Role based specification and security analysis of cryptographic protocols using asynchronous product automata

GMD Report 151

Sigrid Gürgens Peter Ochsenschläger
Carsten Rudolph
Fraunhofer - Institute for Secure Telecooperation SIT

March 8, 2002

Abstract

Cryptographic protocols are formally specified as a system of protocol agents using asynchronous product automata (APA). APA are a universal and very flexible operational description concept for communicating automata. Their specification, analysis and verification is supported by the SH-verification tool (SHVT). The local state of each agent is structured in several components describing its knowledge of keys, its "view" of the protocol and the goals to be reached within the protocol. Communication is modelled by adding messages to and removing them from a shared state component Network. Cryptography is modelled by symbolic functions with certain properties. In addition to the regular protocol agents an intruder is specified, which has no access to the agents' local states but to Network. The intruder may intercept messages and create new ones based on his initial knowledge and on what he can extract from intercepted messages. Violations of the security goals can be found by state space analysis performed by the SHVT. The method is demonstrated using the symmetric Needham-Schroeder protocol, and an attack is presented that does not involve compromised session keys. Our approach defers from others in that protocol specifications do not use implicit assumptions, thus protocol security does not depend on whether some implicit assumptions made are reasonable for a particular environment. Therefore, our protocol specifications explicitly provide relevant information for secure implementations.

Keywords: Cryptographic protocols, role based specification, security analysis, model checking, formal specification, asynchronous product automata

Part of this work is based on results of the project Valikrypt being funded by the German Bundesamt für Sicherheit in der Informationstechnik (BSI), Referat II 2.4.

Contents

1 Introduction

The principal goal of cryptographic protocols is to provide certain security services. If a protocol is not designed correctly it may fail to provide the required security service even if the underlying cryptographic algorithms are secure. Numerous cases exist where protocol flaws have been very hard to detect. Consequently, the need for formal analysis of cryptographic protocols is widely accepted and various different techniques including logics of authentication and model-checking have been proposed and successfully applied to find security flaws.

However, the security analysis of cryptographic protocols is mostly carried out in idealised models of the computational environment and detached from the development of systems the protocols are supposed to be used in. Protocol specifications used for implementations are often not completely formal. Protocols are specified as lists of messages to be exchanged using certain cryptographic operations. Internal actions of protocol agents, verification of messages and message parts on receipt, or assumptions about underlying cryptographic algorithms are not formally specified. Therefore, the security of the protocol depends on information that is either given as informal textual description or implicitly assumed.

For example, many formal models for cryptographic protocols assume that all messages are strongly typed and that in careful implementations these types are always checked (e.g. [4], [9]). However, in real applications, attacks based on type confusion cannot always be neglected. Smartcard applications for example allow to use message structures like ASN.1 but often use implicitly defined structures. This is the case for the authentication protocol of the German DIN standard for digital signature smartcards [6]. Here the structure of the messages is fixed and the specification of the algorithm includes the specification of the message structure. Other smartcard applications use so-called *headers* to define the message structure. The header specifies the type and length of each message part and the message is interpreted accordingly. In these cases the messages itself do not carry any type information.

We propose a formal model for cryptographic protocols that provides sufficient flexibility for detailed protocol specifications. In particular, all nuances of typing from assuming that no types are checked up to strong typing can be specified. In fact, in our model, all message checks on receipt have to be explicitly specified.

By inserting a powerful intruder into the model, automated finite state analysis using the SH verification tool [8] can be efficiently carried out.

In this paper we demonstrate our approach on the well known symmetric Needham-Schroeder protocol and present an attack on this protocol that does not involve compromised session keys and that was first published in [3]. The rest of the paper is organized as follows. In the next section we give a general definition of APA and introduce our APA model for protocols. In particular, we specify the crucial step of the symmetric Needham-Schroeder protocol that allows the arity attack. In section 3 we extend the model to include an intruder, we define its abilities, we specify a concrete scenario for the Needham-Schroeder protocol, and finally we explain the arity attack. Section 4 gives our conclusions.

2 Asynchronous Product Automata

We model a system of protocol agents using asynchronous product automata (APA). APA are a universal and very flexible operational description concept for cooperating systems [8]. It "naturally" emerges from formal language theory [7]. APA are supported by the SH-verification tool that provides components for the complete cycle from formal specification to exhaustive analysis and verification [8].

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are "glued" by shared components of their state sets. Elementary automata can "communicate" by changing the content of shared state components.

We view protocols as cooperating systems, thus APA are an adequate means for protocol formalization. Figure 1 shows the structure of an asynchronous product automaton modelling a system of three protocol agents A, B and S. The circles represent state components and boxes are elementary automata. Each agent P taking part in the protocol is modelled by one elementary automaton P that performs the agent's actions, and four state components Symkeys_P, Asymkeys_P, State_P, and Goals_P to store the symmetric and asymmetric keys of P, P's local state and the security goals P should reach within the protocol, respectively. The only state component shared between all agents (all elementary automata) is the component Network, which is used for communication. A message is sent by adding it to the content of Network and received by removing it from Network. The neighbourhood relation N (graphically represented by an arc) indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of the elementary automaton. For example, automaton A may change State_A and Network but cannot read or change the state of State_B. The figure shows the structure of the automaton. The full specification of the automaton includes the state sets (the data types), transition relations of the elementary automata and the initial state.

