

David Basin, Cas Cremers, Catherine Meadows

# Model Checking and Security Protocols

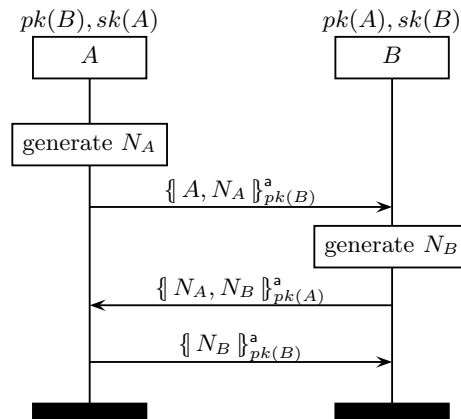
April 5, 2011

Springer

## 24.1 Introduction

Cryptographic protocols are communication protocols that use cryptography to achieve security goals such as secrecy, authentication, and agreement in the presence of adversaries. Examples of well-known cryptographic protocols are SSL/TLS [DR06], IKEV2 [KHNE10], and Kerberos [NHR05], which can be used, respectively, to secure web-based traffic, setup virtual private networks, and perform authentication in distributed environments. In order to ensure that such protocols always achieve their goals, they are designed under the assumption that the network is completely controlled by an adversary (also called the intruder or attacker). This means that the adversary can intercept, redirect, and alter data, have access to any operation that is available to legitimate agents, and even control one or more legitimate agents and thus access their keys. Given the hostility of the intended environment, it is not surprising that cryptographic protocols are difficult to design and are subject to subtle flaws, even when the cryptographic primitives used are themselves secure.

To give an idea of what can go wrong, consider Lowe's often-cited attack [Low96] on the Needham-Schroeder public key protocol [NS78]. In this protocol, two agents  $A$  and  $B$  agree on a pair of shared secrets,  $N_A$  and  $N_B$ , where  $N_A$  is a random number (or nonce) generated by  $A$  and  $N_B$  is a nonce generated by  $B$ . The protocol relies on public key encryption: anyone can send a message to an agent  $A$  using  $A$ 's public key, but only  $A$  can decrypt it. We write  $\{M\}_{pk(A)}^a$  to denote the (asymmetric) encryption of the message  $M$  with  $A$ 's public key. A message sequence chart describing the protocol is shown in Figure 24.1.

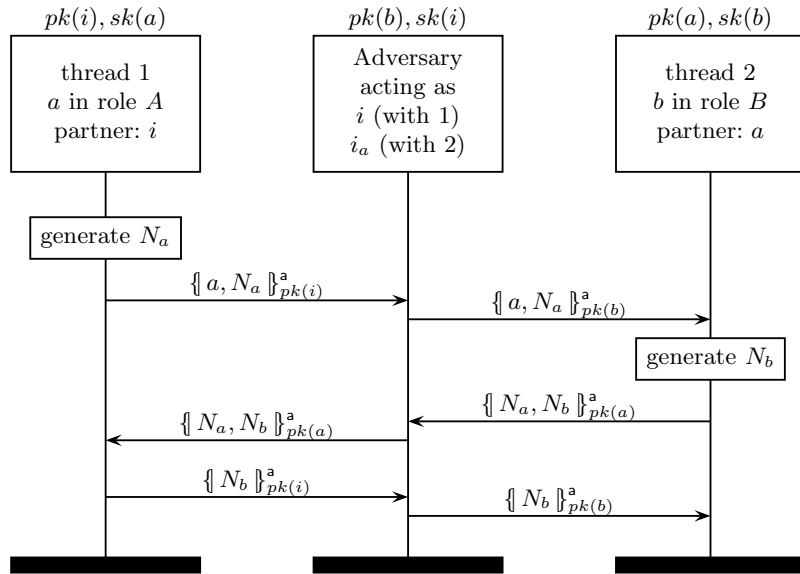


**Fig. 24.1.** Needham-Schroeder protocol (NS)

Let us go through the protocol steps and their intuitive rationale.

1.  $A \rightarrow B : \{ \{ A, N_A \}_{pk(B)}^a \}$   
*B* receives *A*'s message and decrypts it. At this point, *B* assumes that *A* has indeed sent the message recently and will try to confirm his assumption in the next two steps. *B* generates a nonce  $N_B$ .
2.  $B \rightarrow A : \{ \{ N_A, N_B \}_{pk(A)}^a \}$   
 When *A* receives this message, she decrypts it. Because the message contains the nonce  $N_A$ , which *A* generated recently and sent encrypted for *B* only, *A* concludes that the message was indeed sent recently by *B*.
3.  $A \rightarrow B : \{ \{ N_B \}_{pk(B)}^a \}$   
 The last message is sent so that *B* can verify his assumptions. He reasons that the message is recent, since  $N_B$  is recent. Moreover, only *A* and *B* know  $N_B$ , and *B* did not send the message. So the message must come from *A*. Finally, *A* will have received  $N_B$  as part of *B*'s message that also contained  $N_A$ , so she would not have responded unless  $N_A$  was also recently sent by *A*.

Despite this convincing-sounding informal argument as to why the protocol should be correct, it suffers from the following attack. Here *i* denotes the adversary and  $i_a$  denotes the adversary impersonating agent *a*. The corresponding message sequence chart is shown in Figure 24.2.



**Fig. 24.2.** Lowe’s man-in-the-middle attack on the Needham-Schroeder protocol

The attack proceeds in the following way.

1.  $a \rightarrow i : \{ \{ a, N_a \}_{pk(i)}^a \}$

- The agent  $a$  initiates the protocol with  $i$  and generates a fresh nonce  $N_a$ .
2.  $i_a \rightarrow b : \{ \{ a, N_a \}_{pk(b)}^a \}$   
The adversary  $i$  uses  $a$ 's nonce to initiate an instance of the protocol with  $b$ , impersonating  $a$ .
  3.  $b \rightarrow a : \{ \{ N_a, N_b \}_{pk(a)}^a \}$   
 $b$  responds to  $a$  correctly, generating his own nonce  $N_b$  in the process. Since  $a$  sees the nonce she sent to  $i$ , she assumes the message is from  $i$ .
  4.  $a \rightarrow i : \{ \{ N_b \}_{pk(i)}^a \}$   
 $a$  responds to  $i$ , following the rules of the protocol.
  5.  $i_a \rightarrow b : \{ \{ N_b \}_{pk(b)}^a \}$   
 $i$  re-encrypts  $N_b$  under  $b$ 's public key and sends the result to  $b$ . Since  $b$  is expecting this response from  $a$ , he concludes that he shares  $N_a$  and  $N_b$  with  $a$  and  $a$  alone.

Since  $b$  thinks he is communicating with  $a$ , and neither  $a$  nor  $b$  deviate from the protocol and their long-term keys are not known to the adversary,  $b$  expects that  $N_a$  and  $N_b$  are only known to  $a$  and himself. Clearly,  $b$ 's assumption is violated by the attack.

There are two things to notice about this attack. First, it is nonintuitive: if one looks closely, one sees that the security argument relies on the assumption that agents do not reveal secrets, and that this assumption is violated by  $i$  when forwarding  $a$ 's nonce to  $b$ . Indeed, Needham and Schroeder explicitly make this assumption in their paper. However, relaxing the assumption has surprising consequences, as this attack makes clear. Second, the attack does not depend on any flaws in the cryptographic primitives used and requires only a very simple adversary model. The only operations on data that the adversary employs are concatenation and splitting of messages, and encryption and decryption.

In general, we would like to establish the correctness of protocols with respect to even more powerful, but realistic, adversaries. We typically also give the adversary the ability to compute a limited set of functions on data. Namely, the adversary can read, redirect, and delete any message sent along a network, impersonate any agent, and create new messages by applying functions available to him to data he has already seen. This results in a model that can capture a large class of potential protocol problems. Such a model, which is both simple and expressive enough to capture a large class of nonintuitive security flaws, lends itself well to model checking. Hence it is not surprising that the analysis of protocols with respect to such models has been a major application of model checking in security. Basically, one uses the model checker to find all possible ways an adversary can interact with a protocol by using arbitrary combinations of interception, redirection, and the other basic operations available to him. The state space thus generated is of course infinite, but as we will see, the problem is decidable under certain limiting but reasonable restrictions, and heuristics and abstraction techniques have been developed to reason about the case in which the restrictions are not assumed.

Model checking cryptographic protocols is not just of intellectual interest. It can do much to streamline the development and adoption of security standards. New cryptographic protocols are constantly being invented as new communication paradigms are introduced. Since a protocol must be widely adopted before it is useful, new protocols are usually introduced through a standardization process. This process can be drawn-out and argumentative, and standards can be difficult to modify once they are in place. Formal analysis can help speed up standardization by finding problems early and giving evidence of security if no flaws are found. Moreover, it can also help prevent flawed protocols from being standardized. Finally, model checking has the advantage that the counter-examples it finds depict actual attacks on the protocol. This gives insight into a protocol's vulnerabilities, and how they can be fixed.

We proceed as follows. In Section 24.2 we give a brief historical overview of the field. Then, in Section 24.3, we describe a basic model for security protocols and their properties. We give an overview of several model checking algorithms in Section 24.4 and describe representative systems based on these algorithms in Section 24.5. We discuss current and future research questions in Section 24.6 and we draw conclusions in Section 24.7.

## 24.2 History

In this section, we give a brief history of model checking cryptographic protocols. In order to keep the historical record clear while maximizing access to technical results, we provide, when necessary, two references for each result we cite. The first citation is for the historical record, giving the first appearance of the result. Since these often appear in technical reports or conference papers that are inaccessible or incomplete, we also provide a reference to the result as it finally appeared, usually in a journal paper.

The first application of a model-checking-like approach to cryptographic protocol analysis was that of Dolev and Yao [DY81, DY83], and shortly later, Dolev, Even, and Karp [DEK82], just after the invention of public key cryptography. They introduced the paradigm now known as the Dolev-Yao model. In this model, a protocol is modeled as a machine consisting of an arbitrary number of honest agents executing the protocol, in which all messages sent are intercepted by the adversary (even if he does no more than forward them), all messages received are sent by the adversary, and any message processing done by the adversary is done using an arbitrary combination of a finite set of operations. The model also formalizes an abstraction of cryptography where messages are represented by terms rather than bit-strings and cryptography is "perfect" in the sense that cryptographic operators do not leak information, e. g., the only way for the adversary to decrypt an encrypted message is to have the decryption key.

The Dolev-Yao model is at the basis of all applications of model-checking to cryptographic protocols, although today’s protocol analysis tools take a very different approach than that taken originally. Dolev and Yao were interested in low-complexity algorithms for proving secrecy, and gave several polynomial-time algorithms for a class of protocols they characterized as “ping-pong” protocols. However, it turned out that the problem quickly became undecidable when the algebraic properties of the cryptographic algorithms were represented more faithfully [EGS85], and interest in the problem petered out.

A few years later, researchers started tackling the problem from another point of view, developing tools that would exhaustively search the problem space or some portion of it. The first tool to take this approach was Millen’s Interrogator [MCF87], followed by the Longley-Rigby search tool [LR92] and the NRL Protocol Analyzer (NPA) [Mea96]. These can be thought of as proto-model checkers. Indeed the NPA offered many features of a model-checker, including an automated means of proving that exhaustive search of a finite space implied exhaustive search of the infinite state space, and later, a temporal logic language, NPATRL [SM96], for describing protocol security properties.

