Developing High-Assurance Secure Systems with UML: A Smartcard-based Purchase Protocol

Jan Jürjens*
Software & Systems Engineering, TU München, Germany

Abstract

Developing high-assurance security-critical systems is difficult and there are many well-known examples of security weaknesses exploited in practice. Thus a sound methodology supporting secure systems development is urgently needed.

Our aim is to aid the task of developing security-critical systems in an approach based on the notation of the Unified Modeling Language (UML).

Towards this aim, we use an extension of UML, called UMLsec, that allows expressing security-relevant information within the diagrams in a system specification. We present tool-support which has been developed for the UMLsec approach. We apply UMLsec to the example of an electronic purse protocol proposed as a global standard. We demonstrate how to detect some vulnerabilities using our approach, suggest improvements, and show that the improved protocol is secure in a precise sense, by using a tool that implements a formal semantics of a simplified fragment of UML.

Keywords. High assurance systems, security software engineering, security engineering, security verification, formal methods in security, security evaluation, security models, cryptographic protocols, electronic purses.

1 Introduction

Modern society and modern economies rely on infrastructures for communication, finance, energy distribution, and transportation. These infrastructures depend increasingly on networked information systems. Attacks against these systems can threaten the economical or even physical integrity of people and organizations. Due to the widespread interconnection of information systems, attacks can be waged anonymously and from a safe distance.

Many problems with security-critical systems arise from the fact that their developers do not always have a strong background in computer security. This is problematic since in practice, security is compromised most often not by breaking mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used. Security mechanisms cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account.

Furthermore, sometimes security mechanisms (such as security protocols) cannot be used off-the-shelf, but have to be designed specifically to satisfy given requirements (for example on the hardware). Such mechanisms are notoriously hard to design correctly, even for experts, as many examples of protocols designed by experts that were later found to contain flaws show.

On the one hand there exist several approaches for security engineering in practice. It would be desirable to communicate such security engineering principles and methods to software engineers who may not have a background in security, by means of an industrially used modeling language. In particular, it would be desirable to consider security aspects already in the design phase, before a system is actually implemented, since removing security flaws in the design phase saves cost and time.

On the other hand, there has been a significant amount of research into using formal methods for secure systems development. However, part of the difficulty of security-critical systems development is that the goal of correctness is often in conflict with that of low development cost. Where thorough methods of system design pose high cost through personnel training and use, they are all too often avoided.

The Unified Modeling Language (UML, [Obj03], the de facto industry-standard in object-oriented modeling) offers an unprecedented opportunity for high-quality critical systems development that is feasible in an industrial context.

- As the de facto standard in industrial modeling, a large number of developers is trained in UML.
- Compared to previous notations with a user community of comparable size, UML is relatively precisely defined.

To support using UML for secure systems development, an extension, UMLsec, of the UML, is given

*http://www.jurjens.de/jan – juerjens@in.tum.de . Supported within the Verisoft Project of the German Ministry for Education and Research.

in [Jür04]. This way knowledge on prudent security engineering is encapsulated and thereby made available to developers which may not be specialized in security.

In the current work, we demonstrate usage of UMLsec by applying it to the example of part of the Common Electronic Purse Specifications (CEPS), an electronic purse protocol proposed as a global standard. With our approach, we detect some vulnerabilities, suggest improvements, and show that these are secure in a precise sense, by using a tool that implements a formal semantics of a simplified fragment of UML.

In the following paragraph, we report on related work. In Sect. 2, we present the used fragment of the UML and define the concepts necessary for security analysis. In Sect. 4, we apply UMLsec to a security analysis of the purchase protocol from the Common Electronic Purse Specifications. We close with a conclusion and indication of future work.

Related Work The body of research in applying formal methods to security-critical systems (in particular to security protocols) has become so large that it is impossible to give a fair overview. A partial overview can be found for example in [Mea95].

Less work has been done about integrating such work into the practical software engineering context. For example, [KAH99] uses the method SCR (Software Cost Reduction) to specify and analyze a cryptographic system. The paper also examines SCR under different aspects including ease of use, benefits, cost-effectiveness, scalability, applicability to security, ease of transition from prose specifications. SCR is a formal method for specifying and analyzing system requirements. It has an associated tool suite SCR* which includes a consistency checker, simulator, model checker, invariant generator, theorem prover, and validity checker [HKLB98].

Other approaches on using UML for developing security-critical systems can be found for example in [JCF$^+$02, JRFF03], including for example [HdBLS02] which defines a UML profile for model-based risk assessment. Work on other non-functional requirements besides security includes [BDCL$^+$01, HMP$^+$02, CSC02], dealing with dependability and reliability analysis of UML models. Specifically, [BDCL$^+$01] augments design tools using UML with validation and analysis techniques for the early phases of system design and with automatic model generation transformations. [HMP$^+$02] reports on an approach which supports quantative dependability analysis of systems modeled using UML statecharts. [CSC02] annotates UML use case, sequence and deployment diagrams with reliability related attributes and performs system reliability prediction based on component and connector failure rates.

[JW01] models a small part of the CEPS purchase protocol using the CASE tool AutoFocus. A different part of the CEPS, the load protocol, is analyzed using UMLsec in [Jür01]. [WJ02] (and other papers referenced in there) uses AutoFocus for model-based test-generation applied to the load protocol in CEPS. In contrast to these article, the present work gives a detailed specification of the CEPS purchase protocol using UML and on tool-supported analysis using Prolog, which is not so far to be found in the literature.

2 Specification language

We can only shortly recall the necessary background; more explanation can be found in [Jür04].

To specify the protocol, we use a simplified version of UML sequence diagrams. We define their abstract syntax.

