DOI 10.1007/s00165-007-0057-0
BCS © 2007
Formal Aspects of Computing (2008) 20: 41–59

Formal Aspects of Computing

Verification of Mondex electronic purses with KIV: from transactions to a security protocol

Dominik Haneberg, Gerhard Schellhorn, Holger Grandy and Wolfgang Reif

Lehrstuhl für Softwaretechnik und Programmiersprachen, Universität Augsburg, D-86135 Augsburg, Germany.
E-mail: haneberg@informatik.uni-augsburg.de

Abstract. The Mondex case study about the specification and refinement of an electronic purse as defined in the Oxford Technical Monograph PRG-126 has recently been proposed as a challenge for formal system-supported verification. In this paper we report on two results.

First, on the successful verification of the full case study using the KIV specification and verification system. We demonstrate that even though the hand-made proofs were elaborated to an enormous level of detail we still could find small errors in the underlying data refinement theory, as well as the formal proofs of the case study.

Second, the original Mondex case study verifies functional correctness assuming a suitable security protocol. We extend the case study here with a refinement to a suitable security protocol that uses symmetric cryptography to achieve the necessary properties of the security-relevant messages. The definition is based on a generic framework for defining such protocols based on abstract state machines (ASMs). We prove the refinement using a forward simulation.

Keywords: Mondex; Refinement; ASM; Verification; Security protocol; Z

1. Introduction

In this paper we describe the efforts done with KIV to solve the challenge of verifying the Mondex refinements.

Our work has two parts. The first half of this paper is concerned with the original development in [SCW00]: refinement of an abstract specification that specifies money transfer using transactions to a communication protocol (the 'concrete level').

We show that verifying the refinement mechanically with KIV can be done with about a person month of effort. The results presented here extend those of [SGHR06a] to the full case study including the operations that archive failure logs from a smart card to a central archive. Since we do not have to repeat a description of the case study, we can give more details and how we encoded the Z specifications in KIV than in [SGHR06a]. We also give some background on KIV in Sect. 2. This should give a better impression how close our work is to the original work.

To do formal proofs for the Mondex case study required that we provide a formalisation of the underlying data refinement theory given in [CSW02]. Section 3 describes a small correction we found when we translated it to formal specifications in KIV. We also give an improved theory that integrates the use of invariants and backward

---

Correspondence and offprint requests to: D. Haneberg. E-mail: haneberg@informatik.uni-augsburg.de simulation for the contract approach to data refinement [WD96]. This improvement allowed us to derive the Mondex development in Sect. 4 as one refinement instead of two as in the original development: the second refinement of [SCW00] is simplified to an invariance proof for the communication protocol. Section 5 describes the verification and gives several small corrections for the invariant.

Mondex Smart Cards have become famous for having been the target of one of the first ITSEC evaluations of the highest level E6 [CB99]. Nevertheless the case study assumes a suitable security protocol rather than proving it. Therefore the second part of our paper discusses a refinement of the communication protocol to a security protocol that uses abstract cryptography.

Section 6 discusses our general approach to the specification of E-commerce protocols in general, that was used in other case studies as well ([HGRS05], [Han06], [GHRS06], [HGRS07]). The approach uses abstract state machines (ASM, [Gur95], [BS03]) and has a generic attacker model.

In Sect. 7 we show how this approach can be applied to define and verify a refinement of the communication protocol of Mondex (i.e. the concrete Mondex level) to a security protocol based on symmetric keys and DES. The verification uses ASM refinement ([BR95], [Sch01], [Bör03]) and the alternative formalisation of the Mondex protocol using ASMs described in [SGHR06a] and [SGH+07].

The work presented here is done in the context of a project in which we develop a systematic approach for the development of E-commerce protocols, starting with informal specifications using UML, using ASMs and ASM refinement for formal verification and ending with verified Java code.

Therefore, current work is in progress to verify Java Card code as a refinement of the security protocol given in this paper. Section 10 gives an outlook on this work and concludes.

For the interested reader, our specifications and proofs as well as the Java code we are currently working on are available as a Web presentation at [KIV].

2. Background on KIV

KIV is an interactive theorem prover based on sequent calculus over many-sorted higher-order Dynamic Logic [HKT00].

The syntax of the logic is

$$\mathsf{S} = \mathsf{bool} \mid \mathsf{nat} \mid \dots$$
$$\mathsf{T} = \mathsf{S} \mid \mathsf{T}^{+} \to \mathsf{T}$$
$$\mathsf{E} = \mathsf{X} \mid \mathsf{OP} \mid \lambda\, \underline{\mathsf{x}}.\ \mathsf{e} \mid \mathsf{e}(\underline{\mathsf{e}'}) \mid \forall\, \underline{\mathsf{x}}.\ \varphi \mid \exists\, \underline{\mathsf{x}}.\ \varphi \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \mid \langle\!|\alpha|\!\rangle\varphi$$

Based on a set of sorts S that always includes booleans and natural numbers, types T are recursively defined to be either sorts or function types. Expressions E are typed (we write $\mathsf{e} : \mathsf{t}$), and boolean expressions $\varphi : \mathsf{bool}$ are used as formulas. An expression is either a variable from some set X, an operation from OP, a lambda expression, an application, a quantified formula or a formula of dynamic logic. Equality, boolean constants true, $\mathsf{false} : \mathsf{bool}$ and operators (e.g. $\wedge : \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool}$) as well as primitive theory of natural numbers that defines 0, successor and an induction principle are built in.

The three dynamic logic operators are used to express properties of a program α. $[\alpha]\varphi$ means "all runs of α that terminate end in a state where φ holds", $\langle\alpha\rangle\varphi$ means "some run of α terminates in a state where φ holds" and $\langle\!|\alpha|\!\rangle\varphi$ expresses "all runs of α lead to a state where φ holds". Using Dijkstra's wp-calculus notation, the formulas are equivalent to $\mathsf{wlp}(\alpha, \varphi)$, $\neg\, \mathsf{wlp}(\alpha, \neg\, \varphi)$ and $\mathsf{wp}(\alpha, \varphi)$.

Programs α can either be a sequential program, an ASM rule or a Java program. All programs use valuations of higher-order variables as states. The dynamic functions of an ASM are represented as function variables. For Java, an extra variable is used to represent the heap (see [Ste04]). The semantics of a program is a relation over states augmented with a $\perp$ element to express nontermination.

KIV's sequent calculus extends standard sequent calculus by rules that execute programs symbolically by computing strongest postconditions [RSSB98].

Basic specifications consist of signatures and axioms over this logic. The semantics of specifications consists of all models (loose semantics). To rule out non-standard models, the semantics can be restricted by clauses, that give constructors for a data type. An example used later on are finite sets, defined in KIV's library: they are generated using the empty set $\emptyset$ and an operation $+\!\!+$ (written infix) that adds an element to a set. Semantically this means that every set can be represented as a (finite) constructor term $\emptyset\ +\!\!+\ \mathsf{a}_1\ +\!\!+\ \mathsf{a}_2\ +\!\!+\ \cdots\ +\!\!+\ \mathsf{a}_n$. For deduction, every generation principle implies a structural induction scheme. Freely generated data types (where different constructor terms always represent different data structures, like natural numbers generated by zero and successor, tuples and enumerations) can be defined using a notation similar to the one of functional languages. From such definitions axioms are generated automatically.

Structured algebraic specifications are built up with the standard operators (see e.g. [CoF04]) union, enrichment, renaming and actualisation of parametric specifications. An additional operation called instantiation that generalises actualisation is explained in the next section, since it is the main operation used to express relations between various forms of data refinement.

KIV also has an extension of this logic with temporal logic operators [BDRS02], which allows one to reason about interleaved programs and statecharts [TOWS04], but this extension is not used here.

3. Specifying the data refinement theory

The data refinement theory underlying the Mondex case study is defined in [CSW02] in three stages: first, the general data refinement theory of [HHS86] is given. Second the contract embedding [WD96] of partial relations is defined and corresponding proof rules for forward and backward simulation are derived. Third the embedding of input and output into the state is discussed.

We have formalized the first two parts of the theory already for [Sch05] using pure algebraic specifications (KIV's extension of Higher-Order Logic to Dynamic Logic is not needed here). A standard encoding of relations as boolean functions is used, since there are no predefined sets or relations in KIV.

As an example, a standard data type $\mathsf{DT} = (\mathsf{GS}, \mathsf{S}, \mathsf{INIT}, \{\mathsf{OP}_i\}_{i \in I}, \mathsf{FIN})$ becomes a generic specification with parameter sorts GS, S and I and operations

$$\mathsf{INIT} : \mathsf{GS} \times \mathsf{S} \to \mathsf{bool} \qquad \mathsf{OP} : \mathsf{I} \to \mathsf{S} \times \mathsf{S} \to \mathsf{bool} \qquad \mathsf{FIN} : \mathsf{S} \times \mathsf{GS} \to \mathsf{bool}$$

The central specification construct needed to relate the three stages of the development of the data refinement theory and also the application of the theory to Mondex is specification *instantiation*, a generalised form of actualisation.

$\mathsf{SP} :=$ **instantiate** $\mathsf{P} < \mathsf{G}$ **with** A **by mapping** σ

is similar to theory interpretation in the IMPS system [Far94] and works as follows: given a (generic) specification G, a subspecification P (the parameter) of G can be instantiated with a theory A (the actual specification). The instantiation uses a mapping σ. Mappings σ generalise morphisms: they allow to map each sort of P to a tuple of types of A and each operation of type t to a tuple of closed expressions of type $\sigma(\mathsf{t})$. σ must also rename operations in $\mathsf{G} \setminus \mathsf{P}$ such that they are disjoint to those of A. The instantiation is correct, if A is more specific than $\sigma(\mathsf{P})$, i.e. if the axioms of A imply $\sigma(\mathsf{Ax})$ for every axiom Ax of P. This must be shown by discharging proof obligations. The resulting specification is $\sigma(\mathsf{G}) \cup \mathsf{A}$. Instantiation can be used for actualisation, e.g. to get lists of natural numbers by instantiating lists ($= \mathsf{G}$) of ordered elements ($= \mathsf{P}$) by natural numbers ($= \mathsf{A}$). σ maps the order of elements to the order on natural numbers, and the axioms for the order on elements have to be proved to hold for natural numbers.

