such as security models for authenticated key exchange. We extend an existing security-protocol analysis tool, Scyther, with our adversary models. This extension systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of state-reveal queries. Furthermore, we introduce the concept of a protocol-security hierarchy, which classifies the relative strength of protocols against different adversaries.

In case studies, we use Scyther to analyse protocols and automatically construct protocol-security hierarchies in the context of our adversary models. Our analysis confirms known results and uncovers new attacks. Additionally, our hierarchies refine and correct relationships between protocols previously reported in the cryptographic literature.

Categories and Subject Descriptors: C.2 [**Computer Systems Organization**]: Computer-Communication Networks; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol verification*; D.2.4 [**Software Engineering**]: Software/Program Verification; D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*

General Terms: Security, Verification

Additional Key Words and Phrases: Security protocols, adversary models, threat models, automated analysis

ACM Reference Format:
Basin, D. and Cremers, C. 2014. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.* 17, 2, Article 7 (November 2014), 31 pages.
DOI: http://dx.doi.org/10.1145/2658996

1. INTRODUCTION

Problem Context. Many cryptographic protocols are designed to work in the presence of various forms of corruption. For example, a Diffie-Hellman key agreement protocol, where signatures are used to authenticate the exchanged public keys, provides perfect forward secrecy [Günther 1990; Menezes et al. 1996]: the resulting key remains secret even after all long-term keys are compromised by the adversary. Designing protocols that work even in the presence of different forms of adversary compromise has considerable practical relevance. It reflects our 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 pseudorandom numbers generated) and the storage of keys and intermediate results.

This article combines and extends results reported in Basin and Cremers [2010a, 2010b].
Author's address: C. Cremers (corresponding author); email: cas.cremers@cs.ox.ac.uk.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
2014 Copyright held by the Owner/Author. Publication rights licensed to ACM. 1094-9224/2014/11-ART7 $15.00
DOI: http://dx.doi.org/10.1145/2658996

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 adversary control. Under this limited view, it is impossible to distinguish between the security provided by early key-exchange protocols such as the Bilateral Key-Exchange protocol [Clark and Jacob 1997] and state-of-the art protocols such as (H)MQV [Krawczyk 2005a; Law et al. 2003]. It is also impossible to discern any benefit from storing the long-term keys in a tamper-proof module or performing part of a computation in a cryptographic coprocessor. While in theory, some of these aspects could be explicitly encoded, no systematic attempts to do so have been made. Despite this, symbolic methods have the advantage that there are numerous effective tools for symbolic protocol analysis (such as [Blanchet 2001; Cremers 2008a; Schmidt et al. 2013]).

In contrast to these, researchers in the computational setting (such as [Bellare and Rogaway 1993, 1995; Bellare et al. 2000; Bresson and Manulis 2008; Canetti and Krawczyk 2001; Cremers and Feltz 2013; Just and Vaudenay 1996; Katz and Yung 2003; Krawczyk 2005a; LaMachia 2007; Shoup 1999]), have defined stronger adversary models, where principals may be selectively corrupted during protocol execution. For example, their short-term or long-term secrets, or the results of intermediate computations, may be revealed (at different times) to the adversary. These models are used to establish stronger properties, such as perfect forward secrecy or resilience to state-reveal attacks. There are, however, drawbacks to these computational models. Namely, they have been defined just for key-agreement protocols, whereas one may expect similar definitions to exist for any security protocol. Moreover, contrary to the security models used in symbolic approaches, there is no automated tool support available for the stronger adversary models.

Our starting point is an operational semantics for security protocols. We parameterize this semantics by a set of rules that formalize adversarial capabilities. These rules capture three fundamental dimensions of compromise: *whose* data is compromised, *which* kind of data it is, and *when* the compromise occurs. For each of these dimensions, we define a partitioning based on the work in the computational setting. For example, for the *whose data* dimension, we distinguish between the *actor* (which is the agent that executes the thread), the intended peers, and other agents. As a result, different rule combinations formalize symbolic analogs of different practically-relevant notions of key and state compromise from the computational setting. The operational semantics gives rise, in the standard way, to a notion of correctness with respect to state and trace-based security properties.

Contributions. We present a framework for analyzing security protocols in the presence of adversaries with a wide range of compromise capabilities. We show how analogs of adversary models studied in the computational setting can be modeled in our framework. For example, we can model attacks against implementations of cryptographic protocols involving the mixed use of cryptographic coprocessors for the secure storage of long-term secrets with the computation of intermediate results in less-secure main memory for efficiency reasons. Such implementations are common and reflect the 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 performs all other cryptographic operations (such as symmetric cryptography and hashing) on the host computer [Gutmann].

Our models bridge another gap between the computational and symbolic approaches by providing symbolic definitions for adversaries and security properties that were

previously only available in the computational setting. Moreover, by decomposing security properties into an adversary model and a basic security property, we unify and generalize many existing security properties.

We introduce the concept of a protocol-security hierarchy in which protocols are classified by their relative strength against different adversaries. Protocol-security hierarchies can be used to select or design protocols based on implementation requirements and the worst-case expectations for adversaries in the application domain. This concept can be generalized to classify arbitrary secure systems and is of independent interest.

Our framework directly lends itself to protocol analysis. As an example, we extend Scyther [Cremers 2008a], a symbolic protocol analysis tool. This results in the first automated tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries that can reveal the local state of agents. We analyze a set of protocols with the tool and rediscover many attacks previously reported in the cryptographic literature, including a so-called *session-state reveal attack* on the MQV and HMQV protocols [Kunz-Jacques and Pointcheval 2006], which can occur when HMQV is partly implemented using a secure component such as an HSM. Furthermore, our tool finds previously unreported attacks. Our results in the domain of key exchange protocols provide evidence that our symbolic methodology can effectively support cryptographers in designing security protocols, complementing the (manual and time-consuming) construction of computational proofs.

We also extend Scyther to automatically compute protocol-security hierarchies. In case studies, we use this extension to compute hierarchies that refine and correct relationships reported in the cryptographic literature. This further shows that symbolic methods can be effectively used for analyses that were previously possible only using a manual computational analysis.

Organization. We review different adversary capabilities and cryptographic models for authenticated key exchange in Section 2. We present our framework in Section 3. In Section 4, we use it to construct protocol-security hierarchies. In Section 5, we show applications of our framework and report on case studies. We also prove general results relating models and properties which aid the construction of protocol-security hierarchies. We discuss related work in Section 6 and draw conclusions in Section 7.

2. BACKGROUND ON SECURITY MODELS FOR AUTHENTICATED KEY EXCHANGE

Although the models we develop are not specific to any type of security protocol, they are inspired by the domain of key-exchange protocols. This is because most research on adversary compromise has been performed in the context of authenticated key-exchange (AKE) protocols in the computational setting. We start by recalling the main elements of AKE security models. We cover the models in some detail because they provide the context for the design choices made for our framework. Readers familiar with computational AKE models and modern AKE protocols can skip this section.

The goal of an AKE protocol is to establish a shared symmetric (session) key between agents. For example, consider the simplified presentation of the HMQV protocol [Krawczyk 2005b] shown in Figure 1. The protocol assumes as given a generator g for a group G and two hash functions that we identify with H and $\bar{H}$.

The protocol has two roles, Initiator and Responder, which we assume are named $\hat{A}$ and $\hat{B}$, respectively. Then, $\hat{A}$'s long-term private key is a and $\hat{B}$'s key is b. Their corresponding long-term public keys are $A = g^a$ and $B = g^b$, respectively.

The protocol proceeds as follows. The initiator generates a fresh random value x and sends g^x to the responder. After receiving g^x, the responder generates a fresh random value y and sends g^y to the initiator. Now they both compute a session key based on

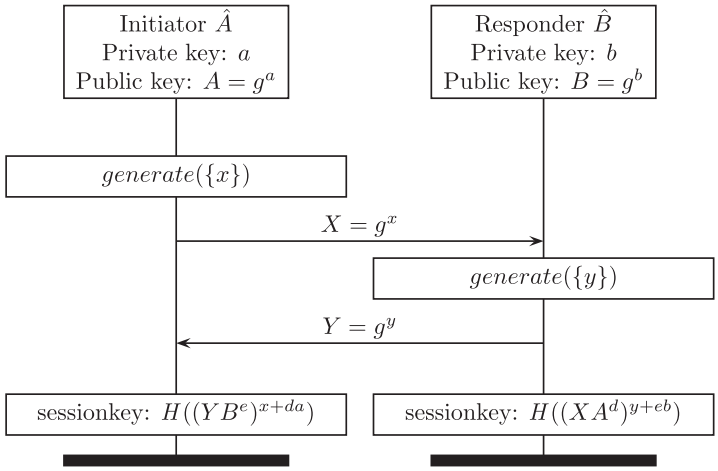

Fig. 1. Simplified presentation of the HMQV protocol, where $d = \bar{H}(X, \hat{B})$ and $e = \bar{H}(Y, \hat{A})$.

the exchanged values and their long-term keys. Due to the algebraic properties of the modular exponentiation, the session keys computed by both parties are identical. An adversary should not be able to compute the session key if he does not have one of the long-term private keys. In fact, even if he obtains long-term private keys, he should not be able to compute the session key unless he also knows either x or y.

For such AKE protocols, advanced security models have been defined (e.g., [Bellare and Rogaway 1993, 1995; Bellare et al. 2000; Bresson and Manulis 2008; Canetti and Krawczyk 2001; Cremers and Feltz 2013; Katz and Yung 2003; Krawczyk 2005a; LaMacchia et al. 2007]. We take these models as a starting point for our analysis.

2.1. The eCK Model

As a concrete example of how security models for key exchange protocols formalize adversarial compromise, we briefly describe the eCK model [LaMacchia et al. 2007].

The eCK model formalizes an adversary that fully controls the network: he can eavesdrop, redirect, or deflect any message, and can insert messages at will. Additionally, the adversary can learn certain values that were intended to be secret. In particular, he can learn:

— the *long-term private keys* of agents, by corrupting the agents;
— agents' *session keys*, by learning the session keys of corrupted agents or by performing cryptanalysis;
— the *randomness* generated by agents, either because weak or corrupted random number generators are used that leak the data they generate, or because the adversary can perform side channel attacks.

