# Formal Security Analysis with Interacting State Machines

David von Oheimb and Volkmar Lotz

Siemens AG, Corporate Technology, D-81730 Munich
{David.von.Oheimb,Volkmar.Lotz}@siemens.com

**Abstract.** We introduce the ISM approach, a framework for modeling and verifying reactive systems in a formal, even machine-checked, way. The framework has been developed for applications in security analysis. It is based on the notion of Interacting State Machines (ISMs), sort of high-level Input/Output Automata. System models can be defined and presented graphically using the AutoFocus tool. They may be type-checked and translated to a representation within the theorem prover Isabelle or defined directly as Isabelle theories. The theorem prover may be used to perform any kind of syntactic and semantic checks, in particular semi-automatic verification. We demonstrate that the framework can be fruitfully applied for formal system analysis by two classical application examples: the LKW model of the Infineon SLE66 SmartCard chip and Lowe's fix of the Needham-Schroeder Public-Key Protocol.

## 1 Introduction

### 1.1 Motivation

In industrial environments, there is an increased demand for rigorous analysis of security properties of systems. Due to restrictions imposed by the application domain, the system environment, and business needs, new security mechanisms and architectures have to be invented frequently, with time-to-market pressure and intellectual property considerations obstructing the chance to gain confidence by exposing a proposed solution to the security community (which has been shown to be appropriate for cryptographic algorithm assessment). Formal analysis of suitable abstractions of systems has instead turned out to be extremely helpful in reasoning about a system's security, since the mathematical precision of the arguments allows for maximal confidence in the results obtained and, thus, in the security of the system being modeled.

The importance of formal analysis – on top of open review – in security assessment is, for instance, reflected by the requirements stated for high assurance levels of criteria like ITSEC [ITS91] and CC [CC99], which include formal security modeling and formal system development steps, and the achievements of the security protocol verification community, which discovered flaws in protocols that failed to be detected by informal approaches.

D. Gollmann et al. (Eds.): ESORICS 2002, LNCS 2502, pp. 212–229, 2002.

However, even in a formal setting it is easy to make – minor and sometimes even major – mistakes: undefined expressions, type mismatches, inconsistent specifications, missing evidence in proofs, false conclusions etc. Therefore, pure pen-and-paper formalizations cannot be considered fully reliable. Machine-checking of formal objects and structures has to be employed in order to significantly reduce the occurrence of such mistakes. Machine support additionally gives the opportunity to represent and deal with formal objects – both specifications and proofs – in an easy-to-comprehend way, which is a prerequisite for introducing formal approaches in an industrial environment characterized by time and cost restrictions.

### 1.2 Goals

A framework for formal security analysis particularly suited for industrial use should enjoy the following properties:

**Expressiveness.** It should be possible to describe any typical security sensitive computation, storage, and communication systems in an abstract way. This requires in particular the notions of state transformation, concurrency and message passing.

**Flexibility.** Since IT systems and their security threats evolve quickly, the models produced within the framework should be easily adaptable and extendable as necessary to reflect the changes.

**Simplicity.** Modeling a system, stating its properties and proving them should require as little expertise and time as possible while maintaining the rigor of a fully formal approach.

**Graphical capabilities.** System models should be representable as diagrams that give a good overview of the system structure and a quick intuition about its behavior.

**Maturity of the semantics.** The semantic foundation of the framework should be well-developed, supporting in particular modular refinement.

**Availability of tools.** The framework should be built from existing widely available (open-source) software like editors and proof tools and require at most minor modifications or extensions to them.

### 1.3 Related Work

The *IOA Language and Toolset* [GL98, Kay01] is a framework for analyzing computational processes with aims very similar to ours. It consists of a specification language and tool support for simulation, theorem proving, model checking, and code generation, where by now the simulation aspect is developed most and theorem proving support is limited to PVS. Its semantic foundation is the notion of *I/O Automata (IOAs)* [LT89] modeling asynchronous distributed computation with synchronous communication. Since the notion is based on transition systems augmented by communication primitives (rather than e.g. a process algebra augmented by local computation primitives), it is fairly easy to understand. It

is equipped with a well-developed meta theory supporting refinement and compositional reasoning. System properties, both safety and lifeness ones, may be described using temporal logics and proved by model checking and interactive theorem proving.

The only — but severe — drawback of IOAs from our perspective, in particular when modeling system security in an abstract way, is that their interaction scheme is rather low-level: buffered communication has to be modeled explicitly, and transitions involving several related input, internal processing and output activities cannot be expressed atomically. Instead, each high-level transition has to be split into multiple low-level transitions, and between these, any number of further input events may take place due to the input-enabledness of IOAs. The solution to this problem is to add extra structure, essentially by interpreting parts of the local state of an automaton as input/output buffers. Our notion of ISMs, introduced in [Ohe02a], provides for that.