Instantiation can also be used to prove that one theory A is more specific than another theory G. Setting $\mathsf{P} := \mathsf{G}$, the axioms of G have to be proved from the axioms of A. As an example the theory of lattices ($= \mathsf{A}$) can be shown to be more specific than the theory of partial orders ($= \mathsf{G}$) by mapping the predicate $\mathsf{a} < \mathsf{b}$ to $\mathsf{a} \sqcap \mathsf{b} = \mathsf{a}$. In our context, we use specification instantiation several times, to show that one data refinement notion is more specific than another:

- First we define two specifications: BW has the backward simulation conditions

  $$\mathsf{CINIT} \mathbin{;} \mathsf{R} \subseteq \mathsf{AINIT} \qquad \forall\, \mathsf{i} \in \mathsf{I} \bullet \mathsf{COP}_\mathsf{i} \mathbin{;} \mathsf{R} \subseteq \mathsf{R} \mathbin{;} \mathsf{AOP}_\mathsf{i} \qquad \mathsf{CFIN} \subseteq \mathsf{R} \mathbin{;} \mathsf{AFIN}$$

  of the standard data refinement approach of [HHS86] as axioms, and REF has the refinement condition

  $$\forall\, \mathsf{is} \in \mathsf{I}^* \bullet \mathsf{CINIT} \mathbin{;} \mathsf{COP}_{\mathsf{is}} \mathbin{;} \mathsf{CFIN} \subseteq \mathsf{AINIT} \mathbin{;} \mathsf{AOP}_{\mathsf{is}} \mathbin{;} \mathsf{AFIN}$$

as axiom. Setting $\mathsf{P} = \mathsf{G} := \mathsf{REF}$, $\mathsf{A} := \mathsf{BW}$ and σ to be identity gives an instantiated specification SP, where we have to prove that the backward simulation conditions imply refinement.

- To show that the contract approach of Z instantiates general data refinement theory, we set up a specification CBW with the proof obligations for backward simulation in the contract approach (see [WD96] and Theorem 3.1 for a variant which supports invariants). We actualize the parameter of states of SP from the previous item with states that include a bottom element, and map operations to operations in the contract embedding. This gives a specification $\mathsf{SP}_\perp$ with parameter $\mathsf{BW}_\perp$ (the backward conditions of the standard approach, but with states that include a bottom element). By setting $\mathsf{G} := \mathsf{SP}_\perp$, $\mathsf{P} := \mathsf{BW}_\perp$ and $\mathsf{A} := \mathsf{CBW}$, the resulting proof obligations require us to show, that the conditions of backward simulation of the contract approach are implied[1] by the standard backward conditions, when states that include $\perp$ are used. The proof is the standard proof of removing $\perp$ from the proof obligations, just as it is done in [CSW02]. Over the resulting specification CREF the refinement theorem for the contract approach is proved from the instantiated generic refinement theorem.
- Embedding of input and output can be shown to specialise the contract approach by instantiating (global, abstract and concrete) states with the tuples consisting of the original state together with an input and output list. As generic specification we therefore use the specifications of the contract approach: $\mathsf{P} := \mathsf{CBW}$ and $\mathsf{G} := \mathsf{CREF}$. The actual specification $\mathsf{A} := \mathsf{IOBW}$ defines the proof obligations for backward simulation from [CSW02] as axioms (the conditions of Theorem 3.2 below). The mapping maps original operations to the embedded operations which consume one input from the input list, and add one output to the output list. Again we prove that the instances of generic proof obligations follow from those of IOBW exactly as in [CSW02].
- Finally, the Mondex refinement can be derived as an instance of the refinement theory by mapping the abstract state AS of the refinement theory to the tuple of the two variables balance and lost (and similar for CS, see next section for more details). The generic index set I of refinement becomes a finite enumeration type startFrom, startTo, req, val, ack, abort, ignore, increase, clearexlog, readexlog, archive and authclear. COP(val) is the value operation, AOP(val) is AbIgnore, the abstract operation that is refined to COP(val).

The final proof obligations we set up in IOBW are slightly different from the ones in [CSW02], since we found that the embedding used was not fully correct: input and output sequences are embedded into the initialisation and finalisation relation using an empty relation (e.g. empty[GO, CO] in Sect. 4.4.1 to embed output in initialisation). This relation is not total and should be replaced with a relation that relates every input to the empty sequence as output (so empty[GO, CO] is then defined as $\mathsf{GO} \times \{\langle\rangle\}$).

With this corrected definition some of the proofs in Sect. 4.4 must be slightly modified. This results in the additional proof obligation "totality of input initialisation" given below.

Compared to generating proof obligations the approach of proving correctness has the obvious advantage of being adaptable to various refinement theories. This allows to do research in setting up and comparing various types of refinement, like we did in [Sch05] and [BDS07]. Instantiation is slightly less efficient: although generic templates must be instantiated too when generating proof obligations, our approach requires to instantiate *all* theorems of the generic specification, not just the proof obligations.

A technical disadvantage of instantiation is, that the main commutativity proof obligation for backward simulation still is one proof obligation that quantifies about all indices i of the finite enumeration type I. Also the simulation relations and invariants cannot be split into several properties with individual proof obligations. In both cases, appropriate lemmas for individual operations and properties must be defined in KIV manually. Finally, a difference to standard proof obligation generation is that there are no conventions about corresponding names on the abstract and concrete level: this gives flexibility, but also requires to set up the correspondence between all operation names explicitly in the mapping.

In the original Mondex case study [SCW00] the proof obligations are applied restricting the state space of the concrete level to those states for which an invariant holds (the 'between' level), and the second refinement basically proves that this invariant indeed holds. This approach can be slightly improved to one refinement by adding invariants directly to the refinement theory.

Adding invariants is trivial for forward simulation, where one can just form a conjunction of invariants and simulation relation (see Theorem 2.4.2 in [DB01]), but it is nontrivial for backward simulation. To prove that it is admissible, we had to go back to backward simulation in the contract approach and prove the following theorem by instantiating the original approach of [HHS86].

[1] we usually prove the reverse implication too, to show that the proof obligations are maximally general.

Theorem 3.1 (Backward Simulation using Invariants) Given an abstract data type $\mathsf{ADT} = (\mathsf{AINIT}, \mathsf{AIN}, \mathsf{AOP}, \mathsf{AFIN}, \mathsf{AOUT})$ with

- total $\mathsf{AINIT} \subseteq \mathsf{GS} \times \mathsf{AS}$
- $\mathsf{AOP_i} \subseteq \mathsf{AS} \times \mathsf{AS}$,
- total $\mathsf{AFIN} \subseteq \mathsf{AS} \times \mathsf{GS}$

a similar data type $\mathsf{CDT} = (\mathsf{CINIT}, \mathsf{CIN}, \mathsf{COP}, \mathsf{CFIN}, \mathsf{COUT})$ which uses states from CS instead of AS, a backward simulation $\mathsf{T} \subseteq \mathsf{CS} \times \mathsf{AS}$ and two invariants $\mathsf{AINV} \subseteq \mathsf{AS}$ and $\mathsf{CINV} \subseteq \mathsf{CS}$, then the refinement is correct using the contract approach when the following proof obligations (in Z [Spi92] notation) hold:

- $\mathsf{CINIT} \subseteq \mathsf{CINV}$, $\mathsf{AINIT} \subseteq \mathsf{AINV}$ (initially invariants)
- $\mathrm{ran}(\mathsf{AINV} \lhd \mathsf{AOP_i}) \subseteq \mathsf{AINV}$, $\mathrm{ran}(\mathsf{CINV} \lhd \mathsf{COP_i}) \subseteq \mathsf{CINV}$ (invariance)
- $(\mathsf{CINIT} \rhd \mathsf{CINV}) \fatsemi \mathsf{T} \subseteq \mathsf{AINIT}$ (initialisation)
- $(\mathsf{CINV} \lhd \mathsf{CFIN}) \subseteq \mathsf{T} \fatsemi (\mathsf{AINV} \lhd \mathsf{AFIN})$ (finalisation)
- $\forall\, \mathsf{i} \in \mathsf{I} \bullet \mathrm{dom}(\mathsf{COP_i}) \ntriangleleft \mathsf{CINV} \subseteq \mathrm{dom}((\mathsf{T} \lhd \mathsf{AINV}) \ntriangleleft \mathrm{dom}(\mathsf{AOP_i}))$ (applicability)
- $\forall\, \mathsf{i} \in \mathsf{I} \bullet \mathrm{dom}(\mathsf{T} \ntriangleleft \mathrm{dom}(\mathsf{AOP_i})) \ntriangleleft (\mathsf{COP_i} \fatsemi \mathsf{T}) \subseteq \mathsf{T} \fatsemi (\mathsf{AINV} \lhd \mathsf{AOP_i})$ (correctness)

The proof proceeds like the standard proof for the contract approach [WD96]. It uses the same embedding for operations, but the embedding for the simulation relation T is

$$\mathring{\mathsf{T}} \mathrel{\hat{=}} (\mathsf{T} \rhd \mathsf{AINV}) \cup \{\mathsf{CS}_\perp \setminus \mathsf{CINV}\} \times \mathsf{AS}_\perp$$

instead of $\mathring{\mathsf{T}} \mathrel{\hat{=}} \mathsf{T} \cup \{\perp\} \times \mathsf{AS}_\perp$. Based on this theorem, input and output can be added to prove (we give implicitly universally quantified proof obligations as defined in KIV):

Theorem 3.2 (Backward Simulation with IO using Invariants)
Assume an abstract data type $\mathsf{ADT} = (\mathsf{AINIT}, \mathsf{AIN}, \mathsf{AOP}, \mathsf{AFIN}, \mathsf{AOUT})$ consisting of