As is common in key exchange models [Bellare and Rogaway 1993], the eCK model describes a game and its winning conditions. A protocol is *secure* with respect to the model if there exists no adversary (modeled as a probabilistic polynomial-time Turing machine) that has more than negligible advantage over guessing in winning the game. Roughly speaking, the rules of the game allow the adversary to interact with agents that are executing instances of the protocol, and the adversary wins the game if he can distinguish a real session key from a random bit string.

An AKE protocol usually consists of two roles: the *Initiator* role (performed by the agent that starts the protocol session) and the *Responder* role (which is activated when

an agent receives the first message). Each agent can perform each role. We call a single role instance, as executed by an agent, a *thread*. Note that in computational models, a thread is often referred to as a *(local) session*. We identify a thread by a thread identifier *tid* from the set of possible thread identifiers *TID*.

eCK game. We consider any number of agents with identities $A, B, C, \ldots$. Each agent has a long-term public/private key pair associated to its identity. Initially, the adversary knows all identities and the long-term public key of each agent, but no long-term private keys. The adversary can perform a given set of actions, historically called *queries*. For the eCK model, the queries are as follows.

(1) Test$(A, B, comm)$. If the message *comm* is the empty message, this query models the adversary scheduling the start of an initiator thread. In particular, the agent A activates a new initiator thread, trying to communicate with B, and computes the first message to send. The result of the query is that the adversary is given the sent message. This models that the adversary can eavesdrop all sent messages.
If *comm* is not the empty message, this query models the adversary sending a message *comm* to the agent A, claiming that the message was sent by B. This message can be accepted by any active thread, updating its state, or it can result in the activation of a new responder thread at A that receives the message. In both cases, the adversary is given the response message computed by A.
(2) Long-TermKeyReveal(A). The adversary is given the long-term private key(s) of A.
(3) EphemeralKeyReveal(tid). *tid* is the thread identifier of an active thread. The adversary is given the *ephemeral keys* of *tid*. This terminology stems from Diffie-Hellman style key exchange. In practice, and along the lines of the examples given in the eCK model, this means that all random values previously generated by the thread are given to the adversary.
(4) Reveal(tid). *tid* is the thread identifier of an active thread that has computed a session key k. The key k is given to the adversary.
(5) Test(tid). This query can only be performed once. *tid* is the thread identifier of an active thread that has computed a session key k. A coin is flipped and its outcome stored as $b \in \{0, 1\}$. If $b = 1$, the adversary is given k (as in the Reveal query). If $b = 0$, a random bit string is drawn from the key space and returned to the adversary.
(6) Guess(b'). This query can only be performed if the Test query has been performed earlier. After the Guess query has been performed, the game ends. The query models the adversary guessing (encoded by the parameter $b' \in \{0, 1\}$) whether he received the real session key or a random bit string as a result of the Test query.

If there were no additional restrictions in the model, the adversary could always guess the correct bit by performing Reveal(tid) and then Test(tid). This would amount to modeling that the adversary can learn all session keys at any time, and hence no protocol could ensure that a session key remains secret in this adversary's presence. Thus, to ensure that at least some protocols are correct with respect to an AKE model, the models restrict the adversary, effectively limiting the queries that he can make. The models aim to specify the fewest possible restrictions but at the same time allow some protocols to be correct.

For the eCK model, we introduce a property *clean* to model when the adversary has met the intended restrictions with respect to a certain thread. We refer to a sequence of queries performed by the adversary as an *experiment*.

To model the clean property, we additionally introduce *matching threads*. Informally speaking, the purpose of a key exchange model is to establish a key *shared* between two

threads. Hence, given a thread tid, the notion of matching threads formalizes when a thread tid' is supposed to compute the same session key as intended by the protocol.

Matching Threads for Two-Message Protocols in the eCK Model. Let tid be a thread. We say its *identifying tuple* is

$$(role, ID, ID*, comm_1, \ldots, comm_n),$$

where *role* is the role (Initiator or Responder) performed by the thread, ID is the identity of the agent executing the thread, $ID*$ is the identity of the intended communication partner, and $comm_1, \ldots, comm_n$ is the sequence of messages communicated (sent or received) so far. For a thread tid of a two-message protocol with the identifying tuple $(r, A, A', comm_1, comm_2)$, another thread tid' is said to be a *matching thread* if and only if its identifying tuple is $(r', A', A, comm_1, comm_2)$, where $r \neq r'$.

Clean Property for the eCK Game. Let E be an experiment, and let tid be a thread executed by A with the intended communication partner B. If a matching thread[1] for tid exists, we denote this by $tid*$. In the context of E, tid is *clean* unless one of the following conditions holds.

(1) E includes Reveal(tid) or Reveal($tid*$).
(2) E includes both Long-TermKeyReveal(A) and EphemeralKeyReveal(tid).
(3) A matching thread $tid*$ exists and E includes both Long-TermKeyReveal(B) and EphemeralKeyReveal($tid*$).
(4) No matching thread $tid*$ exists and E includes Long-TermKeyReveal(B).

Winning Condition for the eCK Game. The adversary wins the eCK game in an experiment E if $b' = b$ and the Test thread is *clean*.

eCK security. A protocol is secure in the eCK model if matching threads compute the same session key, and no probabilistic polynomial-time adversary has more than negligible advantage (over guessing) in winning the eCK game.

2.2. Observations on AKE Models

Security models such as the eCK model are complex, monolithic definitions that are hard to disentangle and for which it is hard to understand the design choices. To further complicate matters, in general, any two computational models are incomparable due to (often minor) differences not only in the adversary notions, but also in the definition of matching threads, the execution models, and security property specifics. The details of some of these definitions and their relationships have been studied (e.g., by [Bresson et al. 2007; Choo et al. 2005a, 2005b; Cremers 2010, 2011; LaMacchia et al. 2007; Menezes and Ustaoglu 2008].

In this article, we would like to model different types of adversaries, irrespective of the execution model and the security property (such as secrecy or authentication). However, separating the execution model, the adversary model, and the security properties in the eCK model is non trivial, because the model is presented in a monolithic way. This observation holds for all AKE security models.

In the next section, we develop an operational semantics with a set of adversary-compromise rules. Our aim is to factor and generalize the adversarial capabilities given by AKE models.

[1]In the context of the protocols so far considered in the eCK model, the matching thread is always unique.

3. COMPROMISING ADVERSARY MODEL

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

3.1. Notational Preliminaries

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

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

3.2. Terms and Events

As we will see in the next section, protocols are executed by instantiating their roles resulting in threads (also known as role instances, runs, local sessions, or strands). Each thread executes a sequence of events that send or receive messages. We introduce these elements in a bottom-up fashion, starting with terms, which represent messages.

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

When role specifications are instantiated in threads, the roles are instantiated with concrete agent names. To bind local terms, such as freshly generated terms or local variables, to a specific thread, we write $T\sharp tid$. This denotes that the term T is local to the thread identified by tid.

Definition 3.1. Terms.

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

For each $X, Y \in Agent$, $sk(X)$ denotes the long-term private key, $pk(X)$ denotes the long-term public key, and $k(X, Y)$ denotes the long-term symmetric key shared between X and Y. Moreover, $\{\!|\, t_1 \,|\!\}^a_{t_2}$ denotes the asymmetric encryption (using a public key) or the digital signature (using a signing key) of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^s_{t_2}$ denotes symmetric encryption. The set *Func* is used to model other cryptographic functions,

such as hash functions. Freshly generated terms and variables are assumed to be local to a thread.

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

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

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

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

Definition 3.2. Events.

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

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

3.3. Protocols and Threads

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

Example 3.3 (*Simple Protocol*). Let $\{\mathrm{Init}, \mathrm{Resp}\} \subseteq Role$, $key \in Fresh$, and $x \in Var$. We define the simple protocol SP as follows.

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

A message sequence chart representation of this protocol is shown in Figure 2.

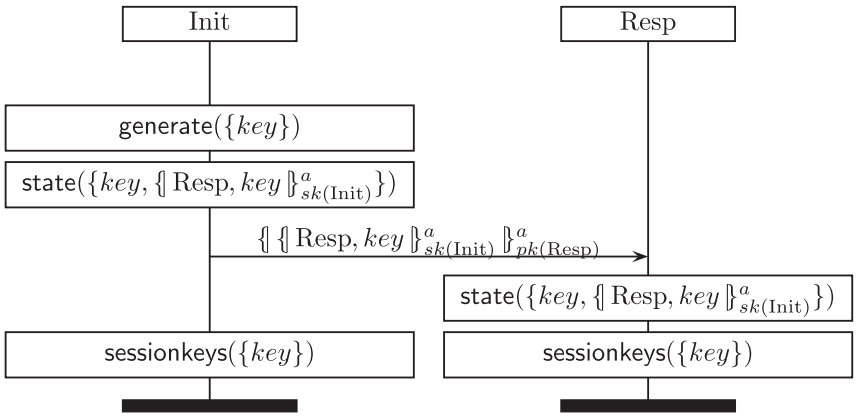

Fig. 2. Message sequence chart depicting the protocol SP from Example 3.3.

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

Protocols are executed by agents who execute roles (represented by sequences of agent events), thereby instantiating role names with agent names. Agents may execute each role multiple times. Each instance of a role is called a thread. We distinguish between the fresh terms and variables of each thread by assigning them unique names, using the function $localize : TID \to Sub$. Note that we abuse notation and extend the domain of substitutions to $Var \cup Role \cup Fresh$.

Definition 3.4 (*Localize*). Let $tid \in TID$. Then,

$$localize(tid) = \bigcup_{cv \in Fresh \cup Var} [\, cv \sharp tid \,/\, cv \,] \,.$$

Using *localize*, we define a function $thread : (AgentEvent^* \times TID \times Sub) \to AgentEvent^*$ that yields the sequence of agent events that may occur in a thread.

Definition 3.5 (*Thread*). Let l be a sequence of events, $tid \in TID$, and let σ be a substitution of role names by agent names and, optionally, variables by terms. Then $thread(l, tid, \sigma) = \sigma(localize(tid)(l))$.

Example 3.6. Let $\{A, B\} \subseteq Agent$. For a thread $t_1 \in TID$ performing the Init role from Example 3.3, we have $localize(t_1)(key) = key \sharp t_1$ and

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

Test Thread. When verifying security properties, we will focus on a particular thread. In the computational setting, this is the thread where the adversary performs a so-called *test query*. In the same spirit, we call the thread under consideration the *test*

thread, with the corresponding thread identifier *Test*. For the test thread, the substitution of role names by agent names, and all free variables by terms, is given by σ_{Test}, and the role is given by R_{Test}. For example, if the test thread is performed by Alice in the role of the initiator, trying to talk to Bob, we have that $R_{Test} = \text{Init}$ and $\sigma_{Test} = [\text{Alice}, \text{Bob} / \text{Init}, \text{Resp}]$.

