Chapter 22
Model Checking Security Protocols

David Basin, Cas Cremers, and Catherine Meadows

Abstract The formal analysis of security protocols is a prime example of a domain where model checking has been successfully applied. Although security protocols are typically small, analysis by hand is difficult as a protocol should work even when arbitrarily many runs are interleaved and in the presence of an adversary. Specialized model-checking techniques have been developed that address both the problems of unbounded, interleaved runs and a prolific, highly nondeterministic adversary. These techniques have been implemented in model-checking tools that now scale to protocols of realistic size and can be used to aid protocol design and standardization.

In this chapter, we provide an overview of the main applications of model checking in security protocol analysis. We explain the central concepts involved in the analysis of security protocols: the abstraction of messages, protocols as role automata, the adversary model, and property specification. We explain and relate the main algorithms used and describe systems based on them. We also give examples of the successful applications of model checking to protocol standards. Finally, we provide an outlook on the field: What is possible with the state of the art and what are the future challenges?

22.1 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 [44], IKEv2 [59], and Kerberos [83], which can be used, respectively, to secure web-

D. Basin
ETH Zürich, Zürich, Switzerland

C. Cremers (✉)
University of Oxford, Oxford, UK
e-mail: cas.cremers@cs.ox.ac.uk

C. Meadows
Naval Research Laboratory, Washington, DC, USA

E.M. Clarke et al. (eds.), *Handbook of Model Checking*,
DOI 10.1007/978-3-319-10575-8_22

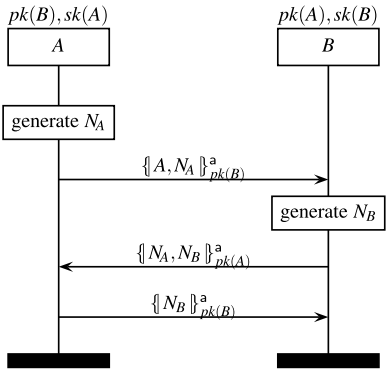


Fig. 1 Needham–Schroeder protocol (NS)

based traffic, set up 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, such as encryption and hash functions, are themselves secure.

To give an idea of what can go wrong, consider Lowe's often-cited attack [66] on the Needham–Schroeder public key protocol [82]. The goal of the protocol is to allow two parties to authenticate each other, i.e., after execution of the protocol they can be sure that they have been communicating with the intended partner. The protocol achieves this by combining two challenge–response interactions. Agents can execute the initiator role A or the responder role B. At the end of the protocol, the initiator and the responder agree on a pair of shared secrets, N_A and N_B, where N_A is a random number (or nonce) generated by the initiator and N_B is a nonce generated by the responder. The protocol relies on public-key encryption: anyone can send a message to an agent X using X's public key $pk(X)$, but only X can decrypt it, using its private key $sk(X)$. We write $\{\!|\, M \,|\!\}^{\mathsf{a}}_{pk(X)}$ to denote the (asymmetric) encryption of the message M with X's public key. A message sequence chart describing the protocol is shown in Fig. 1.

Let us go through the protocol steps and their rationale.

1. $A \to B : \{\!|\, A, N_A \,|\!\}^{\mathsf{a}}_{pk(B)}$
   The responder receives the initiator's message and decrypts it. At this point, the responder assumes that the initiator has indeed sent the message recently and will try to confirm his assumption in the next two steps. The responder generates a nonce N_B.
2. $B \to A : \{\!|\, N_A, N_B \,|\!\}^{\mathsf{a}}_{pk(A)}$
   When the initiator receives this message, she decrypts it. Because the message contains the nonce N_A, which the initiator generated recently and sent encrypted

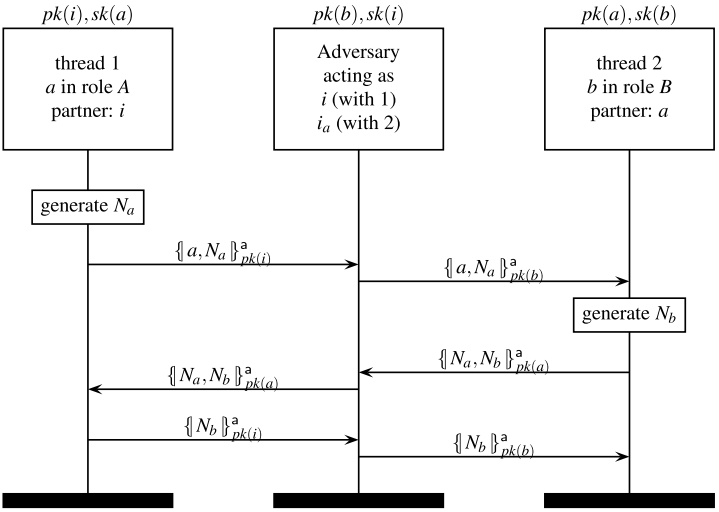


Fig. 2 Lowe's man-in-the-middle attack on the Needham–Schroeder protocol

for the responder, the initiator concludes that the message was indeed sent recently by the responder.

3. $A \to B : \{\!|\, N_B \,|\!\}^{\mathsf{a}}_{pk(B)}$
   The last message is sent so that the responder can verify his assumptions. He reasons that the message is recent, since N_B is recent. Moreover, only the initiator and the responder know N_B, and the responder did not send the message. So the message must come from the initiator. Finally, the initiator will have received N_B as part of the responder's message that also contained N_A, so she would not have responded unless N_A was also recently sent by the initiator.

Although this informal correctness argument may seem convincing, 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 Fig. 2.