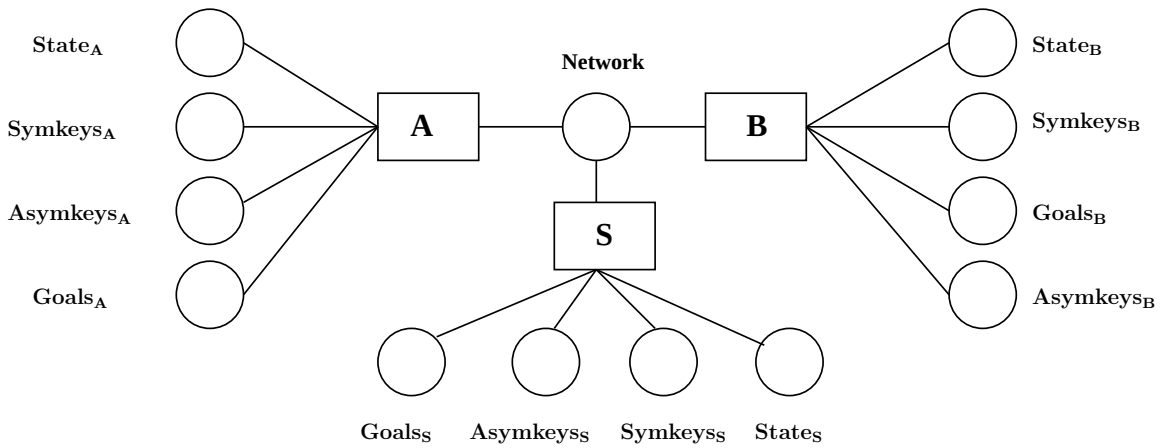

Figure 1: Structure of the APA model for agents A, B and S

2.1 Formal APA definition

In this section we give the formal definition of APA.

An *Asynchronous Product Automaton* consists of a family of *State Sets* $Z_S, S \in \mathbb{S}$, a family of *Elementary Automata* $(\Phi_e, \Delta_e), e \in \mathbb{E}$ and a *Neighbourhood Relation* $\mathrm{N} : \mathbb{E} \to \mathcal{P}(\mathbb{S})$; $\mathcal{P}(X)$ is the power set of X and $\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and elementary automata. For each Elementary Automaton (Φ_e, Δ_e),

- Φ_e is its *Alphabet* and
- $\Delta_e \subseteq \bigtimes_{S \in N(e)}(Z_S) \times \Phi_e \times \bigtimes_{S \in N(e)}(Z_S)$ is its *State Transition Relation*

For each element of Φ_e the state transition relation Δ_e defines state transitions that change only the state components in $N(e)$.

Remark: The alphabets Φ_e of the elementary automata can be used in protocol specifications to assign a label to any set of state transitions representing a particular protocol step. This motivates the state transition notation described below.

An APA's (global) *States* are elements of $\bigtimes_{S \in \mathbb{S}}(Z_S)$. To avoid pathological cases it is generally assumed that $\mathbb{S} = \bigcup_{e \in \mathbb{E}}(N(e))$ and $N(e) \neq \emptyset$ for all $e \in \mathbb{E}$. Each APA has one *Initial State* $s_0 = (q_{0S})_{S \in \mathbb{S}} \in \bigtimes_{S \in \mathbb{S}}(Z_S)$.
In total, an APA $\mathbb{A}$ is defined by $\mathbb{A} = ((Z_S)_{S \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, s_0)$.

The behaviour of an APA is represented by all possible sequences of state transitions starting with initial state s_0. The sequence $(s_0, (e_1, a_1), s_1)(s_1, (e_2, a_2), s_2)(s_2, (e_3, a_3), s_3) \ldots$ with $a_i \in \Phi_{e_i}$ represents one possible sequence of actions of an APA.

State transitions $(s_i, (e, a), s_{i+1})$ may be interpreted as labeled edges of a directed graph whose nodes are the states of an APA: $(s_i, (e, a), s_{i+1})$ is the edge leading from s_i to s_{i+1} and labeled by (e, a). The subgraph reachable from the node s_0 is called the *reachability graph* of an APA.

2.2 The Needham-Schroeder protocol

To illustrate how APA can be used to model cryptographic protocols, we present the well-known symmetric Needham-Schroeder protocol [5] the purpose of which is to distribute a session key to agents A and B with the help of a key server S. The messages of the protocol are as follows:

1. $A \longrightarrow S: \quad A, B, R_A$
2. $S \longrightarrow A: \quad \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \longrightarrow B: \quad \{K_{AB}, A\}_{K_{BS}}$
4. $B \longrightarrow A: \quad \{R_B\}_{K_{AB}}$
5. $A \longrightarrow B: \quad \{R_B - 1\}_{K_{AB}}$

In the first message agent A sends to S its name A, the name B of the desired communication partner, and a random number R_A which it generates for this protocol run. S then generates a ciphertext for A, using the key K_{AS} that it shares with A. This ciphertext includes A's random number, B's name, the new key K_{AB}, and a ciphertext intended for B. The usage of the key K_{AS} shall prove to agent A that the message was generated by S. The inclusion of R_A ensures A that this ciphertext and in particular the key K_{AB} is generated after the generation of R_A, i.e. during the current protocol run. Agent A also checks that the ciphertext includes B's name, making sure that S sends the new key to B

and not to someone else, and then forwards $\{K_{AB}, A\}_{K_{BS}}$ to B. Then, in the last two steps of the protocol, the fact that B's random number $R_B - 1$ is enciphered using the key K_{AB} shall convince B that this is a newly generated key. However, the well-known attack on this protocol presented by Denning and Sacco [1] makes clear that this conclusion cannot be drawn. This attack assumes that secrecy of session keys holds only for a limited time period.