A sequence diagram $D = (\mathsf{Obj}(D), \mathsf{Cncts}(D))$ is given by

- a set $\mathsf{Obj}(D)$ of pairs (O, C) where O is an object of class C whose interaction with other objects is described in D and
- a (finite) sequence $\mathsf{Cncts}(D)$ consisting of elements of the form $l = (\mathsf{source}(l), \mathsf{guard}(l), \mathsf{msg}(l), \mathsf{target}(l))$ (so-called *connections*) where
  - $\mathsf{source}(l) \in \mathsf{Obj}(D)$ is the source object of the connection,
  - $\mathsf{guard}(l) \in \mathbf{BoolExp}$ is a Boolean expression that is the guard of the connection,
  - $\mathsf{msg}(l) \in \mathbf{Events}$ is the message of the connection, and
  - $\mathsf{target}(l) \in \mathsf{Obj}(D)$ is the target object of the connection.

As an example, a sequence diagram is given in the top part of Fig. 3 (with explanation of its behavior given there).

A guard syntactically equal to *true* may be omitted in the diagram. As a simplification, one may also use the notation $var := exp$ to assign an expression to a local variable var. This is only a syntactic shortcut; before assigning a semantics to the diagrams, the variable var should be replaced by the expression exp at each occurrence.

Stereotypes can be used to attach tagged values and constraints as pseudo-attributes of the stereotyped model elements (pseudo-attributes because their semantics is outside the scope of the UML standard). All model elements labelled by a particular stereotype receive the corresponding values and constraints in addition to the attributes, associations, and superclasses that the element has in the standard UML.

Our tool-support implements a formal semantics of the simplified fragment of UML used (see [Jür04] for details). The fragment used here consists of simplified sequence diagrams, class diagrams, deployment diagrams, and subsystems and was selected to keep a formal treatment feasible. We can only outline the basic concepts of this semantics. A complete account of the formal semantics is given in [Jür04], as well as further concepts of the extension UMLsec not needed here.

The objects or components in a UML specification may communicate through messages received in their input buffers and released to their output buffers. Thus for each object or component C of a given system, our semantics defines an input/output state machine $[\![C]\!]()$ which

- takes a multi-set I of input messages and a state S (the state of the component) and
- outputs a set $[\![C]\!](I,S)$ of pairs (O,T) where O is a multi-set of output messages and T the new state of the machine (it is a *set* of pairs because of the non-determinism that may arise)

together with an *initial state* S_0 of the state machine. The behavioral semantics of an object O in a sequence diagram D is given by a machine $[\![D.O]\!]()$ modeling the execution of O.

The different objects have their own in- and out-buffers. The execution of each object is performed in parallel, each with its own dispatcher only dispatching events sent to it. This is composed sequentially with the execution of a scheduler that takes the events from the out-buffers of the client objects requesting a service from another object and distributes them to the in-buffers of the server objects requested to provide the service, subject to a possible interference from an attacker. This interference is determined by the threat scenario arising from the physical environment of the system, as explained below. The physical environment is given in the UML specification as a deployment diagram. For an example, see the bottom part of Fig. 3: in that diagram, there are three nodes (Card, PSAM, and Display), each containing one component (Cardapp, PSAMapp, resp. Dispapp). The PSAM is connected to both other nodes by a wire, which allows messages to be sent between the respective components.

A UML specification in our setting also contains a class diagram, describing the messages and attributes of each object. This may contain further security-relevant information following the UMLsec definition, but we do not need this here.

Then the behavioral interpretation of a UML specification $\mathcal{S}$ is a machine $[\![\mathcal{S}]\!]()$ defined as follows:

(1) It takes a multi-set of input events.

(2) The events are taken from the input multi-set and from the link buffers and given as argument to the machines defining the behavior of the intended recipients in $\mathcal{S}$.

(3) The output messages from these machines are distributed to the link buffers of the links connecting the sender of a message to the receiver, or given as the output from $[\![\mathcal{S}]\!]()$ when the receiver is not part of $\mathcal{S}$.

When performing security analysis, after the last step, the adversary model may modify the contents of the link buffers in a certain way explained below.

Security Analysis We assume a set **Keys** with a partial injective map $(\_)^{-1} : \mathbf{Keys} \to \mathbf{Keys}$. The elements in its domain are called *encryption keys*, those in its set of values *decryption keys*. Keys k satisfying $k^{-1} = k$ are called *symmetric*. We fix a set **Var** of *variables* and a set **Data** of *data values* (which may include *nonces* and other secrets).

The *algebra of expressions* **Exp** is the quotient of the term algebra generated from the set $\mathbf{Var} \cup \mathbf{Keys} \cup \mathbf{Data}$ with the operations below. Here $\_$ denotes an argument place-holder; thus for example $\_ :: \_$ represents a binary operation.

- $\_ :: \_$ (concatenation) and the empty expression ε,
- $\{\_\}_\_$ (encryption)
- $\mathcal{D}ec_\_(\_)$ (decryption)
- $\mathcal{S}ign_\_(\_)$ (signing)
- $\mathcal{E}xt_\_(\_)$ (extracting from signature)
- $\mathcal{H}ash(\_)$ (hashing)

by factoring out the equations

- $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$ (for $K \in \mathbf{Keys}$),
- $\mathcal{E}xt_K(\mathcal{S}ign_{K^{-1}}(E)) = E$ (for $K \in \mathbf{Keys}$),
- and the usual laws regarding concatenation and ε.

For an explanation of the cryptographic concepts, we have to refer to the standard literature.

For a security analysis of a given UMLsec subsystem specification $\mathcal{S}$, we need to model potential adversary behavior. We model specific types of adversaries that can attack different parts of the system in a specified way. For this, we assume a function $\mathsf{Threats}_A(s)$ which takes an *adversary type* A and a stereotype s and returns a subset of $\{\mathsf{delete}, \mathsf{read}, \mathsf{insert}\}$. Note that we do not need a threat modify since this is subsumed by a combination of delete and insert, in combination with the message recombination abilities of the adversary. Then we model the actual behavior of an adversary adversary of type A as a *type A adversary machine* that maps the contents of the link buffers in $\mathcal{S}$ and a state S to the new contents of the link buffers in $\mathcal{S}$ and a new state T non-deterministically[1] in the following way:

- the contents of links stereotyped s where $\mathsf{delete} \in \mathsf{Threats}_A(s)$ may be mapped to $\emptyset$ and

[1] "Non-deterministically" means that the machine has a *set* of possible outputs.