The attack proceeds in the following way.

1. $a \to i : \{\!|\, a, N_a \,|\!\}^{\mathsf{a}}_{pk(i)}$
   The agent a initiates the protocol in the initiator role, aiming to communicate with i, and generates a fresh nonce N_a.
2. $i_a \to b : \{\!|\, a, N_a \,|\!\}^{\mathsf{a}}_{pk(b)}$
   The adversary i uses a's nonce to impersonate a and initiates an instance of the protocol with b, who executes the responder role.
3. $b \to a : \{\!|\, N_a, N_b \,|\!\}^{\mathsf{a}}_{pk(a)}$
   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. $a \to i : \{\!|\, N_b \,|\!\}^{\mathsf{a}}_{pk(i)}$
   a responds to i, following the rules of the initiator role of the protocol.

5. $i_a \rightarrow b : \{\!|\, N_b \,|\!\}^{\mathsf{a}}_{pk(b)}$
   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 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 Sect. 22.2 we give a brief historical overview of the field. Then, in Sect. 22.3, we describe a basic model for security protocols and their properties. In Sect. 22.4 we give an overview of several issues that arise in model checking security protocols, and approaches that have been taken to address them. In Sect. 22.5 we describe representative systems based on these approaches. We discuss current and future research questions in Sect. 22.6 and we draw conclusions in Sect. 22.7.

22.2 History

In this section, we give a brief history of model checking cryptographic protocols.

The first symbolic approach to cryptographic protocol analysis, and the basis of the methods used by current model-checking tools, was that of Dolev and Yao [46, 47], and shortly later, Dolev, Even, and Karp [45], 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 [52], 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 [78], followed by the Longley–Rigby search tool [65] and the NRL Protocol Analyzer (NPA) [71]. 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 [96], 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 [66], described above. The fact that he could demonstrate a problem that had

gone unnoticed for 17 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. [80] 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 [35].

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. Indeed it is possible to create protocols that are only vulnerable to attacks that require interleaving an arbitrarily large number of sessions [76]. In [48, 49], 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 [88] 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 [79], the Constraint-Logic-based Attack Searcher (CL-Atse) [98], the On-the-Fly Model Checker (OFMC) [20], and the SAT-based Model Checker SAT-MC [5]. The same period saw the development of unbounded-session model checkers relying on abstraction (such as ProVerif [29]) and heuristics (such as Maude-NPA [50] and Athena [94], the latter of which formed the basis for the current tool Scyther [42]). We present these tools in more detail in Sect. 22.5.

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 [4, 100], 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 [72] and the Group Domain of Interpretation (GDOI) protocol [74]. In the case of GDOI, the tool was instrumental in catching some vulnerabilities early on that were straightforward to fix at 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 [27] and the smart card protocol InfoCard [28]. Scyther has been used in the analysis of the MQV family

of protocols [16], 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 [17] and the IKE key exchange protocols in the IPsec standard [43].

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 for 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 indistinguishability, extending soundness results down to the code level, and handling probabilistic behavior. These topics will be discussed in further detail in Sect. 22.6.

22.3 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 symbolic approach presented in Sect. 22.2. In this section we present a basic symbolic model (based on [16]) 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.

22.3.1 Notational Preliminaries

Let f be a function. We write $dom(f)$ and $ran(f)$ to denote f's domain and range, respectively. We write $f[a \mapsto b]$ 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 $f : X \twoheadrightarrow Y$ to denote a partial function mapping some elements from X to elements from Y. We write f^n to denote the n-fold composition of f, for $n \geq 0$, where f^0 is the identity function.

For any set S, $\mathscr{P}(S)$ denotes the power set of S and S^* denotes the set of finite sequences of elements from S. We write $\langle s_0, \ldots, s_n \rangle$ (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., [7]) for manipulating terms. Let $\mathscr{S}ub$ denote the set of substitutions of terms for variables (we will define these syntactic categories shortly). We write $[t_0, \ldots, t_n/x_0, \ldots, x_n] \in \mathscr{S}ub$ to denote the substitution of t_i for x_i, for $0 \leq i \leq n$. We extend the functions dom and ran to substitutions. We write $\sigma \cup \sigma'$ to denote the union of two substitutions, which is defined when $dom(\sigma) \cap dom(\sigma') = \emptyset$. We write $\sigma(t)$ for the application of the substitution σ to t.

For R a binary relation, we write R^{-1} to denote the *inverse* of R, i.e., $R^{-1} = \{(y, x) \mid (x, y) \in R\}$. Furthermore, R^+ denotes the transitive closure of R.

22.3.2 Terms and Events

We assume we have 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 $t \sharp tid$. This denotes that the term t is local to the protocol role instance identified by the thread identifier tid, where $tid \in TID$.

Definition 1 Basic terms

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

Definition 2 Terms

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

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

Depending on the protocol analyzed, we assume that symmetric or asymmetric long-term keys have been distributed prior to protocol execution. We assume the existence of an inverse function on terms, where t^{-1} denotes the inverse key of t. We have $pk(X)^{-1} = sk(X)$, $sk(X)^{-1} = pk(X)$ for all $X \in Agent$, 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 $t_0, \ldots, t_n \in Term$ and let $f \in Func$. We define $\vdash$ as the smallest relation satisfying:

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

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

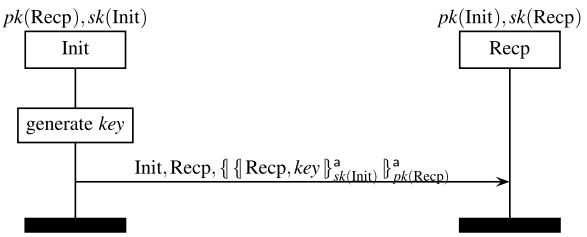
An agent can engage in the following events.

