Modeling and Analyzing Security in the Presence of Compromising Adversaries

David Basin and Cas Cremers*

Department of Computer Science, ETH Zurich

Abstract. We present a framework for modeling adversaries in security protocol analysis, ranging from a Dolev-Yao style adversary to more powerful adversaries who can reveal different parts of principals' states during protocol execution. Our adversary models unify and generalize many existing security notions from both the computational and symbolic settings. We extend an existing symbolic protocol-verification tool with our adversary models, resulting in the first tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of state-reveal queries. In case studies, we automatically find new attacks and rediscover known attacks that previously required detailed manual analysis.

1 Introduction

Problem context. Many cryptographic protocols are designed to work in the face of various forms of corruption. For example, a Diffie-Hellman key agreement protocol, where signatures are used to authenticate the exchanged half-keys, provides perfect forward secrecy [17,30]: the resulting key remains secret even when the signature keys are later compromised by the adversary. Designing protocols that work even in the presence of different forms of adversary compromise has considerable practical relevance. It reflects the multifaceted computing reality with different rings of protection (user-space, kernel space, hardware security modules) offering different levels of assurance with respect to the computation of cryptographic functions (e. g., the quality of the pseudo-random numbers generated) and the storage of keys and intermediate results.

Symbolic and computational approaches have addressed this problem to different degrees. Most symbolic formalisms are based on the Dolev-Yao model. These offer, with few exceptions, a limited view of honesty and conversely corruption: either principals are honest from the start and always keep their secrets to themselves or they are completely malicious

* This work was supported by ETH Research Grant ETH-30 09-3 and the FP7-ICT-2007-1 Project no. 216471 (AVANTSSAR).

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 [12] and state-of-the art protocols such as (H)MQV [23,27]. 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.

In contrast to the above, researchers in the computational setting, such as [7,9,21,25,32], have explored stronger adversary models, whereby 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 against state-reveal attacks. There are, however, drawbacks to these computational models. These models 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.

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 co-processors for the secure storage of long-term secrets with the computation of intermediate results in less-secure main memory for efficiency reasons.

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.

Our framework directly lends itself to protocol analysis. As an example, we extend Scyther [14], 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. Furthermore, our tool finds previously unreported attacks, including a novel attack on HMQV. This shows that

symbolic methods can be effectively extended for analyses that previously were possible only using a manual computational analysis.

Organization. We present our framework in Section 2 and show several applications in Section 3. We discuss related work in Section 4 and conclude in Section 5.

2 Compromising Adversary Model

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

Notational preliminaries.

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

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

2.1 Terms and events

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

These identifiers single out a distinguished "point of view" thread of an arbitrary agent and an adversary thread, respectively.

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

Definition 1. *Terms*

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

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

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

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

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

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

Definition 2. *Events*

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

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 $tid_{\mathcal{A}}$.

2.2 Protocols and threads

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

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

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

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

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

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

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

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

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

2.3 Execution model

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

Definition 4 ($\mathcal{T}est\mathcal{S}ub_P$). *Given a protocol P, we define the set of* test substitutions *$\mathcal{T}est\mathcal{S}ub_P$ as the set of ground substitutions σ_{Test} such that*

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

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

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

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

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

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

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

Fig. 1. Execution-model rules

$dom(\sigma_{Test}) = dom(P) \cup \{v\sharp Test \mid v \in Var\}$ *and* $\forall r \in dom(P).\ \sigma_{Test}(r) \in Agent$.