3.4. Execution Model

We define the set *Trace* as $(TID \times Event)^*$, representing possible execution histories. The state of our system is a four-tuple $(tr, IK, th, \sigma_{Test}) \in Trace \times \mathcal{P}(Term) \times (TID \nrightarrow Event^*) \times Sub$, whose components are (1) a trace tr, (2) the adversary's knowledge IK, (3) a partial function th mapping thread identifiers of initiated threads to sequences of events, and (4) the role to agent and variable assignments of the test thread. To facilitate defining the partner function later, we include the trace as part of the state and fix the test substitution at the start of the system.

Definition 3.7 ($TestSub_P$). Given a protocol P, we define the set of *test substitutions* $TestSub_P$ as the set of ground substitutions σ_{Test} such that $dom(\sigma_{Test}) = dom(P) \cup \{v \sharp Test \mid v \in Var\}$ and $\forall r \in dom(P).\ \sigma_{Test}(r) \in Agent$.

We define the initial adversary knowledge AK_0 as

$$AK_0 = Agent \cup \{pk(a) \mid a \in Agent\} \cup \{c \sharp tid_{\mathcal{A}} \mid c \in Fresh\}.$$

The adversary initially knows the names of all agents, their public keys, and a set of adversary-generated constants, which we denote by fresh symbols that are bound to the adversary's thread identifier $tid_{\mathcal{A}}$. In contrast to most Dolev-Yao models, the initial adversary knowledge does not include any long-term secret keys. The adversary may learn these from long-term key reveal (LKR) events.

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

$$IS(P) = \bigcup_{\sigma_{Test} \in TestSub_P} \{(\langle\rangle, AK_0, \emptyset, \sigma_{Test})\}.$$

The semantics of a protocol $P \in Protocol$ is defined by a transition system that combines the execution rules from Figure 3 with a set of adversary rules from Figure 4. We first present the execution rules.

The create_P rule starts a new instance (a *thread*) of a role R of the protocol P. A fresh thread identifier *tid* is assigned to the thread, thereby distinguishing it from existing threads, the adversary thread, and the test thread. The rule takes the protocol P as a parameter. The role names of P, which can occur in events associated with the role, are replaced by agent names by the substitution σ. Similarly, the createTest_P rule starts the test thread. However, instead of choosing an arbitrary role, it takes an additional parameter R_{Test}, which represents the test role and will be instantiated in the definition of the transition relation in Definition 3.10. Additionally, instead of choosing an arbitrary σ, the test substitution σ_{Test} is used.

The send rule sends a message m to the network. In contrast, the receive rule accepts messages from the network that match the pattern pt, where pt is a term that may contain free variables. The resulting substitution σ is applied to the remaining protocol steps l. Note that, as is standard, we have identified the network with the adversary. Hence messages are sent directly to and received from IK.

$$\frac{R \in dom(P) \quad \sigma \in Role \to Agent \quad tid \notin (dom(th) \cup \{tid_{\mathcal{A}}, Test\}) \quad l = thread(P(R), tid, \sigma)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{create}(R, \sigma(R)))\rangle, IK, th[l \leftarrow tid], \sigma_{Test})}[\mathsf{create_P}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad Test \notin dom(th) \qquad l = thread(P(R_{Test}), Test, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(Test, \mathsf{create}(R_{Test}, a))\rangle, IK, th[l \leftarrow Test], \sigma_{Test})}[\mathsf{createTest_P}]$$

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

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

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

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

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

Fig. 3. Execution rules.

The last three rules support our adversary rules, given shortly. The generate rule marks the fresh terms that have been generated,[2] the state rule marks the current local state, and the sessionkeys rule marks a set of terms as session keys.

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

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

For traces, we define an operator $\downarrow$ that projects traces on events belonging to a particular thread identifier. For all $tid, tid' \in TID$, $e \in Event$, and $tr \in (TID \times Event)^*$, we define $\langle\rangle \downarrow tid = \langle\rangle$ and

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

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

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

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

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

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

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

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

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

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

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

Fig. 4. Adversary-compromise rules.

During protocol execution, the test thread may intentionally share some of its short-term secrets, such as a session key, with other threads. Hence some adversary rules require distinguishing between the intended *partner threads* and other threads. There are many notions of partnering in the literature. Here we use partnering based on matching histories for protocols with two roles. A similar notion for protocols with any number of roles is non-injective (message) agreement, as in Cremers et al. [2006].

Definition 3.8 (*Matching Histories*). For sequences of events l and l', we define $\mathrm{MH}(l, l') \equiv \big(l \lfloor \mathsf{recv} = l' \lfloor \mathsf{send}\big) \wedge \big(l \lfloor \mathsf{send} = l' \lfloor \mathsf{recv}\big)$.

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

Definition 3.9 (*Partnering*). Let R be the non-test role, that is, $R \in dom(P)$ and $R \neq R_{Test}$. For tr a trace, $Partner(tr, \sigma_{Test}) = \big\{tid \mid tid \neq Test \wedge \big(\exists a.\mathsf{create}(R, a) \in tr \downarrow tid\big) \wedge \exists l \,.\, \mathrm{MH}(\sigma_{Test}(P(R_{Test})), (tr \downarrow tid)\hat{}l)\big\}$.

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

Some protocols, such as HMQV, are symmetric in both communication and key computation. For such protocols, a slightly modified definition of partnering is required, where requirement (2) is dropped. For details on these subtleties we refer to Cremers [2011].

3.5. Adversary-Compromise Rules

We define our adversary-compromise rules in Figure 4. 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 (e.g., [Krawczyk 2005a]), whereas other combinations have simply been overlooked.

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

The $\mathsf{LKR}_{\mathsf{others}}$ rule formalizes the adversary capability used in the symbolic analysis of security protocols since Lowe's man-in-the-middle attack on the Needham-Schroeder protocol [Lowe 1996]: the adversary can learn the long-term keys of any agent a that is not an intended partner of the test thread. Hence, if the test thread is executed by Alice, communicating with Bob, the adversary can learn, for example, Charlie's long-term key.

The $\mathsf{LKR}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term key of the agent executing the test thread, also called the *actor*. The intuition is that a protocol may still function as long as the long-term keys of the other partners are not revealed. This rule allows the adversary to attempt so-called Key Compromise Impersonation attacks [Just and Vaudenay 1996]. The rule's second premise is required because our model does not allow the compromise of the partner's key before the end of the test session, but it does allow agents to communicate with themselves: in such cases, revealing the actor's long-term key would also reveal the partner's key.

The $\mathsf{LKR}_{\mathsf{after}}$ and $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rules restrict when the compromise may occur. In particular, they allow the compromise of long-term keys only after the test thread has finished. This is captured by the premise $th(Test) = \langle\rangle$. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{after}}$, it is said to satisfy Perfect Forward Secrecy (PFS) [Günther 1990; Menezes et al. 1996]. $\mathsf{LKR}_{\mathsf{aftercorrect}}$ has the additional premise that a finished partner thread must exist for the test thread. This condition stems from Krawczyk [2005a] and prevents the adversary from both inserting fake messages during protocol execution and learning the key of the involved agents later. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{aftercorrect}}$, it is said to satisfy weak Perfect Forward Secrecy (wPFS). This property is motivated by a class of implicitly authenticated protocols sketched in Krawczyk [2005a], whose members fail to satisfy PFS, although some satisfy this weaker property.

Figure 5 depicts the relationships between our long-term key compromise rules in the relevant dimensions: the rows specify *when* the compromise occurs and the columns specify *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). The ovals specify the effects of each long-term key reveal rule.

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 Figure 6, we show the relevant dimensions: *whose* data, specified by the columns, and *which* kind of data, specified by the rows. Whereas we assumed a long-term key compromise reveals all long-term keys of an agent, we differentiate here between the different

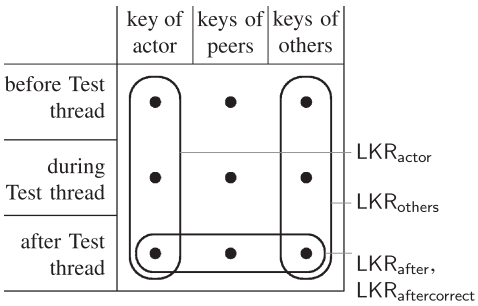

Fig. 5. Relating long-term data reveal rules.

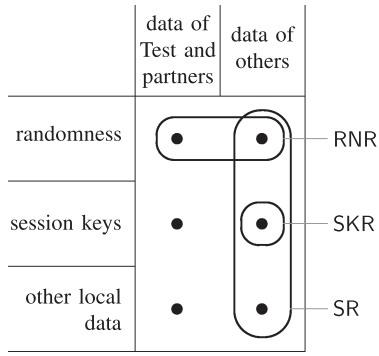

Fig. 6. Relating short-term data reveal rules.

kinds of local data. Because we assume that local data does not exist before or after a session, we can ignore the temporal dimension.

We differentiate between three kinds of local data: *randomness*, *session keys*, and *other local data* such as the results of intermediate computations. The notion that the adversary may learn the randomness used in a protocol stems from LaMacchia et al. [2007]. Considering adversaries that can reveal session keys, for example, by cryptanalysis, is found in many works, such as Bellare and Rogaway [1993]. An adversary capable of revealing the local state is described in Canetti and Krawczyk [2001].

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

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

Note that the premise of the SKR and SR rules requires that the compromised thread is not a partner thread. In contrast, the premise of the RNR rule allows for the compromise of all threads, including the partner threads. This rule is inspired by LaMacchia et al. [2007], where it is shown that it is possible to construct protocols that are correct in the presence of an adversary capable of RNR.

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

We call each subset of the set of adversary rules from Figure 4 an *adversary model*.

3.6. Transition Relation

Given a protocol and an adversary model, we define the protocol's behavior as a set of reachable states.