Definition 3 Events

$$Event ::= \mathsf{create}(Role, \mathscr{S}ub) \mid \mathsf{send}(Term) \mid \mathsf{recv}(Term)$$

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.

Fig. 3 Simple protocol



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., $\sigma(\mathsf{send}(m)) = \mathsf{send}(\sigma(m))$.

22.3.3 Protocols and Threads

A protocol is a mapping from role names to event sequences, i.e., $Protocol : Role \twoheadrightarrow Event^*$. 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 $\{\text{Init}, \text{Recp}\} \subseteq Role$, $key \in Fresh$, and $x \in Var$. Let P be the protocol defined as follows.

$$P(\text{Init}) = \langle \mathsf{send}(\text{Init}, \text{Recp}, \{\!|\, \{\!|\, \text{Recp}, key \,|\!\}^{\mathsf{a}}_{sk(\text{Init})} \,|\!\}^{\mathsf{a}}_{pk(\text{Recp})}) \rangle$$

$$P(\text{Recp}) = \langle \mathsf{recv}(\text{Init}, \text{Recp}, \{\!|\, \{\!|\, \text{Recp}, x \,|\!\}^{\mathsf{a}}_{sk(\text{Init})} \,|\!\}^{\mathsf{a}}_{pk(\text{Recp})}) \rangle$$

The message sequence chart for this protocol is shown in Fig. 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 $localize : TID \to \mathscr{S}ub$. 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

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

We define a function $thread : (Event^* \times TID \times \mathscr{S}ub) \to Event^*$ 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 σ be a substitution. Then

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

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 $localize(t_1)(key) = key\sharp t_1$ and

$$\begin{aligned} &thread\big(P(\text{Init}), t_1, [A, B/\text{Init}, \text{Recp}]\big) \\ &= \big\langle \mathsf{send}\big(A, B, \{\!|\, |\!\}^{\mathsf{a}}_{sk(A)} \{\!|\, B, key\sharp t_1 \,|\!\}^{\mathsf{a}}_{pk(B)}\big)\big\rangle. \end{aligned}$$

22.3.4 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

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

We define the initial adversary knowledge IK_0 as

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

22.3.5 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 $Trace \times \mathscr{P}(Term) \times (TID \nrightarrow Event^*)$. 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

$$\frac{R \in dom(P) \quad \sigma \in dom(P) \to Agent \quad tid \notin dom(th)}{(tr, IK, th) \longrightarrow (tr^\frown\langle(tid, \mathsf{create}(R, \sigma))\rangle, IK, th[tid \mapsto thread(P(R), tid, \sigma)])}\,[\mathsf{create}_P]$$

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

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

Fig. 4 Execution-model rules

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.[1]

The initial system state s_{init} is defined as

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

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 in Fig. 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, the substitution σ associates the role names $dom(P)$ with agents, and tid is a fresh thread identifier. In the state transition in the conclusion, the successor state's trace is extended, reflecting that the thread identified by tid executed the create event, and the thread mapping ("thread pool") is extended with the thread assigned to tid.

The send rule sends a message m to the network. The premise refers to a thread identified by 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 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 σ, 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 σ is applied to the remaining protocol steps l.

Definition 6 (Transition relation) Let P be a protocol. We define a transition relation $\to_P$ using the execution-model rules from Fig. 4. For states s and s', $s \to_P s'$

[1] 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.

iff there exists an execution-model rule with the premises $Q_1(s), \ldots, Q_n(s)$ and the conclusion $s \to s'$ such that all of the premises hold.

Given a protocol P and a set of states T, let Post_P and Pre_P denote the successors and predecessors of T, respectively, i.e.,

$$\mathrm{Post}_P(T) = \left\{ s' \in \mathit{State} \;\middle|\; \exists s \in T \,.\, s \to_P s' \right\}$$
$$\mathrm{Pre}_P(T) = \left\{ s \in \mathit{State} \;\middle|\; \exists s' \in T \,.\, s \to_P s' \right\}.$$

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

$$\mathit{Reachable}(P) = \bigcup_{n=0}^{\infty} \mathrm{Post}_P^n\left(\{s_{\mathrm{init}}\}\right).$$

We will see Pre and Post again when we discuss model-checking algorithms in Sect. 22.4.

22.3.6 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 σ a substitution. We write $\mathrm{HT}(s, tid, \sigma)$ to denote

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

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

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

$$\forall tid, \sigma \,.\, \mathrm{HT}(s, tid, \sigma) \Rightarrow \neg\big(IK \vdash (t \sharp tid)\big) \,.$$

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 Fig. 3 ensures secrecy of the initiator's key. In contrast, the Needham–Schroeder protocol from Fig. 1 does not ensure secrecy of the nonces.

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

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

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

For example, the Needham–Schroeder protocol from Fig. 1 satisfies weak aliveness.

Stronger forms of authentication (see, e.g., [67]) 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.

22.3.7 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 Sect. 22.6 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 fully model protocols that support multiple options and subprotocols, such as the Internet Key Exchange (IKE) [54], or that require loops, such as the stream authentication protocol TESLA [57, 86]. 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 Sect. 22.5.

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 he obtains a decryption key may be unrealistic in some scenarios. We will consider alternative adversary models in Sect. 22.6.