In this paper we show how our analysis approach can find a different attack first shown in [3]. For a successful attack the intruder need not know past session keys. In fact, it is possible that agent B accepts a random number generated by the intruder as a new session key. This attack becomes possible when the number of components of a decrypted message is not checked and random numbers can be accepted as session keys. The attack and measures to avoid it are explained in more detail in section 3.

2.3 APA specification of the Needham-Schroeder Protocol

The APA structure in figure 1 shows the general structure of the APA for protocol specification. As mentioned above, in order to model a specific cryptographic protocol, we need to specify the set of messages, in particular cryptographic algorithms and keys, the state sets for all state components and the state transition relations of the elementary automata. In this section we only describe the details of APA protocol specifications that are needed for the specification of the Needham-Schroeder protocol. In general, different types of cryptographic algorithms with special properties (e.g. the homomorphic property of RSA [10]) can be modelled, a variety of data types can be defined, and additional security predicates describing undesired states of the system can be specified.

State sets, messages and cryptography In the present paper we restrict our model to very basic data types and the model of cryptography to symmetric algorithms for encryption and decryption. For the definition of the domains of the state components as well as for the definition of the set of messages, we need the following basic sets:

$\mathcal{A}gents$	set of agents' names
$\mathcal{K}eywords$	{*new session key, session key, start, respond,agent,server*}
$\mathcal{K}eys$	$\{(P, Q, sym) \mid P, Q \in \mathcal{A}gents\}$
$\mathcal{P}redicates$	set of predicates on global states
$\mathbb{N}$	set of natural numbers

As we assume in this paper that agents cannot check the type of a message, the random numbers used in a protocol are just natural numbers. However, if we want to model an environment where agents do check types of messages, we can easily add basic message sets as $\mathcal{R}andom$, $\mathcal{N}once$ etc. to enable type checks. The elements of $\mathcal{K}eywords$ are used to direct the state transitions.

The union of the sets $\mathcal{A}gents$, $\mathcal{K}eywords$, $\mathcal{K}eys$, and $\mathbb{N}$ represents the set of *atomic messages*, based on which we define a set $\mathcal{M}$ of messages in the following way:

1. Every atomic message is element of $\mathcal{M}$.
2. If $m_1, \ldots, m_r \in \mathcal{M}$, then $(m_1, \ldots, m_r) \in \mathcal{M}$.
3. If $k, m \in \mathcal{M}$, then $encrypt(k, m) \in \mathcal{M}$ and $decrypt(k, m) \in \mathcal{M}$.

4. If $m \in \mathcal{M}$, then $sub(m, 1) \in \mathcal{M}$.

We define the standard functions $elem(k, \ldots)$ and $length$ on tuples $(m_1, \ldots, m_r)$ which return the kth component (or, if $k \geq r$, the rth component) and the number of components of a tuple, respectively.

For all $P \in \mathcal{A}gents$ and $k, m \in \mathcal{M}$ we define the following properties of the symbolic functions *encrypt* and *decrypt*, and a symmetric relation *inv* on $\mathcal{K}eys$, respectively:

1. $decrypt(k, encrypt(k, m)) = m$ and $encrypt(k, decrypt(k, m)) = m$
2. $inv((P, Q, sym), (P, Q, sym))$ and $(P, Q, sym) = (Q, P, sym)$

Our general model provides additional symbolic functions for the specification of other cryptographic protocols. In this paper we restrict the model to those specifically necessary for the specification of the symmetric Needham-Schroeder protocol.

The above properties define for each $m \in \mathcal{M}$ a unique shortest normal form (modulo commutativity). The set $\mathcal{M}essages$ is the set of all these normal forms of elements $m \in \mathcal{M}$.

Now elements of $\mathcal{M}essages$ constitute the content of State components, while Network contains tuples of $\mathcal{A}gents \times \mathcal{M}essages$, where the first component gives the intended message recipient. A symmetric key *key* is stored in Symkeys_P using a tuple (Q, sym, key), where Q is the name of the agent that P's automaton will use to find the key and *sym* is the flag specifying a symmetric key used for encryption and decryption. (We use different flags to specify other types of keys used for example for the generation of a message authentication code.)

The symbolic functions *encrypt* and *decrypt* together with the above listed properties model a symmetric algorithm. For this paper, we assume "perfect encryption", i.e. we assume that keys cannot be guessed, that for generating $encrypt(k, m)$, both k and m need to be known, and that $encrypt(k, m) = encrypt(k', m')$ implies $k = k'$ and $m = m'$. Using more elaborated normal forms, we can relax these assumptions and model, for example, the symbolic function *blockcrypt* with the property $blockcrypt(k, (m_1.m_2)) = (blockcrypt(k, m_1).blockcrypt(k, m_2))$, thus modelling the fact that some cryptographic algorithms under some assumptions allow ciphertexts to be generated by concatenation of blocks of ciphertexts.

Formally, for each state component $C \in \mathbb{S}$ its state set Z_C has to be defined. The state of a state component is a multiset. Therefore, state sets are sets of multisets. A multiset of a set M is formally defined as a function $f \in \mathbb{N}^M$, where for $x \in M$, $f(x)$ indicates the multiplicity of x in the multiset of M. If $f(x) > 0$ for $x \in M$, we say that x is element of the multiset f of M (denoted by $x \in f$). $\mathbb{N}^M$ is the set of all multisets of M, which we shortly denote by $\bar{\mathcal{P}}(M)$. For each state component $C \in \mathbb{S}$ a set M_C has to be specified such that the state set Z_C is defined as $Z_C = \mathbb{N}^{M_C} = \bar{\mathcal{P}}(M_C)$.