A further related work that provided inspiration for our framework is AutoFocus [HSSS96] – see also §2.2. Even though developed primarily for modeling and verifying functional properties of embedded systems, it is used also for the securiy analysis of general distributed systems [WW01, JW01].

Other related approaches combine state-oriented and message-oriented description methods, for example translating CSP to B [But99] or Z to CSP [Fis00]. The drawback of such hybrids is that the user has to deal with two different nontrivial formalisms. Moreover, theorem proving support respecting the structure of the mixed-style specifications seems not to be available.

## 2 Preliminaries

In this section, we briefly introduce the two software tools we rely on and comment on their suitability for the ISM approach.

### 2.1 Isabelle/HOL

*Isabelle* [Pau94] is a generic theorem prover that has been instantiated to many logics, in particular the very practical *Higher-Order Logic (HOL)*. Isabelle/HOL [PNW+] is a predicate logic based on the simply-typed $\lambda$-calculus and thus in a sense combines logical and functional programming. Being quite expressive and supporting automatic type inference, it is the most important and best supported logic of Isabelle. The lack of dependent types introduces a minor nuisance for applications like ours: for each system modeled there is a single type of message contents into which all message data has to be injected, and analogously for the local states of automata.

Proofs are conducted primarily in an interactive fashion where automatic and semi-automatic methods are available to tackle the routine parts. The Isabelle system is well-documented and well-supported, is freely available (including sources) and comes with the excellent user interface ProofGeneral [AGKS99]. We consider it the most flexible and mature verification environment available.

Using Isabelle/HOL, security properties can be expressed easily and adequately and verified with powerful proof methods.

### 2.2 AutoFocus

*AutoFocus* [HSSS96] is a freely available specification and simulation tool for distributed systems. Components and their behavior are specified by a combination of *system structure diagrams (SSDs)*, *state transition diagrams (STDs)* and auxiliary *data type definitions (DTDs)*. Their execution can be visualized using *extended event traces (EETs)*. Various back-ends including code generators and interfaces to model checkers may be acquired by purchase from Validas [S+].

We employ AutoFocus for its strengths concerning graphical design and presentation, which is important when setting up models in collaboration with clients (where strong familiarity with formal notations cannot be assumed), when documenting our work and publishing its results. For abstract security modeling, there are currently two problems. First, expressiveness is limited concerning the type system and the handling of underspecification. These weaknesses are going to be removed in the near future. Second, due to the original emphasis of AutoFocus on embedded systems, the underlying semantics is still clock-synchronous. In contrast, for the most of our applications, in particular communication protocols, an asynchronous (buffered) semantics is more adequate, which is under consideration also for future versions of AutoFocus. Using an alternative semantics implies that we cannot make use of the simulation, code generation and model checking capabilites of current AutoFocus and its back-ends. Yet this is not a real obstacle for us since we are interested mainly in its graphic capabilities and the offered specification syntax is general enough to cover also our deviating semantics.

## 3 The ISM Approach

We first introduce the core of our modeling and verification framework, viz. ISMs, and then give some general comments how they are handled by AutoFocus and Isabelle.

### 3.1 Interacting State Machines

An *Interacting State Machine (ISM)* is an automaton whose state transitions may involve multiple input and output simultaneously on any number of ports. As the name suggests, the key concepts of ISMs are states (and in particular the transitions between them) and interaction. By interaction we mean explicit buffered communication via named ports, where on each of them one receiver listens to possibly multiple senders. A *system* consists of the parallel composition of any number of ISM components where the state of the whole system is essentially the Cartesian product of the states of its components.

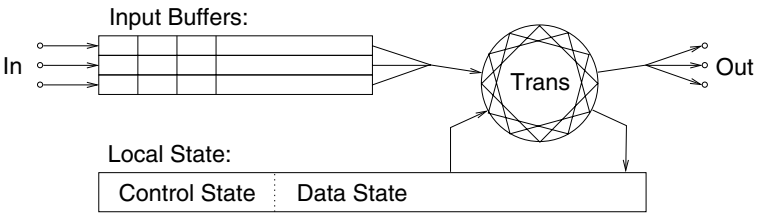


**Fig. 1.** ISM structure

The *state* of an ISM consists of the local state and its input buffers. The *local state* may have arbitrary structure but typically is the Cartesian product of a *control state* which is of finite type, and a *data state* that typically is a record of named fields. Transitions between states may be nondeterministic and can specified in any relational style. Thus the user has the choice to define them in an operational (i.e., executable) or axiomatic (i.e., property-oriented) fashion or a mixture of the two.

Each ISM declares two sets of port names, one for input and the other for output. The local input buffers are a family of unbounded FIFOs indexed by port names. Input of individual messages is triggered by any ISM and cannot be blocked, i.e. may occur at any time, appending the received value to the corresponding FIFO. Values stored in the input buffers may be processed by the ISM when it is ready to do so. This is done in a transition specified as follows: under a given precondition, the ISM consumes as much input from the buffers as appropriate, makes a transition of the local state, and produces output values at its discretion. These values are forwarded to the input buffers of all ISMs listening to the respective port, which may include feedback to the current component.