- parameter sorts $\mathsf{GS}, \mathsf{GI}, \mathsf{GO}, \mathsf{AS}, \mathsf{AI}$ and AO
- $\mathsf{AINIT} : \mathsf{AS} \to \mathsf{bool}$ (the set of initial states)
- $\mathsf{AIN} : \mathsf{GI} \times \mathsf{AI} \to \mathsf{bool}$, (inputs initialised from global inputs)
- $\mathsf{AOP} : \mathsf{I} \to \mathsf{AI} \times \mathsf{AS} \times \mathsf{AS} \times \mathsf{AO} \to \mathsf{bool}$ (operations read an input, modify the state and produce output)
- $\mathsf{AFIN} : \mathsf{AS} \times \mathsf{GS} \to \mathsf{bool}$ (finalising a local state gives a global state)
- $\mathsf{AOUT} : \mathsf{AO} \times \mathsf{GO} \to \mathsf{bool}$ (finalising output to global output)

together with an invariant $\mathsf{AINV} : \mathsf{AS} \to \mathsf{bool}$ and a concrete data type $\mathsf{CDT} = (\mathsf{CINIT}, \mathsf{CIN}, \mathsf{COP}, \mathsf{CFIN}, \mathsf{COUT})$ with invariant $\mathsf{CINV} : \mathsf{CS} \to \mathsf{bool}$ are given. CDT uses the same global data $\mathsf{GI}, \mathsf{GS}, \mathsf{GO}$ but different local data $\mathsf{CI}, \mathsf{CS}, \mathsf{CO}$. Then a backward simulation consisting of $\mathsf{IT} : \mathsf{CI} \times \mathsf{AI} \to \mathsf{bool}$, $\mathsf{T} : \mathsf{CS} \times \mathsf{AS} \to \mathsf{bool}$ and $\mathsf{OT} : \mathsf{CO} \times \mathsf{AO} \to \mathsf{bool}$ proves correctness of the refinement (in the same sense as in [CSW02]), if the following proof obligations can be verified:

$$\begin{array}{ll} \quad\ \mathsf{CINV(cs)} \wedge \mathsf{COP(i)(cin, cs, cs', cou')} \wedge \mathsf{T(cs', as')} \wedge \mathsf{AINV(as')} \wedge \mathsf{OT(cou', aou')} & \\ \ \wedge\, (\forall\, \mathsf{as, ain.\ T(cs, as)} \wedge \mathsf{AINV(as)} \wedge \mathsf{IT(cin, ain)} \to \mathsf{(ain, as)} \in \mathrm{dom}(\mathsf{AOP(i)})) & \\ \to \exists\, \mathsf{as, ain.\ IT(cin, ain)} \wedge \mathsf{T(cs, as)} \wedge \mathsf{AINV(as)} \wedge \mathsf{AOP(i)(ain, as, as', aou')} & \text{(correctness)} \end{array}$$

$$\begin{array}{ll} \ \ \mathsf{CINV(cs)} \wedge \mathsf{(cin, cs)} \notin \mathrm{dom}(\mathsf{COP(i)}) & \\ \to \exists\, \mathsf{as, ain.\ T(cs, as)} \wedge \mathsf{AINV(as)} \wedge \mathsf{IT(cin, ain)} \wedge \mathsf{(ain, as)} \notin \mathrm{dom}(\mathsf{AOP(i)}) & \text{(applicability)} \end{array}$$

$$\begin{array}{ll} \mathsf{AINIT(as)} \to \mathsf{AINV(as)} & \text{(initially abstract invariant)} \\ \mathsf{CINIT(cs)} \to \mathsf{CINV(cs)} & \text{(initially concrete invariant)} \\ \mathsf{CINV(cs), COP(i)(cin, cs, cs', cou')} \to \mathsf{CINV(cs')} & \text{(abstract invariant preserved)} \\ \mathsf{AINV(as), AOP(i)(ain, as, as', aou')} \to \mathsf{AINV(as')} & \text{(concrete invariant preserved)} \\ \mathsf{CINIT(cs)} \wedge \mathsf{T(cs, as)} \to \mathsf{AINIT(as)} & \text{(state initialisation)} \\ \mathsf{CIN(gin, cin)} \wedge \mathsf{IT(cin, ain')} \to \mathsf{AIN(gin, ain')} & \text{(input initialisation)} \\ \mathsf{CINV(cs)} \wedge \mathsf{CFIN(cs, gs)} \to \exists\, \mathsf{as.\ AINV(as)} \wedge \mathsf{T(cs, as)} \wedge \mathsf{AFIN(as, gs)} & \text{(state finalisation)} \\ \mathsf{COUT(cou, gou')} \to \exists\, \mathsf{aou.\ OT(cou, aou)} \wedge \mathsf{AOUT(aou, gou')} & \text{(output finalisation)} \\ \exists\, \mathsf{cs.\ CINIT(cs)} & \text{(totality of state initialisation)} \end{array}$$

$\exists$ cin′. CIN(gin, cin′) (totality of input initialisation)
CINV(cs) $\rightarrow \exists$ gs′. CFIN(cs, gs′) (totality of state finalisation)
$\exists$ gou′. COUT(cou, gou′) (totality of output finalisation)
$\exists$ ain. IT(cin, ain) (totality of input)

The proof of this theorem uses the corrected *empty*-relation and instantiates the previous theorem, but otherwise proceeds like the one in [CSW02]. The new proof obligations now can be used to verify the two Mondex refinements in one instead of two steps.

4. Specification of the Mondex refinement

The two most important concepts of Z are its built-in set theory, and the use of schemata to build up and structure specifications. To translate a Z specification into a KIV specification, one has to express these concepts in the higher-order dynamic logic KIV uses.

For sets there are basically two options, depending on whether we want finite sets or arbitrary ones. For finite sets KIV's library offers a predefined data type as described in Sect. 2.

Infinite sets like toInEpv[2] are usually represented as a characteristic predicate toInEpv, just like the operations of the last section.

An alternative is to use a specification of infinite sets. We defined such a specification for the ether, since it is directly modified by operations on sets in the protocol. The specification was copied from the original specification of the library, and the term generatedness axiom was removed, which caused KIV's correctness management to leave only those theorems valid which had a proof that did not depend on finiteness of the sets. The theorems that became invalid were removed. A function {p} (written as brackets around the argument to have an intuitive notation) is added to the specification that allows to construct the set of all elements that satisfy the characteristic predicate p. {p} is the set {a : p(a)}. Using the test predicates isStartFrom and isStartTo that check whether a message is a startFrom resp. startTo message the initial ether is specified as (using overloading for the brackets: $\{\perp\}$ and {readExLog} are singleton sets)

$$\text{ether} = \{\text{isStartFrom}\} \cup \{\text{isStartTo}\} \cup \{\perp\} \cup \{\text{readExLog}\}$$

Schemata are used in Z for various purposes. There are some in the Mondex case study that just define free data types, like messages. Others define invariants like the specification of BetweenWorld. Still others define data types and restrictions on them, like PayDetails, which defines a five tuple and the restriction from $\neq$ to. Still others define operations and their composition.

In KIV these issues are separated: Data types are put into specifications, e.g. messages and PayDetails are specified as free data types. The invariants of BetweenWorld as well as the restriction from $\neq$ to are encoded as predicates. Schemata that define operations are translated to ASM rules. This suggests itself, since operations will be implemented by programs on smart cards. It also makes control structure explicit. This can be exploited by KIV's heuristics that symbolically execute programs (by computing strongest postconditions), improving automation of proofs.

Since the semantics of programs is a relation too, the translation is relatively simple: Equations $x' = f(x)$ become assignments $x := f(x)$ and nondeterministic relations like nextSeqNo′ > nextSeqNo are translated to nondeterministic choice:

choose n **with** n > nextSeqNo **in** nextSeqNo := n

Preconditions of schemata are encoded as the tests of conditionals. Schema composition and disjunction are translated to compounds and nondeterministic choice, e.g. Abort ⨾ StartFrom $\vee$ Abort is translated to ABORT#; STARTFROM# $\vee$ ABORT# in KIV. The # sign is added by convention to distinguish program identifiers from predicates. The refinement theory of Mondex from [CSW02] requires that all operations must be total. This is naturally expressed as the requirement that the corresponding ASM rules should terminate.

[2] [WF06] noted that toInEpv is specified in [SCW00] as a *finite* set, resulting in an inconsistency. We specified a predicate, not noting the requirement.

Schema promotion that is used to lift operations from one purse to the set of authentic purses is not directly available in KIV. In our encoding of operations as programs we could have used auxiliary ASM rules, that have one purse as argument. We decided not to do that, since it prevents the following simplification of the state: the original specification defines a (partial) function AbAuthPurse (and similarly ConAuthPurse) that maps authentic names of purses to a tuple of name, abalance and lost. A constraint enforces that the duplicated name in the domain and range of this function are the same. Our specification simply uses two total functions abalance and lost from names to the respective values and thereby avoids the duplication. Authentic purses (the domain of AbAuthPurse) is characterised by the predicate authentic. We abbreviate the tuple of abalance and lost as astate in the following. For the concrete state we similarly have cstate = balance, exLog, status, nextSeqNo, pdAuth, ether, archive.

The translation of each Z operations OP results in an always terminating program OP#, whose semantics is a relation on the state modified by OP#. This state consists of the tuple of state variables used in the program (state := balance, lost for the abstract level of Mondex).

To apply the algebraic refinement theory of the previous section, we need to lift the relational semantics of ASM rules to an algebraic operation on this state. This is easily done by the axiom

$$\mathsf{OP}(\mathsf{state}, \mathsf{state}') :\leftrightarrow \langle \mathsf{OP\#} \rangle\ \mathsf{state} = \mathsf{state}' \tag{1}$$

Formally, this defines relation OP to hold between state and state′ iff the program OP# has some terminating run that starts in state and modifies state to be equal to state′ at the end of the run.

As an example we show the translation of the operation handling requests (the promoted ReqPurseOkay from p. 32 of the original work):

REQ#(msg, receiver)
if msg ∈ ether ∧ msg = req(pdAuth(receiver)) ∧ state(receiver) = epr
then balance(receiver) := balance(receiver) − pdAuth(receiver).value
    status(receiver) := epa
    outmsg := val(pdAuth(receiver))
else IGNORE#

The conditions of the test msg = req(pdAuth(receiver)) (AuthenticReqMessage expanded) and status = epr are from ReqPurseOkay, the test msg ∈ ether is from expanding the promotion scheme ΦBOp, p.45. The **else** case makes the fact explicit, that when the test is negative, only an Ignore can be executed in the schema disjunction (p. 50) that defines Req:

$$\mathsf{Req} \mathrel{\widehat{=}} \mathsf{Ignore} \vee \exists\, \Delta\mathsf{ConPurse} \bullet \Phi\mathsf{BOp} \wedge \mathsf{ReqPurseOkay}$$

Finding out, which operation is possible in which situation, by collecting the preconditions of the various schemata used was one of the main difficulties we had to understand the Z specification. In our translated version, the different cases are explicit. Finally the Req operation is defined, using the schema (1) above:

Req(msg, cstate, cstate′, outmsg′)
↔ ⟨IGNORE#(; outmsg) ∨ **choose** receiver **with** authentic(receiver) **in** REQ#; SENDMSG#(outmsg)⟩
    (cstate = cstate′ ∧ outmsg = outmsg′)

SENDMSG# adds the output message outmsg to the ether. Since we prove the refinement from abstract to concrete level directly SENDMSG# may also drop messages when constructing the new ether:

SENDMSG#(outmsg)
choose ether′ **with** ether′ ⊆ ether ∪ {outmsg} **in** ether := ether′

Theorem 3.2 is applied using the Req operation as one COP(i). Simulation relations R, RIN, ROUT as well as the definitions of global states, inputs and outputs are copied from [SCW00]. An invariant AINV for the abstract states is not needed, we set it to true. The only difficult part is the definition of the invariant CINV for the concrete level. We found that the required formulas to do this are distributed in 3 places:

- The property of payment details that requires pd.from ≠ pd.to for every relevant pd used (Sect. 4.3.2).
- The properties of purses P-1 to P-4 (Sect. 4.6).
- The properties B-1 to B-16 of the intermediate state that define an invariant for the concrete state (Sect. 5.3).