Using the sets defined above and appropriate sets for $\mathcal{S}ymflags$ and $\mathcal{A}symflags$ (in this paper, $\mathcal{S}ymflags = \{sym\}$), domains of the state components of a protocol APA can now be defined as follows:

$$
\begin{array}{lcl}
Z_{Network} & = & \bar{\mathcal{P}}(\mathcal{A}gents \times \mathcal{M}essages) \\
Z_{State_P} & = & \bar{\mathcal{P}}(\mathcal{M}essages) \\
Z_{Symkeys_P} & = & \bar{\mathcal{P}}(\mathcal{A}gents \times \mathcal{S}ymflags \times \mathcal{M}essages) \\
Z_{Asymkeys_P} & = & \bar{\mathcal{P}}(\mathcal{A}gents \times \mathcal{A}symflags \times \mathcal{K}eys) \\
Z_{Goals_P} & = & \bar{\mathcal{P}}(\mathcal{P}redicates)
\end{array}
$$

Initial state To model a protocol, it is necessary to give the initial state, that is, the content of all state components in the state the protocol starts with. For the symmetric Needham-Schroeder protocol this can be specified in the following way:

$$
\begin{array}{lcl}
\text{State}_\text{A} & := & \{(\mathbf{B}, \mathit{agent}), (\mathbf{S}, \mathit{server}), (\mathit{start}, \mathbf{B})\} \\
\text{Symkeys}_\text{A} & := & \{(\mathbf{S}, \mathit{sym}, (\mathbf{A}, \mathbf{S}, \mathit{sym}))\} \\
\text{State}_\text{B} & := & \{(\mathbf{A}, \mathit{agent}), (\mathbf{S}, \mathit{server}), (\mathit{respond}, \mathbf{A})\} \\
\text{Symkeys}_\text{B} & := & \{(\mathbf{S}, \mathit{sym}, (\mathbf{B}, \mathbf{S}, \mathit{sym}))\} \\
\text{State}_\text{S} & := & \{(\mathbf{A}, \mathit{agent}), (\mathbf{B}, \mathit{agent})\} \\
\text{Symkeys}_\text{S} & := & \{(\mathbf{A}, \mathit{sym}, (\mathbf{A}, \mathbf{S}, \mathit{sym})), (\mathbf{B}, \mathit{sym}, (\mathbf{B}, \mathbf{S}, \mathit{sym}))\}
\end{array}
$$

Agents A and B know each other and know the server S, A can start a protocol run with B, while B can respond to protocol runs started by A (indicated by the keywords *start* and *respond*, respectively). Each of the agents owns a symmetric key shared with S. S on the other hand knows the agents A and B and owns the respective keys. The agents do not use asymmetric keys in this protocol, thus the components Asymkeys as well as Network are empty in the initial state.

State transition relation To specify the agents' actions we use so-called *state transition patterns* describing state transitions of the corresponding elementary automaton. Step 4 of the Needham-Schroeder protocol for example, where agent B receives message $\{K_{AB}, A\}_{K_{BS}}$ forwarded by A, can be specified as shown in Table 1.

step 4	$(A, S, M, K_{BS}, K_{AB}, R_B)$			Variables used in the pattern
	$(B, M) \in \text{Network}$			Message M with B as recipient in Network
	$(\mathit{respond}, A) \in \text{State}_\text{B}$			B can respond to protocol started by A
	$(A, \mathit{agent}) \in \text{State}_\text{B}$			B knows agent A
	$(S, \mathit{server}) \in \text{State}_\text{B}$			B knows server S
	$(S, \mathit{sym}, K_{BS}) \in \text{Symkeys}_\text{B}$			B shares symmetric key K_{BS} with S
	$\mathit{elem}(2, \mathit{decrypt}(K_{BS}, M)) = A$			after decryption with K_{BS}, second component of resulting plaintext is A
	$\xrightarrow{\mathbf{B}}$			state transition is performed by B
	$K_{AB} := \mathit{elem}(1, \mathit{decrypt}(K_{BS}, M))$			B assigns first plaintext component to K_{AB}
	$R_B \in \mathit{new\_random}$			B generates new random numberR_B
	(B, M)	$\hookleftarrow$	Network	B removes message from Network
	$(\mathit{new\ session\ key}, A, K_{AB}, R_B)$	$\hookrightarrow$	State_B	B stores K_{AB} as new session key
	$(A, (\mathit{encrypt}(K_{AB}, R_B)))$	$\hookrightarrow$	Network	B sends next message

Table 1: Step 4 of the Needham-Schroeder protocol

The lines above $\xrightarrow{\mathbf{B}}$ indicate the necessary conditions for automaton B to transform a state transition, the lines behind specify the changes of the state. $\hookrightarrow$ and $\hookleftarrow$ denote that some data is added to and removed from a state component, respectively. B does not perform any other changes within this state transition.

See the appendix for the full specification of the state transition patterns for the symmetric Needham-Schroeder protocol. The syntax and semantics of state transition patterns for APA is explained in more detail in [2].

So far the specification defines the functionality of one protocol run. A complete specification in the APA framework additionally contains security relevant information. Exemplary, in the following two paragraphs we give the information relevant for the Needham-Schroeder example.

Role specification of cryptographic protocols In the above state transition pattern as well as in the protocol description given in section 2.2, A, B, etc. are addressed as *agents*. However, any agent acting as B will behave in the above described way. Thus, the state transition patterns describe the actions of an agent acting in *role B* rather than describing an agent's actions.