Each ISM has a single[1] initial state with empty input buffers. The execution of a system is a finite (but unbounded) sequence of nondeterministically interleaved steps of any of its components. Finiteness implies that we can handle safety, but not liveness properties. Transitions of different ISMs are related only by the causality wrt. the messages interchanged. Execution gets stuck when there is no component that can perform any step. As typical for reactive systems, there is no built-in notion of final or "accepting" states.

The representation of ISMs and their semantics consists of several layers. Its details, as well as a translation to IOAs, may be found in [Ohe02a].

[1] If a non-singleton set of initial states is required, these may be simulated by spontaneous nondeterministic transitions originating from a single dummy initial state.

### 3.2 AutoFocus Representation

By design, ISMs have almost the same structure as the automata definable with AutoFocus, and thus we can use AutoFocus as a graphical front-end to our Isabelle implementation.

In a typical application of our framework, ISMs are first "painted" using AutoFocus, saved in the so-called *Quest* file format, and then translated into suitable Isabelle theory files by a tool program.

### 3.3 Isabelle Representation

ISMs can be defined in special sections of Isabelle theories. This abstract (and almost semantics-independent) representation has essentially a one-to-one correspondence to the AutoFocus representation. As the cases studies below show, the structure of this section is almost self-explanatory. The formal definition of both the syntax and the semantics of ISMs in Isabelle/HOL is given in [Ohe02a].

## 4 LKW Model for the Infineon SLE66

We give an improvement of the LKW model for the Infineon SLE66 SmartCard processor. We demonstrate that, with the ISM approach, transition systems can be adequately modeled and their security properties stated and proven.

The *LKW model* [LKW00] is one of the first formal models for security properties of hardware chips. It has been used successfully within the security evaluation process for the SLE66 on ITSEC level E4 and the corresponding Evaluation Assurance Level 5 (*semiformally designed and tested*, which includes a formal security model) [CC99]. Recently, a slight extension was introduced [OLW02] in order to reflect additional application-oriented security objectives as defined in the SmartCard IC Platform Protection Profile [AETS01].

The LKW model gives an abstract system model for the SLE66 based on an ad-hoc automaton formalism, formalizes the security requirements in terms of properties of automaton runs and proves that the system meets the given requirements. All this is done as a pen-and-paper work, i.e. without tool assistance. Thus it is inevitable that the model contains many (mostly minor) syntactical, typographical and semantical slips as well as type errors, but also omissions like missing assumptions and incomplete proofs. Therefore it was desirable to formalize the model in a machine-checked way, applying a well-developed meta theory. Using ISMs, the LKW model can be represented adequately, including some improvements.

### 4.1 AutoFocus Diagrams

On the abstract level of the LKW model, the system architecture of the SLE66 is rather trivial: there is one component with one input port named `In` and one output port named `Out`, as depicted by Figure 2. The data state of the

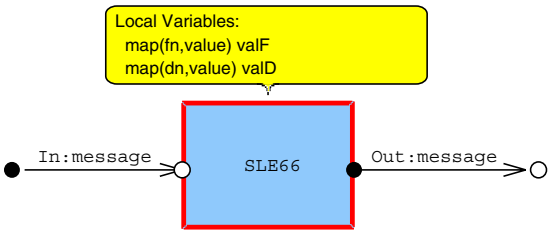


**Fig. 2.** SLE66 System Structure Diagram

component consists of two stores mapping names of functions and data objects to corresponding values.

Much more involved is the structure of the state transitions. There are four control states corresponding to the *phases* of the SLE66 life cycle:

**Phase** `0`**:** construction of the chip
**Phase** `1`**:** upload of Smartcard Embedded Software, personalization
**Phase** `2`**:** normal usage
**Phase** `Error`**:** locked mode from which there is no escape

In order to keep the state transition diagram clear, Figure 3 contains all control states and transitions, but instead of showing the preconditions, inputs, outputs, and changes to the data state, we just label the transitions with the names of the corresponding transition rules. Part of the rules are described in