Collecting these properties and the required definitions of AuxWorld (Sect. 5.2) gives a suitable definition of CINV: full details for the version without archives can be found in the technical report [SGHR06b], the variant used here adds the two original properties (with the small correction described below) for archives from BetweenWorld.

To allow direct verification of the refinement from the abstract to the concrete level, there is still one detail we have to add: the concrete ether may lose messages, while BetweenWorld gives properties for an ether that only hold if ether has *not* lost messages. Therefore we collect all properties that talk about the ether in a predicate etherok(ether, ...) and we use

$$\exists\ \mathsf{fullether}.\ \mathsf{ether} \subseteq \mathsf{fullether} \land \mathsf{etherok}(\mathsf{fullether}, \ldots)$$

in the invariant CINV. fullether is always the ether that has never dropped any message.

Otherwise there are the following minor technical differences and optimisations:

- The distinction between states eaFrom and eaTo is unnecessary. Therefore these states have been merged into one idle state.
- Instead of directly using quantified formulas when defining CINV we have defined predicates for them. This allows to define rewrite rules for the predicates, which often avoids the need to instantiate the quantifier in the main proof.
- We have grouped related properties (e.g. properties 9-12 of BetweenWorld that concern the ether) under one universal quantifier. This allows to instantiate the quantifier of several related properties at once.
- Predicates are not defined in the context of Z schemata, which provide implicit references to prior definitions. We have to provide parameters explicitly. For example, the set fromLogged (defined in Sect. 5.2. of [SCW00]) does not just become a predicate with argument pd of type PayDetails. Since it's definition depends on allLogs, which in turn depends on the exception logs exLog and the archive, the predicate has three arguments fromLogged(exLog, archive, pd)
- Set maybeLost (and similarly definitelyLost and chosenLost) must be a finite set in the definition of the simulation relation R. Since this set is not finite a priori, but only during the run of the system (where always only a finite number of purses may be in a protocol run), we have specified a predicate maybeLost. To enforce, that during runs the set is finite one has to claim that

  $$\exists\ \mathsf{maybelostSet}.\ \forall\ \mathsf{pd}.\ (\mathsf{pd} \in \mathsf{maybeLostSet} \leftrightarrow \mathsf{maybeLost}(\mathsf{pd}))$$

  where maybeLostSet is a variable that ranges over finite sets. The simulation relation R therefore existentially quantifies over the current sets maybelostSet, definitelyLostSet and chosenLostSet.
- Finiteness of the number of authentic purses (necessary to compute the sum of all balances) and the existence of at least one authentic purse (necessary to ensure totality of operations) is specified by a predicate authentic and the axiom

  $$\exists\ \mathsf{authenticSet}.\ \mathsf{authenticSet} \neq \emptyset \land\ \forall\ \mathsf{na}.\ (\mathsf{na} \in \mathsf{authenticSet} \leftrightarrow \mathsf{authentic}(\mathsf{na}))$$

  Again, authenticSet ranges over finite sets. The existence of at least two purses, necessary to enforce consistency of the Z specification of payment details[3] is not necessary in KIV, since the restriction $\mathsf{from} \neq \mathsf{to}$ is just used in the invariant of the communication protocol, not to restrict the (free) type of payment details.

5. Verification of the Mondex refinement

The only difficult proof obligations in the proof of refinement according to Theorem 3.2 are "Concrete invariant preserved" and "Correctness" for every protocol operation.

To prove these two proof obligations, we first reduce them to properties of the elementary programs REQ#, IGNORE#, ABORT# etc. This step roughly corresponds to the derivation of lemmas for the individual operations in [SCW00], but we did not use the original proof structure any further. For the Req operation, whose definition we gave in the previous section, we get one lemma for invariance

$$\begin{array}{l} \quad\quad \mathsf{authentic}(\mathsf{receiver}) \land \mathsf{msg} \in \mathsf{ether} \land \mathsf{msg} = \mathsf{req}(\mathsf{pdAuth}(\mathsf{receiver})) \land \mathsf{state}(\mathsf{receiver}) = \mathsf{epr} \\ \quad \land\ \mathsf{CINV}(\mathsf{cstate}) \land \langle \mathsf{REQ\#}(\mathsf{msg}, \mathsf{receiver})\# \rangle\ \mathsf{cstate} = \mathsf{cstate}' \\ \to \mathsf{CINV}(\mathsf{state}') \end{array}$$

and one for backward simulation. Since Req refines AbTransfer this lemma is

[3] again, this inconsistency was found in [WF06].

$$\begin{aligned}&\mathsf{authentic(receiver)} \wedge \mathsf{msg} \in \mathsf{ether} \wedge \mathsf{msg} = \mathsf{req(pdAuth(receiver))} \wedge \mathsf{state(receiver)} = \mathsf{epr}\\ &\wedge \langle \mathsf{REQ\#(msg, receiver)\#} \rangle\ \mathsf{cstate} = \mathsf{cstate}' \wedge \mathsf{R(cstate', astate')} \wedge \mathsf{CINV(cstate)}\\ &\rightarrow \exists\ \mathsf{astate.}\ \mathsf{R(cstate, astate)} \wedge \langle \mathsf{ABTRANSFER\#} \rangle\ (\mathsf{astate} = \mathsf{astate}')\end{aligned}$$

The preconditions guarantee, that the test of the conditional in the definition of REQ# is positive, the lemma for IGNORE# covers the negative case. The proofs of the lemmas then are done using the sequent calculus of KIV, that reduces the initial goal to simpler goals until axioms are reached.

For the first lemma the proof starts automatically by symbolic execution of REQ#. In this case we get one premise, other lemmas where the program contains a conditional (e.g. for Abort in LogIfNeeded) give several premises. In each premise the two occurrences of CINV in the pre- and postcondition are then unfolded automatically by heuristics. This leaves a conjunction of properties to prove. Those properties which are unchanged by REQ# are trivially implied by the corresponding property from the precondition. Some others can be proved automatically using rewrite rules. For the rest, in particular when the proof requires to use *several other* properties from the precondition, some interaction is necessary to unfold definitions and to instantiate quantifiers. In a few cases, where we were unsure whether a goal was provable, it was helpful to be able to look up the corresponding case in [SCW00]. This probably saved one or two days of extra work.

For the second lemma the proof structure is similar, but there are two additional quantifiers to instantiate. The first is the existential quantifier for astate = abalance, lost. For all operations that do not transfer money, astate can be set to be astate′.

The two exceptions are REQ# and the case of ABORT# where logging takes place. For REQ# there are three cases, which are split manually. In the first case toInEpv is false, the two others depend on whether pdAuth(receiver) ∈ chosenlost. These cases correspond to the ones in Sect. 18.3 of [SCW00] e.g. in the first case astate must modify astate′ by moving pdAuth(receiver).value from lost(receiver) to abalance(receiver) to accommodate the fact, that AbTransferLost will be executed.

The second quantifier is the quantification over the sets maybeLostSet, definitelyLostSet, chosenLostSet. These sets must be instantiated with modified variants of their value maybeLostSet′, definitelyLostSet′ and chosenLostSet′ after the operation. The instances again must be given interactively. Getting these instances right is the main creative step of the proof. Since the correct instances are hidden in various subproofs in [SCW00] we summarise them here:

- None of the three sets changes in STARTFROM# and STARTTO# since aborting directly after these transitions will not lose money. Dually, the sets remain unchanged in ACK#, since the money has already been transfered successfully. The sets also are unchanged in IGNORE# and INCREASE# as well as in the operations for archiving exception logs.
- When REQ# executes successfully, there are two cases to consider: either the to purse who sent the request is still in state epv: then the payment details of the request message enter maybelost and possibly chosenlost. Reasoning backwards the payment details must be deleted from both sets. Otherwise the to purse has already aborted the transaction after sending the request, so reasoning backwards the payment details must be deleted from definitelylost. The two cases correspond to the two cases of a successful respectively failed transaction in ABTRANSFER# (all other operations refine skip).
- Successful execution of VAL# means that the payment details of the message leave maybelost, so reasoning backwards they must be added to maybelost.
- For ABORT# there are three cases to consider: First, if the purse does not log (in LOGIFNEEDED#), then no critical transaction is in progress and all sets remain unchanged. Second, if the purse logs and is in state epv, and if the corresponding from purse is either in state epa or has already logged the payment details, then aborting moves the payment details from maybelost and chosenlost to definitelylost. Reasoning backwards the payment details must be added to maybelost and chosenlost, and deleted from definitelylost. In the remaining third case the payment details are already in definitelylost and all sets remain unchanged.

Doing the proofs in KIV showed several minor flaws, that resulted in proof goals which could not be closed. Each time the problem could be traced back to a specific point of the proof in [SCW00], where the argument must be revised.

- The first problem is in Sect. 29.4 in the proof of B-10 where it must be proved that

$$\mathsf{toInEpv} \vee \mathsf{toLogged} \Rightarrow \mathsf{req} \wedge \neg\ \mathsf{ack}$$

for all payment details pd. Now the problem is as follows: the implication *is* provable for pdAuth(receiver), where receiver is the (to) purse receiving the val message (to which it responds with an ack message). But this is not sufficient: if it would be possible that receiver is *different* from na := pdAuth(receiver).to but has status(na) = epv and pdAuth(na) = pdAuth(receiver), then for *this* na the implication would be violated. The solution to this problem is to add pdAuth(receiver).to = receiver when status(receiver) = epv to P-3.