Interest in model-checking cryptographic protocols really took off with Lowe’s use of the FDR model-checker to analyze the Needham-Schroeder public key protocol [Low96], described above. The fact that he could demonstrate a problem that had gone unnoticed for seventeen years alerted people to the power of model checking. Other researchers began applying their own model checkers to the problem, most notably the use of the Murphi model checker by Mitchell et al. [MMS97] to analyze variations on the TLS protocol. From there it was a short step to the development of special purpose model checkers, such as Clarke et al.’s Brutus [CJM00].

Parallel to this was research on the complexity of model checking. The key feature turned out to be the number of sessions involved, where a *session* refers to a single (potentially partial) execution of the protocol. As we see from the attack on the Needham-Schroeder protocol, attacks often interleave different sessions together. Indeed it is possible to create attacks that interleave an arbitrarily large number of sessions. In [DLMS99, DLM04], Durgin et al. show that, given a model similar to the one described in this chapter, the secrecy problem (that is, the problem of deciding whether or not the adversary learns a particular term) is undecidable if the number of sessions and nonces is unbounded. Rusinowitch and Turuani [RT01] later showed that the secrecy problem is NP complete if the number of sessions is bounded. Moreover, their procedure applies more generally to arbitrary state properties, and thus can also be applied to other reachability properties such as authentication.

The decidability of security for the bounded session case led to the development of *bounded session* model checkers, in which the user specifies the number of sessions the model checker should search. Bounded session model checkers are of practical significance, since most attacks on realistic protocols require only a few sessions. Indeed, an attack that requires interleaving many sessions would not be practical to implement. Bounded session model

checkers include Shmatikov and Millen’s constraint based tool [MS01], the Constraint-Logic-based Attack Searcher (CL-Atse) [Tur06], the On-the-Fly Model Checker (OFMC) [BMV05b], and the SAT-based Model Checker SAT-MC [AC04]. The same period saw the development of unbounded session model checkers relying on abstraction (such as ProVerif [Bla01]) and heuristics (such as Maude-NPA [EMM07] and Athena [Son99], the latter of which formed the basis for the current tool Scyther [Cre08]). We present these tools in more detail in Section 24.5.

In recent years, as the field matures, researchers are increasingly concentrating on making the tools available for others to use, and are applying them to practical problems such as the verification of standards. Some of the tools that have seen the widest use are the AVISPA tool suite [ABB<sup>+</sup>05, Vig06], which is a set of model checkers (the above-mentioned CL-Atse, SAT-MC, and OFMC), with a common front end, and the above-mentioned ProVerif tool. Many tools have been used in the analysis of standards, sometimes detecting problems that would have gone unnoticed otherwise. For example, the NPA was used in the verification of two IETF protocols: the Internet Key Exchange Protocol [Mea99] and the Group Domain of Interpretation (GDOI) protocol [MSC04]. In the case of GDOI, the tool was instrumental in catching some vulnerabilities early on that were straightforward to fix in the design stage but could have led to problems if they had not been caught in time. The AVISPA tools have been applied to a suite of protocol standards. ProVerif has been used in the production of formally verified implementations of TLS [BFCZ08] and the smart card protocol InfoCard [BFGS08]. Scyther has been used in the analysis of the MQV family of protocols [BC10b], and has found attacks against members of that family when the adversary is able to compromise parts of the local state of the agents. Furthermore, Scyther has been used for the analysis of the entity authentication protocols in the ISO/IEC 9798 standard [BC11].

Ultimately, we would expect model-checking to become a standard tool for cryptographic protocol design, as it has become in hardware design. This has not quite happened yet. Although model checking has proved useful in the analysis of standards, it has not yet become part of the standards designer’s basic toolbox. However, designers of new protocols are starting to accompany them with formal analyses. The field is still actively growing and changing, and we would not be surprised to see model checking being more widely adopted in the near future.

Although the basic decidability results and model-checking algorithms in what is commonly accepted as the standard Dolev-Yao model are well understood, there is still much to be learned, and there is currently active research going on in a number of areas. These include making the Dolev-Yao model more precise and expressive, e. g., by including equational properties of cryptographic algorithms such as the associativity-commutativity of exponentiation, and incorporating cryptographic theories of correctness, reasoning about non-trace properties such as non-repudiation, non-interference, and indistin-

guishability, extending soundness results down to the code level, and handling probabilistic behavior. These topics will be discussed in further detail in Section 24.6.

### 24.3 Formal model

Each of the tools mentioned in the previous section is based on a formal model of cryptographic protocols and the actions available to the adversary. Although these models differ in their details, they have a number of important features in common, since they are all based on the Dolev-Yao model presented in Section 24.2. In this section we present a basic symbolic model (based on [BC10b]), for formalizing and reasoning about security protocols, which captures the main features shared by these different models. We will use this model as a reference point when we describe the different tools and approaches later in this chapter.

As the model is symbolic, messages are represented by terms in a term algebra. Protocols themselves are described by a set of roles, each role with an associated script that describes the sequence of events taken by the agents executing the role. The protocols are given an operational semantics where agents may play in multiple roles giving rise to arbitrarily many role instances (also called threads). This gives rise to a semantics formalized by an infinite-state transition system, with an associated notion of trace.

For expository purposes, we present a simple model that handles only a restricted class of security protocols. We describe these restrictions along with extensions and other design options. We also explain how basic security properties can be formalized within this model.

#### 24.3.1 Notational preliminaries

Let  $f$  be a function. We write  $dom(f)$  and  $ran(f)$  to denote  $f$ 's domain and range, respectively. We write  $f[b \leftarrow a]$  to denote  $f$ 's update, which is the function  $f'$  where  $f'(x) = b$  when  $x = a$  and  $f'(x) = f(x)$  otherwise. We write  $f : X \rightarrow Y$  to denote a partial function mapping some elements from  $X$  to elements from  $Y$ .

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

We use standard notions (see e. g., [BN98]) for manipulating terms. Let  $Sub$  denote the set of substitutions of terms for variables (we will define these syntactic categories shortly). We write  $[t_0, \dots, t_n/x_0, \dots, x_n] \in Sub$  to denote

the substitution of  $t_i$  for  $x_i$ , for  $0 \leq i \leq n$ . We extend the functions  $dom$  and  $ran$  to substitutions. We write  $\sigma \cup \sigma'$  to denote the union of two substitutions, which is defined when  $dom(\sigma) \cap dom(\sigma') = \emptyset$ . We write  $\sigma(t)$  for the application of the substitution  $\sigma$  to  $t$ .

For  $R$  a binary relation, we write  $R^{-1}$  to denote the *inverse* of  $R$ , i. e.,  $R^{-1} = \{(y, x) \mid (x, y) \in R\}$ . We overload the  $(\cdot)^*$  notation: when  $R$  is a binary relation,  $R^*$  denotes the transitive closure of  $R$ .

### 24.3.2 Terms and events

We assume given the pairwise-disjoint infinite sets  $Agent$ ,  $Role$ ,  $Fresh$ ,  $Var$ ,  $Func$ ,  $TID$ , and  $AdvConst$  of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, function names, thread identifiers, and adversary-generated constants.

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

**Definition 1.** *Basic terms*

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

**Definition 2.** *Terms*

$$\begin{aligned} Term ::= & BasicTerm \mid (Term, Term) \\ & \mid pk(Term) \mid sk(Term) \mid k(Term, Term) \\ & \mid \{\!\{ Term \}\!\}_a^{Term} \mid \{\!\{ Term \}\!\}_s^{Term} \mid Func(Term^*) \end{aligned}$$

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

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

As noted in the introduction, one of the distinguishing features of model checking security protocols is that they operate in an environment controlled by an adversary. To formalize the powers of a Dolev-Yao-style adversary, 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  $t_0, \dots, t_n \in \text{Term}$  and let  $f \in \text{Func}$ . We define  $\vdash$  as the smallest relation satisfying:

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

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

An agent can engage in the following events.

**Definition 3.** *Events*

$$\text{Event} ::= \text{create}(\text{Role}, \text{Sub}) \mid \text{send}(\text{Term}) \mid \text{recv}(\text{Term})$$

Note that the send and receive events do not include explicit sender or recipient fields. The messages sent or received can, of course, include subterms identifying the sender and the intended recipient, although this information is not a priori authentic. As is standard, the adversary receives all messages sent, independent of the intended recipient.

We will explain the interpretation of the events shortly. Here we note that the events are conventional and are given a standard interpretation in a setting with concurrently executing, communicating processes: starting a thread, sending a message, and receiving a message.

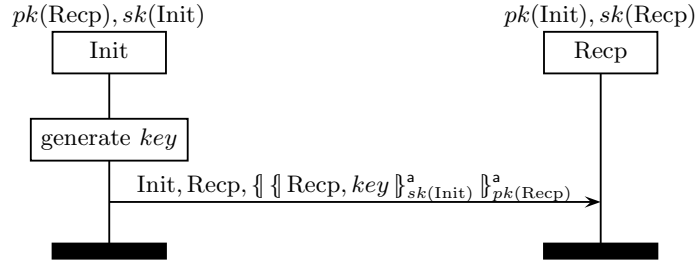
We extend the domain of substitutions over events and sequences of events in the standard way, i. e.,  $\sigma(\text{send}(m)) = \text{send}(\sigma(m))$ .

### 24.3.3 Protocols and threads

A protocol is a mapping from role names to event sequences, i. e.,  $\text{Protocol} : \text{Role} \rightarrow \text{Event}^*$ . We require that no thread identifiers occur as subterms of events in a protocol definition. The following is a very simple example of a protocol with two roles: an initiator and a recipient.

*Example 1 (Simple protocol).* Let  $\{\text{Init}, \text{Recp}\} \subseteq \text{Role}$ ,  $\text{key} \in \text{Fresh}$ , and  $x \in \text{Var}$ . Let  $P$  be the protocol defined as follows.

$$\begin{aligned}
P(\text{Init}) &= \langle \text{send}(\text{Init}, \text{Recp}, \{\!\!| \text{Recp}, \text{key} \}\!\!|_{sk(\text{Init})}^a \{\!\!|_{pk(\text{Recp})}^a) \rangle \\
P(\text{Recp}) &= \langle \text{recv}(\text{Init}, \text{Recp}, \{\!\!| \text{Recp}, x \}\!\!|_{sk(\text{Init})}^a \{\!\!|_{pk(\text{Recp})}^a) \rangle
\end{aligned}$$



**Fig. 24.3.** Simple protocol

The message sequence chart for this protocol is shown in Figure 24.3. Here, the initiator generates a key and sends it (together with the recipient's name) signed and encrypted, along with the initiator and recipient names. The recipient expects to receive a message of this form.

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

**Definition 4 (Localize).** *Let  $tid \in TID$ . Then*

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

We define a function  $thread : (Event^* \times TID \times Sub) \rightarrow Event^*$  that yields the sequence of agent events that may occur in a thread.