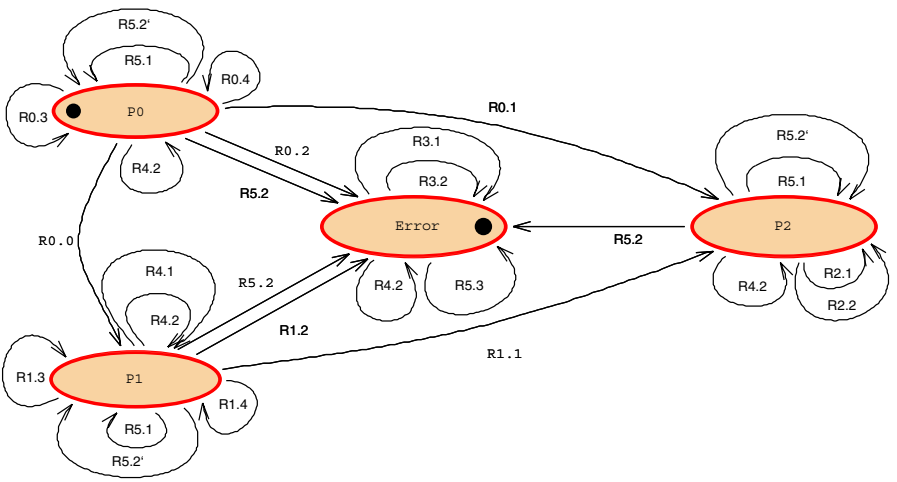


**Fig. 3.** SLE66 State Transition Diagram

detail in the next subsection. Here we give an informal general description of the transitions.

**R0.0 thru R0.4** describe the execution of functions in the initial phase 0. Only the processor manufacturer is allowed to invoke functions in this phase and the required function must be enabled.

**R0.0** states that if the function belongs to class `FTest0` and the corresponding test succeeds, phase 1 will be entered, and the test functions of that class are disabled.

**R0.1** describes a shortcut leaving out phase 1: if the function belongs to class `FTest1` and the test succeeds, phase 2 will be entered, and all test functions are disabled.

**R0.2** states that if a test fails, the system will enter the error state.

**R0.3** models the successful execution of any other (enabled) function, in which case the function may change the chip state and yield a value.

**R0.4** states that in all remaining cases of function execution the chip responds with `No` and its state is unchanged.

**R1.1 thru R1.4** describe the execution of functions in the upload phase 1 analogously to R0.1 thru R0.4.

**R2.1 and R2.2** describe the execution of functions in the usage phase 2 analogously to R0.3 and R0.4.

**R3.1 and R3.2** describe the execution of functions in the error phase analogously to R0.3 and R0.4, except that the only function allowed to be executed in this phase is chip identification.

**R4.1 and R4.2** describe the effects of a specific operation used for uploading new (operating system and application) functionality on the chip. This must be done by subjects trusted by the processor manufacturer and is allowed only in phase 1.

**R4.1** describes the admissible situations, and

**R4.2** describes all other cases.

**R5.1 thru R5.3** describe the effects of attacks. Any attempts to tamper with the chip and to read security-relevant objects via physical probing on side channels (by mechanical, electrical, optical, and/or chemical means), for example differential power analysis or inspecting the silicon with a microscope, are modeled as a special "spy" input. Note that modeling physical attacks in more detail is not feasible because this would require a model of physical hardware. In particular, the conditions (and related mechanisms) under which the processor detects a physical attack is beyond the scope of the model.

**R5.1** describes the innocent case of reading non-security-relevant objects in any regular phase, which actually reveals the requested information.

**R5.2** describes the attempt to reading security-relevant objects in any regular phase. The chip has to detect this and enters the error phase, while the requested object may be revealed or not. This concept is called "destructive reading": one cannot rule out that attacks may reveal information even about security-relevant objects, but after the first of any such attacks, the processor hardware will be "destroyed", i.e. cannot be used regularly.

**R5.3** states that in the error phase no (further) information is revealed.

### 4.2 Isabelle Theory

Next we give the Isabelle/HOL representation of the SLE66 model. For lack of space, of course we can show only the most essential definitions and a very small selection of the transition rules. Yet we do describe the slight extension mentioned above. For a detailed formal description of the LKW model see [LKW00]. The full Isabelle definitions are contained in [Ohe02b].

Objects stored on the chip may be either functions or data and are referred to by object names:

```
datatype on = F fn | D dn
```

Objects are classified as security-relevant (demanding secrecy and integrity) by including their names in the sets `F_Sec` or `D_Sec`, whose union is called `Sec`. In order to meet the additional requirements of [AETS01], the domain of security relevant functions `F_Sec` of the original LKW model has been refined to the disjoint union of `F_PSec` and `F_ASec`, which control the protection of the processor and application functionality, respectively.

The four control states of the SLE66 are defined as

```
datatype ph = P0 | P1 | P2 | Error
```

The data state consists of two fields, a function store and a data store:

```
record data =
  valF :: "fn ⇝ val"
  valD :: "dn ⇝ val"
```

The initial data state is declared but not actually defined. This is a typical example of underspecification, an important modeling technique.

```
const s0 :: data
```

We need only two port names, one for input and one for output:

```
datatype interface = In | Out
```

Possible input to the chip consists of either the two kinds of regular input messages (modeling function execution and load commands to the SLE66), or the `Spy` operation. The chip may respond with a value or a status message indicating success or failure.

```
datatype message =
  Exec sb fn | Load sb fn val | Spy on
| Val val | Ok | No
```

The subjects `sb` performing regular operations identify themselves to the chip via physical means. The actual authentication mechanism, as well as many other implementation details, is beyond the scope of this article.