- A similar problem also exists when status(receiver) = epa (property P-4). For this case the formula pdAuth(receiver).from = receiver has to be added.
- The fact that every val(pd) message in the ether has authentic(pd.from) has to be added: like property authentic(pd.to) (B-1) is needed to have pd.to in the domain of the partial function ConAuthPurse in B-2, this property is needed in order to have a determined value for ConAuthPurse pd.from in B-3.
- Empty sets of payment details must be avoided in exLogResult and exLogClr messages, since the hash function is defined to be injective for nonempty sets only.
- Assertions that pdAuth(receiver).to resp. .from must be authentic in P-3 and P-4. Our early proof attempts lacked the authentic clauses in the definition of the predicates toInEpr, toInEpv and toInEpa. Without these clauses the assertions were definitely necessary. After the correction we did not check whether the addition was still necessary, but [RJ06] and [WF06] confirm that they still are.

With these modifications the invariance proof was successful. The final proofs of the invariance lemmas together have 889 proof steps and 189 interactions.

The lemmas for the simulation proof are similarly difficult, they required 752 proof steps and 173 interactions. Deriving the proof obligations “Concrete invariant preserved” and “Correctness” of Theorem 3.2 from the lemmas, proving the remaining proof obligations and auxiliary lemmas requires a lot of technical overhead (1,467 proof steps and 261 interactions) but these proofs are all much simpler. Also none of these proofs were affected by the corrections of the invariant.

When we started this case study we first specified two abstract state machines (ASMs, [Gur95], [BS03]) and tried the two main proofs for these. With this approach we already found all the problems present in the main protocol (see [SGHR06a] for more details) except the correction for the archiving protocol, which was not specified. Therefore the effort to do the full case study was as follows

- 1 week was needed to get familiar with the case study and to set up the initial ASMs.
- 1 week was needed to prove the essential proof obligations “correctness” and “invariance”, and to get a correct invariant.
- 1 week was needed to specify the Mondex refinement theory of [CSW02] and to generalise the proof obligations to cope with invariants (Sect. 3).
- 1 week was necessary to prove the data refinement as described in this section. This short time is due to the fact, that the lemmas for invariance and simulation are nearly the same as the ones we proved for the ASM.
- 3 days were needed to add the archiving protocol. Since the protocol for archiving is independent of the main protocol, most proofs can be replayed. The two new properties that have to be added to the invariant increase the proof size somewhat, but they do not cause any difficult problems.

Altogether one person month was needed to formally prove the Mondex case study. Of course the time needed to do the verification and the automation of proofs is strongly influenced by the level of expertise with formal verification in general and with the KIV system in particular.

Also getting the work done in this time was immensely helped by having a (nearly) correct simulation relation. Usually most of the time is not needed to verify the correct solution, but to find invariants and simulation relations incrementally.

KIV offers some help for incremental development by allowing proofs to be replayed after corrections, and by a correctness management, that keeps track of proof dependencies between Lemmas, and that invalidates only a minimal set of theorems on changes (see [RSSB98] for more details).

Nevertheless the effort needed to develop a solution is still much bigger that the effort to verify an existing solution. This can also be seen from our effort to develop a systematic verification using ASM refinement, which took 2 months (see [SGH+07]).

The effort required for Mondex can be compared to the effort required for refinement proofs from another application domain which we did at around the same time as the original Mondex case study: verification of a compiler that compiles Prolog to code of the Warren abstract machine ([SA97], [SA98], [Sch99], [Sch01]). This case study was based on refinement steps and mathematical proofs from [BR95], which were much less detailed than the proofs of Mondex in [SCW00]. The case study required 9 refinements, and the statistical data ([Sch99, Chap. 19]) show that each refinement in this case study needed on average about the same number of proof steps in KIV as the Mondex case study. The overall effort was half a year, which gives an effort per refinement that is comparable to the Mondex effort.

The ratio of interactions to proof steps is somewhat better in the WAM case study, since automation of refinement proofs increases over time: investing time to improve automation by adding rewrite rules becomes more important when similar steps are necessary in several refinements and when proofs have to be repeated due to iterative development.

6. The PROSECCO framework

PROSECCO (**Pro**tocols for **Sec**ure **Co**mmunication) is an ASM-based framework for the specification and verification of smart card applications. PROSECCO is described in [Han06]. It combines UML diagrams for the description of the application with a formal protocol model based upon ASMs. The formal model is the basis for the verification of properties, either by refinement of by direct proof attempts. In the graphical model a class diagram is used to specify the state of the agents, a deployment diagram describes the communication network and the attacker's abilities, activity diagrams are used to model the security protocols. *Agents* represent the active components taking part in the application; in the Mondex case study these are the Mondex purses, the Mondex wallets (card terminals), the users and the attacker.

The formal PROSECCO model of an application consists of two parts, first a structured algebraic specification defining data types, the communication network and the attacker. Second, an ASM describing the protocol steps which can be performed by the agents of the application.

The algebraic specification consists of a library of basic definitions concerning the possible communication between the agents which is extended by some application specific axioms which tailor the library to the concrete application. An important part of the algebraic specification is the model of the communication network and the description of the abilities of the attacker. PROSECCO uses a detailed model of the communication based on *connections* representing possible communication links between the different *ports* of the agents which represent communication interfaces (cf. [HGRS07]). For each connection the specification contains axioms determining if the attacker can read, manipulate or suppress data transmitted over the connection. PROSECCO is therefore not limited to a Dolev-Yao [DY81] style attacker, i.e. an attacker that has unrestricted control over the complete communication, can intercept messages, can fake messages using information currently stored in its memory, can act like a regular participant of the protocol but cannot break cryptography. The attacker is represented in the ASM by his knowledge attacker–known. This is the set of data the attacker has acquired by eavesdropping on the communication and analysing the data. The set is extended when the attacker learns new messages and it is used by the attacker to create new messages on his own.

The PROSECCO ASM specifies the dynamic aspects of the application, it describes operationally all steps which are possible for the different agents of the application. The agents are modeled with an explicit internal state used to store application specific data which is manipulated by the ASM rule. This is different to most approaches for cryptographic protocol verification since in these approaches the agents do not have a state, they usually only have certain data for cryptography, e.g. a key, which is not modified in the protocol runs. The value of former nonces is taken from the trace of the current run (see e.g. [Pau98]). Similar to other approaches is the idea of an application-independent data type to model the transmitted data and a uniform model of the attacker's treatment of these data.

The PROSECCO approach uses a uniform model of the data used for communication to give way to an application independent treatment of the attacker and to the possibility to build a library of reusable parts for the PROSECCO ASMs. This data type definition is inspired and quite similar to the messages (data type msg) defined in [Pau98]. Our document model consists of two mutually recursive freely generated data types, document and documentlist[4]:

document = ⊥
| intdoc(. .int : int) **with** is–intdoc
| keydoc(. .key : key) **with** is–keydoc
| noncedoc(. .nonce : nonce) **with** is–noncedoc

[4] dots indicate pre-, post- and infix operands.

| secretdoc(. .secret : secret) **with** is–secretdoc
| hashdoc(. .doc : document) **with** is–hashdoc
| encdoc(. .key : key; . .doc : document) **with** is–encdoc
| sigdoc(. .key : key; . .doc : document) **with** is–sigdoc
| doclist(. .list : documentlist) **with** is–doclist
documentlist = []
| . + . (. .first : document; . .rest : documentlist)

A document can therefore be a simple number (intdoc), some secret information (secretdoc), data used in cryptography (noncedoc, keydoc), the result of a cryptographic operation (hashdoc, encdoc, sigdoc) or a list of documents (doclist). Based on these data types the axioms for analysing and constructing documents are given, e.g. the axiom

$$\mathsf{encdoc(key, doc)} \in \mathsf{attacker\text{–}known} \wedge \mathsf{key} \in \mathsf{attacker\text{–}known} \wedge \mathsf{symmetric(key)} \rightarrow \mathsf{doc} \in \mathsf{attacker\text{–}known}$$

states that a symmetrically encrypted document can be decrypted if the correct key is available. Analysing documents is specified similar to the function analz of [Pau98]. Our model of cryptography is based on the usual perfect cryptography assumption: decryption without the correct key is impossible, generation of a hash value is injective. The axioms for generating a new document doc from attacker knowledge (attacker–known $\vdash$ doc) are somewhat dual to the axioms for analysing documents, e.g. a signature can be constructed if the used key is known:

$$\begin{aligned} &(\mathsf{attacker\text{–}known} \vdash \mathsf{key}) \wedge (\mathsf{attacker\text{–}known} \vdash \mathsf{doc}) \\ \rightarrow &(\mathsf{attacker\text{–}known} \vdash \mathsf{sigdoc(key, doc)}) \end{aligned}$$

The formalisation of the generation of documents is similar to the function synth of [Pau98].

7. Extending the Mondex challenge

The original Mondex case study omits one aspect which is rather important for the security of the application: how is it ensured that the req, val and ack messages cannot be forged by the attacker? In the original work it is just assumed that it is possible to secure the messages needed for the communication protocol.

As proposed in [SGHR06a] we developed a cryptographic protocol which corresponds to the Mondex value transfer protocol and uses symmetrically encrypted documents to represent req, val and ack. This extension bridges the gap between the analysis of Mondex as it was done in [SCW00] and the usual verification of cryptographic protocols. It considers the properties of cryptographic methods and therefore ensures that the assumed unforgeability of the important messages of the Mondex protocol is effectively guaranteed. Our refinement also introduces finiteness constraints (e.g. finite number of entries in the exception logs and a maximum current balance of 32767, the greatest number that fits in a signed short) that are a first step towards a Java implementation.

7.1. A protocol with symmetric encryption

According to [Cla96] at least the first versions of Mondex cards used symmetric encryption, so the protocol we proposed in [SGHR06a] is not unlikely. The security of the communication protocol of the concrete Mondex level is based on the fact that no future req, val or ack messages are available to the attacker, i.e. contained in the ether. The cryptographic protocol must be designed in a way that ensures that this is actually true. Since the real security protocol of the Mondex smart cards has never been published, our protocol must be seen as a proposal. Alternatives using RSA would have payloads that would be too big to be transferred by single messages[5]. This would have led to modifications of the protocol structure since some messages of the communication protocol would have been split into several messages on the security protocol level.

[5] In reality communication with smart cards is done using APDUs (Application Protocol Data Units) which have a fixed maximal size of 255 Bytes.

A symmetric encryption key K_S shared between all Mondex smart cards is the foundation on which the security of our cryptographic protocol relies. This key must not be known by the attacker, otherwise money could be generated. The key is used to encrypt the data contained in a req, val or ack message, i.e. the payment details of the transaction in which the Mondex card issuing the message currently participates. As only genuine Mondex cards know this secret key, such a correctly encrypted message is accepted as authentic.