In some environments it may not be possible that an agent acting in role A or B can also act as a server S. Additionally, some agent's roles may be regarded as secure, that is, we may assume these agents' roles cannot be played by an intruder. Thus for the formal specification of a protocol, the combination of roles that is forbidden has to be specified and those roles that might be compromised have to be determined. A possible specification for the Needham-Schroeder protocol is to forbid agents acting as A or B to act as S and vice versa, and to assume that A and B may be compromised.

Security goals The security of a protocol is defined relative to the specified security goals. Therefore the security goals are part of the protocol specification. Typical security goals are, for example, authenticity and confidentiality of transmitted data, such as new session keys, or the authenticity of agents taking part in the protocol. In our model, the state components Goals are used to specify security goals. Whenever an agent P performs a state transition after which a specific security goal shall hold from the agent's view, a predicate representing the goal is added to the state of Goals_P. Note that the content of Goals has no influence on state transitions.

A protocol is secure (within the scope of our model) if a predicate is true whenever it is in a Goals component.

In the Needham-Schroeder protocol, one security goal is that after B's final step the new session key which is added to State_B is not known to any agent except A and B, i.e. the key is not contained in any state component State_P for all $P \in \mathcal{A}gents \setminus \{A, B\}$. In terms of transition patterns the relevant parts of step 5 are:

$\xrightarrow{\mathbf{B}}$

$(new\ key, K_{AB}, A)$	$\hookrightarrow$	State_B	The value of variable K_{AB} is accepted as new session key
$(\forall P \in \mathcal{A}gents \setminus \{A, B\} : \text{not\_knows}(P, K_{AB}))$	$\hookrightarrow$	Goals_B	

3 Protocol analysis

3.1 APA framework for protocol analysis

The APA specification of a protocol maintains an abstraction level where details like random number generation, number of runs that shall be performed, the actions of agents, etc. are

specified on an abstract level. This has to be transfered to an abstraction level where the SHVT can actually perform a state search. This includes to specify the number and nature of runs that shall be validated (only finitely many runs can be checked), the agents taking part in these runs and their roles, and additional details like data that will be lost (for example, a session key might get lost after the end of the protocol). Additionally, the intruder has to be added to the model.

Interpretation of roles Since in the APA specification, A,B, etc. represent roles rather than agents, (**B**, *agent*) in A's State component in the initial state does not mean that an agent in role A only knows one single agent in role B. Rather, (**B**, *agent*) represents a subset of $\mathcal{A}gents \times \{agent\}$ which contains one element for each agent that can act in role B. Thus, (**B**, *agent*) can be viewed as a characteristic element for the possible interpretations. If, for example, Alice and Bob can act in role B, then (**B**, *agent*) represents the set {*(Alice,agent)*, *(Bob,agent)*}.

When interpreting such a tuple, the name of the agent owning the state component the tuple is an element of is excluded from the interpretation. For determining the initial state of $\text{Symkeys}_{\text{Bob}}$, the only value possible for **B** in the tuple $\{(\mathbf{S}, sym, (\mathbf{B}, \mathbf{S}, sym))\}$ is *Bob*, as Bob knows all keys he shares with all servers, but does not know keys some other agent shares with a server. As servers may vary, the set may contain tuples $\{(\mathbf{S_1}, sym, (\mathbf{B}, \mathbf{S_1}, sym))\}$, $\{(\mathbf{S_2}, sym, (\mathbf{B}, \mathbf{S_2}, sym))\}$, etc.

To derive a concrete scenario for the analysis, the characteristic elements of the role specification are replaced by the respective sets. The initial state for a state component is then the result of the union of all these sets.

The analysis APA Every concrete agent P acting honestly (i.e. according to the protocol specification) is modelled by one elementary automaton P and the state components State_{P}, $\text{Symkeys}_{\text{P}}$, $\text{Asymkeys}_{\text{P}}$, Goals_{P} and Network in its neighbourhood relation. The elementary automaton P performs state transitions according to the patterns given in the protocol specification. Additionally, an analysis APA contains a state component Global that can be accessed by all elementary automata. This state component can be used for the generation of random numbers, for the storage of data that can be used to interrupt a protocol run, etc. The APA also contains a further elementary automaton G to globally direct some state transitions. G has access to all agents' and the intruder's State components (see below) and to Network and Global.

The resulting APA together with a specification of analysis details (which are the agents acting, how many runs, etc.) can be used to check that the protocol specification without malicious behaviour results in the desired state transition sequence and that in particular the expected final states of the agents are reached.

Including an intruder In order to perform a security analysis, our model includes the explicit specification of an intruder $E \in \mathcal{A}gents$. For this we further extend the analysis APA and add state components $\text{Symkeys}_{\text{E}}$ and $\text{Asymkeys}_{\text{E}}$ to store E's keys, State_{E} to store what E learns from messages while listening to Network, and Sent_{E} in order to avoid sending of messages more than once. E also owns an elementary automaton E that has access to E's state components, Network and Global, and that performs E's actions. Figure 2 shows the complete APA for the analysis.