Stereotype	$\mathsf{Threats}_{insider}()$
Internet	{delete, read, insert}
LAN	{delete, read, insert}
wire	{delete, read, insert}
smart card	$\emptyset$

Figure 1: Threats from the insider attacker *card issuer*

- the contents of links stereotyped s where $\mathsf{insert} \in \mathsf{Threats}_A(s)$ may be enlarged by elements from the contents of links stereotyped t where $\mathsf{read} \in \mathsf{Threats}_A(t)$.

The intention is that this way, we can define adversary types, by prescribing what kind of actions an adversary type may apply to a communication link with the respective stereotype. Here delete means that the adversary may delete the messages in the corresponding link buffer, read means that the adversary may read the messages in the link buffer, and insert means that he may insert messages in the link buffer that he has previously learned. At any point of the execution, the *knowledge set* of the adversary type A is the set of messages so far exchanged over links stereotyped s where $\mathsf{insert} \in \mathsf{Threats}_A(s)$. As an example of the function $\mathsf{Threats}_A(s)$, Fig. 1 gives the insider attacker relevant to the case-study in Sect. 4.

To evaluate the security of the system with respect to the given type of adversary, we define the *execution of the subsystem $\mathcal{S}$ in presence of an adversary of type A* to be the machine $[\![\mathcal{S}]\!]_A()$ defined from $[\![\mathcal{S}]\!]()$ by applying the adversary machine to the link buffers as a fourth step in the definition of $[\![\mathcal{S}]\!]()$ as follows:

(4) The type A adversary machine is applied to the link buffers as detailed above.

3 Tool-support for UMLsec

For the ideas that were presented in the previous section to be of benefit in practice, it is important to have advanced tool-support to assist in using them, which we present in this section. The developed tools can be used to check security requirements associated with UMLsec models using stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use. We also explain a framework for implementing verification routines for the constraints associated with the stereotypes. The goal is that advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes.

3.1 Tool Functionality

There are several possible degrees of functionality in the verification of UMLsec models, depending on which of the stereotypes in Sect. 2 should be verified.

Static features: The tool should be able to verify security properties included as stereotypes in structure and deployment diagrams.

Simple dynamic features: The model behavior, described by the statechart and sequence diagrams, is analyzed to verify basic security requirement defined on the behavioral level.

Complex dynamic features: The UMLsec model describing dynamic behavior is translated into the input language of an analysis tool (such as a temporal logic formula in the case of a model-checker), and can thus be verified against even subtle dynamic properties.

The following aspects have to be considered when trying to construct tool-support for secure systems development with UML following the UMLsec approach.

To be able to apply verification tools (such as model checkers), one needs a front-end which automatically produces a semantic model and includes the relevant formalized security requirements, when given a UMLsec model. This avoids requiring the software developers themselves to perform this formalization, which usually needs a high level of specialized training in formal methods. UMLsec supports this approach by offering predefined security primitives (such as security requirements or mechanisms) with a strictly defined semantics, which can be applied by a developer without an extensive training in security by simply including the relevant stereotypes in the UML model. These primitives are translated into the targeted formal language, protecting from potential errors in manual formalization of the security properties (see Sect. 2 for a short account and [Jür04] for more details). Since security requirements are usually defined relative to an adversary, to analyse whether the UML specification fulfills a security requirement, the tool-support has to automatically include the adversary model arising from the physical view contained in the UML specification.

The architecture and basic functionality of the UMLsec analysis suite are illustrated in Fig. 2. The overall architecture is divided between the UML drawing tool in use and the analysis suite. This way the analysis suite can be offered as a web-application (where the users use their drawing tools to construct the UML model which is then uploaded to the analysis suite), which facilitates maintenance. Additionally, a locally installable version is under development at the time of writing. Plugins for various UML drawing tools are also planned.

The usage of the analysis suite as illustrated in Fig. 2 proceeds as follows: The developer creates a model and stores it in the UML 1.4 / XMI 1.2 file format. The file is imported by the UMLsec tool into the internal MDR repository. The MDR (MetaData Repository) library is part of the Netbeans project. The tool accesses the model through the JMI interfaces generated by the MDR library. The Java Metadata Interface (JMI) standard defines a mapping from the Meta-Object-Facility (MOF) used to define UML to Java. The Static Checker parses the model, verifies its static features, and delivers the results to the Error Analyzer. The Dynamic Checker