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

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

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

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

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

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

The last three rules support our adversary rules, given shortly. The $\mathsf{generate}$ rule marks the fresh terms that have been generated, the state rule marks the current local state, and the $\mathsf{sessionkeys}$ 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', and tr, we define $\langle\rangle \downarrow tid = \langle\rangle$ and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 2. Adversary-compromise rules

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

2.4 Adversary-compromise rules

We define the adversary-compromise rules in Figure 2. They factor the security definitions from the cryptographic protocol literature along three dimensions of adversarial compromise: *which* kind of data is compromised, *whose* data it is, and *when* the compromise occurs. Not all combinations of capabilities have been used for analyzing protocols. Some combinations are not covered because of impossibility results (e. g. [23]), whereas other combinations appear to have been previously 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 $\mathsf{LKR_{others}}$ rule formalizes the adversary capability used in the symbolic analysis of security protocols since Lowe's attack on Needham-Schroeder [28]: the adversary can learn the long-term keys of any agent a that is not an intended partner of the test thread. Hence, if the test thread is performed by Alice, communicating with Bob, the adversary can learn, e. g., Charlie's long-term key.

The $\mathsf{LKR}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term key of the agent executing the test thread (also called the *actor*). The intuition is that a protocol may still function as long as the long-term keys of the other partners are not revealed. This rule allows the adversary to perform so-called Key Compromise Impersonation attacks [21]. The rule's second premise is required because our model allows agents to communicate with themselves.

The $\mathsf{LKR}_{\mathsf{after}}$ and $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rules restrict when the compromise may occur. In particular, they allow the compromise of long-term keys only after the test thread has finished, captured by the premise $th(Test) = \langle\rangle$. This is the sole premise of $\mathsf{LKR}_{\mathsf{after}}$. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{after}}$, it is said to satisfy perfect forward secrecy (PFS) [17,30]. $\mathsf{LKR}_{\mathsf{aftercorrect}}$ has the additional premise that a finished partner thread must exist for the test thread. This condition stems from [23] and excludes the adversary from both inserting fake messages during protocol execution and learning the key of the involved agents later. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{aftercorrect}}$, it is said to satisfy weak perfect forward secrecy (wPFS). This property is motivated by a class of protocols given in [23] whose members fail to satisfy PFS, although some satisfy this weaker property.

The left-hand side of Figure 3 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 of the long-term key reveal rules.

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 the right-hand side of Figure 3, 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 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 compu-

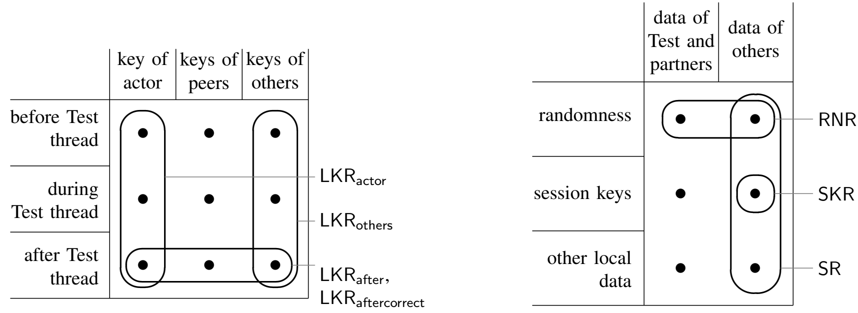

Fig. 3. Relating long-term and short-term data reveal rules

tations. The notion that the adversary may learn the randomness used in a protocol stems from [25]. Considering adversaries that can reveal session keys, e. g., by cryptanalysis, is found in many works, such as [4]. An adversary capable of revealing an agent's local state was described in [9].

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

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

The rules SKR and SR allow for the compromise of session keys and the contents of a thread's local state. Their premise is that the compromised thread is not a partner thread. In contrast, the premise of the RNR rule allows for the compromise of all threads, including the partner threads. This rule stems from [25], where it is shown that it is possible to construct protocols that are correct in the presence of an adversary capable of RNR.

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

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

Security property	Decomposition	
	Basic property	Adversary model
Perfect Forward Secrecy [17,30]	Secrecy	$\{\mathsf{LKR}_{\mathsf{after}}\}$
Weak Perfect Forward Secrecy [23]	Secrecy	$\{\mathsf{LKR}_{\mathsf{aftercorrect}}\}$
Known-Key Security [30]	Secrecy (of session key)	$\{\mathsf{SKR}\}$
Key Compromise Impersonation [21]	Authentication	$\{\mathsf{LKR}_{\mathsf{actor}}\}$

Table 1. Decomposing security properties

2.5 Transition relation and security properties

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

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

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

We provide a symbolic definition of session-key secrecy which, when combined with different adversary models, gives rise to different notions of secrecy found in the literature. Other security properties, such as secrecy of general terms, symbolic indistinguishability, or different variants of authentication, can be defined analogously.