**Definition 5 (Thread).** *Let  $l$  be a sequence of events,  $tid \in TID$ , and let  $\sigma$  be a substitution. Then*

$$thread(l, tid, \sigma) = \sigma(localize(tid)(l)).$$

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

$$thread(P(Init), t_1, [A, B / Init, Recp]) = \langle \text{send}(A, B, \{ \{ B, key \# t_1 \}_{sk(A)}^a \}_{pk(B)}^a) \rangle.$$

#### 24.3.4 Initial adversary knowledge

We assume that the adversary initially knows all agent names and can generate an unbounded set of constants  $AdvConst$ , where  $AdvConst \subset Func$  and no

protocol description contains elements of  $AdvConst$ . The set  $AdvConst$  represents the set of fresh values that are generated by the adversary. The adversary additionally knows the long-term public keys of all agents. We also assume that the adversary has compromised the long-term private keys of some of the agents. We model this by partitioning the set  $Agent$  into the honest agent set  $Honest$  and the compromised agent set  $Compromised$ . Moreover, we include the long-term private keys of the compromised agents in the initial knowledge of the adversary.

The long-term secret keys of an agent  $a$  are defined as

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

We define the initial adversary knowledge  $IK_0$  as

$$IK_0 = Agent \cup AdvConst \cup \bigcup_{a \in Agent} pk(a) \cup \bigcup_{a \in Compromised} LongTermKeys(a).$$

### 24.3.5 Execution model

We define the set  $Trace$  as  $(TID \times Event)^*$ , which represents possible execution histories. Using this set, we define the set  $State$  of system states as  $Trace \times \mathcal{P}(Term) \times (TID \rightarrow Event^*)$ . The components of a state  $(tr, IK, th) \in State$  are (1) a trace  $tr$ , (2) the adversary knowledge  $IK$ , and (3) a partial function  $th$  mapping thread identifiers of initiated threads (executing or completed) to event traces. Note that in conventional model checking approaches, (1) would not be part of the state but would be defined over runs of the transition system. We include the trace as part of the state to facilitate defining the partner function later.<sup>1</sup>

The initial system state  $s_{init}$  is defined as

$$s_{init} = (\langle \rangle, IK_0, \emptyset),$$

where  $IK_0$  is the initial adversary knowledge as defined above.

The semantics of a protocol  $P \in Protocol$  is defined by a transition system whose transitions are given by the rules from Figure 24.4. Each rule describes how the execution of one of the three events causes a state transition. We describe each rule in turn.

**Execution-model rules.** The **create** rule starts a new instance of a protocol role  $R$  (a *thread*). In the premises,  $dom(P)$  refers to the role names,  $\sigma$  associates these with agents, and  $tid$  is a fresh thread identifier. In the state transition in the conclusion, the successor state's trace is extended, reflecting

<sup>1</sup> Actually there are a number of representation options here. For example,  $IK$  and  $th$  can be computed directly from  $tr$ . We explicitly include them in the state to improve the readability of our operational semantics.

$$\begin{array}{c}
\frac{R \in \text{dom}(P) \quad \sigma \in \text{Sub} \quad \text{dom}(\sigma) = \text{dom}(P) \quad \text{ran}(\sigma) \subseteq \text{Agent} \quad \text{tid} \notin \text{dom}(th)}{(tr, IK, th) \longrightarrow (tr \hat{\cdot} \langle \text{tid}, \text{create}(R, \rho) \rangle), IK, th[\text{thread}(P(R), \text{tid}, \rho) \leftarrow \text{tid}]} [\text{create}_P] \\
\\
\frac{th(\text{tid}) = \langle \text{send}(m) \rangle \hat{\cdot} l}{(tr, IK, th) \longrightarrow (tr \hat{\cdot} \langle \text{tid}, \text{send}(m) \rangle), IK \cup \{m\}, th[l \leftarrow \text{tid}]} [\text{send}] \\
\\
\frac{th(\text{tid}) = \langle \text{recv}(pt) \rangle \hat{\cdot} l \quad IK \vdash \sigma(pt) \quad \text{dom}(\sigma) = FV(pt)}{(tr, IK, th) \longrightarrow (tr \hat{\cdot} \langle \text{tid}, \text{recv}(\sigma(pt)) \rangle), IK, th[\sigma(l) \leftarrow \text{tid}]} [\text{recv}]
\end{array}$$

**Fig. 24.4.** Execution-model rules

that the thread identified by  $\text{tid}$  executed the create event, and the thread mapping (“thread pool”) is extended with the thread assigned to  $\text{tid}$ .

The send rule sends a message  $m$  to the network. The premise refers to a thread identified by  $\text{tid}$ , whose next event is to send the message  $m$ . In the conclusion, the trace is updated with this event, the adversary knowledge is updated with  $m$ , and thread for  $\text{tid}$  is updated.

Finally, the receive rule models an agent, running a thread, receiving a message from the network. The message must match the message pattern  $pt$  under a substitution  $\sigma$ , where  $pt$  is a term that may contain free variables. Note that by the second premise, the adversary must be able to infer  $\sigma(pt)$  from his current knowledge  $IK$ . One can see this as formalizing that the adversary controls the network and effectively determines who receives which messages. In our model, recipients accept all messages that match the message pattern  $pt$ , and block on any other messages. The resulting substitution  $\sigma$  is applied to the remaining protocol steps  $l$ .

**Definition 6 (Transition relation).** *Let  $P$  be a protocol. We define a transition relation  $\rightarrow_P$  from the execution-model rules from Figure 24.4. For states  $s$  and  $s'$ ,  $s \rightarrow_P s'$  iff there exists an execution-model rule with the premises  $Q_1(s), \dots, Q_n(s)$  and the conclusion  $s \rightarrow s'$  such that all of the premises hold.*

Given a protocol  $P$  and a set of states  $T$ , let  $\text{Post}_P$  and  $\text{Pre}_P$  denote the successors and predecessors of  $T$ , respectively, i. e.,

$$\begin{aligned}
\text{Post}_P(T) &= \{ s' \in \text{State} \mid \exists s \in T . s \rightarrow_P s' \} \\
\text{Pre}_P(T) &= \{ s \in \text{State} \mid \exists s' \in T . s \rightarrow_P s' \}.
\end{aligned}$$

Let  $f^n$  denote the  $n$ -fold composition of  $f$ , for  $n \geq 0$ , where  $f^0$  is the identity function.

**Definition 7 (Reachable states).** *Let  $P$  be a protocol. We define the set of reachable states of  $P$  as*

$$\text{Reachable}(P) = \bigcup_{n=0}^{\infty} \text{Post}_P^n(\{s_{\text{init}}\}).$$

We will see Pre and Post again when we define model checking algorithms in Section 24.4.

### 24.3.6 Property specification

We focus on basic security properties that can be expressed as reachability properties, i. e., properties of reachable states. Because the adversary knows the long-term private keys of the compromised agents, protocol sessions that involve compromised agents cannot guarantee the secrecy of data such as shared keys or exchanged terms. This is reflected in the definition of most security properties by considering only those threads that do not involve compromised agents.

We introduce an auxiliary predicate HT (for *honest thread*) that identifies completed threads that do not involve compromised agents.

**Definition 8 (HT).** Let  $s = (tr, IK, th)$  be a state,  $tid$  a thread identifier, and  $\sigma$  a substitution. We write  $HT(s, tid, \sigma)$  to denote

$$\exists R . (tid, \text{create}(R, \sigma)) \in tr \wedge th(tid) = \langle \rangle \wedge \text{ran}(\sigma) \cap \text{Compromised} = \emptyset .$$

For example, let  $s = (tr, IK, th)$  be the state reached after the attack trace represented in Figure 24.2. We have that both thread 1 and 2 are completed, i. e.  $th(1) = th(2) = \langle \rangle$ , and the trace  $tr$  contains  $(1, \text{create}(A, [a, i/A, B]))$  and  $(2, \text{create}(B, [a, b/A, B]))$ . Hence we have that  $HT(s, 2, [a, b/A, B])$  but there exists no  $\sigma$  such that  $HT(s, 1, \sigma)$ , because  $a$  in thread 1 starts a thread to communicate with the compromised agent  $i$ .

**Definition 9 (Secrecy).** Let  $t \in \text{Fresh}$ . We say that a state  $s = (tr, IK, th)$  satisfies secrecy of  $t$  if and only if

$$\forall tid, \sigma . HT(s, tid, \sigma) \Rightarrow \neg(IK \vdash (t\#tid)) .$$

We say that a protocol  $P$  ensures secrecy of  $t$  if and only all reachable states of  $P$  satisfy secrecy of  $t$ .

For example, the protocol in Figure 24.3 ensures secrecy of the initiator's key.

*Authentication* is an important property for many security protocols and numerous notions of authentication have been proposed in the literature. As a simple example, consider *weak aliveness*. This weak form of authentication guarantees only that if a non-compromised agent completes a thread of the protocol under the assumption that he is communicating with a non-compromised agent  $a$  executing role  $R$ , then it is indeed the case that  $a$  previously started a thread in role  $R$ .

**Definition 10 (Weak Aliveness).** Let  $R$  be a role. We say a state  $s = (tr, IK, th)$  satisfies weak aliveness of  $R$  if and only if

$$\forall tid', \sigma' . HT(s, tid', \sigma') \Rightarrow \exists tid, \sigma . (tid, \text{create}(R, \sigma)) \in tr \wedge \sigma'(R) = \sigma(R) .$$

We say that a protocol  $P$  ensures weak aliveness if and only if all reachable states of  $P$  satisfy weak aliveness of all roles  $R \in \text{dom}(P)$ .

Stronger forms of authentication (e. g., [Low97]) impose additional requirements on the state. For example, they require that the threads' assumptions on agents match, e. g., by requiring  $\sigma = \sigma'$ , or they place additional requirements on the exchanged messages or the instantiation of variables, or they require that messages are recent. Some of these properties require instrumenting the model with additional markers, such as labeling communications or introducing signal events to simplify expressing agreement on the contents of variables.

### 24.3.7 Alternatives

We have intentionally kept our formalism simple to highlight the main ideas. At each step along the way there are design options, reflecting the class of protocols and adversaries one intends to capture.

To begin with, we have formalized cryptographic messages using a free term algebra, where term equality is therefore just syntactic equality. Additional cryptographic operators can be added, but this requires formalizing how the adversary can construct and reason about terms built from them. In Section 24.6 we describe how this can be done for operators formalized by sets of equations.

Our protocol roles are specified by straight-line sequences of events, without control flow primitives such as branching or loops. This is sufficient to model many security protocols, provided we ignore error cases, for example, where a thread receives an unexpected message. Such cases are handled implicitly: no transition is enabled and hence the thread simply does not progress. In contrast, a richer execution model would be needed to model protocols that support multiple options and subprotocols, such as the Internet Key Exchange (IKE) [HC<sup>+</sup>98], or that require loops, such as the stream authentication protocol TESLA [PT02, HL04]. It is not difficult to add control flow primitives or alternatively to base the execution model on a process calculus or some transition-system formalism. We will give examples of tools whose input languages are based on such formalisms in Section 24.5.

The simplicity and power of the Dolev-Yao adversary model has made it extremely popular. However, for many real-world scenarios an adversary who has complete control of the network may be unrealistic, and therefore protocols that offer weaker security guarantees may be preferred for efficiency reasons. At the other end of the spectrum, assuming that the adversary can learn nothing about encrypted data unless it obtains a decryption key may be unrealistic in other scenarios. We will consider alternative adversary models in Section 24.6.