Definition 3.10 (*Transition Relation and Reachable States*). Let P be a protocol, Adv an adversary model, and R_{Test} a role. We define a *transition relation* $\rightarrow_{P,Adv,R_{Test}}$ from the execution rules from Figure 3 and the rules in Adv. The variables P, Adv,

and R_{Test} in the adversary rules are instantiated by the corresponding parameters of the transition relation. For states s and s', $s \rightarrow_{P,Adv,R_{Test}} s'$ if and only if there exists a rule in either Adv or the execution rules with the premises $Q_1(s), \ldots, Q_n(s)$ and the conclusion $s \rightarrow s'$ such that all of the premises hold. We define the set of *reachable states* RS as

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

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

Definition 3.11 (*Order on Adversary-Compromise Models* $\leq_{\mathcal{A}}$). For all adversary models Adv and Adv',

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

We write $Adv =_{\mathcal{A}} Adv'$ if and only if $Adv \leq_{\mathcal{A}} Adv'$ and $Adv' \leq_{\mathcal{A}} Adv$. As is standard, we write $Adv <_{\mathcal{A}} Adv'$ if and only if $Adv \leq_{\mathcal{A}} Adv'$ and not $Adv =_{\mathcal{A}} Adv'$.

3.7. Security Properties

We now provide two examples of security property definitions. We give a symbolic definition of *session-key secrecy* which, when combined with different adversary models, gives rise to different notions of secrecy from the literature. We also define *aliveness*, which is one of the many forms of authentication [Cremers et al. 2006; Lowe 1997]. Other security properties, like secrecy of general terms, symbolic indistinguishability, or other variants of authentication, can be defined analogously.

Definition 3.12 (*Session-Key Secrecy*). Let $(tr, IK, th, \sigma_{Test})$ be a state. We define the *secrecy of the session keys in* $(tr, IK, th, \sigma_{Test})$ as

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

Definition 3.13 (*Aliveness for Two-Party Protocols*). Let $(tr, IK, th, \sigma_{Test})$ be a state. We say that $(tr, IK, th, \sigma_{Test})$ satisfies *aliveness* if and only if

$$\begin{aligned} th(Test) = \langle\rangle \Rightarrow \exists i, i', j, R_{Test}, R, tid, a.\, tr_i = (Test, \mathsf{create}(R_{Test}, a)) \\ \wedge\, tr_{i'} = last(tr \downarrow Test) \wedge tr_j = (tid, \mathsf{create}(R, \sigma_{Test}(R))) \wedge R \neq R_{Test} \wedge j < i'. \end{aligned}$$

Intuitively, the preceding property formalizes that if the test thread ended, the trace contains the first and last events of the test thread (tr_i and $tr_{i'}$) and additionally the intended peer ($\sigma_{Test}(R)$) started a thread in the other role before the last event of the test thread.

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

Decomposing Security Properties. Many definitions of security properties, such as perfect forward secrecy, contain elements of adversary capabilities. In our framework, such properties are cleanly separated into a basic security property (such as secrecy or authentication) and an adversary model. In Table I, we decompose different security properties from the literature this way.

Our way of modeling security properties provides a uniform view of protocol properties in an execution environment where adversaries have given capabilities. It also allows for direct generalizations of security properties. This leads to new, practically relevant combinations of adversary models and basic security properties. For example,

Table I. Decomposing Security Properties

Security property	Decomposition	
	Basic property	Adversary model
Perfect Forward Secrecy	Secrecy	$\{\mathsf{LKR}_{\mathsf{after}}\}$
Weak Perfect Forward Secrecy	Secrecy	$\{\mathsf{LKR}_{\mathsf{aftercorrect}}\}$
Known-Key Security & Unknown-Key Share	Secrecy of session key	$\{\mathsf{SKR}\}$
Key Compromise Impersonation	Authentication/Secrecy of sess. key	$\{\mathsf{LKR}_{\mathsf{actor}}\}$

for a hardware security module restricted to protecting long-term keys, relevant properties could be secrecy or agreement, resilient to state-reveal. Other properties arise by considering the combination of our adversary models with other basic properties like nonrepudiation, (plausible) deniability, anonymity, or resistance to denial-of-service attacks.

3.8. Relations between Models and Properties

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

To begin with, our partial order on adversary models $\leq_{\mathcal{A}}$ has implications for security protocol verification. Given a state property ϕ like those from Section 3.6, a protocol that satisfies ϕ in a model also satisfies ϕ in all weaker models. Equivalently, falsification in a model entails falsification in all stronger models. Formally, if $Adv \leq_{\mathcal{A}} Adv'$, then for all protocols P and state properties ϕ, $sat(P, Adv', \phi) \Rightarrow sat(P, Adv, \phi)$ and, equivalently, $\neg sat(P, Adv, \phi) \Rightarrow \neg sat(P, Adv', \phi)$.

Since adding adversary rules only results in more transitions and hence more reachable states, we have the following.

LEMMA 3.14 (ADDING RULES STRENGTHENS THE ADVERSARY). *Let r be an adversary rule from Figure 4 and Adv be an adversary model, that is, a set of adversary rules. Then, $Adv \leq_{\mathcal{A}} Adv \cup \{r\}$.*

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

LEMMA 3.15 (RULE INDEPENDENCE). *Let Adv be an adversary model. Then we have for all adversary rules r from Figure 4,*

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

PROOF OF ($\Rightarrow$). Let $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}} \in Adv$. Each transition using $\mathsf{LKR}_{\mathsf{aftercorrect}}$ can be simulated using $\mathsf{LKR}_{\mathsf{after}}$. Hence the sets of reachable states on both sides of the above equality are equal and thus $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. □

PROOF OF ($\Leftarrow$). Let $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. Suppose $r \neq \mathsf{LKR}_{\mathsf{aftercorrect}}$. Then there are transitions enabled by r that are not enabled by the other rules. In particular, even if $\mathsf{LKR}_{\mathsf{aftercorrect}} \in Adv$, there are protocols with roles that can be completed without matching sessions, whereby $\mathsf{LKR}_{\mathsf{after}}$ enables transitions not enabled by $\mathsf{LKR}_{\mathsf{aftercorrect}}$. Hence we have a contradiction, and therefore $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$. Now suppose $\mathsf{LKR}_{\mathsf{after}} \notin Adv$. Then some transitions enabled by r are not enabled by $Adv \setminus \{r\}$, contradicting $Adv \setminus \{r\} =_{\mathcal{A}} Adv \cup \{r\}$. Hence $r = \mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}} \in Adv$. □

COROLLARY 3.16. *The rules in Figure 4 give rise to $2^5 \times 3 = 96$ models with distinct sets of reachable states.*

This corollary follows from Lemmas 3.14 and 3.15. In particular, the five rules except for $\mathsf{LKR}_{\mathsf{aftercorrect}}$ and $\mathsf{LKR}_{\mathsf{after}}$ are independent, giving rise to 2^5 models. The $\mathsf{LKR}_{\mathsf{after}}$ rule strictly subsumes the transitions enabled by the $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rule, and therefore the union of the two rules does not yield additional transitions. The possible combinations of these two rules therefore yield three distinct sets of enabled transitions.

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

Definition 3.17 (*Post-Test Invariant Properties*). We define the set of *post-test invariant properties* as all state properties $\phi \in \Phi$ that satisfy

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

Aliveness, as given in Definition 3.13, is post-test invariant: the transitions that occur after the end of the test thread do not influence the property. Other authentication properties such as various forms of agreement [Lowe 1997] or synchronization [Cremers et al. 2006] are also post-test invariant. Secrecy, however, is not such a property.

THEOREM 3.18 (POST-TEST INVARIANT PROPERTIES AND FUTURE CAPABILITIES). *Let r be an adversary rule from Figure 4 and ϕ be a post-test invariant property. Then for all protocols P and adversary models Adv,*

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

Put differently, post-test invariant properties are resilient to future capabilities.

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

Note that some alternate approaches use a fixed adversary model but use different assumptions in different security goals to constrain which adversary actions are considered in that goal. In those approaches, our partial order on adversary models is mirrored by the partial order of logical entailment among the assumptions.

4. PROTOCOL-SECURITY HIERARCHIES

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

It follows from Corollary 3.16 that determining for which adversary models a protocol satisfies its state properties involves analyzing the protocol with respect to 96 models. Since this is infeasible to do by hand, we will use automated analysis methods for this task.

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

Definition 4.1 (*Recursive Approximation of sat*). We say that a function $f \in Protocol \times A \times \Phi \to \{F, \bot, V\}$ *recursively approximates sat* if and only if f is recursive and for all protocols P, adversary models Adv, and state properties ϕ, we have $f(P, Adv, \phi) \neq \bot \Rightarrow \big(f(P, Adv, \phi) = V \Leftrightarrow sat(P, Adv, \phi)\big)$.

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

Definition 4.2 (*Protocol-Security Hierarchy*). Let Π be a set of protocols, ϕ a state property, A be a set of adversary models, and let f recursively approximate *sat*. The *protocol-security hierarchy* with respect to Π, A, ϕ, and f is a directed graph $H = (N, \to)$ that satisfies the following properties.

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

LEMMA 4.3. *Let $H = (N, \to)$ be a protocol-security hierarchy with respect to Π, ϕ, A, and f. Let $\leq_H$ be defined as follows: for all $\pi, \pi' \in N$, $\pi \leq_H \pi'$ iff $\pi \to \pi'$. Then $\leq_H$ is a partial order.*

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

Effectively, a node in a protocol-security hierarchy represents an equivalence class of protocols whose members satisfy the security property with respect to the same set of adversary models. When visualizing a protocol-security hierarchy, we display two sets for each node: the set of protocols P and the set of adversary models S for which the protocols satisfy the security property. Roughly speaking, each of the protocols in P satisfies the security property with respect to the adversary model m if and only if there exists an adversary model m', such that $m \leq_{\mathcal{A}} m'$ and $m' \in S$. Note that we omit implied weaker adversary models from the visualization, that is, if P satisfies its properties with respect to m and m', where $\{m, m'\} \subseteq A$ and $m \leq_{\mathcal{A}} m'$, we include m' but not m in the set of adversary models on the node.

Formally, we annotate each node π with all adversary models $a \in A$ for which
$$\forall a' \in A, P \in \pi. (a <_{\mathcal{A}} a' \Rightarrow f(P, a', \phi) = F) \wedge (a' \leq_{\mathcal{A}} a \Rightarrow f(P, a', \phi) \neq F),$$
where we use $\leq_{\mathcal{A}}$ and $<_{\mathcal{A}}$ from Definition 3.11. □

Example 4.4 (*Protocol-Security Hierarchy*). In Figure 7, we show an example of a simple protocol-security hierarchy for two well-known protocols, the Needham-Schroeder protocol and Lowe's variant, with respect to the synchronization property.