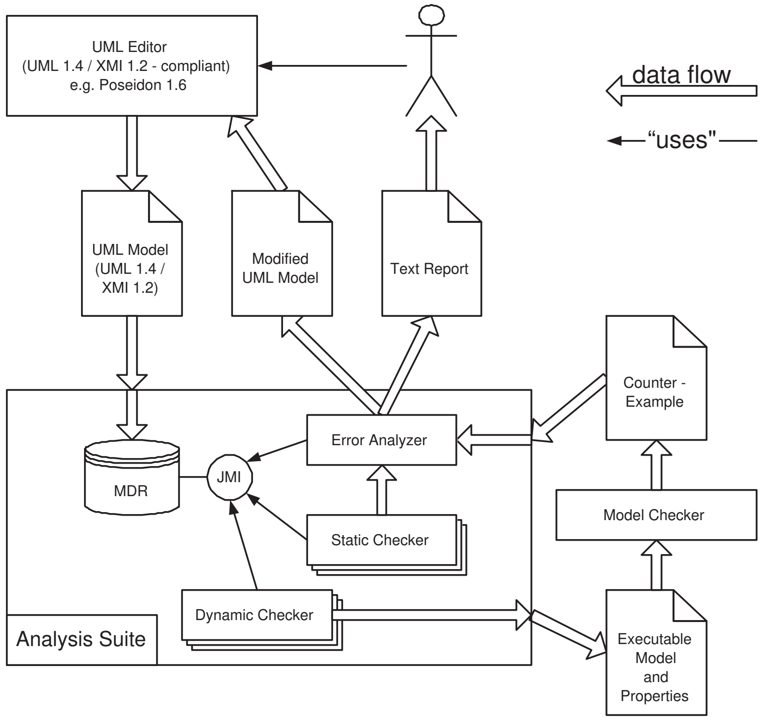

Figure 2: UML tools suite

translates the relevant fragments of the UML model into the model checker input language. The results of the model checker (currently, a prototypical binding with Prolog is used here) are delivered back to the Error Analyzer. The Error Analyzer uses the information received from both Static Checker and Dynamic Checker to produce a Text Report for the developer describing found problems, and a Modified UML Model, where the found errors are visualized and (as far as possible) corrected.

3.2 Implementation Details

We now explain a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. The goal is that advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes. In particular, the framework includes the UMLsec tool web interface, so that new routines are also accessible over this interface.

The idea behind the framework is thus to provide a common programming framework for the developers of different verification modules (*tools*). Thus a tool developer should be able to concentrate on the verification logic and not be required to become involved with the input/output interface. Different tools, implementing verification logic modules (*Static Checkers* or *Dynamic Checkers* in Fig. 2) can be independently developed and integrated. At the time of writing, there exist verification modules for most UMLsec stereotypes.

An added tool implementation needs to obey the following assumptions.

- It is given a default UML model it operates on. It may load further models if necessary.
- The tool exposes a set of commands which it can execute.
- Every single command is not interactive. It receives parameters, executes, and delivers feedback.
- The tool can have its internal state which is preserved between commands.
- Each time the tool is called with a UML model, it may give back a text report and also a UML model.

These assumptions were made in order for the framework to cover as much common functionality as possible while not becoming overly complicated. Experience indicates that the assumptions are not too restrictive, given the architecture at hand (see Fig. 2).

The tool architecture then allows the development of the verification logic independently of the input and output media with minimum effort. Each tool is required to implement the ITextMode interface which exposes tool functionality in text mode, with string array as input and text as output. The framework provides default wrappers for the Graphical User Interface (GUI) GuiWrapper and the Web mode WebWrapper. These wrappers enable using the tool without modifications in the GUI application (which is part of the framework) or through a web interface by rendering the output text on the respective media. However, each tool may itself implement the IGuiMode and/or IWebMode to fully exploit the machineality of the corresponding media, for example to fully use GUI mode capabilities to display graphical information.

At the time of writing, the UMLsec tools are working with the UML 1.4 version, which can be stored in a XMI 1.2 (XML Metadata Interchange, [Obj02]) format by a number of existing UML design tools (including Poseidon 1.6 Community Edition [Gen03]).[2]

3.2.1 Translation to Prolog

We explain our translation to Prolog in some more detail.

The idea is to use a predicate $\mathsf{knows}(E)$ meaning that the adversary may get to know E during the execution of the protocol. For any data value s specified to be confidential in the UMLsec model using the tag $\{\mathsf{secret}\}$, one thus has to check whether one can derive $\mathsf{knows}(s)$.

The set of predicates defined to hold for a given UMLsec specification is defined as follows.

For each publicly known expression E, one defines $\mathsf{knows}(E)$ to hold.

To model the fact that the adversary may enlarge his set of knowledge by constructing new expressions from the ones he knows (including the use of encryption and decryption), one defines

$$\begin{aligned}\forall E_1, E_2.\quad & (\mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2) \Rightarrow \\ & \mathsf{knows}(E_1 :: E_2) \wedge \\ & \mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(\mathcal{D}ec_{E_2}(E_1)) \wedge \\ & \mathsf{knows}(\mathcal{S}ign_{E_2}(E_1)) \wedge \mathsf{knows}(\mathcal{E}xt_{E_2}(E_1)) \wedge \\ & \mathsf{knows}(\mathcal{H}ash(E_1)) \wedge \\ & \mathsf{knows}(\mathbf{head}(E_1)) \wedge \mathsf{knows}(\mathbf{tail}(E_1))).\end{aligned}$$

Suppose we are given a connection $l = (\mathsf{source}(l), \mathsf{guard}(l), \mathsf{msg}(l), \mathsf{target}(l))$ in a sequence diagram with $\mathsf{guard}(l) \equiv \mathit{cond}(\mathit{arg}_i, \ldots, \mathit{arg}_j)$, and $\mathsf{msg}(l) \equiv \mathit{exp}(\mathit{arg}_i, \ldots, \mathit{arg}_j)$, where the parameters arg_i of the guard and the message are variables which store the data values exchange during the course of the protocol. For each such connection, we define a clause as follows:

$$\begin{aligned}\forall \mathit{exp}_1, \ldots, \mathit{exp}_n.\quad & (\mathsf{knows}(\mathit{exp}_1) \wedge \ldots \wedge \mathsf{knows}(\mathit{exp}_n) \\ & \wedge \mathit{cond}(\mathit{arg}_i, \ldots, \mathit{arg}_j) \\ & \Rightarrow \mathsf{knows}(\mathit{exp}(\mathit{exp}_1, \ldots, \mathit{exp}_n))\end{aligned}$$

This way, the adversary knowledge set is approximated from above (because one abstracts away for example from the message sender and receiver identities and the message order). This means, that one will find all possible attacks, but one may also encounter "false positives". However, this treatment turns out to be rather efficient.

Note that due to the undecidability of first-order predicate logic, one cannot in general automatically establish that the adversary does *not* get to know a certain data value. This is the opposite situation for example from that in [Sch97]. We aim to combine the two approaches to try to minimize the situations that cannot be decided automatically (see Sect. 5).