## 24.4 Model checking algorithms for protocols

### 24.4.1 Forward versus backward search

We consider security properties that can be expressed as reachability properties, i. e., as a set of states  $S$ . We say that a protocol  $P$  satisfies the property  $S$  if and only if

$$\text{Reachable}(P) \subseteq S. \quad (24.1)$$

Let  $\overline{S} = \text{State} \setminus S$  be the property's complement, representing possible attacks. Then the above formula can be rewritten as

$$\text{Reachable}(P) \cap \overline{S} = \emptyset. \quad (24.2)$$

Then, we can rewrite Formula 24.2 either as

$$\left( \bigcup_{n=0}^{\infty} \text{Post}_P^n(\{s_{\text{init}}\}) \right) \cap \overline{S} = \emptyset \quad (24.3)$$

or alternatively as

$$s_{\text{init}} \notin \bigcup_{n=0}^{\infty} \text{Pre}_P^n(\overline{S}). \quad (24.4)$$

Algorithms that iteratively compute a representation of (all, or some subset of)  $\text{Post}_P^n(\{s_{\text{init}}\})$ , as in Formula 24.3, are said to use *forward search*. If, for some  $n$ , an element is found that is also in  $\overline{S}$ , a counterexample can be constructed representing an attack. Alternatively, if there is an  $n$  where we reach a fixpoint, i.e.  $\text{Post}_P^n(\{s_{\text{init}}\}) = \text{Post}_P^{n+1}(\{s_{\text{init}}\})$ , and additionally  $\text{Post}_P^n(\{s_{\text{init}}\}) \cap \overline{S} = \emptyset$ , then the property holds of the protocol. A fixpoint will be reached, in general, only in finite-state models.

In contrast, *backward search* iteratively constructs  $\text{Pre}_P^n(\overline{S})$ , as in Formula 24.4. Similar observations hold as for the forward search, except that we check whether  $s_{\text{init}}$  occurs in the constructed set.

In the analysis of security protocols, the set of reachable states is infinite, as new threads can always be created. Hence the closure in forward search contains infinitely many states. Similarly, the closure in backward search contains infinitely many states, but for a different reason: for the properties we consider here,  $\overline{S}$  contains infinitely many states.

The main idea behind searching infinite sets of states is to use finite representations of the infinite sets. The selection criteria for such a finite representation include the complexity of computing  $\text{Pre}_P$  or  $\text{Post}_P$ , and the complexity of evaluating whether or not all elements of the represented set satisfy the property  $S$ .

When exploring infinite state spaces, it is often efficient to use as much information as possible about the states. In general, the negation of the security property  $\overline{S}$  provides more information about the states than the initial

state  $\{s_{\text{init}}\}$ . For example, the negation will specify that particular events must have occurred or that the adversary knows certain terms. As a result, backward search is often employed when exploring infinite sets of states.

A simpler case occurs if the number of reachable states is restricted to a finite set, for example, by limiting the number of threads or sessions that can be created. In this case, forward search for violations of secrecy or authentication properties can be trivially implemented: checking that a given state satisfies these properties can be done using either Definition 9 or 10. A bounded backward search starts from the finite set of attack states  $\bar{S}$ , from which  $\text{Pre}_P^n(\bar{S})$  can be computed. Depending on the property and the (finite) size of the set of states, the size of (the representation of)  $\bar{S}$  can be significant. In practice, forward search is commonly used to explore finite sets of states.

### 24.4.2 Bounded instances

The execution model presented in Section 24.3 gives rise to an infinite-state transition system. Infinitely many states arise in two distinct ways. First, the `create` rule may start unboundedly many new protocol threads. Second, under the `receive` rule, there are unboundedly many different messages that a thread could receive from the network. This is modeled by the rule's second premise, which formalizes that a thread can be updated with any message  $\sigma(pt)$  that is in the closure of the adversary knowledge.

The first source of infinity is a fundamental problem. As mentioned in Section 24.2, even relatively simple properties such as secrecy are undecidable for security protocols formalized using operational semantics similar to ours [DLM04]. If we restrict the number of sessions (or threads), then the problem becomes NP complete [RT01], provided messages are formulated as terms in a free term algebra. Note that in practice, when analyzing real-world protocols, it is usually only necessary to consider a small number of threads. If there is an attack on the protocol there is normally an attack where the number of threads is at most a small factor more than the number of roles, e.g., twice the number of roles, which allows for messages from one protocol session to be replayed in another session. For a class of protocols where attacks require arbitrarily many threads, see [Mil99].

The second source of infinity, an infinite space of messages, turns out not to be a problem. The NP completeness result of Rusinowitch and Turuani [RT01] establishes that if there is an attack then there is one where the size of the messages involved is polynomially bounded in the size of the protocol and the number of threads, provided messages are represented by directed acyclic graphs. As a result, assuming a finite number of threads, and hence fresh data, one can bound a priori the messages that must be considered in protocol analysis. We will see below how both bounds on the number of threads and messages have been used by different protocol analysis tools.

### 24.4.3 Representing states

The Equations 24.3 and 24.4 can be directly used for forwards or backwards model-checking after fixing a representation of states for which one can effectively compute successors or predecessors. There are different options here depending on whether states are explicitly or symbolically represented.

It is simple to turn the operational semantics given in Section 24.3 into an explicit state model checker. A state is just a triple, all of whose components can be finitely encoded. The problem in practice is efficiently representing large sets of states, i. e., reducing the impact of state-space explosion. An example of a model-checker for security protocols based on forward search using explicit state enumeration is Murphi [SD95]. This tool uses techniques inspired by explicit state model checkers like SPIN [Hol02], such as hash tables and hash compaction to improve its efficiency. Another example of an explicit state coding is implemented by Lowe’s Casper system [Low98], which encodes the operational semantics of the protocol and the adversary as a (finite state) CSP process and uses the FDR model checker to either identify attacks or verify the protocol for instances with a bounded number of threads and messages.

The second possibility is to encode sets of states using formulas as in symbolic model checking. Most current tools take this approach. Here terms, in particular messages, are represented by non-ground terms (cf. *message patterns* in Section 24.3.5) which contain variables. These variables may be instantiated during search. For example, under our operational semantics, this instantiation would occur when applying the rules using unification. In approaches based on rewriting, such as Maude-NPA, instantiation occurs during rewriting by narrowing.

When working with symbolic representations, unification is often combined with constraint solving. In the formal model we have given, the need for constraint solving arises from the second premise of the *recv* rule,  $IK \vdash \sigma(pt)$ . When applying rules of the operational semantics backwards (either in forward or backward search), this gives rise to a subgoal often called the *intruder deduction problem*. The simplest version of this problem is to determine whether  $IK \vdash m$ , for a ground message  $m$  using the rules formalizing the Dolev-Yao adversary given in Section 24.3.2. During symbolic reasoning, the non-ground problem arises: determining whether there exists a substitution  $\sigma$  such that  $\sigma(IK) \vdash \sigma(pt)$ . The ground problem can be tackled by considering a restricted class of normal form derivations [CLT04]. The non-ground problem is generally solved either by unification-based procedures or specialized constraint solvers such as those used in OFMC or CL-Atse.

An alternative symbolic approach is that of bounded model-checking. In the simplest case, the closure specified in Equation 24.3 is simply unrolled some bounded number of times  $k$ , thereby specifying the existence of an attack, given by  $k$  or fewer applications of rules from the operational semantics. If this finite unrolling is combined with a bound on the number of messages

that may appear (and the result of Rusinowitch and Turuani gives us an exponential bound), then the resulting formula can be encoded within propositional logic and SAT solvers can be used to search for attacks, as shown in Chapter 10. Different encodings and optimizations for using SAT-based model-checking for security protocols have been explored by Armando and Compagna and implemented in the model-checker SAT-MC.

Symbolic representations of terms can be combined with partially-ordered finite sets of events to represent (possibly infinite) sets of states or traces. Such a partially-ordered set  $E$  is used to represent all traces of the protocol that contain an instance of  $E$  as a substructure. By applying the operational semantics backwards to the events in  $E$ , additional constraints on its traces can be derived from  $E$ , such as adding additional preceding events, unifying messages, or adding constraints on the adversary knowledge. This process can either lead to a contradiction, in which case  $E$  represents no traces of the protocol, or to a witness trace of the protocol that contains an instance of  $E$ . Such representations form the basis of Athena and Scyther.

#### 24.4.4 Optimizations

##### Partial Order Reduction (POR)

Partial order reduction, as discussed in Chapter 11, is a natural optimization technique in the context of model-checking security protocols. The reasons for this are twofold. First, separate threads are largely independent processes, which communicate only through a single shared channel. Second, secrecy (Definition 9) depends only on the adversary knowledge and, in our definition, on the communication partners of completed threads.

*Example 3 (POR).* As a simple example, consider a state  $s$  with two threads identified by  $tid_1$  and  $tid_2$ . Assume that the next actions of both threads are respectively the receive events  $e_1$  and  $e_2$  of the patterns  $pt_1$  and  $pt_2$ , and that there exist messages in the adversary knowledge such that both  $e_1$  and  $e_2$  can be executed. More formally, consider the state  $s = (tr, IK, th)$ , two sequences  $l_1, l_2$ , and two substitutions  $\sigma_1, \sigma_2$ , such that for all  $i \in \{1, 2\}$ ,

$$e_i = \text{recv}(pt_i) \wedge th(tid_i) = \langle e_i \rangle^{\wedge} l_i \wedge IK \vdash \sigma_i(pt_i) \wedge \text{dom}(\sigma_i) = FV(pt_i),$$

and where  $FV(pt_1) \cap FV(pt_2) = \emptyset$ .

Observe that in this state,  $e_1$  and  $e_2$  can be executed in any order. This results in either  $s_1$  or  $s_2$ , where

$$\begin{aligned} s &\rightarrow_P^* s_1, \text{ and } s_1 = (tr^{\wedge} \langle (tid_1, \sigma_1(e_1)) \rangle^{\wedge} \langle (tid_2, \sigma_2(e_2)) \rangle, IK, th'), \\ s &\rightarrow_P^* s_2, \text{ and } s_2 = (tr^{\wedge} \langle (tid_2, \sigma_2(e_2)) \rangle^{\wedge} \langle (tid_1, \sigma_1(e_1)) \rangle, IK, th'), \end{aligned}$$

and where  $th' = th[\sigma_1(l_1) \leftarrow tid_1][\sigma_2(l_2) \leftarrow tid_2]$ . Observe that  $s_1$  is identical to  $s_2$  except for its trace component. Because the premises of the transition

rules do not depend on a state's trace component, the successor states of  $s_1$  are identical to those of  $s_2$  except for the traces.

To simplify the example, we additionally assume that for  $i \in \{1, 2\}$ ,

$$\forall tid, \sigma . \text{HT}(s, tid, \sigma) \Leftrightarrow \text{HT}(s_i, tid, \sigma).$$

Hence, if the secrecy property is violated in state  $s_1$ , then it is also violated in  $s_2$ , and vice versa. Thus, we can safely explore only one of these successor states: if there is a state reachable from  $s_1$  that violates secrecy, then there will also be a state reachable from  $s_2$  that is identical up to the trace component and thus secrecy is violated, and vice versa.

Similarly, for secrecy properties, one can consider only paths in which threads with send actions are executed first (and ignore paths in which these same sends are executed later) as this will only provide the adversary with more knowledge earlier.

For authentication properties, POR techniques are not necessarily sound: authentication properties depend on the order in which events occur, and therefore ignoring some orderings may cause attacks to be missed. For each property (or class of properties), the soundness of a particular partial order reduction scheme must be individually proven. The main proof obligation is to show that if there is an attack in a state that is not explored by the POR scheme, then there is also an attack on a state that is explored in the POR scheme.