22.4 Issues in Developing Model-Checking Algorithms for Security Protocols

Here we present a number of issues that arise in model checking security protocols, and the approaches that have been taken to address them. In particular, we indicate various design decisions, such as forward or backward search, state representations, and bounding the state space.

22.4.1 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

$$Reachable(P) \subseteq S. \tag{1}$$

Let $\overline{S} = State \setminus S$ be the property's complement, representing possible attacks. For example, for the secrecy of a term t as in Definition 9, $\overline{S}$ is defined as:

$$\left\{ s \in State \;\middle|\; \exists tid, \sigma \,.\, \mathrm{HT}(s, tid, \sigma) \wedge IK \vdash (t \sharp tid) \right\}.$$

Using the complement $\overline{S}$, Formula (1) can be rewritten as

$$Reachable(P) \cap \overline{S} = \emptyset. \tag{2}$$

Then, we can rewrite Formula (2) either as

$$\left(\bigcup_{n=0}^{\infty} \mathrm{Post}_P^n \left(\{ s_{\mathrm{init}} \} \right) \right) \cap \overline{S} = \emptyset \tag{3}$$

or alternatively as

$$s_{\text{init}} \notin \bigcup_{n=0}^{\infty} \text{Pre}_P^n(\overline{S}). \tag{4}$$

Algorithms that iteratively compute a representation of (all, or some subset of) $\text{Post}_P^n(\{s_{\text{init}}\})$, as in Formula (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 (4). Similar observations hold as for forward search, except that we check whether s_{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 Pre_P or 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 $\overline{S}$ of the security property 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 $\overline{S}$, from which $\text{Pre}_P^n(\overline{S})$ can be computed. Depending on the property and the (finite) size of the set of states, the size of (the representation of) $\overline{S}$ can be significant. In practice, forward search is commonly used to explore finite sets of states.

22.4.2 Bounded Instances

The execution model presented in Sect. 22.3 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 Sect. 22.2, even relatively simple properties such as secrecy are undecidable for security protocols formalized using operational semantics similar to ours [48]. If we restrict the number of sessions (or threads), then the problem becomes NP-complete [88], 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, then 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 [76].

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 [88] 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.

22.4.3 Representing States

Formulas (3) and (4) can be directly used for forward or backward 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 Sect. 22.3 into an explicit-state model checker. As defined in Sect. 22.3.5, 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 [95]. This tool uses techniques inspired by explicit-state model checkers like SPIN [56], 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 [68], 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. Many tools take this approach. Here terms, in particular messages, are represented by non-ground terms (cf. *message patterns* in Sect. 22.3.5) 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 Sect. 22.3.2. This problem is often decidable, which can be shown by using the notion of locality [18, 38] to bound the size of terms occurring in derivations. During symbolic reasoning, the non-ground problem arises: determining whether there exists a substitution σ such that $\sigma(IK) \vdash \sigma(pt)$. The ground problem can be tackled by considering a restricted class of normal form derivations [38]. 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 Formula (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 Chap. 9 of this handbook [69]. 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 [6].

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 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, Scyther, and Tamarin.

22.4.4 Partial-Order Reduction

Partial-order reduction, as discussed in Chap. 6 of this handbook [85], 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 σ_1, σ_2, such that for all $i \in \{1, 2\}$,

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

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

$$s \to_P^* s_1, \quad \text{and} \quad s_1 = \big(tr^\frown\langle(tid_1, \sigma_1(e_1))\rangle^\frown\langle(tid_2, \sigma_2(e_2))\rangle, IK, th'\big),$$
$$s \to_P^* s_2, \quad \text{and} \quad s_2 = \big(tr^\frown\langle(tid_2, \sigma_2(e_2))\rangle^\frown\langle(tid_1, \sigma_1(e_1))\rangle, IK, th'\big),$$

and where $th' = th[tid_1 \mapsto \sigma_1(l_1)][tid_2 \mapsto \sigma_2(l_2)]$. 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 $i \in \{1, 2\}$,

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

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 in which 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 scheme.

POR techniques are used in several tools. Examples include Brutus [35], OFMC [20], and Maude-NPA [50].

22.4.5 Handling Equations

The formal model introduced in Sect. 22.3 uses a free algebra to represent operations on data. Thus the adversary's ability to perform encryption is represented by the deduction rule stating that $\{\!|\, t_1 \,|\!\}^{\mathsf{a}}_{t_2}$ may be deduced from t_1 and t_2, and his ability to perform decryption is represented by the deduction rule stating that t_1 may be deduced from $\{\!|\, t_1 \,|\!\}^{\mathsf{a}}_{t_2}$ 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

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

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 [77] and also by Küsters and Truderung, who develop inference rules for the theory governing exclusive-or [61] and a subtheory of the theory governing Diffie–Hellman exponentiation [62]. 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. Researchers have concentrated on two main techniques. 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 and Turuani's proof of decidability

of security in the bounded-session model. Decision procedures have subsequently been given for a large class of equational theories relevant for model checking cryptographic protocols [1]. 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 [23, 34, 39].

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 σ 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* [51], which can be applied to a class of theories that satisfy a property known as the *finite variant property* [37]. 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 Sect. 22.5.1.

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 [91], 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. [60] 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 [77]. 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. [25], but there is still much to be learned.

22.5 Systems and Algorithms

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

22.5.1 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 [74]. 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 [72], the Group Domain of Interpretation Protocol [74], and the Simmons Selective Broadcast Protocol [70].