The Needham-Schroeder protocol [Needham and Schroeder 1978] is resilient to adversaries capable of $\mathsf{LKR}_{\mathsf{after}}$ and SKR. By Theorem 3.18, all authentication properties of any protocol are resilient to $\mathsf{LKR}_{\mathsf{after}}$. The fact that the protocol is resilient to SKR is not surprising, as the protocol does not contain any session keys. Conversely, if the adversary has the $\mathsf{LKR}_{\mathsf{others}}$ capability, the Needham-Schroeder protocol is vulnerable

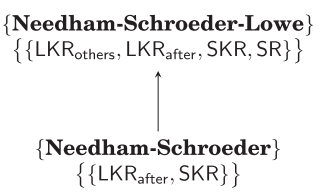

Fig. 7. A simple protocol-security hierarchy for authentication.

to a man-in-the-middle attack. There are also attacks on the protocol if the adversary has any capability from the set $\{\mathsf{LKR}_{\mathsf{actor}}, \mathsf{RNR}, \mathsf{SR}\}$. For example, the protocol is vulnerable to SR because the missing identity in the second message allows the adversary to exploit a nonmatching session to decrypt this message, in which he uses SR to reveal the nonce of the first message. Therefore, the Needham-Schroeder protocol satisfies synchronization with respect to exactly four adversary models: $\emptyset$, $\{\mathsf{LKR}_{\mathsf{after}}\}$, $\{\mathsf{SKR}\}$, and $\{\mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}\}$. These four models are all weaker than or equal to $\{\mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}\}$. Hence we label the node with (1) the singleton set of the Needham-Schroeder protocol, and (2) the (singleton) set of adversary models in which it is correct, where we omit all weaker adversary models.

Lowe's version of the protocol [Lowe 1996] was designed to prevent the man-in-the-middle attack that exploits $\mathsf{LKR}_{\mathsf{others}}$. Perhaps surprisingly, his fix also prevents the SR attack. In the fixed version of the protocol, sessions that do not match the test session will not accept messages that contain the test-session's nonce. Thus, the SR rule can no longer be used to reveal these nonces, preventing the attack. Combining these results, we draw a node for the Needham-Schroeder-Lowe protocol that lists the protocol's resilience to the adversary model $\{\mathsf{LKR}_{\mathsf{others}}, \mathsf{LKR}_{\mathsf{after}}, \mathsf{SKR}, \mathsf{SR}\}$ and any weaker models.

Because the Needham-Schroeder-Lowe protocol meets the security property with respect to all adversary models (and more) than those for which the Needham-Schroeder protocol meets the property, we draw an edge between the nodes.

Automatic Generation of Protocol-Security Hierarchies. Because of the large number of possible adversary models, manual construction of protocol-security hierarchies is infeasible for large sets of protocols and is best aided by a model checker.

The Scyther tool [Cremers 2008a] recursively approximates *sat* and produces an output from $\{F, \perp, V\}$. F denotes that Scyther found an attack, thereby falsifying the property, V denotes that Scyther verified the property, and $\perp$ denotes that either a timeout occurred or Scyther could not verify the property without bounds.

Using our extended version of the Scyther tool, the properties of a protocol can be automatically analyzed with respect to all adversary models. Given a set of protocols, the tool computes a protocol-security hierarchy by combining this data for each of the protocols with the order $\preceq_{\mathcal{A}}$ on the adversary models. The protocol description files, analysis tools, and graph generation scripts can be downloaded from Cremers [2014].

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

Definition 4.5 (*Strictness of Edges in a Protocol-Security Hierarchy*). We say an edge $\pi \rightarrow \pi'$ in a protocol-security hierarchy is *strict* if the following two properties hold.

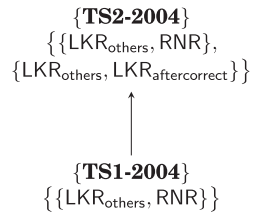

Fig. 8. A simple protocol-security hierarchy for secrecy. The edge is nonstrict.

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

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

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

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

All edges in the authentication hierarchies in Figure 7 and Figure 12 are strict. This reflects Scyther's success in either verifying or falsifying these protocols. In contrast, for the secrecy hierarchies in Figure 8 and Figure 11, most protocols contain Diffie-Hellman exponentiation, for which Scyther does not provide (unbounded) verification. Therefore, the edges in these figures are only based on attacks. Because they are not strict, they might not occur in the corresponding hierarchy based on *sat*.

Intuitively, the interpretation of the set of adversary models of a node π, for hierarchies with non-strict edges, is the following.

(1) No attacks have been found on the protocols for these models and all weaker ones.
(2) For each node π' higher in the hierarchy, there exists an adversary model a in the annotation of π' such that an attack was found on all protocols from π for the adversary model a.

In Section 5.4, we perform an extensive case study that includes the automatic generation of larger protocol security hierarchies. We also show how such hierarchies can be used in protocol analysis.

5. APPLICATIONS AND CASE STUDIES

5.1. Modeling Adversary 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 [Canetti and Krawczyk 2001; Bellare et al. 2000; Bellare and Rogaway 1995; Krawczyk 2005a; LaMacchia et al. 2007]. We focus on the adversary capabilities only, abstracting from subtle differences between the computational models. For example, the model in Canetti and Krawczyk [2001] has an execution model that restricts the agents' choice of thread identifiers, leading to a different notion of partner threads than in other models. Here we define partnering uniformly by matching histories. We refer the reader to previous works [Bresson et al. 2007; Choo et al. 2005a, 2005b; Menezes and Ustaoglu 2008] for further details on the differences between computational models.

Table II provides an overview of different adversary models, interpreted as instances of our semantics. We write Adv_{CK} to denote the adversary model extracted from the CK model [Canetti and Krawczyk 2001] and similarly for other models. A check (✓) denotes that the rule labeling the column is included in the adversary model named in the row.

Table II. Mapping Adversary-Compromise Models from the Literature

	Long-term data				Short-term data			
	Owner		Timing		Type			
Name	others	actor	after	aftercorrect	SessionKey	State	Random	Origin of model
Adv_{EXT}								external Dolev-Yao
Adv_{INT}	✓							Dolev-Yao [Lowe 1996]
Adv_{CA}		✓						Key Compromise Impersonation [Just and Vaudenay 1996]
Adv_{AFC}				✓				Weak Perfect Forward Secrecy [Krawczyk 2005a]
Adv_{AF}			✓	✓				Perfect Forward Secrecy Secrecy [Günther 1990, Menezes et al. 1996]
Adv_{BR93}					✓			BR93 [Bellare and Rogaway 1993]
Adv_{BR95}	✓				✓			BR95 [Bellare and Rogaway 1995]
Adv_{BPR00}	✓		✓	✓	✓			BPR00 [Bellare et al. 2000]
Adv_{CKw}	✓	✓		✓	✓	✓		CK2001-wPFS [Krawczyk 2005a]
Adv_{CK}	✓		✓	✓	✓	✓		CK2001 [Canetti and
								Krawczyk 2001]
$Adv_{\text{eCK-1}}$	✓				✓		✓	eCK [LaMacchia et al. 2007]
$Adv_{\text{eCK-2}}$	✓	✓		✓	✓			

5.2. Tool Support

We extended the symbolic security-protocol verification tool Scyther [Cremers 2008a; 2008b] with our adversary rules from Figure 4. We used this tool to automatically analyze a set of protocols, described next. The tool, all protocol models, and test scripts can be downloaded from Cremers [2014].

In Figure 9, we show the concrete input file for the Scyther tool that specifies the two-message signed Diffie-Hellman protocol. In the description, the `Ticket` type of the variables alpha and beta corresponds to all possible terms. The abstract functions `exp` and `exp-g`, respectively, denote modular exponentiation of two terms and raising the generator g to the power of a term. Additional rules to approximate the Diffie-Hellman equational theory have been omitted from the figure but can be found in the downloadable model archives. The claim events `claim(I,SKR,...)` and `claim(R,SKR,...)` encode the desired security property. Here, `SKR` denotes that the following term is a session-key, which implies that (a) it can be revealed through the SKR rule, and (b) it should be secret as in Definition 3.12. The adversary model is specified independently, either through the tool's GUI or through its command-line/batch mode. Scyther coarsely approximates Diffie-Hellman using additional roles that model a subset of the equations for modular exponentiation, see Cremers [2014].

5.3. Attack Examples

MQV, HMQV, and Variants. The MQV protocol family [Krawczyk 2005b; Ustaoglu 2008; Law et al. 2003] is a class of authenticated key-exchange protocols designed to provide strong security guarantees. The HMQV protocol was proven secure with

SIG-DH-core.spdl

```
/* Two-move Diffie-Hellman in the unauthenticated-links model.
 * From CK2001, p. 20
 */
// Collision-resistant one-way functions
hashfunction exp-g,exp;

// The protocol description
protocol SIG-DH-UM(I,R) {
        role I {
                fresh s: Nonce;
                fresh x: Nonce;
                var beta: Ticket;

                send_1(I,R, I,s,exp-g(x) );
                recv_2(R,I, R,s,beta, { R,s,beta,exp-g(x),I }sk(R) );
                send_3(I,R, I,s, { I,s,exp-g(x),beta,R }sk(I) );

                claim(I,SKR, exp(beta,x) );
        }
        role R {
                fresh y: Nonce;
                var s: Nonce;
                var alpha: Ticket;

                recv_1(I,R, I,s,alpha );
                send_2(R,I, R,s,exp-g(y), { R,s,exp-g(y),alpha,I }sk(R) );
                recv_3(I,R, I,s, { I,s,alpha,exp-g(y),R }sk(I) );

                claim(R,SKR, exp(alpha,y) );
        }
}
```

Fig. 9. Input file for the Scyther tool, specifying the two-message signed Diffie-Hellman protocol.

respect to the adversary model in Krawczyk [2005a]. 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. Our tool automatically finds that under certain definitions of the session-state, the HMQV protocol is insecure in adversary models that contain SR rules, such as the CK model Canetti and Krawczyk [2001]. The attack found is similar to the one described in [Kunz-Jacques and Pointcheval 2006].