#### 24.4.5 Handling equations

The formal model introduced in Section 24.3 uses a free algebra to represent operations on data. Thus the adversary's ability to perform encryption is represented by the deduction rule that says that  $\{t_1\}_{t_2}^a$  may be deduced from  $t_1$  and  $t_2$ , and his ability to perform decryption is represented by the deduction rule that says that  $t_1$  may be deduced from  $\{t_1\}_{t_2}^a$  and  $t_2^{-1}$ . This approach is in general adequate for a large class of operators, but we run into trouble when we include operations such as exclusive-or which obey different equational theories. In this case, we need to include not only a rule such as

$$M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash t_1 \oplus t_2$$

But also the set of equations

$$\begin{array}{ll} t \oplus 0 = t & t_1 \oplus t_2 = t_2 \oplus t_1 \\ t \oplus t = 0 & t_1 \oplus (t_2 \oplus t_3) = (t_1 \oplus t_2) \oplus t_3 \end{array}$$

Similar problems arise when we introduce protocols based on Diffie-Hellman exponentiation or protocols based on homomorphic encryption, such as those involving blind signatures.

There are two ways of dealing with this problem. The first is to replace the equational theory with a set of inference rules that is equivalent under certain syntactic restrictions on the protocol. This is the approach followed in the previously mentioned work on encryption-decryption [Mil03] and also by Küsters and Truderung, who develop inference rules for the theory governing exclusive-or [KT08] and a subtheory of the theory governing Diffie-Hellman exponentiation [KT09]. This approach is most appropriate when one wishes to use a tool that does not directly support reasoning about equational properties.

The second approach is to adapt the reasoning used by the tool to the equational theory at hand. A substantial amount of work has been done in this area. There are two main techniques that researchers have concentrated on. One is an extension of the intruder deduction problem to include equational theories. Thus, we now ask: given a set of terms  $M$ , a term  $t$ , and an equational theory  $E$ , is it possible to determine whether or not  $M \vdash t$  modulo  $E$ ? The decidability of the intruder deduction problem in the free theory was the main component of Rusinowitch's and Turuani's proof of decidability of security in the bounded session model. Decision procedures have been subsequently given for a large class of equational theories relevant for model checking cryptographic protocols [AC06]. These theories include a class of rewrite theories known as subterm-convergent (for which the intruder deduction problem is decidable in polynomial time), and other theories such as homomorphic encryption and exclusive-or. Algorithms for verifying intruder deduction modulo finite convergent rewrite theories have been implemented in several tools [BCD09b, CDK09, CBC10].

The other technique is equational unification: given an equational theory  $E$  and two terms  $t$  and  $s$ , find a complete description of all the substitutions  $\sigma$  such that  $\sigma s = \sigma t$  modulo  $E$ . This is useful for determining which states can immediately follow (or precede) a given state: one unifies the current state with the output (or input) of a state transition. Equational unification was a well-known technique long before it was applied to cryptographic protocol analysis. Indeed, anyone who has solved an arithmetic equation such as  $x + 7 = 12 + y$  has applied equational unification. However, unification research generally concentrated on algorithms for special-purpose theories. In cryptographic protocol analysis, it is necessary to be able to apply unification to a range of theories, and to different combinations of theories. Thus generic approaches that apply to classes of theories that can be easily combined are preferable to special purpose algorithms, even when the special purpose algorithms may be more efficient. One technique that follows this approach is the process known as *variant narrowing* [EMS09], which can be applied to a class of theories that satisfy a property known as the *finite variant property* [CLD05]. This is satisfied by a large number of equational theories of interest for cryptographic protocol analysis, with the major exception being homomorphic encryption. A version of variant narrowing has been implemented in the Maude-NPA protocol analysis tool, discussed in Section 24.5.1.

We note that although there is a large overlap between subterm convergent and finite variant theories, the finite variant property does not imply subterm convergence, since by definition no associative-commutative theories are subterm convergent. Whether the converse holds is, to the best of our knowledge, still an open problem.

The most pressing open problem in the short term is how to extend the available tools to handle additional theories. In some cases (e. g. the intruder deduction problem for non-subterm-convergent theories) the exact complexity is not known and needs to be understood better. In other cases (for example some of the unification problems connected with different homomorphic encryption theories), the problem is known to be intractable or undecidable, and what instead needs to be investigated are (preferably syntactic) conditions on cryptographic protocol specifications that make the problems tractable. There is also the issue of combining theories. For example, in the case of unification, general algorithms for combining unification algorithms for different theories have been known for some time [SS89], but their generality forces them to be highly nondeterministic, and thus they are usually too slow to be practical. It may be possible to restrict ourselves to particular types of theories such that practical algorithms can be developed. However, it is still unknown whether such classes of theories contain the main theories of interest for cryptographic protocol analysis.

More generally, there is the problem of determining what one actually learns from an analysis with respect to an equational theory. For example, Kremer et al. [KRS10] have been using subterm-convergent theories to analyze voting protocols. However, the primitives that they describe with subterm-convergent theories are implemented by algorithms satisfying richer equational theories, usually involving Abelian groups. Are the higher-level descriptions in terms of subterm-convergent theories safe approximations? Conversely, when would a free theory be a safe approximation for a subterm-convergent theory? This question has been answered for the subterm-convergent theories involving both shared and public key encryption [Mil03]. Can this result be extended to more general cases? Finally, when is any theory a safe approximation of a computational theory of protocol correctness? Some initial work on this last problem has been done by Baudet et al. [BCK09], but there is still much to be learned.

## 24.5 Systems and algorithms

In this section, we give some representative examples of systems based on the algorithmic approaches just presented.

### 24.5.1 NPA and Maude-NPA

The NRL Protocol Analyzer, or NPA, was one of the earliest tools for verifying the security of cryptographic protocols. Although not originally designed

as a model checker, it later took on many of the features of one, including the ability to check properties expressed in a temporal logic language, NPATRL [MSC04]. In NPA, both the actions of honest agents and the adversary were specified in terms of state transitions. Model checking employed backward search, where the output of a state transition was unified with the current state. NPA had limited support for equational unification and could, for example, model properties such as the cancellation of encryption and decryption.

One of the most interesting properties of NPA was that it was an unbounded session model checker. It included built-in inductive techniques for building grammars defining languages of infinite search paths. Once NPA reached a state that contained a term in the grammar, it would not explore beyond that state. This technique often allowed NPA to terminate after a finite number of steps, although of course termination was not guaranteed. It has been used to verify a number of protocols and protocol standards, including the Internet Key Exchange Protocol [Mea99], the Group Domain of Interpretation Protocol [MSC04], and the Simmons Selective Broadcast Protocol [Mea92].

Maude-NPA is a descendant of NPA, implemented in the Maude rewriting language [CDE<sup>+</sup>07]. It shares many of NPA's features, including the use of unification to implement backward search and the reliance on grammars to ensure termination. There are two main differences. First, it is implemented in the Maude rewriting language, which gives it a formal basis in rewriting logic. In particular, backward search in Maude-NPA is implemented via narrowing over a simple transition model that is expressed via a small set of rewrite rules. Narrowing is a technique for deduction using rewrite rules and unification that given a term  $t$ , finds a substitution  $\sigma$  that unifies a subterm with the left-hand side of a rewrite rule  $l \rightarrow r$ . The subterm  $\sigma l$  of  $\sigma t$  is replaced by  $\sigma r$ . When the rewrite rules describe state transitions, narrowing gives an algorithm for state space exploration. When the arrows are reversed, as they are in Maude-NPA, narrowing can be used to implement backward search from a final goal. A second difference is that Maude-NPA is devoted to reasoning about the different equational theories that describe the behavior of cryptographic algorithms. Maude-NPA makes use of unification modulo different equational theories as the unification step in its narrowing algorithm.

Maude-NPA also incorporates two state space reduction techniques that were originally used in NPA, but have been refined and extended in Maude-NPA. The first is subsumption-based partial order reduction, in which the unreachability of a Maude-NPA state description is implied by the unreachability of another state description if a certain subsumption relation exists between them. The second is a super-lazy intruder. This is similar to the lazy intruder of the OFMC, except that it is adapted for backward search. If a variable term or a term constructed out of variable terms appears in the adversary knowledge part of the state, then this term is not searched any further (that is, it is removed from the state), since the adversary should be able

to find it using arbitrary terms.<sup>2</sup> However, it is not deleted entirely but kept around in a ghost state. If any of the variables in the term are instantiated, the ghost state is resuscitated. Among the equational theories that Maude-NPA can currently handle are a subclass of subterm-convergent theories, as well as exclusive-or, Abelian groups, modular exponentiation, bounded associativity and homomorphic encryption over a free operator. Work is now ongoing on incorporating more general homomorphic encryption.

### 24.5.2 AVISPA and related tools

The AVISPA tool [ABB<sup>+</sup>05] is a popular model checker that integrates several different model-checking approaches. AVISPA provides a high-level specification language HLPSL, for specifying protocols and their properties. Protocols are specified in HLPSL in terms of their roles, using control flow patterns, data structures, alternative adversary models, as well as different cryptographic primitives and their algebraic properties. HLPSL specifications have a declarative semantics based on Lamport’s Temporal Logic of Actions [Lam94] and an operational semantics defined in terms of a rewrite-based formalism called the intermediate format, or IF. Different model-checking backends interpret the IF and can be used for (bounded) verification or falsification.

The main backends in the AVISPA tool are the Constraint-Logic-based Attack Searcher CL-Atse, the On-the-fly Model-Checker OFMC, and the SAT-based Model-Checker SAT-MC. We previously discussed SAT-MC in Section 24.4.3 and restrict our attention here to CL-Atse and OFMC.

CL-Atse, like the other AVISPA backends, operates on IF specifications of protocols. CL-Atse represents protocol states symbolically as collections of non-ground facts, which record the states of different threads, the messages sent to the network, and the adversary knowledge. In particular, constraints are used to describe what the different agents know and a constraint calculus is used to solve for what they can know, from messages previously exchanged, i. e., the calculus is used to solve a variant of the non-ground intruder deduction problem. CL-Atse was designed to allow the easy integration of new deduction rules and operator properties. In particular, CL-Atse integrates a version of Baader and Schulz’s unification algorithm [BS96] with modules for xor, exponentiation, and associative paring.

OFMC combines a number of techniques to enable the efficient analysis of security protocols. First, OFMC uses lazy data types (in a functional programming setting) as a simple way of building efficient on-the-fly model checkers for protocols with very large, or even infinite, state spaces. A lazy data type is one where data constructors (such as *cons* for building lists or *node* for building trees) build data without evaluating their arguments; this allows one to represent and compute with infinite data (e. g., streams or infinite trees),

---

<sup>2</sup> Maude-NPA does not have secret types, so we assume that the adversary can create at least one term of any type.

generating arbitrary prefixes of the data on demand. In [Bas99], lazy data types are used to build, and compute with, models of security protocols: a protocol and a description of the powers of an adversary are formalized as an infinite tree. Lazy evaluation is used to decouple the model from search and heuristics, building the infinite tree on the fly, in a demand-driven fashion. Second, OFMC models the adversary in a lazy fashion (the so-called “lazy intruder”), where adversary communication is represented symbolically and solved during search. To this end, like CL-Atse, it integrates a constraint solver for the non-ground intruder deduction problem. Effectively OFMC performs search at two levels: search in the space of symbolic states, and search in the space of constraints. Third, while OFMC performs verification for a bounded number of sessions, it works with *symbolic session generation* which avoids enumerating all possible ways of instantiating possible sessions. Fourth, OFMC exploits a state-space reduction technique, inspired by partial-order reduction, called constraint differentiation [MVB10]. Constraint differentiation works by eliminating certain kinds of redundancies that arise in the search space when using constraints to represent and manipulate the messages that may be sent by the adversary. Namely, different symbolic states may describe overlapping sets of ground states. Constraint differentiation essentially compute a set-difference symbolically, to minimize these overlaps. This can be seen as generalizing the kind of subsumption-based partial-order reduction used in Maude-NPA. Finally, OFMC also provides some limited support for handling different equationally specified operators on messages [BMV05a].