Maude-NPA is a descendant of NPA, implemented in the Maude rewriting language [36]. It shares many of NPA's features, including the use of unification to

implement backward search and 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 σ 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 [20], 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 the search proceeds no further on this term (that is, it is removed from the state), since the adversary should be able to find it using arbitrary terms.[2] 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 ongoing on incorporating more general homomorphic encryption.

22.5.2 AVISPA and Related Tools

The AVISPA tool [4] is a 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, and 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 [63] and an operational semantics defined in terms of

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

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 Sect. 22.4.3 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 [8] with modules for xor, exponentiation, and associative pairing.

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), generating arbitrary prefixes of the data on demand. In [14], 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 [81]. 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 computes 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 [19].

22.5.3 Athena, Scyther, and Tamarin

The Athena [94] and Scyther [42] algorithms implement model checking with respect to the unbounded model described in Sect. 22.3.5 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.

$$
\begin{aligned}
e_1 &= \big(tid, \mathsf{recv}\big(a, b, \{\!|\, a, X \,|\!\}^{\mathsf{a}}_{pk(b)}\big)\big)\\
e_2 &= \big(tid, \mathsf{send}\big(b, a, \{\!|\, X, N_B \sharp tid \,|\!\}^{\mathsf{a}}_{pk(a)}\big)\big)\\
e_3 &= \big(tid, \mathsf{recv}\big(a, b, \{\!|\, N_B \sharp tid \,|\!\}^{\mathsf{a}}_{pk(b)}\big)\big)\\
e_4 &= \mathsf{AdversaryKnows}(N_B \sharp tid)\\
PT &= \big(\{e_1, e_2, e_3, e_4\}, \big\{(e_1, e_2), (e_2, e_3)\big\}^+\big)
\end{aligned}
$$

This pattern represents an infinite set of traces of the Needham–Schroeder protocol, e.g., the attack from Fig. 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.

The Tamarin prover [90] is a generalization of the algorithms underlying Athena and Scyther. Tamarin uses a backward search that can handle more expressive protocol and property specifications. Protocols and adversary capabilities can be specified using multiset rewriting rules, allowing the specification of protocols with branching and loops. Tamarin provides support for Diffie–Hellman exponentiations and a class of user-defined equational theories. Protocols can be analyzed with respect to properties specified in a guarded fragment of first-order logic that supports quantification over timepoints.

22.5.4 ProVerif

The ProVerif tool [29] 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 overapproximation 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 [30]. These properties express that when event e_1 occurs, event e_2 must have occurred earlier with related parameters ρ_1 and ρ_2. For example ρ_1 could be a nonce and ρ_2 a variable that is supposed to be instantiated with ρ_1. In this case, it does not suffice to prove that the values of ρ_1 and ρ_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 ρ_1 and ρ_2, we increase the precision of the verification at the cost of efficiency and termination.

22.6 Research Problems

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

22.6.1 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 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 [41].

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. [11], 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 [10] and exclusive-or [9]. This is perhaps not surprising, given the divergence between the two models and the difficulty of bringing the two together. Hence this is still an active area of research.

22.6.2 Corruption Models

In the basic Dolev–Yao model described in Sect. 22.3, 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. [33, 58, 93], 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 [75]: 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 [15, 16], 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., [3, 53],

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.

22.6.3 Channel Properties

The standard Dolev–Yao model gives the adversary control over all communication channels. The adversary sees all communications, 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. However, a growing number of cryptographic protocols either use 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 [32] and secure localization protocols [99], 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 [13], such as a human reading a sequence of numbers off a computer screen, to bootstrap key distribution in the absence of a public key infrastructure. Work has been ongoing on developing methods for formal analysis of protocols that use these channels, e.g., [21, 73, 89, 97], 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.

22.6.4 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 labeled "high" should have no discernible effect on events labeled "low". Non-interference is closely related to the cryptographic notion of indistinguishability, mentioned in Sect. 22.6.1. 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 [31].

Another property related to indistinguishability is *static equivalence* [2]. 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 [26], 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 further communication. As shown in [22, 40], 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 [24], KISS [34], and FAST [39] tools. ProVerif also supports the analysis of off-line guessing attacks, but based on a different formalization of guessing due to [40].

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.

22.6.5 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 Chap. 28 of this handbook [12]. Probabilistic model checking has been successfully applied to different kinds of security protocols. Examples include protocols for anonymity [92], non-repudiation [64], and contract signing [84].

As an example, in [92], Shmatikov models the protocol underlying Crowds [87], 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 required anonymity properties of the system in the probabilistic temporal logic PCTL. Given this model, he can use the PRISM model checker [55] 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.

22.7 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 has infinitely many states and the active adversary results in considerable nondeterminism. Addressing these challenges has motivated a number of specialized model-checking techniques.

State of the art methods and tools are able to handle classical authentication and key-exchange protocols of realistic, but limited, complexity. Abstract models of large protocol suites such as the Internet Key Exchange protocol have been analyzed. However, an automatic analysis of the full protocol suite, with all its variations, Diffie–Hellman equational reasoning, and a full model of its branching and looping behaviors, would lead to a state-space explosion and is beyond the state of the art. A state-space 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 reasoning using abstraction is required.

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, may play an important role in enabling new classes of applications.