Having defined its various parameters, we can now give the new **ism** theory element that specifies the SLE66 model:

```
ism SLE66 =
    ports interface
```

```
    inputs    "{In}"
    outputs   "{Out}"
    messages  message
  state
    control P0 :: ph
    data    s0 :: data
  transitions

R0.0: P0 -> P1
      pre "f ∈ fct s∩FTest0", "test f s"
      in   In  "[Exec Pmf f]"
      out  Out "[Ok]"
      eff  "valF := valF s⌊-FTest0"
```

Rule R0.0 specifies execution of a test function *f* by the processor manufacturer *Pmf* from the initial phase *P0*: if the test is successful then the SLE66 enters the next phase *P1*, disables the test functions *FTest0*, and answers with *Ok*. This rule is typical for interactions with the SLE66 in the sense that a single input triggers a single output. Note that the direct relation of input and output is expressed easily using ISMs, whereas using IOAs, two transitions would be required whose relation would be cumbersome to express and to use during verification.

```
R5.2: ph -> Error
      pre "ph ≠ Error", "oname ∈ Sec",
          "v ∈ {[],[Val (the (val s oname))]}"
      in   In  "[Spy oname]"
      out  Out "v"
      eff  "valF := fs, valD := ds"
```

Rule R5.2 specifies the typical reaction of the SLE66 upon attacks trying to read (the representation of) a secret object: The desired value may be output or not, but in any case the *Error* phase is reached. Note that R5.2 is a generic transition from any regualar phase to the *Error* phase. Furthermore, two sorts of nondeterminism are involved: *v* denotes either the empty output or the singleton output giving the desired value, and the attack may corrupt the function and data stores arbitrarily.

In contrast to the original LKW model, the *Load* operation may upload not only non-security-relevant functions but also functions of the application security domain, as long as they are not overwritten:

```
R4.1: P1 -> P1
      pre "f ∈ F_NSec ∪ (F_ASec - fct s)"
      in   In  "[Load Pmf f v]"
      out  Out "[Ok]"
      eff  "valF := valF s(f↦v)"
```

All remaining transition rules and further details on the system model may be found in [LKW00] and [Ohe02b].

### 4.3 Properties

The original security objectives for the SLE66 were stated as follows.

**SO1.** "The hardware must be protected against unauthorised disclosure of security enforcing functionality."
**SO2.** "The hardware must be protected against unauthorised modification of security enforcing functions."
**SO3.** "The information stored in the processor's memory components must be protected against unauthorised access."
**SO4.** "The information stored in the processor's memory components must be protected against unauthorised modification."
**SO5.** "It may not occur that test functions are executed in an unauthorised way."

Later, an additional requirement concerning the confidentialiy and integrity of Smartcard Embedded Software, which is not part of the security enforcing functionality, has been added [AETS01, §4.1].

Having defined the SLE66 system model, these informal statements can now be expressed formally as predicates on the system behavior, describing unambiguously and in detail which states may be reached under which circumstances, which data may be modified, and which output may appear on the output channel.

After formalizing the security objectives, it is natural to ask if the chip behavior, as specified in the system model, actually fulfills these requirements. The corresponding proofs have been conducted first using pen and paper, as reported in [LKW00]. Within the ISM framework, we meanwhile have verified these properties even mechanically (and thus with maximal reliability) using Isabelle.

Due to the abstract specification style where the semantics of parts of the chip functionality is not fully specified, it turns out that in order to prove the properties, a few general axioms are required. These assert for example that security-relevant functions do not modify security-relevant functions:

```
Axiom1: "f∈fct s∩F_Sec ⟹ valF (change f s)⌊F_Sec = valF s⌊F_Sec"
```

In comparison to the version of this axiom in the original model, the scope of functions `f` has been extended from "initially available" to "security-relevant", reflecting the changes to rule `R4.1`. Part of the lemmas as well as the formalized security objective FSO2.1 change accordingly:

```
FSO21: "⟦((ib,(ph,s)),p,(ib',(ph',s'))) ∈ Trans; ph' ≠ Error;
         g ∈ fct s∩fct s'∩F_Sec⟧ ⟹ valF s' g = valF s g"
```

The proof of this property is — as usual — by induction on the construction of all runs the SLE66 ISM can perform, which boils down to a case distinction over all possible transitions. Most cases are trivial except for those where function execution may change the stored objects, which are described by the rules `R0.3`, `R1.3`, and `R2.1`. Here an argumentation about the invariance of security-relevant functions `g` is needed, which follows easily from `Axiom1` and `Axiom2` stating the analogous property for non-security-relevant functions `f`.