Next we describe the attack, which shows that MQV and HMQV are insecure in, for example, Adv_{CKw}, if the final exponentiation in the computation of the session key is performed in the local state. It is possible for an adversary to reuse the inputs to this exponentiation to impersonate an agent in future sessions. The attack is not covered in Krawczyk [2005a; 2005b], because both the proof and the extended analysis given there assume that the local state contains only the ephemeral keys (the temporary private keys), which in this case correspond to the random values x and y.

A description of the HMQV protocol was given in Section 2; see Figure 1. We show the SR attack on HMQV in Figure 10, where $d_1 = \bar{H}(X, \mathrm{Bob})$, $e_1 = \bar{H}(Z, \mathrm{Alice})$, and $e_2 = \bar{H}(Y, \mathrm{Alice})$. The attack starts with Bob receiving a message g^z apparently coming from Alice. This message may have been sent by another agent or may 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 thread 1's state, accessing $x + d_1 b$. At any desired time, the adversary sends X to the responder test thread of Alice. Alice computes and sends $Y = g^y$ and computes the session key based on X and y. The adversary intercepts Y and computes $H((YA^{e_1})^{x+d_1 b})$. This yields the session key of the test thread.

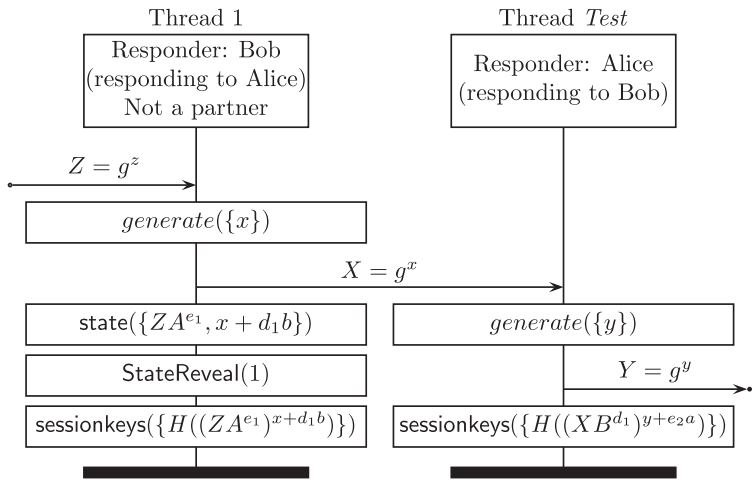

Fig. 10. SR attack on HMQV.

We assume that in critical scenarios, the protocol is implemented entirely in a tamper-proof module or cryptographic coprocessor, and the local state is therefore empty, preventing this attack. Conversely, if (H)MQV is implemented entirely in unprotected memory, the state will also include the long-term keys, which enables an attack where the adversary compromises these keys using SR. This example shows how analysis with respect to our models can help sharpen protocol implementation requirements.

The YAK protocol [Hao 2010] adds additional information and zero-knowledge proofs to the initial HMQV message exchange. In our analysis, YAK achieves the same security guarantees as the HMQV protocol with respect to our adversary models. In contrast, the CF protocol [Cremers and Feltz 2011] adds explicit signatures to the initial HMQV exchange to achieve perfect forward secrecy directly. Our analysis reveals that CF indeed achieves perfect forward secrecy, confirming the results proven for CF in the computational setting [Cremers and Feltz 2011].

KEA+ and Naxos. KEA+ [Lauter and Mityagin 2006] and Naxos [LaMacchia et al. 2007] belong to the same class of protocols. In Lauter and Mityagin [2006], KEA+ is proven correct with respect to a variant of the adversary model of Krawczyk [2005a], 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 SR attacks and therefore are insecure in, for example, the Adv_{CK} model. Additionally, for KEA+ we find an attack using the $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rule, and hence KEA+ does not satisfy weak perfect forward secrecy. This attack cannot be modified to work for Naxos.

Yahalom. Paulson [2001] 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 (SKR). At first sight, this appears to contradict Paulson's result; however, the discrepancy is due to different properties being considered. In Paulson [2001], the adversary may compromise other session keys. In contrast, our

Table III. Attacks Found on Secrecy with Respect to the Adversary Models from Table II

	EXT	INT	CA	AFC	AF	BR	CKw	CK	eCK-1	eCK-2
DH-GS									×	
DH-GS-C								×	×	
BCNP-1				×	×		×	×	×	×
BCNP-2					×		×	×	×	
CF							×	×		
DHKE-1							×		×	×
HMQV-C							×	×		
HMQV					×		×	×		
NAXOS					×		×	×		
SIG(NAXOS)							×	×		
TS1 (2004)			×	×	×	×	×	×	×	×
TS1 (2008)			×	×	×		×	×		×
TS2 (2004)			×		×	×	×	×	×	×
TS2 (2008)			×		×		×	×		×
TS3 (2004)			×				×	×	×	×
TS3 (2008)			×				×	×	×	×
UM			×		×	×	×	×	×	×
YAK					×		×	×		
KEA+				×	×		×	×	×	×
BKE			×	×	×		×	×	×	×
Yahalom-Paulson			×	×	×	×	×	×	×	×

SKR rule, following the definitions from key-agreement literature, allows the adversary to compromise keys of threads that are not partners of the test thread.

Other Case Studies. In Table III, we summarize the attacks found using our tool on protocols with a secrecy requirement with respect to the adversary models from Table II. A cross (×) denotes that an attack was found. The set of protocols includes both formally analyzed protocols (BKE and Yahalom) as well as protocols recently proposed in computational settings (HMQV, UM, and SIG(NAXOS)). Our tool rediscovers the attacks described in the literature, for example, that signed Diffie-Hellman is insecure in the eCK model [LaMacchia et al. 2007] and that the implicitly authenticated two-message protocols KEA+, Naxos, and HMQV do not satisfy perfect forward secrecy. Additionally, our tool finds new attacks, for example, on KEA+. The time needed for finding the attacks listed ranged from less than a second to three minutes per attack. We present further results obtained for these protocols in Section 5.4.

5.4. Examples of Protocol-Security Hierarchies

In Figure 11, we show the protocol-security hierarchy for the secrecy property of a set of protocols with respect to all possible sets of adversary rules from Figure 4. In Figure 12, we show a protocol-security hierarchy for authentication properties.

Analyzing Protocols Using Protocol-Security Hierarchies. Protocol-security hierarchies provide a novel way for choosing an optimal protocol for a given application domain, for example, exchanging a secret as illustrated here. We next discuss the protocols included in the protocol-security hierarchies in Figure 11 and 12. Both protocol-security hierarchies are automatically generated by the Scyther tool. We only converted the generated Graphviz graphs to PGF/Tikz format for visual consistency. In Figure 11, we added annotations at the top and on the righthand side to highlight information already present in the graph.

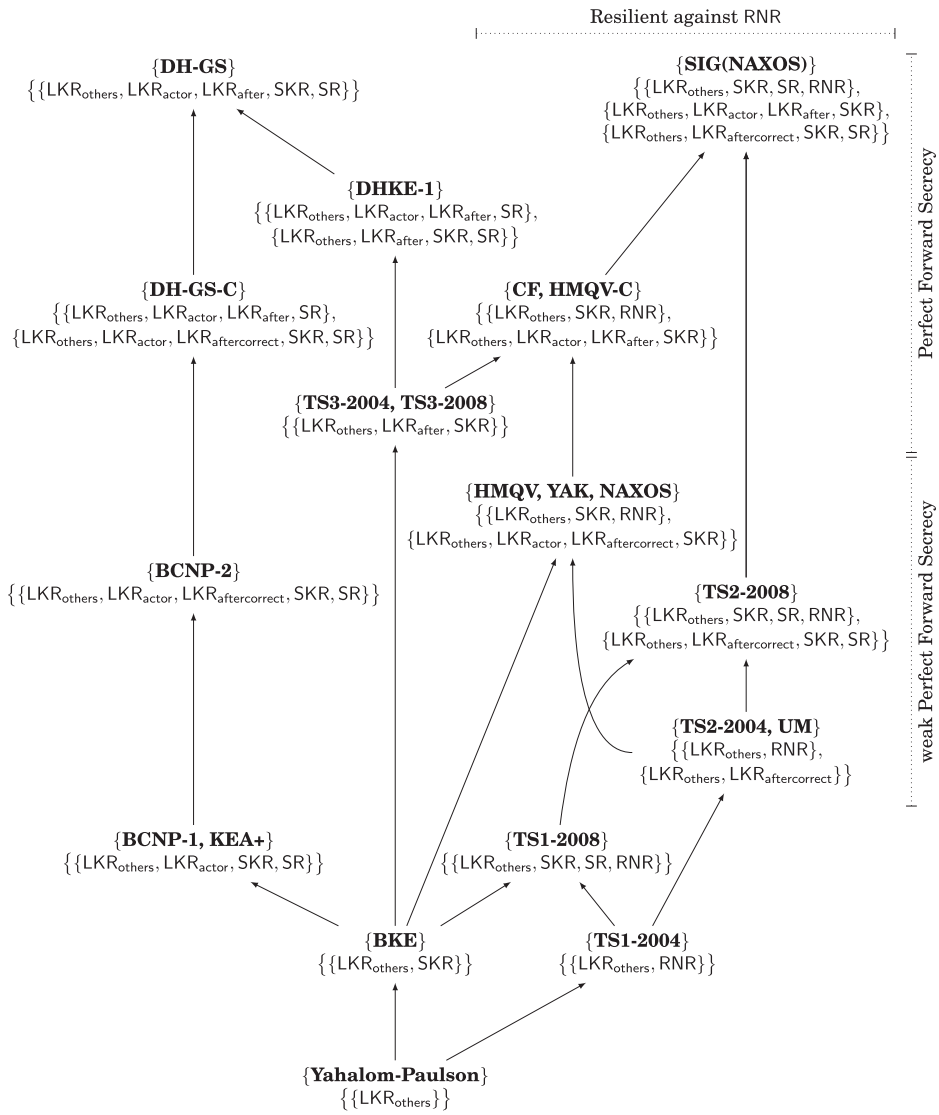

Fig. 11. Protocol-security hierarchy for secrecy of the 21 protocols from Table III with respect to all 96 adversary models.

We show how the resulting hierarchies facilitate fine-grained protocol comparisons that often refine or even contradict comparisons made in the literature. We start by discussing the hierarchy in Figure 11. This is a hierarchy for secrecy for the 21 protocols from Table III with respect to all 96 adversary models.