4 Common Electronic Purse Specifications

We apply UMLsec to a security analysis of the Common Electronic Purse Specifications (CEPS) [CEP01]. CEPS is a candidate for a globally interoperable electronic purse standard supported by organizations (including Visa International) representing 90 percent of the world's electronic purse cards and likely to become an accepted standard, making its security an important goal.

Stored value smart cards ("electronic purses") have been proposed to allow cash-free point-of-sale (POS) transactions offering more fraud protection than credit cards: Their built-in chip can perform cryptographic operations which allows transaction-bound authentication (whereas credit card numbers are valid until the card is stopped, enabling misuse). The card contains an account balance that is adjusted when loading the card or purchasing goods.

Here we consider a central part of CEPS, the purchase transaction, an off-line protocol which allows the cardholder to use the electronic value on a card to pay for goods. We give a slightly simplified account to keep the presentation feasible.

The participants involved in the off-line purchase transaction protocol are the customer's card and the merchant's POS device. The POS device contains a Purchase Security Application Module (PSAM) (for example a smart card) that is used to store and process data (and assumed to be tamper-resistant). After the protocol, the account balance in the customer's card is decremented, and the balance in the PSAM is incremented by the corresponding amount. The card issuer later receives transaction logs.

In addition to transactions using public terminals it is also intended to use CEPS cards for transactions over the Internet [CEP01, Bus. Req. ch.X].

In Fig. 3 we give a specification of the purchase transaction as a UML specification $\mathcal{P}$. We explain the specification below.

Apart from incremental transactions (not considered here), security machineality is provided only by the PSAM (and not the rest of the POS device). Thus our protocol participants are the CEP card C (with identity ID_C and public resp. private keys K_C resp. K_C^{-1}) and the PSAM P (with identity ID_P and pub-

[2] An upgrade to UML 2.0 is in development at the time of writing.

lic resp. private keys K_P resp. K_P^{-1}). Both also have stored the public key K_{CA} of the certification authority before the transaction.

We assume for simplicity that the sequence of transaction amounts M_{NT} (indexed by the transaction number NT) is given, as well as the sequence of (mutually independent) session keys SK_{NT}.

We leave implicit the actual adjustment of the balance on the card (which includes checking that the balance is greater than the charged amount).

At the beginning of its execution in the POS device, the PSAM creates a transaction number NT with value 0. Before each protocol run, NT is incremented. If a certain limit is exceeded, the PSAM stops machineing (to avoid rolling over of NT to 0). The protocol between the card C, the PSAM P and the display D is supposed to start after the card C is inserted into a POS device containing P and D and after the amount M is communicated to the PSAM (by typing it into a terminal assumed to be secure).

Each protocol run consists of the parallel execution of the card's and the PSAM's part of the protocol. Card and PSAM begin the protocol by exchanging certificates $ID_C, K_C, \mathcal{S}ign_{K_{CA}^{-1}}(ID_C :: K_C)$ resp. $ID_P, K_P, \mathcal{S}ign_{K_{CA}^{-1}}(ID_P :: K_P)$ containing their identifier ID_C resp. ID_P and their public key K_C resp. K_P, together with the same information signed with the private key K_{CA}^{-1} of the certification authority. Both check the validity of the received certificate (i. e. they check that the signature consists of the received identifier and public key, signed with the private key K_{CA}^{-1} of the certification authority, by verifying the signature with the public key K_{CA}). Note that the card has no information regarding the identity of P that ID_P itself could be verified against. The PSAM then proceeds by sending the Debit-for-Purchase message containing the transaction number NT, and an encryption of the following data under the public key k_C received in the card's certificate: The concatenation of the price M_{NT} of the good to be purchased, a (symmetric) session key SK_{NT}, and the following data signed with the private key K_P^{-1}: the amount M_{NT}, the key SK_{NT}, P's identifier ID_P, the data id_C earlier received as C's identifier, and the transaction number NT. The card then checks the validity of the signature with the earlier received public key k_P against the received data amount m, the received key sk, the received identifier id_P, the own identifier ID_C and the received transaction number nt. The card then returns firstly, E, which consists of the values ID_C, id_P, m, and nt, signed with the private key K_C^{-1} and encrypted under the key sk, and secondly the values m and E signed with the key sk. The PSAM verifies that the second part of the received message is the concatenation of the amount M_{NT} sent out previously and the first part of the message, signed with the key SK_{NT} sent out earlier, and verifies that the first part of the message, after decryption with the key SK_{NT}, gives the signature of the concatenation of the values id_C, ID_P, M_{NT} and NT. If all the verifications succeed, the protocol finishes, otherwise the execution of the protocol stops at the failed verification.

Security threat model The CEP specifications require the smart card and the PSAM (but not the POS device [CEP01, Bus.req. p.13, Funct.req. p.20]) to be tamper-proof. The purchase transaction is supposed to provide mutual authentication between the terminal and the card using a certificate issued by a Certification Authority and containing the card's or PSAM's public key.

The smart card is inserted into a POS device and can thus communicate with the PSAM. Since there is no direct communication between the cardholder and her card, the information displayed by the POS device regarding the transaction has to be trusted at the point of transaction. Security for the customer against fraud by the merchant is supposed to be provided by checking the card balance after the transaction and complaining to the merchant, and if necessary to the card issuer, in the case of incorrect processing. Similarly, security for the merchant against the customer is supposed to be provided by exchanging the purchased good only for a signed message from the card containing the transaction details, for which the merchant will receive the corresponding monetary amount from the issuer in the settlement process afterwards. More precisely, the merchant possessing the PSAM with identifier ID_P will, when presenting the signature $E = \mathcal{S}ign_{K_C^{-1}}(ID_C :: ID_P :: M_{NT} :: NT)$, receive the monetary amount M_{NT} from the account of the cardholder of the card with identifier ID_C, once for each NT.