The third (and last) axiom introduced in the LKW model states that in phase 2, a non-security-relevant function may not "guess" or (accidentally) reveal security-relevant information.

When machine-checking the proofs contained in [LKW00] with Isabelle, we noticed that a fourth axiom was missing that makes an implicit but important assumption explicit: if a function object may be referenced in two (different) ways and one of them declares the function to be security-relevant, the other has to do the same. Such experience demonstrates how important machine support is when conducting formal analysis.

Another omission was that in the proof of the security objective FSO5 an argumentation about the accessibility of certain functions was not given in a rigorous way. We fix this by introducing an auxiliary property (where, as typical with invariants, finding the appropriate one is the main challenge) and proving it to be an invariant of the ISM:

```
  no_FTest_invariant :: "state ⇒ bool"
 "no_FTest_invariant ≡ λ(ph,s).
  ∀f ∈ fct s. (ph = P1 ⟶ f ∉ FTest0) ∧ (ph = P2 ⟶ f ∉ FTest)"
```

Exploiting the invariant, we can prove the desired property easily:

```
FSO5: "⟦((ib,(_,s)),(p,_,(_,s'))) ∈ Trans; ib In = Exec sb f#r;
        f ∈ FTest⟧ ⟹ sb = Pmf ∨ p Out = [No] ∧ s' = s"
```

The Isabelle proofs of all six theorems formalizing the security objectives and the two lemmas required are well supported by Isabelle: each of them takes just a few steps, about half of which are automatic.

The formalization of the remaining security objectives as well as their proofs wrt. the system model may be found in [LKW00] and [Ohe02b].

## 5 Needham-Schroeder Public-Key Protocol

As an example of an interaction-oriented system modeled with ISMs, we take Lowe's fix of the *Needham-Schroeder public-key authentication protocol* [Low96], which we call *NSL*. The emphasis here is not to provide new insights to the protocol, but to take a well-known (and thus easy to compare) benchmark system in order to show that, using the ISM approach, not only high-level requirements analysis but also low-level analysis of distributed systems can be done in a both rigorous and elegant way.

### 5.1 AutoFocus Diagrams

The system consists of an agent called `Alice` aiming to establish an authenticated session with another agent called `Bob` in the presence of an `Intruder` according to the Dolev-Yao attacker model [DY83]. As will be motivated in §5.2, we introduce a server `NGen` generating nonces. The corresponding system structure diagram in Figure 4 shows the four components with their data state (reflecting the expectations of the two agents, the set of messages the intruder

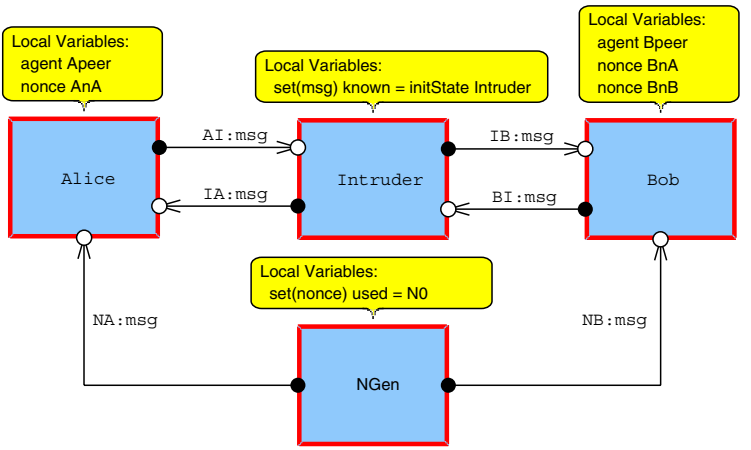


**Fig. 4.** NSL System Structure Diagram

knows of, and the set of already used nonces, respectively) and the named connections between them. Even if sometimes neglected, agents involved in communication protocols do have state: their current expectations and knowledge. This is made explicit in a convenient way by describing their interaction behavior with state transition diagrams. Figure 5 shows the three states of the agent `Alice` and the transitions between them, which have the general format `guard : inputs : outputs : assignments`.

In the initial state, `Alice` decides which agent she wants to talk to and sends the corresponding request. In the next state she awaits the response from the prospective peer before sending an acknowledgment. The third state represents (hopefully) successful session establishment. From the example of `Alice`'s tran-

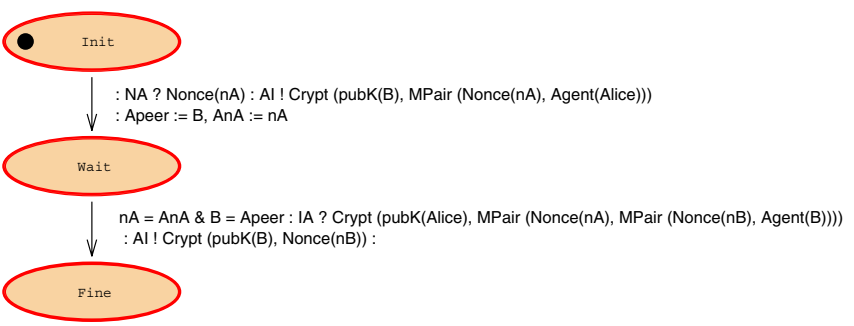


**Fig. 5.** NSL State Transition Diagram: `Alice`

sitions we realize that control state information is the natural way to fix the order of protocol steps.