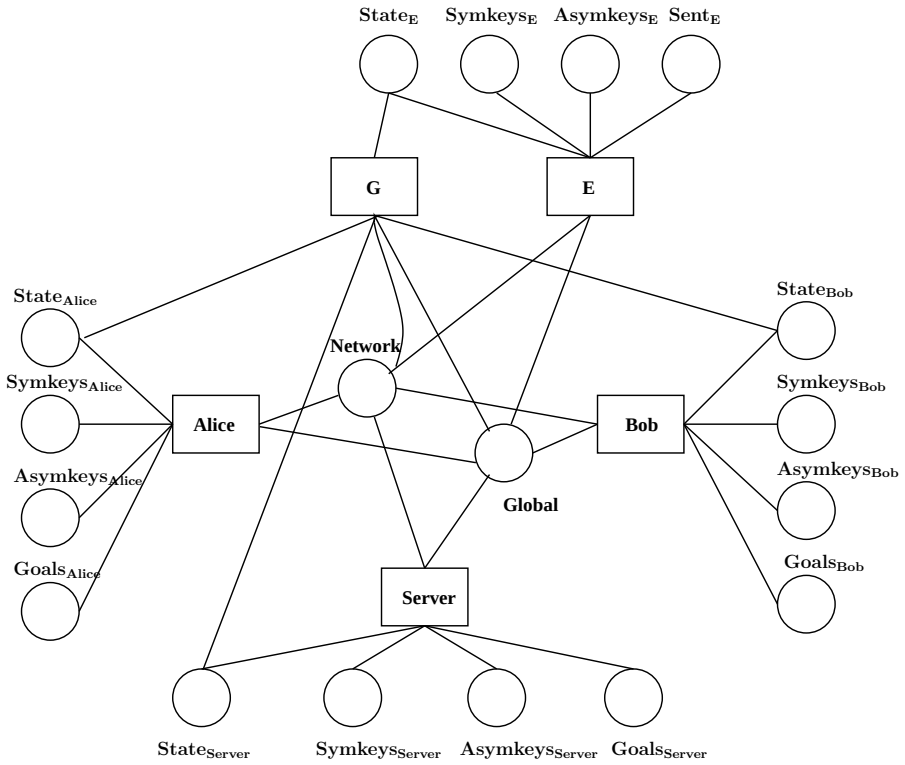

Figure 2: Structure of the analysis APA for agents Alice, Bob and Server, including E

The intruder's elementary automaton E can remove all tuples from Network independently from being named as the intended recipient. It can extract new knowledge from the messages and add this knowledge to the respective state components. Additionally E can add the intended recipient of a message as a possible recipient of his own messages to his knowledge base. The intruder's knowledge can be defined recursively in the following way:

1. The intruder knows all data being part of his initial state (for example the names of other protocol agents, the keys he shares with the key server, etc.)
2. The intruder knows all agents' names he extracts from a tuple in Network as the intended recipient of that message.
3. The intruder knows all messages he removed (as the second component of a tuple) from Network.
4. The intruder knows all parts of messages he removed from Network (where a ciphertext is viewed as one message part).
5. The intruder knows all messages he can generate by concatenation of messages he knows.
6. The intruder can generate new messages by applying the function *encrypt* to messages he knows. (Note that he can use any message in his knowledge base as a key.)
7. The intruder can generate random numbers.

8. The intruder knows all plaintexts he can generate by deciphering a ciphertext he knows, provided he knows the necessary key as well.

9. The intruder knows all messages he can generate by applying the symbolic function *sub* to messages he knows.

With new messages generated according to the above rules, in every state of the system, the intruder's elementary automaton E can add new tuples (*recipient,message*) to Network, *recipient* being one possible message recipient in his State component.

3.2 A concrete scenario for the Needham-Schroeder protocol

We now introduce a concrete scenario for the symmetric Needham-Schroeder protocol. Using the SH verification tool we want to automatically analyse a scenario where *Alice* (in role A) starts a protocol run with *Bob* (in role B), and *Server* acts in role S. These agents are acting honestly, thus the system includes an outside intruder E not taking part in the protocol run as an honest agent, which means in particular that E does not own a symmetric key shared with the server. However, E knows agents' and the server's names. We then want to analyse whether there is a state in which Bob believes to share a session key with Alice which is known by the intruder, i.e. whether or not every predicate $(\forall P \in \mathcal{A}gents \setminus \{Alice, Bob\} : \text{not\_knows}(P, K))$ holds in the state in which Bob adds this predicate to the state component Goals_B (which is the last step of the protocol). The initial state for this scenario includes

$$
\begin{array}{lcl}
\text{State}_\text{Alice}(s_0) & = & \{(Bob,agent),(Server,server),(start,Bob)\} \\
\text{Symkeys}_\text{Alice}(s_0) & = & \{(Server,sym,(Alice,Server,sym))\} \\
\text{State}_\text{Bob}(s_0) & = & \{(Alice,agent),(Server,server),(respond,Alice)\} \\
\text{Symkeys}_\text{Bob}(s_0) & = & \{(Server,sym,(Bob,Server,sym))\} \\
\text{State}_\text{Server}(s_0) & = & \{(Bob,agent),(Alice,agent)\} \\
\text{Symkeys}_\text{Server}(s_0) & = & \{(Alice,sym,(Alice,Server,sym)),(Bob,sym,(Bob,Server,sym))\} \\
\text{State}_\text{E}(s_0) & = & \{(Alice,agent),(Bob,agent),(Server,server)\}
\end{array}
$$

The Asymkeys and Goals components, Global, Network and Symkeys_E and Sent_E are empty in the initial state.

Starting with the initial state, the SHVT computes all reachable states until it finds a state in which the intruder knows a key that is accepted as a new session key by Alice or Bob. The SHVT outputs the state indicating a successful attack. Now one can let the SHVT compute a path from the initial state to this attack state. This path shows how the attack works. The remainder of this section explains the actions of Bob and the intruder E. Alice is not involved in the attack at all.