The idea is that risk of fraud is kept small since fraud should be either prevented or at least later detected in the settlement, and certificates of cards or PSAMs actively involved in fraud can be revoked (using revocation lists whose treatment is omitted here). Note that some kinds of fraud can only be detected after a transaction. For example, the cardholder is unable to communicate with the card directly to authorize the transaction. Therefore, the POS device could simply charge a higher amount to the card than shown in its display.

Thus we have the following three security goals.

Cardholder security The merchant can only claim the amount which is registered on the card after the transaction (and thus can be checked with the cardholder's cardreader).

Merchant security The merchant receives a valid signature in exchange for the sold good.

Card issuer security The sum of the balances of all valid cards and all valid PSAMs remains unchanged by the transaction.

When investigating the threats, one needs to take into account that the protocol is also expected to be

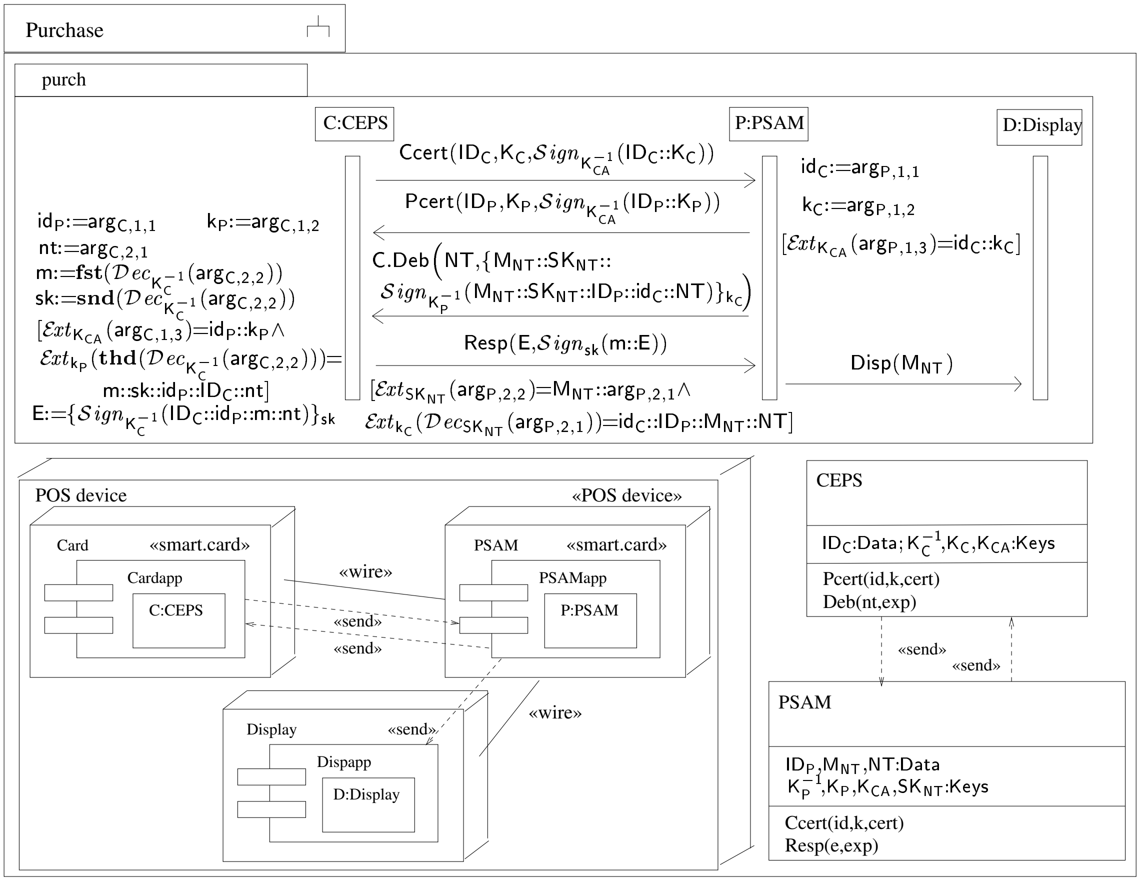

Figure 3: Specification for the CEPS purchase transaction

used over the Internet, and that the POS device in which the PSAM resides and which provides the communication link between the card and the PSAM is not considered to be within the security perimeter, as mentioned above.

The above discussion leads us to the following more precise formulations of the security goals.

We call a key K_X *valid* for a card or PSAM with identifier ID_X if there exists $\mathcal{S}ign_{K_{CA}^{-1}}(ID_X :: K_X)$.

Cardholder security For all $ID_C, ID_P, M_{NT}, NT, K_C^{-1}$ such that K_C is valid for ID_C, if P is in possession of $\mathcal{S}ign_{K_C^{-1}}(ID_C :: ID_P :: M_{NT} :: NT)$ then C is in possession of $\mathcal{S}ign_{K_P^{-1}}(M_{NT} :: SK_{NT} :: ID_P :: ID_C :: NT)$ (for some SK_{NT} and K_P^{-1} such that the corresponding key K_P is valid for ID_P).

Merchant security Each time D receives the value M_{NT}, P is in possession of $\mathcal{S}ign_{K_{CA}^{-1}}(ID_C :: K_C)$ and $\mathcal{S}ign_{K_C^{-1}}(ID_C :: ID_P :: M_{NT} :: NT)$ for some ID_C, K_C^{-1} and a new value NT.