To prevent the misuse of a message, e.g. by using a req message as a val message and thereby creating money, it is necessary to distinguish the three kinds of messages. To allow the smart cards to distinguish the different messages, three pairwise distinct constant values REQ, VAL and ACK are introduced and the corresponding constant is added to the data part which is encrypted when a req, val or ack message is created.

Our protocol for the Mondex value transfer, written in a commonly used standard notation for cryptographic protocols [Car94], is:

1. to $\rightarrow$ from : $\{\mathtt{REQ},\mathrm{pdAuth(to)}\}_{K_S}$
2. from $\rightarrow$ to : $\{\mathtt{VAL},\mathrm{pdAuth(from)}\}_{K_S}$
3. to $\rightarrow$ from : $\{\mathtt{ACK},\mathrm{pdAuth(to)}\}_{K_S}$

Another possible approach to formulate a value transfer protocol based on common knowledge is the usage of a HMAC (keyed hashing [BCK96]). In this case smart cards with support for an appropriate hash algorithm are required and the common secret as well as the payment details would be hashed. The resulting hash value proves the authenticity of the message.

7.2. The PROSECCO model of Mondex

The substitution of the cryptographic protocol for the communication protocol is not the only difference between the two levels, the PROSECCO model is more detailed in several other aspects as well. The Mondex models described in [SCW00] do not deal with the technical conditions of real smart card applications. In [SCW00] two purses communicate directly and the commands sent to the smart cards are chosen nondeterministically from the current ether. In the PROSECCO model two smart cards cannot communicate directly (this is not possible in the real world), instead we have modeled explicitly the Mondex wallet, a small computer with two smart card readers which executes the value transfer protocol between two Mondex purses. The real Mondex smart cards and the Mondex wallet do not start protocol runs on their own, instead they must be activated by the card owner. The PROSECCO model reflects this and therefore has some user agents that send commands to the Mondex wallets and start value transfers. The PROSECCO model has another agent which is not present in the models in [SCW00]: the attacker. The attacker is a malicious agent that represents the threats that the Mondex application faces.

Furthermore the Mondex purses on the PROSECCO level have additional operations: the request of the current sequence number and the purse's name needed for startFrom and startTo messages and the request of the current balance.

PROSECCO's detailed model of the possible communication and the application specific description of the attacker's abilities replaces the assumptions over the attacker which were implicit in the definition and the treatment of the ether in [SCW00]. The attacker model used in the PROSECCO ASM for the Mondex case study is a Dolev–Yao [DY81] style attacker, which has access to all communication channels and analyses and composes documents following the usual perfect cryptography assumption. The additional complexity on this level compared to the concrete world of [SCW00] arises from the fact that we have to prove that analysing and composing the messages available to the attacker does not lead to the possibility that a future req, val or ack message is created by the attacker. Therefore it is most important that the shared key remains unknown to the attacker. If this is ensured, one can prove that the messages that represent req, val and ack actually cannot be forged and therefore faithfully mimic the req, val and ack messages of [SCW00].

The possible steps of the application are described operationally by the main rule of the PROSECCO ASM. For example the part of the PROSECCO ASM responsible for treating a req message is:

```
PREQ
  if get–inpd(indoc) = pdAuth(agent)
  then balance(agent) := balance(agent) – get–part(pdAuth(agent), 5).int
       state(agent) := EPA
       outdoc := encdoc(key(agent), doclist(intdoc(VAL) + pdAuth(agent)))
```

The ASM rule checks if the payment details received in the input message (get–inpd(indoc)) are equivalent to the purse's current payment details. If this is the case, the balance is decremented, the state of the purse is set to EPA and the encrypted val message is generated as output. This ASM rule corresponds to the operation REQ# presented in Sect. 4 with one exception: The test if the purse's state is EPR is done before the ASM rule PREQ is executed.

The PROSECCO ASM contains all steps of the communication protocol, i.e. all steps necessary to transfer money between the purses, but we left out the archiving of exception logs because we wanted to focus on the security protocol.

8. Refinement by forward simulation

The link between the concrete level of the original Mondex case study and the PROSECCO ASM is established using ASM refinement ([BR95], [Sch01], [Bör03]). We prove a forward simulation that guarantees that for all runs of the PROSECCO model of Mondex there exists a corresponding run of the communication protocol, using the ASM model of the communication protocol that is defined in [SGHR06a] and verified in [SGH+07]).

The library of the KIV system offers a generic theory of ASM refinement which has to be instantiated appropriately. The main proof task is to prove that each step of the PROSECCO ASM either refines a step of the communication protocol or refines an empty step.

The following section describes this central proof obligation and the definitions necessary for the proof. These are a state mapping σ that links the states of the purses on the abstract and the concrete level and a simulation relation that describes how the knowledge of the attacker is linked to the abstract ether.

On the PROSECCO level the state of the Mondex purses is described by functions $\mathsf{name} : \mathsf{agent} \to \mathsf{int}$, $\mathsf{balance} : \mathsf{agent} \to \mathsf{int}$, $\mathsf{status} : \mathsf{agent} \to \mathsf{int}$, $\mathsf{pdAuth} : \mathsf{agent} \to \mathsf{document}$, $\mathsf{exLog} : \mathsf{agent} \to \mathsf{documentlist}$ and $\mathsf{nextSeqNo} : \mathsf{agent} \to \mathsf{int}$. These encode the same information as in the communication protocol. Additionally $\mathsf{key} : \mathsf{agent} \to \mathsf{key}$ stores the symmetric key and $\mathsf{exLogCounter} : \mathsf{agent} \to \mathsf{int}$ counts the number of exception logs. Similar to astate and cstate in Sect. 4 we use pstate to abbreviate this tuple. Application specific data types of the communication protocol have been replaced with documents from the PROSECCO library. For example payment details are represented by a documentlist:

$$\begin{aligned}&\sigma(\mathsf{mkpd}(\mathsf{fromname}, \mathsf{fromno}, \mathsf{toname}, \mathsf{tono}, \mathsf{value})) = \\ &\quad \mathsf{doclist}(\mathsf{intdoc}(\mathsf{name2int}(\mathsf{fromname})) + \mathsf{intdoc}(\mathsf{fromno}) + \mathsf{intdoc}(\mathsf{name2int}(\mathsf{toname})) \\ &\qquad + \mathsf{intdoc}(\mathsf{tono}) + \mathsf{intdoc}(\mathsf{value}))\end{aligned}$$

and a request message req(pd) containing payment details pd is represented by an encrypted document

$$\sigma(\mathsf{req}(\mathsf{pd})) = \mathsf{encdoc}(\mathsf{K_S}, \mathsf{doclist}(\mathsf{intdoc}(\mathtt{REQ}) + \sigma(\mathsf{pd})))$$

where $\mathsf{K_S}$ is the symmetry key stored by every authentic purse.

Auxiliary functions are used to convert elementary data to integers, e.g. name2int with type $\mathsf{name} \to \mathsf{int}$ is a bijection between authentic names and those integers which identify authentic purses. Not using application specific data types allows a generic treatment of the attacker and the reuse of components of the ASM model in various applications.

The state mapping used in the simulation relation is a bijection between the states of the purses with authentic names on the two levels and describes how the values of the fields on the PROSECCO level are derived from the values of the fields on the concrete level of [SCW00]. The state of purses without authentic name is left unspecified. The complete axiom of the state–mapping relation is:

$$\mathsf{state\text{–}mapping}(\mathsf{cstate}, \mathsf{pstate}) \leftrightarrow \forall\, \mathsf{na}.\ \mathsf{authentic}(\mathsf{na}) \to \mathsf{pstate}(\sigma(\mathsf{na})) = \sigma(\mathsf{cstate}(\mathsf{na}))$$

σ represents the joint application of the different mapping functions necessary to transform the different parts of the state of a purse.

Given the state mapping which links cstate and pstate as the core, the forward simulation relation can be defined. Besides the link between the states of the purses three further properties are needed in the simulation relation:

1. The ether of the abstract level and the attacker's knowledge of the PROSECCO level must be equally powerful, i.e. the same relevant documents can be generated from the ether and the set attacker–known representing

the attacker's knowledge in the PROSECCO ASM. More precisely the definition of the equivalence between ether and attacker–known is that it is possible to generate the same documents from the attacker's knowledge attacker–known and from the set of documents resulting from applying the mapping function on all elements of the ether:

$$\mathsf{ether} \equiv \mathsf{attacker\text{–}known} \leftrightarrow \forall\ \mathsf{doc}.\ ((\mathsf{attacker\text{–}known} \vdash \mathsf{doc}) \leftrightarrow\ (\sigma(\mathsf{ether}|_{\mathsf{RVA}}) \vdash \mathsf{doc}))$$

$\mathsf{ether}|_{\mathsf{RVA}}$ is the restriction of the ether to the security relevant encrypted messages req, val and ack. The predicate $\equiv$ means that (from the attacker's point of view) the ether and the set of documents attacker–known are equally powerful, i.e. the attacker can derive the same new documents from them. This is exactly what the above axiom states: a documents doc can be generated from attacker–known if and only if doc can be generated from $\sigma(\mathsf{ether}|_{\mathsf{RVA}})$. This is the set of documents that results from taking all the req, val and ack messages from the ether and transforming them in their document representation.
2. The messages $\perp$ and all startTo and startFrom messages must be contained in the ether.
3. The PROSECCO ASM maintains an invariant PINV(pstate) which expresses well-formedness conditions on the internal state of the purses (e.g. payment details of the current transaction contain valid documentlists and the key stored by a purse is indeed the common secret key) as well as conditions concerning the attacker (e.g. the common secret key of the Mondex purses is not contained in attacker–known).