First, the intruder E pretends to be Bob starting a protocol run with Alice. E generates a random number *randomE* and sends the first message to Server (see the appendix for the respective state transition pattern). This results in

$$
\begin{array}{lcl}
\text{State}_\text{E}(s_1) & = & \{randomE,(Alice,agent),(Bob,agent),(Server,server)\} \\
\text{Network}(s_1) & = & \{(Server,(Bob,Alice,randomE))\}
\end{array}
$$

The server as the intended recipient of this message removes it from Network. It then generates a new session key K and a ciphertext *encrypt((Bob,Server,sym)*, (*randomE,Alice*, K, *encrypt((Alice, Server,sym)*, (K, Bob))))) using the symmetric keys it shares with Bob and Alice, respectively. The server adds Bob's name as the intended recipient and adds the complete tuple to Network:

$$\begin{aligned}\text{Network}(s_2) &= \{(Bob, (encrypt((Bob,Server,sym), (randomE,Alice, K,\\ &\quad encrypt((Alice,Server,sym), (K, Bob))))))\}\end{aligned}$$

In the next state transition, according to the state transition pattern presented in section 2.3, Bob's elementary automaton checks that he owns a server's key to decipher the ciphertext with, that the second component of the resulting plaintext contains the name of an agent Bob knows (namely Alice), that he is allowed to respond to a protocol run started by Alice, and then accepts this message. Thus Bob accepts this message (that was meant to be the second message of a protocol run) as the third message of a protocol run started by Alice. Therefore, Bob removes the message from Network, stores the first part of the plaintext, namely *randomE*, as a new session key to be shared with Alice, generates a random number *randomB*, enciphers this random number using the new "session key" *randomE*, and then adds *(Alice,encrypt(randomE,(randomB)))* to Network:

$$\begin{aligned}\text{State}_{\text{Bob}}(s_3) &= \{(new\ session\ key,Alice,randomE,randomB),(Alice,agent),\\ &\quad (Server,server)\}\\ \text{Network}(s_3) &= (Alice,(encrypt(randomE,(randomB))))\end{aligned}$$

The intruder E removes this tuple from Network, deciphers the ciphertext (as he knows *randomE*), applies the symbolic function *sub* and adds *(Bob,(encrypt (randomE,(sub(randomB,1)))))* to Network:

$$\text{Network}(s_4) = (Bob,(encrypt(randomE,(sub(randomB,1)))))$$

Finally, Bob removes this tuple from Network, deciphers it using again what he takes to be the new session key, checks that the result is $sub(randomB, 1)$ (using *randomB* stored in his State component) and stores *randomE* as the final session key shared with Alice. Finally, the predicate describing that *randomE* is a session key shared with Alice is added to Bob's Goals component:

$$\begin{aligned}\text{State}_{\text{Bob}}(s_5) &= \{(session\ key,Alice,randomE),(Alice,agent),(Server,server)\}\\ \text{Goals}_{\text{Bob}}(s_5) &= \{(\forall P \in \mathcal{A}gents \setminus \{Alice, Bob\} : \text{not\_knows}(P, randomE))\}\end{aligned}$$

So the final state includes *randomE* being stored by Bob as a new session key shared with Alice, while at the same time *randomE* is element of the intruder's State component. Thus the analysis has found a state where the *not_knows* predicate is in a Goals component and does not hold, i.e. the analysis shows that the Needham-Schroeder protocol is not secure. The attack the SHVT has found is the arity attack described in [3]. It can be avoided if Bob's actions when receiving message 3 of the protocol include the check that the ciphertext contains exactly two message items (*length(decrypt(Bob,Server,sym),M)* = 2). However, it is not trivial to realize such a check in real environments, as the length of the ciphertext itself does not necessarily allow to deduce the number of message items. For example, it is not obvious how to distinguish between random numbers used as padding data and other message parts. Note that the attack is also not possible if the protocol is run in an environment where agents can distinguish between types of keys and random numbers. We can model this by adding a check $K_{AB} \in \mathcal{K}eys$ to the state transition pattern describing Bob's actions when receiving message 3 of the protocol.

The attack was automatically found after computing 579 states in less than a minute on a Pentium III, 550 MHz system.

4 Conclusions

In this paper we have presented a methodology for the security analysis of cryptographic protocols. We view the agents taking part in such a protocol as communicating systems and model them using asynchronous product automata (APA), each agent being represented by one elementary automaton. Cryptography is modelled by symbolic functions with certain properties. The analysis APA explicitly includes the specification of an intruder's actions. Security of a protocol is defined relative to the specified security goals which are part of the specification. A protocol is secure (within the scope of our model) if whenever a predicate describing a security goal is in the state of an agent, it holds in the global state. We use the SH-verification tool [8] to perform a protocol analysis. We have demonstrated our methods using the well-known symmetric Needham-Schroeder protocol and presented an attack (the arity attack, see [3]).

Our methods do not provide proofs of security, but are similar to model checking analysis approaches where a finite state space is searched for insecure states (see, for example, [4] or [11]). Different approaches are concerned with the verification, i.e. the proof of certain security properties relative to a particular computational model (see for example [9] or [12]).

Compared to other approaches, our methods are both very flexible and minimal with respect to implicit assumptions used. Lowe and others as well for example use the assumption that every check an agent can perform is indeed performed, and that agents are able to distinguish between different data types. Our methods on the other hand allow the analysis of systems where these assumptions hold by adding the respective checks as well as the analysis of systems where these assumptions do not hold. Such systems exist, smartcard applications for example often use implicit data structures (see [6]).