Definition 8 (Session-key secrecy). *Let P be a protocol and Adv an adversary model. We say that P satisfies* session-key secrecy *with respect to Adv if and only if*

$$\begin{aligned} &\forall R_{Test} \in dom(P).\, \forall (tr, IK, th, \sigma_{Test}) \in \mathrm{RS}(P, Adv, R_{Test}). \\ &\qquad th(Test) = \langle\rangle \Rightarrow \forall k \in union((tr \downarrow Test) \downharpoonright \mathsf{sessionkeys}).\, IK \nvdash k\,. \end{aligned}$$

Many definitions of security properties, such as perfect forward secrecy, also contain elements of adversary capabilities. In our framework, such properties are cleanly separated into a basic security property (e. g. secrecy or authentication) and an adversary model. In Table 1, we decompose different security properties from the literature this way.

	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 [28]
Adv_{CA}		✓						Key Compromise Impersonation [21]
Adv_{AFC}				✓				Weak Perfect Forward Secrecy [23]
Adv_{AF}			✓	✓				Perfect Forward Secrecy [17,30]
Adv_{BR}	✓				✓			BR93 [4], BR95 [5]
Adv_{CKw}	✓	✓		✓	✓	✓		CK2001-wPFS [23]
Adv_{CK}	✓		✓	✓	✓	✓		CK2001 [9]
$Adv_{\mathrm{eCK\text{-}1}}$	✓				✓		✓	eCK [25]
$Adv_{\mathrm{eCK\text{-}2}}$	✓	✓		✓	✓			

Table 2. Mapping adversary-compromise models from the literature

Our way of modeling security properties provides a uniform view and also allows for direct generalizations of security properties. This leads to new, practically relevant combinations of adversary models and basic security properties. For example, for a hardware security module restricted to protecting long-term keys, relevant properties could be secrecy or agreement, resilient against state-reveal. Further properties arise by considering the combination of our adversary models with other basic properties like non-repudiation, plausible deniability, anonymity, or resistance to denial-of-service attacks.

3 Applications and case studies

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 [3,5,9,23,25]. We focus on the adversary capabilities only, abstracting from subtle differences between the computational models. For example, the model in [9] 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 [8,10,11,29] for further details on the differences between computational models.

Table 2 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 [9] and similarly for other models. We use a check (✓) to denote that the rule labeling the column is included in the adversary model named in the row.

Tool support. We extended the symbolic security-protocol verification tool Scyther [14,15] with our adversary rules from Figure 2. We used this

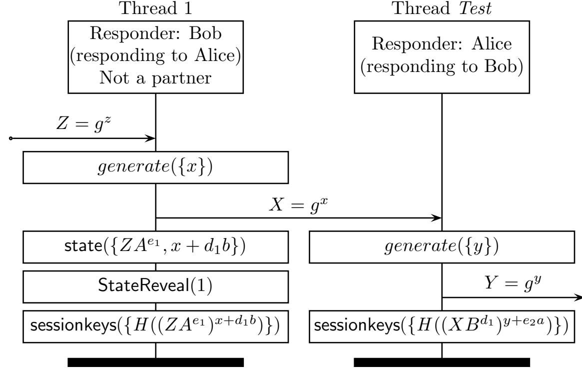

Fig. 4. SR attack on HMQV

tool to automatically analyze a set of protocols, described below. The tool, all protocol models, and test scripts can be downloaded from [13].