### 24.5.3 Athena and Scyther

The Athena and Scyther algorithms implement model-checking with respect to the unbounded model described in Section 24.3.5 by performing a backward-style search. For these methods, the model is extended with *adversary events* for *encrypting*, *decrypting*, *hashing*, and *knowing* messages. Infinite sets of states are represented by *(trace) patterns*: partially-ordered sets of events that must occur in the traces, and whose messages may contain variables. (In the Athena model, patterns are referred to as *semi-bundles*.) The events in patterns must satisfy a number of criteria that follow from the semantics. For example, if an event occurs in the pattern from a role  $R$  with thread identifier  $tid$ , then (a) the pattern does not contain events from other roles with thread identifier  $tid$  and (b) the event is preceded in the pattern by all events that precede it in the role  $R$ , with identical substitutions and thread identifier  $tid$ . However, it is not required for receive events in patterns that the received term can be inferred from the union of the initial knowledge and the messages that occur in preceding send events within the pattern. Patterns allow for specifying properties such as secrecy.

*Example 4 (Secrecy pattern).* The following pattern  $PT$  specifies the violation of secrecy of the nonce of the responder role of the Needham-Schroeder protocol, performed by  $b$  when trying to communicate with  $a$ , where  $X$  and  $tid$  are

variables. The `AdversaryKnows` event is used to encode the secrecy violation.

$$\begin{aligned}
e_1 &= (tid, \text{rcv}(a, b, \{ \{ a, X \}_{pk(b)}^a \})) \\
e_2 &= (tid, \text{send}(b, a, \{ \{ X, N_B \# tid \}_{pk(a)}^a \})) \\
e_3 &= (tid, \text{rcv}(a, b, \{ \{ N_B \# tid \}_{pk(b)}^a \})) \\
e_4 &= \text{AdversaryKnows}(N_B \# tid) \\
PT &= \left( \{e_1, e_2, e_3, e_4\}, \{(e_1, e_2), (e_2, e_3)\}^* \right)
\end{aligned}$$

This pattern represents an infinite set of traces of the Needham-Schroeder protocol, e. g., the attack from Figure 24.2 and all traces that additionally include arbitrarily interleaved threads. In contrast, the corresponding pattern for the initiator role (also with agents  $a$  and  $b$ ) represents the empty set of traces.

By introducing restrictions on variable instantiations into the algorithm, and replacing  $a$  and  $b$  by variables that can only be instantiated by non-compromised agents, we can faithfully represent all violations of the secrecy property for the protocol in the above example.

During backward search, a case distinction on the source of messages is used for branching and the patterns are extended by either adding events, adding ordering constraints, or unifying terms. The search can terminate in two ways. First, the pattern can be proven to be empty, i. e., it contains no traces of the protocol. The main mechanism here is detecting cyclic dependencies of the messages. Second, the receive events in the pattern meet all premises of the receive event: the adversary can produce an appropriate message from the preceding events. In such a case, the pattern is called *realizable* and it corresponds to an infinite set of actual traces; a representative trace (of minimal length) from this set can be generated by linearizing the non-adversary events and instantiating the remaining variables from the adversary knowledge.

Scyther differs from Athena in how it makes the case distinction and in the possible outcomes of the analysis. With respect to the possible outcomes, Scyther bounds the size of the patterns but detects whether the bound is reached. By bounding the size of the patterns, termination is guaranteed and one of three possible results occurs. First, if a realizable pattern is found, a representative (attack) trace is constructed. Second, if no realizable patterns are found, and the bound is not reached, no realizable patterns exist (for any bound). In case of an attack pattern, this corresponds to the absence of attacks. Third, if no realizable patterns are found but the bound is reached, the result can be interpreted as verification with respect to a bounded number of sessions and is similar to the guarantees provided by, e. g., OFMC or CL-AtSe when they do not find attacks.

#### 24.5.4 ProVerif

The ProVerif tool uses abstractions to obtain an efficient analysis method. In particular, it employs two main abstractions compared to the operational

semantics presented before. First, individual fresh values are abstracted into sets of fresh values. Second, each action of a thread can be executed multiple times.

The abstracted protocol model can be represented as a set of Horn clauses. These Horn clauses are analyzed using a two-phase resolution algorithm. In the first phase, the Horn clauses are saturated in a forward fashion until a fixpoint is reached. This phase combines multiple derivations into single rules, thereby optimizing the rule set. Facts can be derived from the optimized rule set if and only if they can be derived from the original rule set. In the second phase, a backward depth-first search is used to try to establish that a fact (usually representing the adversary knowing a message that is supposed to be secret) cannot be derived from the saturated Horn clauses. If this cannot be established because the fact can be derived, an attempt is made to reconstruct a corresponding protocol trace.

If we assume that the fact represents a secret, we can interpret each of the four possible results of running the tool in the following way. First, if the fact cannot be derived, the over-approximation ensures that no protocol execution will leak the secret, and hence the protocol satisfies secrecy. Second, if the fact can be derived and the derivation can be translated into a corresponding (attack) trace, the protocol does not satisfy secrecy, as witnessed by the trace. Third, if the fact can be derived but no corresponding trace can be reconstructed from the derivation, the result is inconclusive. Finally, either of the two phases in the algorithm may not terminate and again no knowledge is gained about the security of the protocol.

ProVerif can handle authentication properties formalized as correspondence properties [Bla09]. These properties express that when event  $e_1$  occurs, event  $e_2$  must have occurred earlier with related parameters  $\rho_1$  and  $\rho_2$ . For example  $\rho_1$  could be a nonce and  $\rho_2$  a variable that is supposed to be instantiated with  $\rho_1$ . In this case, it does not suffice to prove that the values of  $\rho_1$  and  $\rho_2$  abstract into the same set (i. e., they are in the same equivalence class in the abstraction) and a finer abstraction is used. By introducing session identifiers in the construction of  $\rho_1$  and  $\rho_2$ , the precision of the verification is increased at the cost of efficiency and termination.

## 24.6 Research problems

Although the research community has come far in the past decades, many research challenges remain. Below we describe some of the most pressing problems being tackled as well as open problems.

### 24.6.1 Link to computational soundness

The Dolev-Yao model, even when equational properties are added, treats cryptosystems as black boxes. If the adversary possesses the appropriate key, he

can learn the contents of an encrypted term. Otherwise he is completely ignorant of the corresponding plaintext. Moreover he is only able to perform the specific operations specified in the protocol. This is very different from definitions used by cryptographers. In these definitions, the adversary is modeled as a probabilistic polynomial time Turing machine. The security properties of the cryptosystems themselves vary depending on what kinds of attacks they are assumed to be secure against, e. g. chosen plaintext or chosen ciphertext. Finally, secrecy is usually specified not in terms of the adversary not being able to obtain a given secret, but in the indistinguishability between two different versions of the protocol, e. g. two versions with different secrets, or one constructed using a secret and one constructed using random data, or in the indistinguishability between the real secret and a random bit string. Is it possible to come up with an approach to proving protocols correct that combines the amenability to exhaustive search of the Dolev-Yao model with the stronger requirements of cryptographic models?

There has been a substantial amount of work on this problem. The general idea is to have two models, a symbolic model and a computational model, and establish some sort of relationship between them, e. g., a simulation relation, so that security in the symbolic model implies security in the computational model. In some approaches, the symbolic specification is trivially secure, while in others it is a Dolev-Yao style specification that can be verified with a model checker. A survey of research in this area is given by Cortier et al. in [CKW10].

Although research in this area has been successful in establishing links between the two models, the results have been criticized both for being so complex as to detract from the advantage of being able to reason at the simpler Dolev-Yao level, and for being too limited in the types of cryptosystems they can handle. For example, the reactive simulatability framework of Backes et al. [BPW07], one of the most prominent models in this area, has been shown to be impossible to extend to include two standard items in the cryptographer's toolbox: one-way hash functions [BPW06] and exclusive-or [BP05]. This is perhaps not surprising, given the divergence between the two models and the difficulty of bringing the two together. But the complete answer is not yet known, and this is still an active area of research.

### 24.6.2 Corruption models

In the basic Dolev-Yao model described in Section 24.3, the Dolev-Yao adversary initially has the long term keys of some of the agents. This formalizes a notion of static corruption where the adversary has compromised some of the agents and can play as an “insider” during protocol execution. However, many protocols are intended to be secure against much more sophisticated corruption models. [CK01, JV96, Sho99], for example, specify adversaries who can dynamically corrupt long term secrets, session keys and other parts of the session state, or even random number generators. For example, a Diffie-Hellman key agreement protocol, where digital signatures are used to authenticate the

exchanged half-keys, provides perfect forward secrecy [MvOV96]: the resulting key remains secret even when the signature keys are later compromised by the adversary.

Some of the earlier model-checkers, such as the NRL Protocol Analyzer, allowed for the dynamic corruption of keys. More recently in [BC10b, BC10a], Basin and Cremers have examined this issue more thoroughly by formalizing a hierarchy of corruption models. Their models extend the operational semantics presented earlier and cover many aspects of the adversary models used in cryptographic models. The Scyther tool supports the evaluation of protocols with respect to these corruption models, which has led to the automatic discovery of many attacks that previously could only be found by manual analysis.

The additional complexity of richer corruption models significantly increases the time required for verification. To make analysis of larger protocols with respect to these models feasible, it would be useful to develop more efficient dedicated model-checking algorithms. Furthermore, traditional compositionality results, e. g., [GT00, ACG<sup>+</sup>08], no longer hold under stronger corruption models. Hence, another research challenge is to develop analogous compositionality results for this purpose.

An alternative direction is to weaken the adversary models. In fact, in many cases, protocols are designed under the assumption that adversaries are not completely dishonest and for various reasons do not perform all the activities available to the Dolev-Yao adversary. One example is the *honest but curious* agent, who acts according to the rules of the protocol, but attempts to learn secret information from the messages that it has received legitimately. A related case is the *passive adversary* who does not even participate in the protocol but simply observes passing traffic. Other protocols, including for example many electronic commerce protocols, are based on the assumption that an adversary will not take any action against its own best interests, such as those that involve revealing its own secret information. Although a fair amount of work has been done on model checking protocols with respect to these various adversary models, a more comprehensive approach, in which the user could specify which adversary model will be used, would be of benefit.