References

1. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. **367**(1–2), 2–32 (2006)
2. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Hankin, C., Schmidt, D. (eds.) Symp. on Principles of Programming Languages (POPL), pp. 104–115. ACM, New York (2001)
3. Andova, S., Cremers, C., Gjøsteen, K., Mauw, S., Mjølsnes, S.F., Radomirović, S.: A framework for compositional verification of security protocols. Inf. Comput. **206**, 425–459 (2008)
4. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
5. Armando, A., Compagna, L.: SATMC: a SAT-based model checker for security protocols. In: Alferes, J.J., Leite, J.A. (eds.) Logics in Artificial Intelligence—Journées Européennes sur la Logique en Intelligence Artificielle (JELIA). LNCS, vol. 3229, pp. 730–733. Springer, Heidelberg (2004)
6. Armando, A., Compagna, L.: SAT-based model-checking for security protocols analysis. Int. J. Inf. Secur. **7**(1), 3–32 (2008)
7. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
8. Baader, F., Schulz, K.U.: Unification in the union of disjoint equational theories: combining decision procedures. J. Symb. Comput. **21**(2), 211–243 (1996)
9. Backes, M., Pfitzmann, B.: Limits of the cryptographic realization of Dolev–Yao-style XOR. In: di Vimercati, S.D.C., Syverson, P.F., Gollmann, D. (eds.) European Conf. on Research in Computer Security (ESORICS). LNCS, vol. 3679, pp. 178–196. Springer, Heidelberg (2005)
10. Backes, M., Pfitzmann, B., Waidner, M.: Limits of the BRSIM/UC soundness of Dolev–Yao models with hashes. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) European Conf. on Research in Computer Security (ESORICS). LNCS, vol. 4189, pp. 404–423. Springer, Heidelberg (2006)
11. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Inf. Comput. **205**(12), 1685–1720 (2007)
12. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
13. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: authentication in ad-hoc wireless networks. In: Network and Distributed System Security Symp. (NDSS) (2002)
14. Basin, D.: Lazy infinite-state analysis of security protocols. In: Secure Networking—CQRE [Secure] '99. LNCS, vol. 1740, pp. 30–42. Springer, Heidelberg (1999)
15. Basin, D., Cremers, C.: Degrees of security: protocol guarantees in the face of compromising adversaries. In: Dawar, A., Veith, H. (eds.) Intl. Workshop Computer Science Logic (CSL). LNCS, vol. 6247, pp. 1–18. Springer, Heidelberg (2010)
16. Basin, D., Cremers, C.: Modeling and analyzing security in the presence of compromising adversaries. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) European Conf. on Research in Computer Security (ESORICS). LNCS, vol. 6345, pp. 340–356. Springer, Heidelberg (2010)
17. Basin, D., Cremers, C., Meier, S.: Provably repairing the ISO/IEC 9798 standard for entity authentication. In: Degano, P., Guttman, J.D. (eds.) Intl. Conf. Principles of Security and Trust (POST). LNCS, vol. 7215, pp. 129–148. Springer, Heidelberg (2012)
18. Basin, D., Ganzinger, H.: Automated complexity analysis based on ordered resolution. J. Assoc. Comput. Mach. **48**(1), 70–109 (2001)

19. Basin, D., Mödersheim, S., Viganò, L.: Algebraic intruder deductions. In: Sutcliffe, G., Voronkov, A. (eds.) Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Lecture Notes in Artificial Intelligence, vol. 3835, pp. 549–564. Springer, Heidelberg (2005)
20. Basin, D., Mödersheim, S., Viganò, L.: OFMC: a symbolic model checker for security protocols. Int. J. Inf. Secur. **4**(3), 181–208 (2005)
21. Basin, D., Čapkun, S., Schaller, P., Schmidt, B.: Let's get physical: models and methods for real-world security protocols. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Intl. Conf. on Theorem Proving in Higher Order Logics (TPHOLs). LNCS, vol. 5674, pp. 1–22. Springer, Heidelberg (2009). Invited paper
22. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: Atluri, V., Meadows, C., Juels, A. (eds.) ACM Conf. on Computer and Communications Security (CCS), pp. 16–25. ACM, New York (2005)
23. Baudet, M., Cortier, V., Delaune, S.: YAPA: a generic tool for computing intruder knowledge. In: Treinen, R. (ed.) Intl. Conf. Rewriting Techniques and Applications (RTA). LNCS, vol. 5595, pp. 148–163. Springer, Heidelberg (2009)
24. Baudet, M., Cortier, V., Delaune, S.: YAPA: a generic tool for computing intruder knowledge. In: Treinen, R. (ed.) Intl. Conf. Rewriting Techniques and Applications (RTA), pp. 148–163. Springer, Heidelberg (2009)
25. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. Inf. Comput. **207**(4), 496–520 (2009)
26. Baudet, M., Warinschi, B., Abadi, M.: Guessing attacks and the computational soundness of static equivalence. J. Comput. Secur. **18**(5), 909–968 (2010)
27. Bhargavan, K., Fournet, C., Corin, R., Zalinescu, E.: Cryptographically verified implementations for TLS. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conf. on Computer and Communications Security (CCS), pp. 459–468. ACM, New York (2008)
28. Bhargavan, K., Fournet, C., Gordon, A.D., Swamy, N.: Verified implementations of the information card federated identity-management protocol. In: Abe, M., Gligor, V.D. (eds.) Symp. on Information, Computer and Communications Security (ASIACCS), pp. 123–135. ACM, New York (2008)
29. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Computer Security Foundations Workshop (CSFW), pp. 82–96. IEEE, Piscataway (2001)
30. Blanchet, B.: Automatic verification of correspondences for security protocols. J. Comput. Secur. **17**(4), 363–434 (2009)
31. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. In: Symp. on Logic in Computer Science (LICS), pp. 331–340. IEEE, Piscataway (2005)
32. Brands, S., Chaum, D.: Distance-bounding protocols. In: Helleseth, T. (ed.) Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT '93), pp. 344–359. Springer, Heidelberg (1994)
33. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) Intl. Conf. on the Theory and Application of Cryptographic Techniques (EUROCRYPT). LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
34. Ciobâca, S., Delaune, S., Kremer, S.: Computing knowledge in security protocols under convergent equational theories. In: Schmidt, R.A. (ed.) Intl. Conf. on Automated Deduction (CADE), pp. 355–370. Springer, Heidelberg (2009)
35. Clarke, E.M., Jha, S., Marrero, W.R.: Verifying security protocols with Brutus. Trans. Softw. Eng. Methodol. **9**(4), 443–487 (2000)
36. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude—A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. LNCS, vol. 4350. Springer, Heidelberg (2007)