Card issuer security After each completed purchase transaction, let S be the sum of all M_{NT} in the sequence consisting of the processed elements of the form $\mathcal{S}ign_{K_C^{-1}}(ID_C :: ID_P :: M_{NT} :: NT)$ (with possibly varying ID_C, ID_P and K_C^{-1}, such that the corresponding key K_C is valid for ID_C and where the NT are mutually distinct for fixed C). Also, let S' be the sum of all $M'_{NT'}$ in the sequence of processed $\mathcal{S}ign_{K_{P'}^{-1}}(M'_{NT'} :: SK'_{NT'} :: ID_{C'} :: ID_{P'} :: NT')$ (with possibly varying $ID_{C'}, ID_{P'}$ and $K_{P'}^{-1}$, such that the corresponding key $K_{P'}$ is valid for $ID_{P'}$, and where the NT' are mutually distinct for fixed C'). Then S is no greater than S'.

Results According to the assumptions of the CEP specifications, we consider a threat scenario where the attacker is able to access the POS device links, and can access other PSAMs over the Internet, but is not able to tamper with the smart cards (i. e. the *insider* attacker from Fig. 1).

Vulnerability Under the current threat scenario, our tool finds the following weakness with regards to a formalization of the above goal of Merchant security, arising from the fact that the POS device is not secured against a potential attacker that may try to betray the merchant, and that CEPS is also to be used over the Internet. The attacker could for example be an employee, which is a realistic scenario. It turns out that the attacker can simply redirect the messages between the card C and the PSAM P to another PSAM P', for example with the goal to buy electronic con-

$$C \xrightarrow{\mathsf{Ccert}(ID_C, K_C, \mathcal{S}ign_{K_{CA}^{-1}}(ID_C::K_C))} A \xrightarrow{\mathsf{Ccert}(ID_C, K_C, \mathcal{S}ign_{K_{CA}^{-1}}(ID_C::K_C))} P'$$

$$C \xleftarrow{\mathsf{Pcert}(ID_{P'}, K_{P'}, \mathcal{S}ign_{K_{CA}^{-1}}(ID_{P'}::K_{P'}))} A \xleftarrow{\mathsf{Pcert}(ID_{P'}, K_{P'}, \mathcal{S}ign_{K_{CA}^{-1}}(ID_{P'}::K_{P'}))} P'$$

$$A \xleftarrow{\mathsf{Deb}(NT, \{M_{NT}::SK_{NT}::\mathcal{S}ign_{K_{P'}^{-1}}(M_{NT}::SK_{NT}::ID_{P'}::id_C::NT)\}_{k_C})} P'$$

$$C \xleftarrow{\mathsf{Deb}(NT, \{M_{NT}::SK_{NT}::\mathcal{S}ign_{K_{P'}^{-1}}(M_{NT}::SK_{NT}::ID_{P'}::id_C::NT)\}_{k_C})} A$$

$$C \xrightarrow{\mathsf{Resp}(E, \mathcal{S}ign_{sk}(m::E))} A \xrightarrow{\mathsf{Resp}(E, \mathcal{S}ign_{sk}(m::E))} P'$$

$$A \xrightarrow{\mathsf{Disp}(M_{NT})} D$$

Figure 4: Attack

tent, and to let the cardholder pay for it using the corresponding amount. The attacker also sends the required message to the display which will then reassure the merchant that he has received the required amount. The attack has good chances to go undetected: The cardholder will not notice anything suspicious, because the deducted amount is correct. Also, the card registers the identifier $id_{P'}$ rather than id_P, but the identifiers are non-self-explanatory data that the cardholder cannot be assumed to verify (and the card has no information about what the identity of P should be). Furthermore, the identifier id_C in the Deb message is as expected, since P' correctly assumes to be in a transaction with C. The merchant who owns P will notice only later a lacking amount of M_{NT}. The message flow diagram corresponding to this attack is given in Fig. 4 (using the notation of Fig. 3 and where $E{:=}\{\mathcal{S}ign_{K_C^{-1}}(ID_C::ID_{P'}::M_{NT}::NT)\}_{sk}$).

Note also that the attack is simplified if we assume that the attacker also can eavesdrop on the connection between the terminal where the amount M_{NT} is entered and the PSAM P (then the attacker only has to intercept M_{NT}, redirect all messages from C to P' and back, and finally send $\mathsf{Disp}(M_{NT})$ to the display). If in addition to that we assume that the cardholder coincides or collaborates with the attacker, the attacker could simply intercept and remove M_{NT} and send $\mathsf{Disp}(M_{NT})$ to the display, because then the cardholder receives the good without having to pay for it.

Proposed solution The problem can be solved by securing the communication link between PSAM and display, for example by using a smart-card with integrated display as the PSAM (and by making sure that this PSAM cannot be replaced without being noticed). We discuss the security of the improved version of the protocol. In fact, it cannot be guaranteed that C and P terminate the protocol with the same value for M_{NT}, since a "redirection attack" similar to the above still applies, only that the display cannot any more be manipulated, which means that it would be noticed immediately if the PSAM received less money than expected.

We consider the formalized security goals from above, which we proved to be satisfied by the proposed solution. Because of the undecidability of first-order logic, the proof was performed manually, see Sect. 3.2.1. We are currently working towards removing this limitation, see Sect. 5. We write $\mathcal{K}_A^p$ for the knowledge of an adversary of type A prior to the execution of the protocol.

Theorem 1 *Consider adversaries of type A = insider with*

$$\mathcal{K}_A^p \cap \Big(\{K_C^{-1}, K_P^{-1}, K_{CA}^{-1}\} \cup \{SK_{NT} : NT \in \mathbb{N}\}$$
$$\cup \{\mathcal{S}ign_{K_P^{-1}}(E) : E \in \mathbf{Exp}\} \cup \{\mathcal{S}ign_{K_C^{-1}}(E) : E \in \mathbf{Exp}\}$$
$$\cup \{\mathcal{S}ign_{SK_{NT}}(E) : E \in \mathbf{Exp} \wedge NT \in \mathbb{N}\}\Big) = \emptyset$$

and such that for each $X \in \mathbf{Exp}$ with $\mathcal{S}ign_{K_{CA}^{-1}}(X::K) \in \mathcal{K}_A^p$, $X = ID_C$ implies $K = K_C$ and $X = ID_P$ implies $K = K_P$. The above security guarantees are provided by $\mathcal{P}'$ in the presence of adversaries of type A.