DH-GS, DH-GS-C, and DHKE-1. The original Diffie-Hellman protocol is only secure in the presence of a passive adversary, since the messages sent are not authenticated. A simple improvement is for agents to sign each message sent, along with the intended recipient, using the sender's long-term signature key. The resulting protocol family is referred to as *signed Diffie-Hellman*.

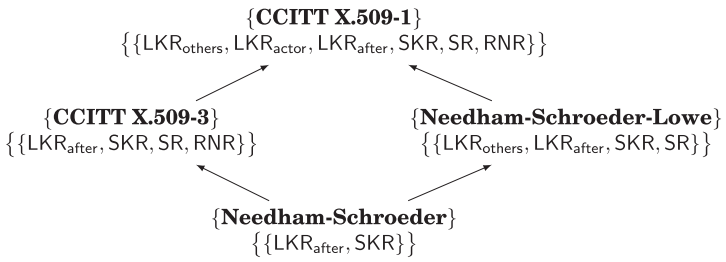

Fig. 12. Protocol-security hierarchy for authentication with respect to all 96 adversary models.

We analyze three variants from Gupta and Shmatikov [2005]. They include a basic two-message version and its extension, a three-message version with key-confirmation. These two protocols appear to be modified versions of ISO entity authentication protocols. We refer to the first protocol as DH-GS and to the second as DH-GS-C. The third protocol is a signed Diffie-Hellman protocol called DHKE-1.

Scyther finds attacks on the signed Diffie-Hellman protocols for all models containing the RNR rule. This is consistent with the proof in Gupta and Shmatikov [2005], which does not consider this rule, as well as with the observation in LaMacchia et al. [2007] that RNR allows an attack on the basic signed Diffie-Hellman protocols.

Contrary to our expectations, the two-message DH-GS protocol ends up higher in our hierarchy than the three-message variant with key-confirmation DH-GS-C. Closer inspection reveals that there are two reasons for this. First, the additional property that the DH-GS-C protocol is meant to achieve, namely, key-confirmation, is not considered in this hierarchy because it is only based on secrecy. Second, the protocol uses an incorrect mechanism to achieve key confirmation: the only difference between the responder's key confirmation part and the initiator's key confirmation part is the order of the included identities, which is insufficient to distinguish them. As a result, Scyther finds an attack on a responder thread in which a party performs a session with itself by performing both roles. After observing the first message and the responder's response, the adversary reuses the responder's confirmation to immediately finish the test thread. The adversary then leaks both long-term private keys and finishes the partial initiator thread with a session identifier i that is different from the one used by the responder test thread. As a result, although both threads compute the same key, they are not partners, as they disagree over the session identifier included in the second and third message. Hence, the adversary can use SKR on the initiator thread to reveal the session key of the test thread.

This attack on DH-GS-C can be prevented, for example, by differentiating between the two confirmation messages by including distinct constants, as is common.

JKL-TS1, JKL-TS2, and JKL-TS3. Jeong et al. propose the protocols TS1, TS2, and TS3 [2004]. TS1 is designed to satisfy *key independence* (keys of nonmatching sessions may be revealed), whereas TS2 and TS3 should additionally satisfy *forward secrecy* (long-term keys of the agents may be revealed after the test session ends). They prove TS1 and TS2 correct in the random oracle model and TS3 in the standard model. Four years later, the authors released a new version of the paper with updated protocols and security models. The main difference is that the original models did not capture certain unknown-key share attacks, and the original protocols where lacking sufficient information on the identity of the participants to prevent these attacks. We differentiate between the two versions by annotating the protocols with (2004) or (2008).

Our protocol-security hierarchy reveals the following. First, the TS3-2004 protocol is incomparable to the other two protocols from the original paper. In contrast to

TS2-2004, TS3-2004 is additionally resilient to $\mathsf{LKR}_{\mathsf{after}}$ and SKR, but it is not resilient to RNR. Second, the TS1-2004 protocol is not resilient to SKR, which implies that it does not satisfy key independence. Indeed, the missing identities in the session identifier of the protocol cause the protocol to be vulnerable to SKR. This is a flaw in the security model in Jeong et al. [2004], which does not adequately capture such attacks. Third, Jeong et al. suggest that the TS2-2004 protocol satisfies forward secrecy. Our analysis shows that it only satisfies weak perfect forward secrecy, that is, resilience to $\mathsf{LKR}_{\mathsf{aftercorrect}}$. The security model [Jeong et al. 2004] requires the adversary to be passive when corrupting agents. This is in contrast to TS3-2004, which does satisfy perfect forward secrecy. In this case, the authors have proven a weaker claim (weak perfect forward secrecy), whereas they might have been able to prove that TS3-2004 satisfies a stronger property.

The repaired protocols from the 2008 version of the paper address the unknown-key share attacks. Subsequently, TS1-2008 and TS2-2008 improve upon their predecessors from 2004 by being resilient to SKR (and, as a side effect, to SR.) The properties of TS3-2008 remain unchanged from its predecessor TS3-2004.

UM. The Unified-Model (UM) protocol [Blake-Wilson and Menezes 1999] was originally proposed by Ankney et al. [1995]. It is a conceptually clean protocol design. However, our analysis confirms that it is vulnerable to both $\mathsf{LKR}_{\mathsf{actor}}$ and SKR, providing similar security to TS2-2004.

BKE. For the bilateral key-exchange (BKE) protocol [Clark and Jacob 1997], we find attacks in all models in our hierarchy except for adversaries capable of $\mathsf{LKR}_{\mathsf{others}}$ or SKR. BKE is therefore among the weakest protocols in our hierarchy. However, because it is resilient to SKR, it is not weaker than TS1 or TS2.

BCNP-1 and BCNP-2. Boyd et al. propose two protocols [2009], which we refer to as BCNP-1 and BCNP-2. When comparing their protocols to others, they focus on two properties, KCI resistance (resilience to $\mathsf{LKR}_{\mathsf{actor}}$) and weak forward secrecy (resilience to $\mathsf{LKR}_{\mathsf{aftercorrect}}$). Additionally, they claim that BCNP-2 provides more security than TS3 [Jeong et al. 2004]. Our analysis allows for a more fine-grained comparison of the different protocols, confirming many remarks made in Boyd et al. [2009], but established by automatic instead of manual analysis. The hierarchy also explains why BCNP-2 does not provide more security than TS3. BCNP-2 is incomparable to TS3 because, unlike TS3, it is KCI-resilient (resilience to $\mathsf{LKR}_{\mathsf{actor}}$) but does not satisfy perfect forward secrecy (resilience to $\mathsf{LKR}_{\mathsf{after}}$). This disproves their claim that BCNP-2 provides more security than TS3.

NAXOS, SIG(NAXOS), and CF. LaMacchia et al. [2007] proposes the Naxos protocol along with a new security model called eCK, claiming that this model is the strongest model for AKE protocols. However, our hierarchy reveals that NAXOS is not stronger than most other protocols in our set because it is vulnerable to SR and $\mathsf{LKR}_{\mathsf{after}}$. This result has independent consequences, reported in Cremers [2010]. NAXOS is resilient to adversaries that are capable of both RNR and SKR.

Cremers and Feltz [2013] show that by applying a signature-based transformation, protocols that are secure in an eCK-like model can be made to additionally satisfy a strong notion of perfect-forward secrecy. The SIG(NAXOS) protocol is the result of applying this transformation to the NAXOS protocol. Our analysis confirms the results from their paper.

Next, we discuss the hierarchy for authentication presented in Figure 12. We verify the protocols with respect to a strong form of authentication called *synchronization* [Cremers et al. 2006]. Protocols that satisfy synchronization also satisfy aliveness.

Needham-Schroeder and Needham-Schroeder-Lowe. These protocols were already discussed in Example 4.4.

CCITT X.509-1 and X.509-3. The CCITT X.509 standard [CCITT 1987] contains several protocol recommendations. Here we consider X.509-1 and X.509-3. X.509-1 satisfies its authentication properties with respect to the strongest possible adversary model, that is, the adversary with all capabilities from Figure 4. The X.509-3 protocol is not resilient to $\mathsf{LKR}_{\mathsf{others}}$ or $\mathsf{LKR}_{\mathsf{actor}}$. However, unlike Needham-Schroeder(-Lowe), it is resilient to RNR.

6. 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., [Bellare and Rogaway 1993, 1995; Bellare et al. 2000; Bresson and Manulis 2008; Canetti and Krawczyk 2001; Cremers and Feltz 2013; Katz and Yung 2003; Krawczyk 2005a; LaMacchia et al. 2007; Shoup 1999]). In general, any two computational models are incomparable due to (often minor) differences not only in the adversary notions, but also in the definitions of partnership, 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 (e.g., Bresson et al. 2007; Choo et al. 2005a, 2005b; Cremers 2011; LaMacchia et al. 2007; Menezes and Ustaoglu 2008]).

The CryptoVerif tool [Blanchet 2006] is a mechanized tool for computational analysis. Its adversary model covers Adv_{INT}, corresponding to static corruption, that is, the classical Dolev-Yao adversary. It also supports multiple test queries, as in Abdalla [2006].

Related Work in Symbolic Analysis. In the symbolic setting, Guttman [2001] 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 ad-hoc constructions of key compromise, made for specific protocols, which can be verified in a Dolev-Yao style adversary model.

Abadi et al. [2007] analyzed the JFK protocol in the Pi calculus and showed it achieves perfect forward secrecy, by giving the adversary all long-term keys at the end of the protocol run. This corresponds to manually instrumenting the analog of our $\mathsf{LKR}_{\mathsf{after}}$ rule. In Blanchet [2009], a form of session-key compromise is modeled in the ProVerif framework by assuming that session-keys are compromised strictly before the target thread starts. This form of session-key compromised is covered by our SKR rule; however, our rule covers many additional scenarios, such as parallel-session key-compromise and unknown-key share attacks. Paulson used his inductive approach to reason about the compromise of short-term data [2001]. To model compromise, he adds a rule to the protocol, called *Oops*, that directly gives short-term data to the adversary. This rule is roughly analogous to our SKR rule. Gupta and Shmatikov [2005, 2006] link a symbolic adversary model that includes dynamic corruptions to an adversary model used in the computational analysis of key-agreement protocols. They describe [Gupta and Shmatikov 2006] a cryptographically-sound logic that can be used to prove security in the presence of adaptive corruptions, that is, the adversary can dynamically obtain the long-term keys of agents.