Attack example. The MQV protocol family [24,27,33] is a class of authenticated key-exchange protocols designed to provide strong security guarantees. The HMQV protocol was proven secure with respect to the adversary model in [23]. This model is the analog of our Adv_{CKw} model, where the local state of HMQV is defined as the random values generated for the Diffie-Hellman key-exchange. Surprisingly, our tool finds that the HMQV protocol is, depending on the definition of the state, insecure in adversary models that contain SR rules, such as the CK model [9].

Below we describe a new attack, which shows that MQV and HMQV are insecure in, e. g., 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 [23] because both the proof and the extended analysis given there assume that the local state contains only the ephemeral keys (the temporary private keys).

Using notation from [23], we show the attack in Figure 4, where $d_1 = \bar{H}(X, \text{Bob})$, $e_1 = \bar{H}(Z, \text{Alice})$, and $e_2 = \bar{H}(Y, \text{Alice})$. The attack starts with Bob receiving a message g^z apparently coming from Alice. This message may have been sent by an agent or have been generated by the adversary. Next, Bob generates x and sends $X = g^x$, which is intercepted by the adversary. Thread 1 is not a partner of the test thread because its history does not match the test thread's. Hence the adversary can compromise 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.

	Adv_{EXT}	Adv_{INT}	Adv_{CA}	Adv_{AFC}	Adv_{AF}	Adv_{BR}	Adv_{CKw}	Adv_{CK}	$Adv_{\text{eCK-1}}$	$Adv_{\text{eCK-2}}$
DH-ISO [18,25]									×(B)	
DH-ISO-C [18]								×	×	
DHKE-1 [18]							×(A)		×	×
HMQV-C [24]							×	×		
HMQV [24]					×(B)		×(A)	×		
NAXOS [16,25]					×(B)		×	×		
KEA+ [26]				×(A)	×(B)		×	×	×	×
NSL [28]			×(A)	×(B)	×(B)		×	×	×	×
BKE [12]			×(A)	×(B)	×(B)		×	×	×	×
Yahalom-Paulson [31]			×(A)	×(B)	×(B)	×(A)	×	×	×	×
NS [28]		×	×(A)	×(B)	×(B)	×	×	×	×	×

Table 3. Attacks found: (A) new, and (B) rediscovered automatically

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, which prevents this attack. Conversely, if (H)MQV will be 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.

Further case studies. In Table 3, we summarize the attacks found using our tool on protocols with respect to the adversary models from Table 2. A cross (×) in the table denotes that an attack was found. Attacks marked (A) were previously unreported. Attacks marked (B) were previously found by manual computational analysis. The set of protocols includes both formally analyzed protocols (NS, NSL, BKE, Yahalom) as well as protocols recently proposed in computational settings (HMQV, DH-ISO, Naxos, KEA+). Our tool rediscovers the attacks described in the literature, e. g., that DH-ISO is insecure in the eCK model [25] and that the implicitly authenticated two-message protocols KEA+, Naxos, and HMQV do not satisfy perfect forward secrecy. Additionally our tool finds new attacks on KEA+ and HMQV. The time needed for finding the attacks in the table ranged from less than a second to three minutes for each attack.

4 Related work

Related work in computational analysis. Most research on adversary compromise has been performed in the context of key-exchange protocols in the computational setting, e. g. Canetti and Krawczyk [9,23], Shoup [32], Bellare et al. [3–5], Katz and Yung [22], LaMacchia et al. [25], and Bresson

and Manulis [7]. 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 by, e. g., Choo et al. [10,11], Bresson et al. [8], LaMacchia et al. [25], and Menezes and Ustaoglu [29].

The CryptoVerif tool of Blanchet [6] is a mechanized tool for computational analysis. Its adversary model covers Adv_{INT}, corresponding to static corruption, i. e., the classical Dolev-Yao adversary.

Related work in symbolic analysis. In the symbolic setting, Guttman [20] 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.

In [1], Abadi, Blanchet, and Fournet 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. Paulson used his inductive approach to reason about the compromise of short-term data [31]. 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 [18,19] link a symbolic adversary model that includes dynamic corruptions to an adversary model used in the computational analysis of key-agreement protocols. They describe in [19] 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.