### 24.6.3 Channel properties

The standard Dolev-Yao model gives the adversary control over all communication channels. The adversary sees all communication, and can block, alter, and redirect traffic at will. Thus all an agent can conclude from receipt of information sent along one of these channels is that somebody sent it. Assurance of other properties must be gained by cryptographic means. But there is a growing number of cryptographic protocols that use either channels with special properties or rely upon assumptions about weaker adversaries. These include anonymous routing protocols such as Tor, which assume an adversary

who is able to spy on only part of the network, distance bounding protocols [BC94] and secure localization protocols [CH06], which rely on the use of timed wireless communications to verify that a prover is within a certain range of a verifier, and protocols that rely on human-verifiable channels [BSSW02], such as a human reading a sequence of numbers off a computer screen, to bootstrap key distribution in the lack of a public key infrastructure. Work has been ongoing on developing methods for formal analysis of protocols that use these channels, e. g. [SSBC09, BCSS09, MPP<sup>+</sup>06, TSG10], but most of it has not yet been applied to model checking, concentrating more on theorem proving or specialized logics. The problem of how best to model and reason about these channels using a model checker is still open.

#### 24.6.4 Other properties, including non-trace properties

The application of formal methods to cryptographic protocol analysis was originally restricted to the study of various forms of secrecy and authentication. These are straightforward to formulate using temporal logics and analyze using model checkers. However, there are other classes of properties that are less straightforward. Often these are properties that are not trace properties themselves, but can be approximated by trace properties such that if the trace property holds, then so does the original property.

One of the earliest properties of this type is *non-interference*, which, roughly speaking, is the property that events labelled “high” should have no discernible effect on events labeled “low”. Non-interference is closely related to the cryptographic notion of indistinguishability, mentioned in Section 24.6.1. Indistinguishability formalizes that an adversary should be unable to distinguish between two protocols, usually either the same protocol using different secrets, or a real protocol and a simulation of the protocol using random data. This can be approximated by a trace-based notion called *observational equivalence*, which has been implemented in ProVerif by running the two protocols in tandem and checking for equivalence at each transition [BAF05].

Another property related to indistinguishability is *static equivalence* [AF01]. This notion is defined with respect to an underlying equational theory and, roughly speaking, two terms are statically equivalent when they satisfy the same equations. As noted in [BWA10], static equivalence is essentially a special case of observational equivalence that does not allow for continued interaction between a system and an observer: the observer gets data once and conducts experiments on its own. Static equivalence has direct application to modeling off-line guessing attacks, which are attacks where the adversary tries to guess a secret and verify his guess, without requiring further communication. As shown in [CDE05, Bau05], static equivalence may be used to specify the absence of off-line guessing attacks by expressing that the adversary cannot distinguish between two versions of the same symbolic trace: one corresponding to a correct guess and the other corresponding to an incorrect guess. Decision procedures for static equivalence have been implemented by

the YAPA [BCD09a], KISS [CDK09], and FAST [CBC10] tools. ProVerif also supports the analysis of off-line guessing attacks, but based on a different formalization of guessing due to [CDE05].

We see that there have been a number of individual solutions to reasoning about special properties. But what would be useful to have is a more general approach to such properties that could be tailored to specific instances. For example, as we have seen from the above, many security properties are expressed in terms of some sort of equivalence between families of traces. One might expect a general procedure to exist for such equivalences.

#### 24.6.5 Probabilities

As noted previously, probabilities are part of standard cryptographic definitions of security. Probabilities also arise when security protocols are based on randomized algorithms or the protocol guarantees themselves are probabilistic. Different options for augmenting transition systems and logics with probabilities are discussed in Chapter 29 of this handbook. Probabilistic model-checking has been successfully applied to different kinds of security protocols. Examples include protocols for anonymity [Shm04], non-repudiation [LMST05], and contract signing [NS06].

As an example, in [Shm04], Shmatikov models the protocol underlying Crowds [RR98], which is a peer-to-peer group communication system based on random message routing among members. He models the behavior of group members and the adversary as a discrete-time Markov chain, and formalizes the anonymity properties for the system in the temporal probabilistic logic PCTL. Given this model, he can use the PRISM model checker [HKNP06] to analyze the anonymity provided by the system, showing, for example, how probabilistic anonymity degrades as the group size increases.

Until now, work on model-checking stochastic systems and model-checking security has been disjoint. The problem of using off-the-shelf general model checkers, whether for stochastic systems or other classes of systems, is that they neither offer optimizations useful in the domain of security (e.g., for handling the Dolev-Yao intruder) nor support cryptographic operators and equational extensions. An open problem is how best to combine the models and algorithms from these two areas.

## 24.7 Conclusions

We have shown in this chapter how to extend transition-system models of concurrent computation to model cryptographic protocols. The main extensions have been with a term language formalizing cryptographic messages and a model of a network (Dolev-Yao) adversary. These extensions themselves are fairly straightforward but they lead to two immediate challenges: the resulting

system is infinite state and the active adversary results in considerable nondeterminism. Addressing these challenges has motivated a number of specialized model-checking techniques.

The state-of-the-art is able to handle classical authentication and key-exchange protocols of realistic, but limited, complexity. A fully automatic analysis of the Internet Key Exchange Protocol, with all its variations, would lead to state explosion and is outside of the current state-of-the-art. State explosion arises due to a combination of the protocol's size and the cryptographic operators used. In the short-term, improved support for equational reasoning will make a difference; in the mid-term and long-term, better support for abstraction.

Security protocols go far beyond authentication and key-exchange in practice. We have sketched some of the challenges in handling more precise, cryptographic notions of security as well as wider classes of cryptographic primitives and properties. Interestingly, advances in other areas of model-checking, such as probabilistic model-checking, combined with advances in security model-checking, may play an important role in enabling new classes of applications.

---

## References

- [ABB<sup>+</sup>05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron, *The AVISPA tool for the automated validation of internet security protocols and applications*, CAV, Lecture Notes in Computer Science, vol. 3576, Springer, 2005, pp. 281–285.
- [AC04] Alessandro Armando and Luca Compagna, *SATMC: A SAT-based model checker for security protocols*, JELIA, Lecture Notes in Computer Science, vol. 3229, Springer, 2004, pp. 730–733.
- [AC06] Martín Abadi and Véronique Cortier, *Deciding knowledge in security protocols under equational theories*, Theor. Comput. Sci. **367** (2006), no. 1-2, 2–32.
- [ACG<sup>+</sup>08] S. Andova, C.J.F. Cremers, K. Gjøsteen, S. Mauw, S.F. Mjølsnes, and S. Radomirović, *A framework for compositional verification of security protocols*, Information and Computation **206** (2008), 425–459.
- [AF01] M. Abadi and C. Fournet, *Mobile values, new names, and secure communication*, Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, 2001, pp. 104–115.
- [BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet, *Automated Verification of Selected Equivalences for Security Protocols*, 20th IEEE Symposium on Logic in Computer Science (LICS 2005) (Chicago, IL), IEEE Computer Society, June 2005, pp. 331–340.
- [Bas99] David Basin, *Lazy infinite-state analysis of security protocols*, Secure Networking — CQRE [Secure] '99 (Düsseldorf, Germany), Lecture Notes in Computer Science, no. 1740, Springer-Verlag, November 1999, pp. 30–42.
- [Bau05] M. Baudet, *Deciding security of protocols against off-line guessing attacks*, Proceedings of the 12th ACM conference on Computer and communications security, ACM, 2005, pp. 16–25.
- [BC94] Stefan Brands and David Chaum, *Distance-bounding protocols*, EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology (Secaucus, NJ, USA), Springer-Verlag New York, Inc., 1994, pp. 344–359.

- [BC10a] David Basin and Cas Cremers, *Degrees of security: Protocol guarantees in the face of compromising adversaries*, 19th EACSL Annual Conference on Computer Science Logic (CSL) (Brno, Czech Republic), Lecture Notes in Computer Science, vol. 6247, Springer-Verlag, 2010, pp. 1–18.
- [BC10b] ———, *Modeling and analyzing security in the presence of compromising adversaries*, Computer Security — ESORICS 2010, Lecture Notes in Computer Science, vol. 6345, Springer, 2010, pp. 340–356.
- [BC11] ———, *Evaluation of ISO/IEC 9798 protocols*, Tech. Report 2014, CRYPTREC, January 2011, <http://www.cryptrec.go.jp/english/estimation.html>.
- [BCD09a] M. Baudet, V. Cortier, and S. Delaune, *YAPA: A generic tool for computing intruder knowledge*, Rewriting Techniques and Applications, Springer, 2009, pp. 148–163.
- [BCD09b] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune, *YAPA: A generic tool for computing intruder knowledge*, RTA, Lecture Notes in Computer Science, vol. 5595, Springer, 2009, pp. 148–163.
- [BCK09] Mathieu Baudet, Véronique Cortier, and Steve Kremer, *Computationally sound implementations of equational theories against passive adversaries*, Inf. Comput. **207** (2009), no. 4, 496–520.
- [BCSS09] David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt, *Let's get physical: Models and methods for real-world security protocols*, 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs) (Munich, Germany), Lecture Notes in Computer Science, vol. 5674, Springer-Verlag, 08 2009, Invited paper, pp. 1–22.
- [BFCZ08] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zalinescu, *Cryptographically verified implementations for TLS*, ACM Conference on Computer and Communications Security, ACM, 2008, pp. 459–468.
- [BFGS08] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Nikhil Swamy, *Verified implementations of the information card federated identity-management protocol*, ASIACCS, ACM, 2008, pp. 123–135.
- [Bla01] Bruno Blanchet, *An efficient cryptographic protocol verifier based on Prolog rules*, CSFW, IEEE Computer Society, 2001, pp. 82–96.
- [Bla09] ———, *Automatic verification of correspondences for security protocols*, Journal of Computer Security **17** (2009), no. 4, 363–434.
- [BMV05a] David Basin, Sebastian Mödersheim, and Luca Viganò, *Algebraic intruder deductions*, LPAR 2005, LNAI, vol. 3835, Springer-Verlag, December 2005, pp. 549–564.
- [BMV05b] David A. Basin, Sebastian Mödersheim, and Luca Viganò, *OFMC: A symbolic model checker for security protocols*, Int. J. Inf. Sec. **4** (2005), no. 3, 181–208.
- [BN98] Franz Baader and Tobias Nipkow, *Term rewriting and all that*, Cambridge University Press, 1998.
- [BP05] Michael Backes and Birgit Pfitzmann, *Limits of the cryptographic realization of Dolev-Yao-Style XOR*, ESORICS, Lecture Notes in Computer Science, vol. 3679, Springer, 2005, pp. 178–196.
- [BPW06] Michael Backes, Birgit Pfitzmann, and Michael Waidner, *Limits of the BRSIM/UC soundness of Dolev-Yao models with hashes*, ESORICS, Lecture Notes in Computer Science, vol. 4189, Springer, 2006, pp. 404–423.