5 Conclusion and Future Work

We applied UMLsec, the UML extension for secure systems development, to a security analysis of the purchase protocol from the Common Electronic Purse Specifications (CEPS). The general aim of this line of work is, on the one hand, to transport established knowledge from security engineering into the context of software development by developers who may not themselves be security experts. On the other hand, the goal is to help putting tools from the formal methods community to a more widespread use in industry, by making use of a notation commonly used in industry.

To conclude, it seems that the more specific goals of performing a security analysis of part of CEPS and of demonstrating usability of UMLsec were addressed successfully: Our approach helped us to discover a potential weakness, correct it, and show that the correction is secure in a precise sense, by referring to a formal semantics for the used fragment of UML. This was done in the context of the relatively intuitive and user-friendly UML notation and using the tool-support provided for this approach, and it demonstrated the usability of UML and more specifically UMLsec for this purpose. Also, it is hoped that demonstrating the usefulness of the approach in the given non-trivial case-study contributes towards the more general aim of putting formal tools to increased use in practice. The method proposed here has already been successfully applied in security-relevant projects in industry, for example in projects involving a large German bank.

In future work in the context of the research project Verisoft, we aim to link our tool to the automatic theorem prover SETHEO. Using the complementary approach from [Sch97] based on BAN logic (see [SC01] for background on this), we hope that this way we can also diminiush the current restrictions of the two approaches inherent to the undecidability of first-order predicate logic (because our approach aims to prove the existence of an attack, while the approach in [Sch97] aims to prove its absence).

Acknowledgements Joint work with the UMLsec group on tool-support for the automated analysis of UMLsec models is gratefully acknowledged (see [JSA+03]).

References

[BDCL+01] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, and G. Savoia. Dependability analysis in the early phases of UML based system design. *Journal of Computer Systems Science and Engineering*, 16:265–275, 2001.

[CEP01] CEPSCO. Common Electronic Purse Specifications, 2001. Business Requirements vers. 7.0, Functional Requirements vers. 6.3, Technical Specification vers. 2.3, available from http://www.cepsco.com.

[CSC02] V. Cortellessa, H. Singh, and B. Cukic. Early reliability assessment of UML based software models. In *Workshop on Software and Performance*, pages 302–309, 2002.

[Gen03] Gentleware. http://www.gentleware.com, 2003.

[HdBLS02] S.-H. Houmb, F. den Braber, M. Soldal Lund, and K. Stolen. Towards a UML profile for model-based risk assessment. In Jürjens et al. [JCF+02], pages 79–92.

[HKLB98] C. L. Heitmeyer, J. Kirby, B. G. Labaw, and R. Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *CAV*, pages 526–531, 1998.

[HMP+02] G. Huszerl, I. Majzik, A. Pataricza, K. Kosmidis, and M. Dal Cin. Quantitative analysis of UML Statechart models of dependable systems. *The Computer Journal*, 45(3):260–277, 2002.

[JCF+02] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. *Critical Systems Development with UML*, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

[JRFF03] J. Jürjens, B. Rumpe, R. France, and E. Fernandez, editors. *Critical Systems Development with UML*, number TUM-I0317 in TUM technical report, 2003. UML'03 satellite workshop proceedings.

[JSA+03] J. Jürjens, P. Shabalin, E. Alter, S. Meng, M. Schwaiger, G. Kokavecz, S. Schwarzmüller, and S. Shen. UMLsec tool, 2003. Webinterface at http://www4.in.tum.de/~umlsec.

[Jür01] J. Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge*, pages 93–108, Paris, 11–13 June 2001. IFIP, Kluwer. Proceedings of SEC 2001 – 16th International Conference on Information Security.

[Jür04] J. Jürjens. *Secure Systems Development with UML*. Springer, March 2004. To be published.

[JW01] J. Jürjens and G. Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In B. Schmid, K. Stanoevska-Slabeva, and V. Tschammer, editors, *Towards the E-Society: E-Commerce, E-Business, and E-Government*, pages 489–506. IFIP, Kluwer, 2001. First IFIP Conference on E-Commerce, E-Business, and E-Government (I3E 2001).

[KAH99] J. Kirby, M. Archer, and C. Heitmeyer. Applying formal methods to an information security device: An experience report. In *4th IEEE International Symposium on High Assurance Systems Engineering (HASE '99)*, pages 81–88. IEEE Computer Society, 1999.

[Mea95] C. Meadows. Formal verification of cryptographic protocols: A survey. In J. Pieprzyk and R. Safavi-Naini, editors, *Asiacrypt 94*, volume 917 of *LNCS*, pages 135–150. Springer, 1995.

[Obj02] Object Management Group. OMG XML Metadata Interchange (XMI) Specification. Available at http://www.omg.org/cgi-bin/doc?formal/2002-01-01, January 2002.

[Obj03] Object Management Group. OMG Unified Modeling Language Specification v1.5: Revisions and recommendations, March 2003. Version 1.5. OMG Document formal/03-03-01.

[SC01] P. F. Syverson and I. Cervesato. The logic of authentication protocols. In *Foundations of Security Analysis and Design (FOSAD 2000)*, volume 2171 of *LNCS*, pages 63–136, Bertinoro, Italy, 2001.

[Sch97] J. Schumann. Automatic verification of cryptographic protocols with SETHEO. In W. McCune, editor, *CADE-14*, volume 1249 of *LNCS*, pages 87–100. Springer, 1997.

[WJ02] G. Wimmel and J. Jürjens. Specification-based test generation for security-critical systems using mutations. In *International Conference on Formal Engineering Methods (ICFEM)*, volume 2495 of *LNCS*, pages 471–482, Shanghai, China, Oct. 22-25 2002. Springer.