The first property is needed in order to prove that all steps which are possible on the PROSECCO level (i.e. for which the attacker can generate the document triggering them) are also possible on the abstract level because the corresponding messages are available in the ether. This is important for the critical messages req, val and ack. This equivalence is the central part of the forward simulation, without a precise description of the dependencies between the ether and the attacker's knowledge the proof of the refinement theorem is likely to fail. The second property is needed because on the PROSECCO level startFrom and startTo messages are represented as lists of integer documents. The attacker can always produce these, therefore on the abstract level the corresponding messages must always be available. The full definition of the simulation relation R is

$$\begin{aligned}\mathsf{R}(\mathsf{cstate}, \mathsf{pstate}) \leftrightarrow\quad & \mathsf{state\text{–}mapping}(\mathsf{cstate}, \mathsf{pstate}) \wedge \mathsf{ether} \equiv \mathsf{attacker\text{–}known}\\ & \wedge \{\perp\} \cup \{\mathsf{isStartTo}\} \cup \{\mathsf{isStartFrom}\} \subseteq \mathsf{ether} \wedge \mathsf{PINV}(\mathsf{pstate})\end{aligned}$$

With this simulation relation the main proof obligation is

$$\begin{aligned}&\mathsf{R}(\mathsf{cstate}, \mathsf{pstate}) \wedge \langle \mathsf{PSTEP\#}(\mathsf{pstate})\rangle\ \mathsf{pstate} = \mathsf{pstate}'\\ \rightarrow\ & \exists\ \mathsf{cstate}'.\ \langle \mathsf{CSTEP\#}(\mathsf{cstate}) \vee \mathbf{skip}\rangle\ (\mathsf{cstate} = \mathsf{cstate}' \wedge \mathsf{R}(\mathsf{cstate}', \mathsf{pstate}'))\end{aligned}$$

It states that given states cstate of the communication protocol level and pstate of the security protocol which are in the simulation relation, and assuming a step PSTEP# of the PROSECCO ASM may lead to a new state pstate′, then either a step CSTEP# of the communication protocol or an empty "skip" step leads to a state cstate′ which is in the simulation relation to pstate′ again. The proof obligation is quite similar to the proof obligation of data refinement except that there is only one ASM rule on each level and that a 0:1 diagram is possible in the proof obligation without the need to explicitly add a skip operation to the communication protocol. A 0:1 diagram represents the case of a purse operation on the PROSECCO-level that refines "skip" on the communication protocol level, i.e. the pre- and the post-state of the operation relate to the same state of the higher level. The ASM rules on the two levels both encompass all protocol steps (i.e. operations) and contain a case distinction over all the different operations. Because on both levels all operations are integrated into one ASM rule, for this refinement we only have one proof obligation instead of individual proof obligations for all operations.

Based on the generic theory of documents and attacker knowledge from Sect. 6 that we developed earlier the proof of the forward simulation itself is quite direct and not complicated. We only have 0:1 and 1:1 diagrams. A successful protocol step of the PROSECCO ASM refines the corresponding step of the communication protocol (1:1 diagram), the processing of malformed or unexpected documents refines ABORT# (1:1 diagram), steps without abstract counterpart but unmodified state of the purses on the PROSECCO level, e.g. attacker and terminal steps and the new purse operations that report the current balance and sequence number refine skip (0:1 diagram).

The proofs for the refinement from the communication protocol level to the security protocol level were done by a student worker who had just taken a one semester practical course on formal methods and KIV. They took him approximately 4 months including the proofs for the invariant on the PROSECCO level. A large part of the additional effort is due to the fact that this invariant developed to be quite large and at certain points quite difficult. Several iterations were necessary to develop a suitable invariant with changes not only to the invariant but also to the ASM on the PROSECCO level. The invariant contains not just information relevant for the refinement from the communication protocol level but also a large part which is only relevant for the next refinement to the Java implementation. Especially this part took at lot of time. In fact most of the changes and the larger part of the invariant were needed to fit together the PROSECCO and the Java levels. This indicates that having "hints" concerning the contents of the invariant, significantly simplifies its development. Although the 4 months required for this refinement are quite long compared to the effort for the refinement from the transaction to the communication protocol level (see Sect. 5) we consider the refinement from the communication protocol to the PROSECCO ASM as the simpler one. The main reason for this is that we have a direct correspondence between the operations on the 2 levels, which means that we didn't have to break atomic steps on the communication protocol level into multiple steps on the PROSECCO level.

The proofs for the forward simulation from the communication protocol level to the security protocol level required approximately 800 interactive proof steps, the total number of proof steps is approximately 4,200, i.e. proof automation is about 80 %. The proof of the invariant PINV of the PROSECCO ASM was done independently of the forward simulation and required approximately 1,200 additional interactive proof steps. Compared to the proofs we reported on in Sect. 5 these proofs required more interactions. This is not surprising since the efficiency of proving is strongly linked to the experience of the person developing the proofs.

9. Related work

Related work describing other specification and verification approaches for the Mondex case study can be found in this issue. Our own work on proving the Mondex refinement using the ASM refinement approach is described in [SGH+07]. An additional, recent approach based on OTS/CafeOBJ and Maude is [KOF07]. This approach models the protocol level only and proves the security properties directly rather than by refinement. It reports that proofs and lemmas have been nevertheless very similar to ours and the orignal ones.

Concerning the graphical modeling of security critical applications and especially security protocols, UMLsec [Jür02, Jür05] is most similar to PROSECCO. UMLsec is a UML profile which extends several UML diagrams with security relevant annotations. UMLsec allows the investigation of various security properties, not just security of cryptographic protocols. To specify cryptographic protocols, UMLsec uses sequence diagrams to describe the messages of the protocol and class diagrams to describe the state of the agents and add annotations marking security relevant information. UMLsec focuses on modeling, proof support for the verification of properties of cryptographic protocols is offered by exporting parts of the model into inputs for a model-checker. The verification also focuses on some standard properties of security protocols (e.g. secrecy). In PROSECCO the formal model is completely embedded in the KIV system. The PROSECCO approach also does not suffer from the limitations of model-checkers (finite state space) and is not limited to standard properties. [Jür05] reports on the verification of CEPS, a proposal for a smart card-based electronic payment system. The proof of the security property for the payment system as presented in [Jür05] is done by hand, i.e. without tool-support.

In the context of verification of cryptographic protocols several approaches have been proposed. One of the most influential publications was [BAN90] introducing the "Logic of Authentication" (aka. BAN-Logic). A lot of protocols were found erroneous when they were analysed with the BAN-Logic. However the BAN-Logic has some serious disadvantages. It does not support reasoning about secrecy since it was developed for reasoning about authentication protocols. Furthermore no proof system for the BAN-Logic was presented limiting the analysis to hand-made proofs. In the area of tool-supported protocol verification fully-automated verification using model-checking is dominating (see e.g. [Low96],[MCJ97], [CJM98], [Zar98], [BMV03], [SBP01]), but these are usually limited to standard properties like authenticity and secrecy (but see T. Ramananandro's work with Alloy in this issue). The security properties of the Mondex case study are application-specific and quite different from those standard properties.

Two interactive approaches are [Pau98] and [RSG+01]. Paulson's Inductive Approach [Pau98] is best-known among the interactive verification techniques for security protocols. Paulson uses the theorem prover Isabelle to verify properties of the protocols based on inductively defined sets of traces. Each of these traces represents a possible sequence of (communication-) events given the formalised protocol and a set of agents and the attacker. Paulson has successfully analysed different large protocols (e.g. TLS [Pau99] and SET [Pau01]). The main differences between PROSECCO and the Inductive Approach are that PROSECCO has a graphical modeling language while in the Inductive Approach the axioms are written down directly, the fact that Paulson does not model the state of the agents whereas PROSECCO explicitly models the internal state of the agents, the different approaches for describing the possible runs of the application, an operational description in PROSECCO and a relational approach in Paulson's work and finally the focus towards a refinement to real code which is very important within the PROSECCO approach.

10. Conclusion and further work

We have specified and formally verified two refinements for Mondex electronic purses. The first refinement solves the challenge to verify the development of the full case study [SCW00].

Our proof is based on an improved theory of backward simulation for the contract approach, that allows to replace the second refinement of the original work by an invariance proof.

The effort to verify the first refinement was about a person month of work. We feel that this effort was rather small compared to the effort we assume it has taken to write down proofs in [SCW00] at nearly calculus level. Having a nearly correct simulation relation and invariants helped a lot to get the proof done quickly. Nevertheless we were still able to find several small flaws: one in the underlying data refinement theory, where a proof obligation was missing and three in the invariant. Therefore we feel justified to recommend doing machine proofs as a means to increase confidence in the results.

As a second contribution we have verified a refinement of the concrete level of Mondex to a security protocol based on symmetric key cryptography. This refinement justifies the security assumptions that underly the original work. The complexity of this refinement is somewhat easier than the one of the original case study.

Together with [SGH+07], where we have developed a systematic approach to verify the Mondex refinement, the second refinement is part of our work to develop verified Java Code for E-Commerce applications using UML for informal specifications [MHSR07] and ASMs for formal protocol definitions (another case-study is Cindy [GHRS06], an electronic ticketing application, which has already been refined to Java [GSR06]).

For Mondex we are currently working on the verification of Java Card code as a refinement of the security protocol level. In [GMB+06] we present three implementations for the Mondex case study using different cryptographic techniques. The refinement proof will be based on the verification kernel approach ([GSR05]), our refinement framework for Java [GSR07] and the Java Card calculus of KIV ([Ste04]). Java code is already available on our web site [KIV].

Acknowledgments

Gerhard Schellhorn would like to thank Jim Woodcock for inviting him to the Mondex workshops. We would also want to thank Bogdan Tofan, a student research assistant who helped us by doing most of the proof work necessary for the refinement from the communication protocol to the security protocol.