In [2], we have built upon the work presented here and introduce the concept of a *protocol-security hierarchy*, which classifies the relative strength of protocols against different forms of compromise.

5 Conclusions

We have provided the first symbolic framework capable of systematically modeling a family of adversaries endowed with different compromise capabilities. Our adversary capabilities generalize those from the computa-

tional setting and combine them with a symbolic model. In doing so, we unify and generalize a wide range of adversary models from both settings.

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, e. g., on DH-ISO, it also found new attacks, e. g., on HMQV and 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.

References

1. M. Abadi, B. Blanchet, and C. Fournet. Just Fast Keying in the Pi calculus. *ACM Transactions on Information and System Security (TISSEC)*, 10(3):1–59, July 2007.
2. D. Basin and C. Cremers. Degrees of security: Protocol guarantees in the face of compromising adversaries. In *19th EACSL Annual Conferences on Computer Science Logic (CSL 2010)*, LNCS Advanced Research in Computing and Software Science, 2010. To appear.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, LNCS, pages 139–155. Springer, 2000.
4. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249. Springer, 1993.
5. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proc. STOC '95*, pages 57–66. ACM, 1995.
6. B. Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, May 2006.
7. E. Bresson and M. Manulis. Securing group key exchange against strong corruptions. In *ASIACCS*, pages 249–260. ACM, 2008.
8. E. Bresson, M. Manulis, and J. Schwenk. On security models and compilers for group key exchange protocols. In *IWSEC*, volume 4752 of *LNCS*, pages 292–307. Springer, 2007.
9. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.
10. K.-K. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment proofs. In *ASIACRYPT*, volume 3788 of *LNCS*, pages 624–643. Springer, 2005.
11. K.-K. Choo, C. Boyd, Y. Hitchcock, and G. Maitland. On session identifiers in provably secure protocols. In *SCN'05*, volume 3352 of *LNCS*, pages 351–366. Springer, 2005.

12. J. Clark and J. Jacob. A survey of authentication protocol literature, 1997. `http://citeseer.ist.psu.edu/clark97survey.html`.
13. C. Cremers. Scyther tool with compromising adversaries extension. Includes protocol description files and test scripts. Available online at `http://people.inf.ethz.ch/cremersc/scyther/`.
14. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. CAV*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.
15. C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proc. of the 15th ACM conference on Computer and communications security*, pages 119–128. ACM, 2008.
16. C. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In *ACNS*, volume 5536 of *LNCS*, pages 20–33. Springer, June 2009.
17. C. Günther. An identity-based key-exchange protocol. In *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, 1990.
18. P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proc. FMSE 2005*, pages 23–32. ACM, 2005.
19. P. Gupta and V. Shmatikov. Key confirmation and adaptive corruptions in the protocol security logic. In *FCS-ARSPA'06*, pages 113–142, 2006.
20. J. D. Guttman. Key compromise, strand spaces, and the authentication tests. *ENTCS*, 45:1–21, 2001. Invited lecture, 17th Annual Conference on Mathematical Foundations of Programming Semantics.
21. M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT 1996*, volume 1163 of *LNCS*, pages 36–49, 1996.
22. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO*, volume 2729 of *LNCS*, pages 110–125. Springer, 2003.
23. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176, 2005. `http://eprint.iacr.org/`, retrieved on April 14, 2009.
24. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO*, volume 3621 of *LNCS*, pages 546–566. Springer, 2005.
25. B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, volume 4784 of *LNCS*, pages 1–16. Springer, 2007.
26. K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *LNCS*, pages 378–394, 2006.
27. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.
28. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.
29. A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In *ACISP 2008*, volume 5107 of *LNCS*, pages 53–68, 2008.
30. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996.
31. L. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
32. V. Shoup. On formal models for secure key exchange (version 4), Nov. 1999. revision of IBM Research Report RZ 3120 (April 1999).
33. B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Des. Codes Cryptography*, 46(3):329–342, 2008.