For the work presented in this paper we have assumed perfect encryption. Ongoing work includes to relax this assumption and use additional symbolic functions to model for example the homomorphic property of RSA [10]. Additionally the way of reducing the state space without restricting the powers of the intruder is subject to future work.

References

[1] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24:533–536, 1982.

[2] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and provability – a formal framework. GMD Report 150, GMD – Forschungszentrum Informationstechnik GmbH, 2001.

[3] S. Gürgens and R. Peralta. Validation of Cryptographic Protocols by Efficient Automated Testing. In *FLAIRS2000*, pages 7–12. AAAI Press, May 2000.

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

[5] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, pages 993–999, 1978.

[6] DIN NI-17. *Chipkarten mit Digitaler Signatur - Anwendung/Funktion nach SigG und SigV - Teil 1: Anwendungsschnittstelle*. DIN, April 2000.

[7] P. Ochsenschläger, J. Repp, and R. Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, June 2000.

[8] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24, 1999.

[9] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[10] R. L. Rivest, A. Shamir, and L. A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[11] B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe. *The modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[12] D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 1,2(9):47–74, 2001.

A Appendix

Step 1 Variables: R_A, B, S

$(start, B) \in \text{State}_\text{A}$
$(B, agent) \in \text{State}_\text{A}$
$(S, server) \in \text{State}_\text{A}$
$\xrightarrow{\mathbf{A}}$
$R_A \in new\_nonce$

$(start, B)$	$\hookleftarrow$	State_A
(B, R_A, S)	$\hookrightarrow$	State_A
$(S, (A, B, R_A))$	$\hookrightarrow$	Network

Step 2 Variables: $A, B, M, K_{AS}, K_{BS}, K_{AB}, R_A$

$(S, M) \in \text{Network}$
$(A, agent) \in \text{State}_\text{S} \wedge elem(1, M) = A$
$(B, agent) \in \text{State}_\text{S} \wedge elem(2, M) = B$
$(A, sym, K_{AS}) \in Symkeys_S$
$(B, sym, K_{BS}) \in Symkeys_S$
$\xrightarrow{\mathbf{S}}$
$R_A := elem(3, M)$
$(A, B, sym) \in new\_key$
$K_{AB} := (A, B, sym)$

(S, M)	$\hookleftarrow$	Network
$(A, (encrypt(K_{AS}, (R_A, B, K_{AB},$ $encrypt(K_{BS}, (K_{AB}, A))))))$	$\hookrightarrow$	Network

Step 3 Variables: $B, S, M, C, K_{AS}, K_{AB}, R_A$

$(A, M) \in \text{Network}$
$(B, R_A, S) \in \text{State}_\text{A}$
$(S, sym, K_{AS}) \in \text{Symkeys}_\text{A}$
$elem(1, decrypt(K_{AS}, M)) = R_A$
$elem(2, decrypt(K_{AS}, M)) = B$
$\xrightarrow{\mathbf{A}}$
$K_{AB} := elem(3, decrypt(K_{AS}, M))$
$C := elem(4, decrypt(K_{AS}, M))$

(A, M)	$\hookleftarrow$	Network
(B, R_A, S)	$\hookleftarrow$	State_A
$(new\ session\ key, B, K_{AB})$	$\hookrightarrow$	State_A
(B, C)	$\hookrightarrow$	Network

Step 4 Variables: $A, S, M, K_{BS}, K_{AB}, R_B$

$(B, M) \in \text{Network}$
$(respond, A) \in \text{State}_\text{B}$
$(A, agent) \in \text{State}_\text{B}$
$(S, server) \in \text{State}_\text{B}$
$(S, sym, K_{BS}) \in \text{Symkeys}_\text{B}$
$elem(2, decrypt(K_{BS}, M)) = A$
$\xrightarrow{\mathbf{B}}$
$K_{AB} := elem(1, decrypt(K_{BS}, M))$
$R_B \in new\_nonce$

(B, M)	$\hookleftarrow$	Network
$(new\ session\ key, A, K_{AB}, R_B)$	$\hookrightarrow$	State_B
$(A, (encrypt(K_{AB}, R_B)))$	$\hookrightarrow$	Network

Step 5a Variables: B, M, K_{AB}, R_B

$(A, M) \in \text{Network}$
$(new\ session\ key, B, K_{AB}) \in \text{State}_\text{A}$
$\xrightarrow{\mathbf{A}}$
$R_B := elem(1, decrypt(K_{AB}, M))$

(A, M)	$\hookleftarrow$	Network
$(new\ session\ key, B, K_{AB})$	$\hookleftarrow$	State_A
$(session\ key, B, K_{AB})$	$\hookrightarrow$	State_A
$(\forall P \in \mathcal{A}gents \setminus \{A, B\} : \text{not\_knows}(P, K_{AB}))$	$\hookrightarrow$	Goals_A
$(B, (encrypt(K_{AB}, sub(R_B, 1))))$	$\hookrightarrow$	Network

Step 5b Variables: A, M, K_{AB}, R_B

$(B, M) \in \text{Network}$
$(new\ session\ key, A, K_{AB}, R_B) \in \text{State}_\text{B}$
$elem(1, decrypt(K_{AB}, M)) = sub(R_B, 1)$
$\xrightarrow{\mathbf{B}}$

(B, M)	$\hookleftarrow$	Network
$(new\ session\ key, A, K_{AB}, R_B)$	$\hookleftarrow$	State_B
$(session\ key, A, K_{AB})$	$\hookrightarrow$	State_B
$(\forall P \in \mathcal{A}gents \setminus \{A, B\} : \text{not\_knows}(P, K_{AB}))$	$\hookrightarrow$	Goals_B