References

[BAN90] Burrows M, Abadi M, Needham R (1990) A logic of authentication. ACM Trans Comput Syst 8(1)

[BCK96] Bellare M, Canetti R, Krawczyk H (1996) Keying hash functions for message authentication. In: Koblitz N (ed) Advances in Cryptology—Crypto 96 Proceedings, vol 1109 of LNCS. Springer, Heidelberg

[BDRS02] Balser M, Duelli C, Reif W, Schellhorn G (2002) Verifying concurrent systems with symbolic execution. J Logic Comput 12(4):549–560

[BDS07] Boiten E, Derrick J, Schellhorn G (2007) Relational concurrent refinement part ii: intenral operations and ouputs. FAC (under consideration)

[BMV03] Basin D, Mödersheim S, Viganò L (2003) An on-the-fly model-checker for security protocol analysis. In: Proceedings of Esorics'03, LNCS 2808, Springer, Heidelberg, pp 253–270

[Bör03] Börger E (2003) The ASM refinement method. Form Aspects Comput 15(1–2):237–257

[BR95] Börger E, Rosenzweig D (1995) The WAM-definition and compiler correctness. In: Beierle C, Plümer L (eds) Logic Programming: Formal Methods and Practical Applications. Studies in Computer Science and Artificial Intelligence 11, pp 20–90. North-Holland, Amsterdam

[BS03] Börger E, Stärk RF (2003) Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Heidelberg

[Car94] Carlsen U (1994) Generating formal cryptographic protocol specifications. In: IEEE Symposium on Research in Security and Privacy. IEEE Computer Society, pp 137–146

[CB99] UK ITSEC Certification Body (1999) UK ITSEC Scheme Certification Report No. P129 Mondex Purse. Technical report, UK IT Security Evaluation and Certification Scheme, URL: http://www.cesg.gov.uk/site/iacs/itsec/media/certreps/CRP129.pdf

[CJM98] Clarke EM, Jha S, Marrero W (1998) Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In: Proceedings of the IFIP Working Conference in Programming Concepts and Methods (PROCOMET'98)
[Cla96] Clarke R (1996) The Mondex Value-Card Scheme, chap. 4. Xamax Consultancy Pty Ltd, URL: http://www.anu.edu.au/people/Roger.Clarke/EC/Mondex.html
[CoF04] CoFI (2004) (The Common Framework Initiative). CASL Reference Manual. LNCS 2960 (IFIP Series). Springer, Heidelberg
[CSW02] Cooper D, Stepney S, Woodcock J (2002) Derivation of Z refinement proof rules: forwards and backwards rules incorporating input/output refinement. Technical Report YCS-2002-347, University of York, URL: http://www-users.cs.york.ac.uk/∼susan/bib/ss/z/zrules.htm
[DB01] Derrick J, Boiten E (2001) Refinement in Z and in Object-Z : foundations and advanced applications. FACIT. Springer, Heidelberg
[DY81] Dolev D, Yao AC (1981) On the security of public key protocols. In: Proceedings of the 22nd IEEE symposium on foundations of computer science, pp 350–357
[Far94] Farmer WM (1994) Theory interpretation in simple type theory. In: Heering J et al (ed) Higher-order algebra, logic, and term rewriting, vol 816. Lecture Notes in Computer Science. Springer, Heidelberg
[GHRS06] Grandy H, Haneberg D, Reif W, Stenzel K (2006) Developing provably secure M-commerce applications. In: Müller G (ed) Emerging trends in information and communication security, vol 3995 of LNCS. Springer, Heidelberg, pp 115–129
[GMB+06] Grandy H, Moebius N, Bischof M, Haneberg D, Schellhorn G, Stenzel K, Reif W (2006) The Mondex case study: from specifications to code. Technical Report 2006-31, University of Augsburg, URL: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/publications
[GSR05] Grandy H, Stenzel K, Reif W (2005) Object-oriented verification Kernels for secure Java applications. In: Aichering B, Beckert B (eds) SEFM 2005—3rd ieee international conference on software engineering and formal methods
[GSR06] Grandy H, Stenzel K, Reif W (2006) A refinement method for Java programs. Technical Report 2006-29, University of Augsburg URL: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/publications
[GSR07] Grandy H, Stenzel K, Reif W (2007) A refinement method for Java programs. In: Formal methods for open object-based distributed systems, vol 4468 of Lecture Notes in Computer Science. Springer, Heidelberg
[Gur95] Gurevich Y (1995) Evolving algebras 1993: Lipari guide. In: Börger E (ed) Specification and validation methods. Oxford University Press, Oxford, pp 9–36
[Han06] Haneberg D (2006) Sicherheit von Smart Card—Anwendungen (in German). PhD thesis, University of Augsburg, Augsburg, Germany
[HGRS05] Haneberg D, Grandy H, Reif W, Schellhorn G (2005) Verifying security protocols: an ASM approach. In: Beauquier D, Börger E, Slissenko A (eds) 12th International workshop on abstract state machines, ASM 05. University Paris 12, Val de Marne, Créteil, France
[HGRS07] Haneberg D, Grandy H, Reif W, Schellhorn G (2007) Verifying smart card applications: an ASM approach. In: Gibbons J, Davies J (eds) Proceedings of the international conference on integrated formal methods (iFM) 2007, vol 4591 of LNCS. Springer, Heidelberg, pp 313–332
[HHS86] Jifeng H, Hoare CAR, Sanders JW (1986) Data refinement refined. In: Robinet B, Wilhelm R (eds) Proceedings of ESOP 86, vol 213 of Lecture Notes in Computer Science. Springer, Heidelberg, pp 187–196
[HKT00] Harel D, Kozen D, Tiuryn J (2000) Dynamic logic. MIT, Cambridge
[Jür02] Jürjens J (2002) UMLsec: Extending UML for secure systems development. In: Jézéquel J-M, Hussmann H, Cook S (eds) UML 2002—The Unified Modeling Language 5th International Conference, Dresden, Germany, Springer, LNCS 2460
[Jür05] Jürjens J (2005) Secure systems development with UML. Springer, Heidelberg, ISBN: 3-540-00701-6
[KIV] Web presentation of the mondex case study in KIV, URL: http://www.informatik.uni-augsburg.de/swt/projects/mondex.html
[KOF07] Kong W, Ogata K, Futatsugi K (2007) Algebraic approaches to formal analysis of the mondex electronic purse system. In: Gibbons J, Davies J (eds) Proceedings of the international conference on integrated formal methods (iFM) 2007, vol 4591 of LNCS. Springer, Heidelberg, pp 393–412
[Low96] Lowe G (1996) Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Tools and algorithms for the construction and analysis of systems (TACAS), vol 1055. Springer, Berlin
[MCJ97] Marrero W, Clarke E, Jha S (1997) A model checker for authentication protocols. In: Proceedings of the DIMACS workshop on design and formal verication of security protocols
[MHSR07] Moebius N, Haneberg D, Schellhorn G, Reif W (2007) A modeling framework for the development of provably secure E-Commerce applications. In: Proceedings of the international conference on software engineering advances 2007. IEEE Computer Society Press (accepted for publication)
[Pau98] Paulson LC (1998) The inductive approach to verifying cryptographic protocols. J Comput Secur. **6**
[Pau99] Paulson LC (1999) Inductive analysis of the Internet protocol TLS. ACM Trans Inf Syst Secur 2(3):332–351
[Pau01] Paulson LC (2001) Verifying the SET protocol. In: Gore R, Leitsch A, Nipkow T (eds) IJCAR 2001: international joint conference on automated reasoning, Siena, Italy. Springer, LNCS 2083
[RJ06] Ramananadro T, Jackson D (2006) Mondex, an electronic purse: specification and refinement checks with the alloy model-finding method, URL: http://www.eleves.ens.fr/home/ramanana/work/mondex
[RSG+01] Ryan PYA, Schneider SA, Goldsmith MH, Lowe G, Roscoe B (2001) The modelling and analysis of security protocols: the CSP approach. Addison-Wesley, Reading
[RSSB98] Reif W, Schellhorn G, Stenzel K, Balser M (1998) Structured specifications and interactive proofs with KIV. In: Bibel W, Schmitt P (eds) Automated deduction—a basis for applications, vol II: Systems and implementation techniques, Chap. 1: Interactive theorem proving. Kluwer, Dordrecht, pp 13–39
[SA97] Schellhorn G, Ahrendt W (1997) Reasoning about abstract state machines: The WAM case study. J Univer Comput Sci (J.UCS) 3(4):377–413, URL: http://www.jucs.org
[SA98] Schellhorn G, Ahrendt W (1998) The WAM case study: verifying compiler correctness for prolog with KIV. In: Bibel W, Schmitt P (eds) Automated deduction—a basis for applications. Kluwer, Dordrecht, pp 165–194

[SBP01] Song D, Berezin S, Perrig A (2001) Athena: a novel approach to efficient automatic security protocol analysis. J Comput Secur (Special Issue CSFW12) 9(1,2):47–74

[Sch99] Schellhorn G (1999) Verification of abstract state machines. PhD thesis, Universität Ulm, Fakultät für Informatik, URL: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/publications

[Sch01] Schellhorn G (2001) Verification of ASM refinements using generalized forward simulation. J Univ Comput Sci (J.UCS), 7(11):952–979, URL: http://www.jucs.org

[Sch05] Schellhorn G (2005) ASM refinement and generalizations of forward simulation in data refinement: a comparison. J Theor Comput Sci 336(2–3):403–435

[SCW00] Stepney S, Cooper D, Woodcock J (2000) An electronic purse specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, URL: http://www-users.cs.york.ac.uk/∼susan/bib/ss/z/monog.htm

[SGH+07] Schellhorn G, Grandy H, Haneberg D, Moebius N, Reif W (2007) A systematic verification approach for mondex electronic purses using ASMs. In: Glässer U, Abrial J-R (ed) Proceedings of the Dagstuhl seminar on rigorous methods for software construction and analysis, LNCS. Springer, Heidelberg (accepted, older version available as Techn. Report 2006-27 at [KIV])

[SGHR06a] Schellhorn G, Grandy H, Haneberg D, Reif W (2006) The Mondex challenge: machine checked proofs for an electronic purse. In: Misra J, Nipkow T, Sekerinski E (eds) Formal methods 2006, Proceedings, vol 4085 of LNCS. Springer, Heidelberg, pp 16–31

[SGHR06b] Schellhorn G, Grandy H, Haneberg D, Reif W (2006) The Mondex challenge: machine checked proofs for an electronic purse. Technical Report 2006-2, Universität Augsburg

[Spi92] Michael Spivey J (1992) The Z notation: a reference manual. International Series in Computer Science, 2nd edn. Prentice Hall, Englewood Cliffs

[Ste04] Stenzel K (2004) A formally verified calculus for full Java card. In: Rattray C, Maharaj S, Shankland C (eds) Algebraic methodology and software technology (AMAST) 2004, Proceedings, Stirling Scotland. Springer, LNCS 3116

[TOWS04] Thums A, Ortmeier F, Reif W, Schellhorn G (2004) Interactive verification of statecharts. In: Ehrig H (ed) Integration of software specification techniques for applications in engineering Springer, LNCS 3147, pp 355–373

[WD96] Woodcock JCP, Davies J (1996) Using Z: Specification, proof and refinement. International Series in Computer Science, Prentice Hall, Englewood Cliffs

[WF06] Woodcock J, Freitas L (2006) Z/eves and the mondex electronic purse. In: Cerone A, Barkaoui K, Cavalcanti A (eds) Theoretical aspects of computing—ICTAC 2006, 3rd international colloquium, LNCS 4281 Tunis, Springer, Heidelberg, pp 14–34

[Zar98] Zarba CG (1998) Model checking the Needham–Schroeder protocol, URL: http://www-step.stanford.edu/case-studies/security

Received 19 January 2007

Accepted in revised form 9 July 2007 by J. C. P. Woodcock

Published online: 8 December 2007