If the analysis needs to include the possibility that an agent takes part in more than one protocol run simultaneously, this can be modeled by multiple instantiation of the respective agent — under the assumption that from that agent's perspective the protocol runs are independent of each other.

### 5.2 Isabelle Theory

We base our ISM model on the formalization by Paulson [Pau98]. His so-called "inductive approach" is tailored to semi-automated verification of cryptographic protocols. Its great advantage is a high degree of automation, due to abstraction to the core semantics of the protocols: event traces. On the other hand, this makes both the models and the properties at least cumbersome to express: state information is implicit, yet often it has to be referred to, which is done by repeating suitable parts of the event history and sometimes even by introducing auxiliary events.

For lack of space, we do not show the definitions of the various state and message components since they are straightforward and analogous to the SLE66 model. Moreover, we show only the ISM definition of agent `Bob` as a typical representative.

```
ism Bob =
   ports channel
     inputs   "{NB,IB}"
     outputs  "{BI}"
     messages msg
   state
     control Idle :: B_control
     data    B0   :: B_data
   transitions
 Resp: Idle -> Resp
      in   NB "[Nonce nB]",
           IB "[Crypt (pubK Bob) {|Nonce nA, Agent A|}]"
      out  BI "[Crypt (pubK A) {|Nonce nA, Nonce nB, Agent Bob|}]"
      eff  "Bpeer := A, BnA := nA, BnB := nB"
 Ack': Resp -> Conn
      pre  "nB' = BnB s"
      in    IB "[Crypt (pubK Bob) (Nonce nB')]"
```

Note that `Bob`'s first transition `Resp` takes two inputs, from the nonce generator and the intruder, and produces one output. If we modeled this transition using IOAs, we would have needed three transitions with intermediate states. The precondition of transition `Ack'` could have been made implicit by moving the comparison as a pattern to the **in** part, yet we make it explicit in order to emphasize its importance. The local variable `BnB` serves to remember the value

of the nonce expected, while the other two variables express Bob's view to whom he is connected in which session. In Paulson's approach, this state information is implicit in the event trace.

Modeling the freshness of nonces is intricate. In Paulson's model [Pau98], nonces are generated under the side condition that they do not already appear in the current event history. This criterion refers to the semantic and system-global notion of event traces — something not available from the (local) perspective of an ISM. We solve the problem by introducing a component called NGen that performs the generation of nonces for all agents in a centralized fashion. In this way we can ensure global freshness with a local criterion. Note that this component is just a modeling aid and thus its correct interplay with the agents does not need to be analyzed. We could alternatively express global freshness by adding an axiom restricting system runs in the desired way, yet we prefer the more constructive approach and derive the required freshness property as a lemma.

### 5.3 Properties

Properties of protocols specified with ISMs may be expressed with reference to both the state of agents and the messages exchanged. In the case of NSL, the most interesting property is authentication of Alice to Bob (actually, even session agreement [Low97] from Bob's view), which we formulate as

```
⟦Alice ∉ bad;  Bob ∉ bad;  (b,s)#cs ∈ Runs;
 Bob_state s = (Conn, (|Bpeer = Alice, BnA = nA, BnB = _|))⟧ ⟹
∃(_,s') ∈ set cs.
 Alice_state s' = (Wait, (|Apeer = Bob, AnA = nA|))
```

This can be quite intuitively read as: if in the current state `s` of the system Bob believes to be connected to Alice within a session characterized by the nonce nA then there is an earlier state `s'` where Alice was in the waiting state after initiating a connection to Bob using the same nonce nA.

It is interesting to compare the above formulation with the one given by Paulson:[2]

```
⟦A ∉ bad;  B ∉ bad;  evs ∈ ns_public;
 Crypt (pubK B) (Nonce NB) ∈ parts (spies evs);
 Says B A (Crypt (pubK A) {|Nonce NA,Nonce NB,Agent B|}) ∈ set evs
⟧ ⟹
 Says A B (Crypt (pubK B) {|Nonce NA,Agent A|}) ∈ set evs
```

This statement is necessarily more indirect since the beliefs of the agents have to be coded by elements of the event history. At least in this case, all messages of the protocol run have to be referred to. Note that this formulation makes stronger assumptions than ours because the value of the nonce NB is involved.

On the other hand, due to the extra detail concerning agent state and the input buffers (which are not actually required here), the inductive proofs within

[2] http://isabelle.in.tum.de/library/HOL/Auth/NS_Public.html

the ISM approach are more painful and require more lemmas on intermediate states of protocol runs than Paulson's inductive proofs.