7. CONCLUSIONS

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

Our adversary capabilities generalize those from the computational setting and combine them with a symbolic model. In doing so, we unify and generalize a wide range of models from both settings. Exploring the exact nature of this generalization as well as mappings between the two settings remains as future work. It would be interesting to develop methods for designing protocols, for example, using refinement, as in Sprenger and Basin [2010, 2012], that are optimized for different adversarial scenarios or strengthening existing protocols.

Our definitions of adversaries and security properties from the computational setting allow us to apply symbolic techniques to problems that were previously tackled only by computational approaches. We developed the first tool capable of systematically handling notions such as weak perfect forward secrecy, key compromise impersonation, and session state compromise. In case studies, our tool not only rediscovered many attacks previously reported in the cryptographic literature, for example, on HMQV and DH-GS, it also found new attacks, for example, on KEA+. These examples show that our symbolic adversary models are surprisingly effective for automatically establishing results that, until now, required labor-intensive manual computational analysis.

Our formalization can serve as a reference for defining the adversarial concepts from the computational setting in other symbolic frameworks. These include Strand Spaces [Guttman 2001], the Pi calculus [Abadi et al. 2007], and the frameworks of Paulson [1998] and Schmidt et al. [2013].

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

REFERENCES

Abadi, M., Blanchet, B., and Fournet, C. 2007. Just fast keying in the pi calculus. *ACM Trans. Inf. Syst. Secur. 10*, 3, 1–59.

Abdalla, M. 2006. Password-based authenticated key exchange in the three-party setting. *IEE Proc. Inf. Secur. 153*, 12, 27–39.

Basin, D. and Cremers, C. 2010a. Modeling and analyzing security in the presence of compromising adversaries. In *Proceedings of the Conference on Computer Security (ESORICS'10)*. Lecture Notes in Computer Science, Vol. 6345, Springer, Berlin, 340–356.

Basin, D. A. and Cremers, C. J. 2010b. Degrees of security: Protocol guarantees in the face of compromising adversaries. In *Proceedings of the 24th International Workshop on Computer Science Logic (CSL'10)*. Lecture Notes in Computer Science, Vol. 6247, Springer, Berlin, 1–18.

Bellare, M., Pointcheval, D., and Rogaway, P. 2000. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology—EUROCRYPT 2000*. Lecture Notes in Computer Science, Vol. 1807, Springer, Berlin, 139–155.

Bellare, M. and Rogaway, P. 1993. Entity authentication and key distribution. In *Advances in Cruptology—CRYPTO*. Lecture Notes in Computer Science, Vol. 773, Springer, Berlin, 232–249.

Bellare, M. and Rogaway, P. 1995. Provably secure session key distribution: The three party case. In *Proceedings of the 27th Annual Symposium in Theory of Computing (STOC'95)*. ACM, 57–66.

Blake-Wilson, S. and Menezes, A. 1999. Authenticated diffie-hellman key agreement protocols. In *Proceedings of the Selected Areas in Cryptography (SAC'98)*. 339–361.

Blanchet, B. 2001. An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE, 82–96.

Blanchet, B. 2006. A computationally sound mechanized prover for security protocols. In *Proccedings of the IEEE Symposium on Security and Privacy*. 140–154.

Blanchet, B. 2009. Automatic verification of correspondences for security protocols. *J. Comput. Secur. 17*, 4, 363–434.

Boyd, C., Cliff, Y., Nieto, J. M. G., and Paterson, K. G. 2009. One-round key exchange in the standard model. *Inf. J. Appl. Crypto. 1*, 3, 181–199.

Bresson, E. and Manulis, M. 2008. Securing group key exchange against strong corruptions. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. ACM, 249–260.

Bresson, E., Manulis, M., and Schwenk, J. 2007. On security models and compilers for group key exchange protocols. In *Proceedings of the 2nd International Workshop on Security (IWSEC)*. Lecture Notes in Computer Science, Vol. 4752, Springer, Berlin, 292–307.

Canetti, R. and Krawczyk, H. 2001. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology—EUROCRYPT*. Lecture Notes in Computer Science, Vol. 2045, Springer, Berlin, 453–474.

CCITT. 1987. The directory authentification framework. Draft Recommendation X.509, Version 7.

Choo, K.-K., Boyd, C., and Hitchcock, Y. 2005a. Examining indistinguishability-based proof models for key establishment proofs. In *Advances in Cryptology—ASIACRYPT*. Lecture Notes in Computer Science, Vol. 3788, Springer, Berlin, 624–643.

Choo, K.-K., Boyd, C., Hitchcock, Y., and Maitland, G. 2005b. On session identifiers in provably secure protocols. In *Proceedings of the 4th International Conference on Security Communication Networks (SCN'05)*. Lecture Notes in Computer Science, Vol. 3352, Springer, Berlin, 351–366.

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

Cremers, C. 2008a. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science, Vol. 5123, Springer, Berlin, 414–418.

Cremers, C. 2008b. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, 119–128.

Cremers, C. 2010. Session-StateReveal is stronger than eCK's EphemeralKeyReveal: Using automatic analysis to attack the NAXOS protocol. *Int. J. Appl. Crypto. 2*, 83–99.

Cremers, C. 2011. Examining indistinguishability-based security models for key exchange protocols: The case of CK, CK-HMQV, and eCK. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS'11)*. ACM, 80–91.

Cremers, C. 2014. Scyther tool with compromising adversaries extension. Includes protocol description files and test scripts. http://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html.

Cremers, C. and Feltz, M. 2011. One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300. http://eprint.iacr.org/.

Cremers, C. and Feltz, M. 2013. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. *Des. Codes Crypt.*

Cremers, C., Mauw, S., and de Vink, E. 2006. Injective synchronisation: An extension of the authentication hierarchy. *Theor. Comput. Sci. 367*, 1, 139–161.

Günther, C. 1990. An identity-based key-exchange protocol. In *Advances in Cryptology—EUROCRYPT'89*. Lecture Notes in Computer Science, Vol. 434, Springer, Berlin, 29–37.

Gupta, P. and Shmatikov, V. 2005. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceedings of the ACM Workshop on Formal Methods in Security Engineering (FMSE'05)*. ACM, 23–32.

Gupta, P. and Shmatikov, V. 2006. Key confirmation and adaptive corruptions in the protocol security logic. In *Proceedings of the Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06)*. 113–142.

Gutmann, P. Performance characteristics of application-level security protocols. Draft paper www.cs.auckland.ac.nz/~pgut001/pubs/app_sec.pdf.

Guttman, J. D. 2001. Key compromise, strand spaces, and the authentication tests. In *Proceedings of the 17th Annual Conference on Mathematical Foundations of Programming Semantics*. 141–161.

Hao, F. 2010. On robust key agreement based on public key authentication. In *Financial Cryptography and Data Security*, R. Sion, Ed. Lecture Notes in Computer Science, Vol. 6052. Springer, Berlin Heidelberg, 383–390.

Jeong, I. R., Katz, J., and Lee, D. H. 2004. One-round protocols for two-party authenticated key exchange. In *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security (ACNS'04)*. Lecture Notes in Computer Science, Vol. 3089, Springer, Berlin, 220–232.

Just, M. and Vaudenay, S. 1996. Authenticated multi-party key agreement. In *Advances in Cryptology—ASIACRYPT 1996*. Lecture Notes in Computer Science, Vol. 1163, Springer, Berlin, 36–49.

Katz, J. and Yung, M. 2003. Scalable protocols for authenticated group key exchange. In *Advances in Cryptology—CRYPTO*. Lecture Notes in Computer Science, Vol. 2729, Springer, Berlin, 110–125.

Krawczyk, H. 2005a. HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176. http://eprint.iacr.org/. (Last Accessed on April 14, 2009).

Krawczyk, H. 2005b. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology—CRYPTO*. Lecture Notes in Computer Science, Vol. 3621, Springer, Berlin, 546–566.

Kunz-Jacques, S. and Pointcheval, D. 2006. A new key exchange protocol based on MQV assuming public computations. In *Security and Cryptography for Networks*, R. Prisco and M. Yung, Eds. Lecture Notes in Computer Science, Vol. 4116, Springer, Berlin, 186–200.

LaMacchia, B., Lauter, K., and Mityagin, A. 2007. Stronger security of authenticated key exchange. In *Proceedings of the 1st International Conference on Provable Security (ProvSec)*. Lecture Notes in Computer Science, Vol. 4784. Springer, Berlin, 1–16.

Lauter, K. and Mityagin, A. 2006. Security analysis of KEA authenticated key exchange protocol. In *Proceedings of the 9th International Conference on Theory and Practice in Public-Key Cryptography (PKC'06)*. Lecture Notes in Computer Science, Vol. 3958, Springer, Berlin, 378–394.

Law, L., Menezes, A., Qu, M., Solinas, J., and Vanstone, S. 2003. An efficient protocol for authenticated key agreement. *Des. Codes Crypto. 28*, 119–134.

Lowe, G. 1996. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*. Lecture Notes in Computer Science, Vol. 1055, Springer, Berlin, 147–166.

Lowe, G. 1997. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE, 31–44.

Menezes, A. and Ustaoglu, B. 2008. Comparing the pre- and post-specified peer models for key agreement. In *Proceedings of the 13th Australasian Conference on Information Security and Privacy (ACISP'08)*. Lecture Notes in Computer Science, Vol. 5107, Springer, Berlin, 53–68.

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

Needham, R. and Schroeder, M. 1978. Using encryption for authentication in large networks of computers. *Commun. ACM 21*, 12, 993–999.

Paulson, L. 1998. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur. 6*, 1–2, 85–128.

Paulson, L. 2001. Relations between secrets: Two formal analyses of the Yahalom protocol. *J. Comput. Secur. 9*, 3, 197–216.

Schmidt, B., Meier, S., Cremers, C., and Basin, D. 2013. The TAMARIN Prover for the symbolic analysis of security protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science, Vol. 8044, Springer, Berlin, 696–701.

Shoup, V. 1999. On formal models for secure key exchange (version 4). Revision of IBM Research Report RZ 3120 (April 1999).

Sprenger, C. and Basin, D. 2010. Developing security protocols by refinement. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'10)*. ACM, 361–374.

Sprenger, C. and Basin, D. 2012. Refining key establishment. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF)*. 230–246.

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

Received December 2013; revised May 2014; accepted July 2014