37. Comon-Lundh, H., Delaune, S.: The finite variant property: how to get rid of some algebraic properties. In: Giesl, J. (ed.) Intl. Conf. Rewriting Techniques and Applications (RTA). LNCS, vol. 3467, pp. 294–307. Springer, Heidelberg (2005)
38. Comon-Lundh, H., Treinen, R.: Easy intruder deductions. In: Dershowitz, N. (ed.) Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday. LNCS, vol. 2772, pp. 225–242. Springer, Heidelberg (2003)
39. Conchinha, B., Basin, D., Caleiro, C.: Efficient decision procedures for message deducibility and static equivalence. In: Degano, P., Etalle, S., Guttman, J.D. (eds.) Intl. Workshop on Formal Aspects of Security and Trust (FAST 2010). LNCS, vol. 6561, pp. 34–49 (2011)
40. Corin, R., Doumen, J., Etalle, S.: Analysing password protocol security against off-line dictionary attacks. Electron. Notes Theor. Comput. Sci. **121**, 47–63 (2005)
41. Cortier, V., Kremer, S., Warinschi, B.: A survey of symbolic methods in computational analysis of cryptographic systems. J. Autom. Reason. **46**(3–4), 225–259 (2011)
42. Cremers, C.: The Scyther tool: verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
43. Cremers, C.: Key exchange in IPsec revisited: formal analysis of IKEv1 and IKEv2. In: Atluri, V., Díaz, C. (eds.) European Conf. on Research in Computer Security (ESORICS), pp. 315–334. Springer, Heidelberg (2011)
44. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.1. Tech. Rep. RFC 4346, Internet Engineering Task Force (2006)
45. Dolev, D., Even, S., Karp, R.M.: On the security of ping-pong protocols. Inf. Control **55**(1–3), 57–68 (1982)
46. Dolev, D., Yao, A.C.C.: On the security of public key protocols (extended abstract). In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 350–357. IEEE, Piscataway (1981)
47. Dolev, D., Yao, A.C.C.: On the security of public key protocols. Trans. Inf. Theory **29**(2), 198–207 (1983)
48. Durgin, N.A., Lincoln, P., Mitchell, J.C.: Multiset rewriting and the complexity of bounded security protocols. J. Comput. Secur. **12**(2), 247–311 (2004)
49. Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: Undecidability of bounded security protocols. In: Workshop on Formal Methods and Security Protocols (1999)
50. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) Foundations of Security Analysis and Design (FOSAD), 2007/2008/2009 Tutorial Lectures. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2007)
51. Escobar, S., Meseguer, J., Sasse, R.: Variant narrowing and equational unification. Electron. Notes Theor. Comput. Sci. **238**(3), 103–119 (2009)
52. Even, S., Goldreich, O., Shamir, A.: On the security of ping-pong protocols when implemented using the RSA. In: Williams, H.C. (ed.) Intl. Cryptology Conf. (CRYPTO). LNCS, vol. 218, pp. 58–72. Springer, Heidelberg (1985)
53. Guttman, J., Thayer, F.: Protocol independence through disjoint encryption. In: Computer Security Foundations Workshop (CSFW), pp. 24–34. IEEE, Piscataway (2000)
54. Harkins, D., Carrel, D., et al.: The internet key exchange (IKE) (1998)
55. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), vol. 3920, pp. 441–444 (2006)
56. Holzmann, G.: The model checker SPIN. Trans. Softw. Eng. **23**(5), 279–295 (2002)
57. Hopcroft, P.J., Lowe, G.: Analysing a stream authentication protocol using model checking. Int. J. Inf. Secur. **3**(1), 2–13 (2004)
58. Just, M., Vaudenay, S.: Authenticated multi-party key agreement. In: Kim, K., Matsumoto, T. (eds.) Advances in Cryptology (ASIACRYPT '96). LNCS, vol. 1163, pp. 36–49 (1996)
59. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P.: Internet key exchange protocol version 2 (IKEV2). Tech. Rep. RFC 5996, Internet Engineering Task Force (September 2010)

60. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) European Conf. on Research in Computer Security (ESORICS). LNCS, vol. 6345, pp. 389–404. Springer, Heidelberg (2010)
61. Küsters, R., Truderung, T.: Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conf. on Computer and Communications Security (CCS), pp. 129–138. ACM, New York (2008)
62. Küsters, R., Truderung, T.: Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In: Computer Security Foundations Symp. (CSF), pp. 157–171. IEEE, Piscataway (2009)
63. Lamport, L.: The temporal logic of actions. Trans. Program. Lang. Syst. **16**(3), 872–923 (1994)
64. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Automatic analysis of a non-repudiation protocol. Electron. Notes Theor. Comput. Sci. **112**, 113–129 (2005)
65. Longley, D., Rigby, S.: An automatic search for security flaws in key management schemes. Comput. Secur. **11**(1), 75–89 (1992)
66. Lowe, G.: Breaking and fixing the Needham–Schroeder public-key protocol using FDR. Softw., Concepts Tools **17**(3), 93–102 (1996)
67. Lowe, G.: A hierarchy of authentication specifications. In: Computer Security Foundations Workshop (CSFW), pp. 31–44 (1997)
68. Lowe, G.: Casper: a compiler for the analysis of security protocols. J. Comput. Secur. **6**(1–2), 53–84 (1998)
69. Marques-Silva, J., Malik, S.: Propositional SAT solving. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
70. Meadows, C.: Applying formal methods to the analysis of a key management protocol. J. Comput. Secur. **1**(1), 5–36 (1992)
71. Meadows, C.: The NRL Protocol Analyzer: an overview. J. Log. Program. **26**(2), 113–131 (1996)
72. Meadows, C.: Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In: Symp. on Security and Privacy (S & P), pp. 216–231. (1999)
73. Meadows, C., Poovendran, R., Pavlovic, D., Chang, L., Syverson, P.: Distance bounding protocols: authentication logic analysis and collusion attacks. In: Poovendran, R., Wang, C., Roy, S. (eds.) Secure Localization and Time Synchronization in Wireless Ad Hoc and Sensor Networks. Springer, Heidelberg (2006)
74. Meadows, C., Syverson, P.F., Cervesato, I.: Formal specification and analysis of the Group Domain of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer. J. Comput. Secur. **12**(6), 893–931 (2004)
75. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
76. Millen, J.K.: A necessarily parallel attack. In: Workshop on Formal Methods and Security Protocols (1999)
77. Millen, J.K.: On the freedom of decryption. Inf. Process. Lett. **86**(6), 329–333 (2003)
78. Millen, J.K., Clark, S.C., Freedman, S.B.: The Interrogator: protocol security analysis. Trans. Softw. Eng. **13**(2), 274–288 (1987)
79. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: Reiter, M.K., Samarati, P. (eds.) ACM Conf. on Computer and Communications Security (CCS), pp. 166–175 (2001)
80. Mitchell, J.C., Mitchell, M., Stern, U.: Automated analysis of cryptographic protocols using Murphi. In: Symp. on Security and Privacy (S & P), pp. 141–151. IEEE, Piscataway (1997)
81. Mödersheim, S., Viganò, L., Basin, D.: Constraint differentiation: search-space reduction for the constraint-based analysis of security protocols. J. Comput. Secur. **18**(4), 575–618 (2010)
82. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978)
83. Neuman, C., Hartman, S., Raeburn, K.: The Kerberos network authentication service (V5). Tech. Rep. RFC 4120, Internet Engineering Task Force (July 2005)

84. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. J. Comput. Secur. **14**(6), 561–589 (2006)
85. Peled, D.: Partial-order reduction. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
86. Perrig, A., Tygar, J.D.: Secure Broadcast Communication in Wired and Wireless Networks. Kluwer Academic, Norwell (2002)
87. Reiter, M., Rubin, A.: Crowds: anonymity for web transactions. Trans. Inf. Syst. Secur. **1**(1), 66–92 (1998)
88. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: Computer Security Foundations Workshop (CSFW), p. 174. IEEE, Piscataway (2001)
89. Schaller, P., Schmidt, B., Basin, D., Čapkun, S.: Modeling and verifying physical properties of security protocols for wireless networks. In: Computer Security Foundations Symp. (CSF), pp. 109–123. IEEE, Piscataway (2009)
90. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Computer Security Foundations Symp. (CSF), pp. 78–94 (2012)
91. Schmidt-Schauß, M.: Unification in permutative theories is undecidable. J. Symb. Comput. **8**, 415–421 (1989)
92. Shmatikov, V.: Probabilistic analysis of an anonymity system. J. Comput. Secur. **12**(3), 355–377 (2004)
93. Shoup, V.: On formal models for secure key exchange (version 4) (1999). Revision of IBM Research Report RZ 3120, (April 1999)
94. Song, D.X.A.: A new efficient automatic checker for security protocol analysis. In: Computer Security Foundations Workshop (CSFW), pp. 192–202 (1999)
95. Stern, U., Dill, D.L.: Improved probabilistic verification by hash compaction. In: Camurati, P.E., Eveking, H. (eds.) Correct Hardware Design and Verification Methods. LNCS, vol. 987, pp. 206–224. Springer, Heidelberg (1995)
96. Syverson, P.F., Meadows, C.: A formal language for cryptographic protocol requirements. Des. Codes Cryptogr. **7**(1–2), 27–59 (1996)
97. Thayer, F.J., Swarup, V., Guttman, J.D.: Metric strand spaces for locale authentication protocols. In: Nishigaki, M., Jøsang, A., Murayama, Y., Marsh, S. (eds.) IFIP International Conference on Trust Management (IFIPTM), vol. 321, pp. 79–94. Springer, Heidelberg (2010)
98. Turuani, M.: The CL-Atse protocol analyser. In: Pfenning, F. (ed.) Intl. Conf. Rewriting Techniques and Applications (RTA). LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)
99. Čapkun, S., Hubaux, J.: Secure positioning in wireless networks. J. Sel. Areas Commun. **24**(2), 221–232 (2006)
100. Viganò, L.: Automated security protocol analysis with the AVISPA tool. Electron. Notes Theor. Comput. Sci. **155**, 61–86 (2006)