## 6 Conclusion

The Interacting State Machines approach turns out to offer good support for formal security analysis in the way required within an industrial environment. ISMs are designed as high-level I/O automata, with additional structure and communication facilities. Like IOAs, ISMs are suitable for describing typical state-based communication systems relevant for security analysis, where ISM provide increased simplicity wrt. specifying component interaction via buffered communication and means to relate input and output actions directly.

The ISM approach offers graphical representation by means of AutoFocus System Structure Diagrams and State Transitions Diagrams. The graphical views are closely related to the formal system specification and verification via a tool translating the AutoFocus representation to an Isabelle theory.

We have shown that the ISM approach is equally applicable to a variety of security analysis tasks, ranging from high-level security modeling and requirements analysis, typically showing less system structure but increased complexity of state transitions, to security analysis of distributed systems including cryptographic protocols, likely to exhibit advanced system structuring. The examples explicate the importance of a fully formalized strategy, in particular, the LKW model has been significantly improved by identifying hidden assumptions and completing sloppy argumentation.

Further work on ISMs includes the extension of the proof support in the ISM level and the provision of a specification language based on temporal logic. Additional AutoFocus capabilities may be made available, including further systems views like event traces and simulation, as well as test case generation.

**Acknowledgments** We thank Guido Wimmel, Thomas Kuhn and some anonymous referees for their comments on earlier versions of this article.

## References

[AETS01] Atmel, Hitachi Europe, Infineon Technologies, and Philips Semiconductors. Smartcard IC Platform Protection Profile, Version 1.0, July 2001. 217, 220, 222

[AGKS99] David Aspinall, Healfdene Goguen, Thomas Kleymann, and Dilip Sequeira. *Proof General*, 1999. 214

[But99] Michael Butler. csp2B : A practical approach to combining CSP and B. In *Proceedings of FM'99: World Congress on Formal Methods*, pages 490–508, 1999. `http://www.dsse.ecs.soton.ac.uk/techreports/99-2.html`. 214

[CC99] Common Criteria for Information Technology Security Evaluation (CC), Version 2.1, 1999. ISO/IEC 15408. 212, 217

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983. 223

[Fis00] Clemens Fischer. *Combination and implementation of processes and data: from CSP-OZ to Java*. PhD thesis, University of Oldenburg, 2000. 214

[GL98] Stephen J. Garland and Nancy A. Lynch. The IOA language and toolset: Support for designing, analyzing, and building distributed systems. Technical Report MIT/LCS/TR-762, Laboratory for Computer Science, MIT, August 1998. 213

[HSSS96] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. Autofocus - a tool for distributed systems specification. In *Proceedings FTRTFT'96 - Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 467–470. Springer-Verlag, 1996. See also `http://autofocus.in.tum.de/index-e.html`. 214, 215

[ITS91] Information Technology Security Evaluation Criteria (ITSEC), June 1991. 212

[JW01] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *Automated Software Engineering*. IEEE Computer Society, 2001. 214

[Kay01] Dilsun Kirli Kaynar. IOA language and toolset, 2001. `http://theory.lcs.mit.edu/tds/ioa.html`. 213

[LKW00] Volkmar Lotz, Volker Kessler, and Georg Walter. A Formal Security Model for Microprocessor Hardware. In *IEEE Transactions on Software Engineering*, volume 26, pages 702–712, August 2000. `http://www.computer.org/tse/ts2000/e8toc.htm`. 217, 220, 221, 222, 223

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer-Verlag, 1996. 223

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997. 226

[LT89] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. `http://theory.lcs.mit.edu/tds/papers/Lynch/CWI89.html`. 213

[Ohe02a] David von Oheimb. Interacting State Machines, 2002. Submitted for publication. 214, 216, 217

[Ohe02b] David von Oheimb. The Isabelle/HOL implementation of Interacting State Machines, 2002. Technical documentation, available on request. 220, 221, 223

[OLW02] David von Oheimb, Volkmar Lotz, and Gerog Walter. An interpretation of the LKW model according to the SLE66CX322P security target. Unpublished, January 2002. 217

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994. For an up-to-date description, see `http://isabelle.in.tum.de/`. 214

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998. 225, 226

[PNW+] Lawrence C. Paulson, Tobias Nipkow, Markus Wenzel, et al. The Isabelle/HOL library. `http://isabelle.in.tum.de/library/HOL/`. 214

[S+] Oscar Slotosch et al. Validas Model Validation AG. `http://www.validas.de/`. 215

[WW01] Guido Wimmel and Alexander Wisspeintner. Extended description techniques for security engineering. In M. Dupuy and P. Paradinas, editors, *International Conference on Information Security (IFIP/SEC)*. Kluwer Academic Publishers, 2001. 214