- [BPW07] ———, *The reactive simulatability (RSIM) framework for asynchronous systems*, Inf. Comput. **205** (2007), no. 12, 1685–1720.
- [BS96] F. Baader and K. U. Schulz, *Unification in the union of disjoint equational theories: Combining decision procedures*, J. Symbolic Computation **21** (1996), 211–243.
- [BSSW02] Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong, *Talking to strangers: Authentication in ad-hoc wireless networks*, Proceedings of Network and Distributed System Security Symposium 2002 (NDSS'02) (San Diego, CA), February 2002.
- [BWA10] M. Baudet, B. Warinschi, and M. Abadi, *Guessing attacks and the computational soundness of static equivalence*, Journal of Computer Security **18** (2010), no. 5, 909–968.
- [CBC10] Bruno Conchinha, David Basin, and Carlos Caleiro, *Efficient decision procedures for message deducibility and static equivalence*, FAST '2010, vol. -, Lecture Notes in Computer Science, 2010, Accepted for Publication.
- [CDE05] R. Corin, J. Doumen, and S. Etalle, *Analysing password protocol security against off-line dictionary attacks*, Electronic Notes in Theoretical Computer Science **121** (2005), 47–63.
- [CDE<sup>+</sup>07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, *All about maude - a high-performance logical framework, how to specify, program and verify systems in rewriting logic*, Lecture Notes in Computer Science, vol. 4350, Springer, 2007.
- [CDK09] S. Ciobâca, S. Delaune, and S. Kremer, *Computing knowledge in security protocols under convergent equational theories*, Automated Deduction: 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings, Springer-Verlag, New York, 2009, pp. 355–370.
- [CH06] S. Capkun and J.P. Hubaux, *Secure positioning in wireless networks*, Selected Areas in Communications, IEEE Journal on **24** (2006), no. 2, 221–232.
- [CJM00] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero, *Verifying security protocols with Brutus*, ACM Trans. Softw. Eng. Methodol. **9** (2000), no. 4, 443–487.
- [CK01] R. Canetti and H. Krawczyk, *Analysis of key-exchange protocols and their use for building secure channels*, EUROCRYPT, Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, 2001, pp. 453–474.
- [CKW10] Véronique Cortier, Steve Kremer, and Bogdan Warinschi, *A survey of symbolic methods in computational analysis of cryptographic systems.*, Journal of Automated Reasoning. (2010), 1–35.
- [CLD05] Hubert Comon-Lundh and Stéphanie Delaune, *The finite variant property: How to get rid of some algebraic properties*, RTA, Lecture Notes in Computer Science, vol. 3467, Springer, 2005, pp. 294–307.
- [CLT04] H. Comon-Lundh and R. Treinen, *Easy intruder deductions*, Verification: Theory and Practice (2004), 182–184.
- [Cre08] Cas J. F. Cremers, *The Scyther tool: Verification, falsification, and analysis of security protocols*, CAV, Lecture Notes in Computer Science, vol. 5123, Springer, 2008, pp. 414–418.

- [DEK82] Danny Dolev, Shimon Even, and Richard M. Karp, *On the security of ping-pong protocols*, Information and Control **55** (1982), no. 1-3, 57–68.
- [DLM04] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell, *Multiset rewriting and the complexity of bounded security protocols*, Journal of Computer Security **12** (2004), no. 2, 247–311.
- [DLMS99] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov, *Undecidability of bounded security protocols*, Workshop on Formal Methods and Security Protocols, 1999.
- [DR06] T. Dierks and E. Rescorla, *The transport layer security (TLS) protocol version 1.1*, Tech. Report RFC 4346, Internet Engineering Task Force, 2006.
- [DY81] Danny Dolev and Andrew Chi-Chih Yao, *On the security of public key protocols (extended abstract)*, FOCS, IEEE, 1981, pp. 350–357.
- [DY83] ———, *On the security of public key protocols*, IEEE Transactions on Information Theory **29** (1983), no. 2, 198–207.
- [EGS85] Shimon Even, Oded Goldreich, and Adi Shamir, *On the security of ping-pong protocols when implemented using the RSA*, CRYPTO, Lecture Notes in Computer Science, vol. 218, Springer, 1985, pp. 58–72.
- [EMM07] Santiago Escobar, Catherine Meadows, and José Meseguer, *Maude-NPA: cryptographic protocol analysis modulo equational properties*, FOSAD, Lecture Notes in Computer Science, vol. 5705, Springer, 2007, pp. 1–50.
- [EMS09] Santiago Escobar, José Meseguer, and Ralf Sasse, *Variant narrowing and equational unification*, Electr. Notes Theor. Comput. Sci. **238** (2009), no. 3, 103–119.
- [GT00] J.D. Guttman and F.J. Thayer, *Protocol independence through disjoint encryption*, CSFW, IEEE Computer Society, 2000, pp. 24–34.
- [HC<sup>+</sup>98] D. Harkins, D. Carrel, et al., *The internet key exchange (IKE)*, 1998.
- [HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, *PRISM: A tool for automatic verification of probabilistic systems*, Tools and Algorithms for the Construction and Analysis of Systems (2006), 441–444.
- [HL04] Philippa J. Hopcroft and Gavin Lowe, *Analysing a stream authentication protocol using model checking*, Int. J. Inf. Sec. **3** (2004), no. 1, 2–13.
- [Hol02] G.J. Holzmann, *The model checker SPIN*, Software Engineering, IEEE Transactions on **23** (2002), no. 5, 279–295.
- [JV96] M. Just and S. Vaudenay, *Authenticated multi-party key agreement*, ASIACRYPT 1996, Lecture Notes in Computer Science, vol. 1163, 1996, pp. 36–49.
- [KHNE10] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen, *Internet key exchange protocol version 2 (IKEV2)*, Tech. Report RFC 5996, Internet Engineering Task Force, September 2010.
- [KRS10] Steve Kremer, Mark Ryan, and Ben Smyth, *Election verifiability in electronic voting protocols*, ESORICS, Lecture Notes in Computer Science, vol. 6345, Springer, 2010, pp. 389–404.
- [KT08] Ralf Küsters and Tomasz Truderung, *Reducing protocol analysis with xor to the xor-free case in the horn theory based approach*, ACM Conference on Computer and Communications Security, ACM, 2008, pp. 129–138.
- [KT09] ———, *Using ProVerif to analyze protocols with Diffie-Hellman exponentiation*, CSF, IEEE Computer Society, 2009, pp. 157–171.
- [Lam94] L. Lamport, *The temporal logic of actions*, ACM Transactions on Programming Languages and Systems (TOPLAS) **16** (1994), no. 3, 872–923.

- [LMST05] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, *Automatic analysis of a non-repudiation protocol*, Electronic Notes in Theoretical Computer Science **112** (2005), 113–129.
- [Low96] Gavin Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using fdr*, Software - Concepts and Tools **17** (1996), no. 3, 93–102.
- [Low97] ———, *A hierarchy of authentication specifications*, CSFW, 1997, pp. 31–44.
- [Low98] ———, *Casper: A compiler for the analysis of security protocols*, Journal of Computer Security **6** (1998), no. 1-2, 53–84.
- [LR92] Dennis Longley and S. Rigby, *An automatic search for security flaws in key management schemes*, Computers & Security **11** (1992), no. 1, 75–89.
- [MCF87] Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman, *The Interrogator: Protocol security analysis*, IEEE Trans. Software Eng. **13** (1987), no. 2, 274–288.
- [Mea92] Catherine Meadows, *Applying formal methods to the analysis of a key management protocol*, Journal of Computer Security **1** (1992), no. 1, 5–36.
- [Mea96] ———, *The NRL Protocol Analyzer: An overview*, J. Log. Program. **26** (1996), no. 2, 113–131.
- [Mea99] ———, *Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer*, IEEE Symposium on Security and Privacy, 1999, pp. 216–231.
- [Mil99] Jonathan K. Millen, *A necessarily parallel attack*, In Workshop on Formal Methods and Security Protocols, 1999.
- [Mil03] Jonathan K. Millen, *On the freedom of decryption*, Inf. Process. Lett. **86** (2003), no. 6, 329–333.
- [MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern, *Automated analysis of cryptographic protocols using Mur-phi*, IEEE Symposium on Security and Privacy, IEEE Computer Society, 1997, pp. 141–151.
- [MPP<sup>+</sup>06] Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson, *Distance bounding protocols: authentication logic analysis and collusion attacks*, Secure Localization and Time Synchronization in Wireless Ad Hoc and Sensor Networks (R. Poovendran, C. Wang, and S. Roy, eds.), Springer Verlag, 2006.
- [MS01] Jonathan K. Millen and Vitaly Shmatikov, *Constraint solving for bounded-process cryptographic protocol analysis*, ACM Conference on Computer and Communications Security, 2001, pp. 166–175.
- [MSC04] Catherine Meadows, Paul F. Syverson, and Iliano Cervesato, *Formal specification and analysis of the Group Domain Of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer*, Journal of Computer Security **12** (2004), no. 6, 893–931.
- [MVB10] Sebastian Mödersheim, Luca Viganò, and David A. Basin, *Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols*, Journal of Computer Security **18** (2010), no. 4, 575–618.
- [MvOV96] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, CRC Press, October 1996.
- [NHR05] C. Neuman, S. Hartman, and K. Raeburn, *The Kerberos network authentication service (V5)*, Tech. Report RFC 4120, Internet Engineering Task Force, July 2005.

- [NS78] Roger M. Needham and Michael D. Schroeder, *Using encryption for authentication in large networks of computers*, Commun. ACM **21** (1978), no. 12, 993–999.
- [NS06] G. Norman and V. Shmatikov, *Analysis of probabilistic contract signing*, Journal of Computer Security **14** (2006), no. 6, 561–589.
- [PT02] Adrian Perrig and J. D. Tygar, *Secure broadcast communication in wired and wireless networks*, Kluwer Academic Publishers, Norwell, MA, USA, 2002.
- [RR98] M.K. Reiter and A.D. Rubin, *Crowds: Anonymity for web transactions*, ACM Transactions on Information and System Security (TISSEC) **1** (1998), no. 1, 66–92.
- [RT01] Michaël Rusinowitch and Mathieu Turuani, *Protocol insecurity with finite number of sessions is NP-complete*, CSFW, IEEE Computer Society, 2001, pp. 174–.
- [SD95] U. Stern and D. Dill, *Improved probabilistic verification by hash compaction*, Correct Hardware Design and Verification Methods (1995), 206–224.
- [Shm04] V. Shmatikov, *Probabilistic analysis of an anonymity system*, Journal of Computer Security **12** (2004), no. 3, 355–377.
- [Sho99] V. Shoup, *On formal models for secure key exchange (version 4)*, November 1999, revision of IBM Research Report RZ 3120 (April 1999).
- [SM96] Paul F. Syverson and Catherine Meadows, *A formal language for cryptographic protocol requirements*, Des. Codes Cryptography **7** (1996), no. 1-2, 27–59.
- [Son99] Dawn Xiaodong Song, *Athena: A new efficient automatic checker for security protocol analysis*, CSFW, 1999, pp. 192–202.
- [SS89] Manfred Schmidt-Schauß, *Unification in permutative theories is undecidable*, J. Symbolic Computation **8** (1989), 415–421.
- [SSBC09] Patrick Schaller, Benedikt Schmidt, David Basin, and Srdjan Capkun, *Modeling and verifying physical properties of security protocols for wireless networks*, 22nd IEEE Computer Security Foundations Symposium, IEEE Computer Society Washington, DC, USA, 2009, pp. 109–123.
- [TSG10] F. Javier Thayer, Vipin Swarup, and Joshua D. Guttman, *Metric strand spaces for locale authentication protocols*, IFIPTM, IFIP Conference Proceedings, vol. 321, Springer, 2010, pp. 79–94.
- [Tur06] Mathieu Turuani, *The CL-Atse protocol analyser*, RTA, Lecture Notes in Computer Science, vol. 4098, Springer, 2006, pp. 277–286.
- [Vig06] Luca Viganò, *Automated security protocol analysis with the AVISPA tool*, Electr. Notes Theor. Comput. Sci. **155** (2006), 61–86.