Design for Trust: Security in M-Commerce

Dominik Haneberg, Alexander Kreibich, Wolfgang Reif, Kurt Stenzel

Lehrstuhl für Softwaretechnik und Programmiersprachen
Fakultät für Informatik
Universität Augsburg
D-86135 Augsburg
E-Mail: {haneberg, kreibich, reif, stenzel}@informatik.uni-augsburg.de

Abstract

We present a method for the development of secure smartcard applications: communication protocols are modelled using UML activity diagrams, while class diagrams are used for the internal state of the participants. These graphical descriptions are converted into an algebraic specification. Problems and challenges are described using a small yet interesting example, an electronic copycard.

1 Introduction

For some time it has been predicted that e- or m-commerce would become an import business segment with a lot of new services. However, we still have not seen a great deal of cutting-edge applications. Among the reasons for this is the lack of confidence both from the customers as well as from the services providers. The problem is that personal data of the customer, electronic goods or signatures and other business goods are exchanged electronically. It is obvious that this poses a threat to the customers privacy as well as the possibility of fraud for the providers. To rule out a threat to the different security properties the exchanged data has to be protected. This is done using cryptographic protocols. Unfortunately good cryptographic protocols are hard to design, they tend to be very error-prone [AN95].

The aim of this paper is to sketch a method [HRS02] to develop e- or m-commerce applications that satisfy highest security demands. This is done using formal specification and verification techniques to guarantee the reliability of the communication and the correctness of the implementations. Our method combines different techniques to reach our different goals:

- We use special UML diagrams to enable a simple yet unambiguous specification of the components and the protocols of the application.
- For the verification of the communication protocols we use an algebraic specification that is basically generated from the diagrams and enriched with certain additional information.
- To ensure the correctness of the implementation we use a dynamic logic calculus capable of performing proofs over genuine JavaCard programs.

The method is implemented in the KIV system [BRS$^+$00] and developed in the Go!Card project[1].

2 An example

As a simple example for a smartcard application we describe an open copy card application for a library or university. We need three components:

- a smartcard that holds ‘value points’;
- a ‘filling station’, i.e. a machine that accepts money and loads value points onto the card;
- a ‘pay station’, i.e. a card reader connected to a copier that subtracts value points before printing a copy.

Usage of the card is simple and illustrated in Fig. 1: the card holder inserts the card into the filling station, sees the number of value points left on the card, selects the number of points to load, and inserts the necessary money into the machine. Then the new points are loaded (added) onto the cardlet. To copy with the card, it is simply inserted into the card reader of the copier. The reader displays the

[1]The Go!Card project is sponsored by the Deutsche Forschungsgemeinschaft (DFG) in the priority programme ”Sicherheit in der Informations- und Kommunikationstechnik”

remaining number of points, and the copier asks the cardlet to pay (subtract) one or more points before printing a copy.

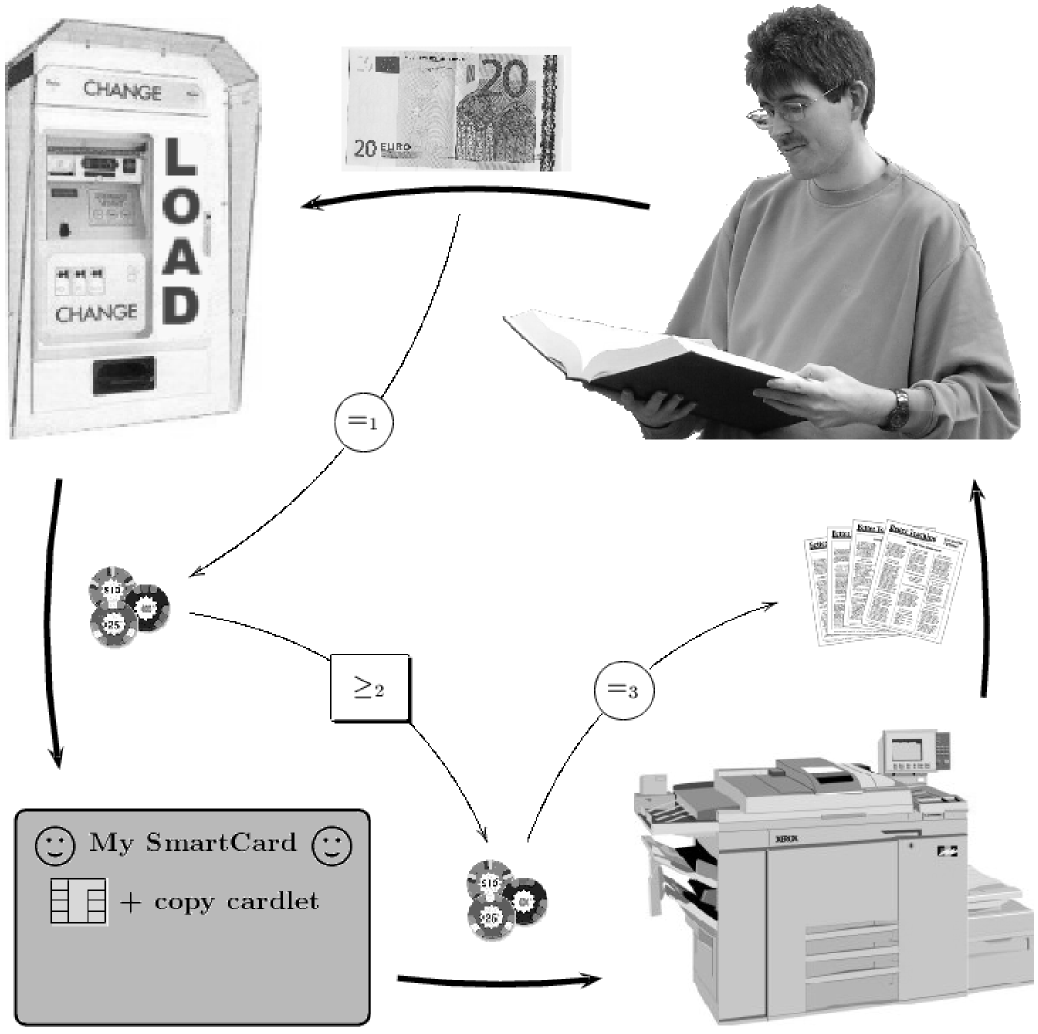

Figure 1: Example scenario

The most important security requirement of the application provider is

> *The sum of all points issued at all filling stations should be larger or equal to the sum of all collected points at all pay stations.*

Given two special attributes of the terminal. the security property can be formulated as follows. The field `collected` counts the number of points that were paid by cards, `issued` counts how many points have been loaded onto cards. The requirement can be written as:

$$\text{local-sec: collected} \leq \text{issued}$$

This is not trivial because an attacker, for example the card holder, could try to cheat by eavesdropping, by manipulating the communication or by using forged cards.

The challenge is to design communication protocols such that the above security requirement is guaranteed in spite of the attacker.

3 The solution in a nutshell

1. **Modelling and formalization**
   (a) Model the relevant parts of the scenario with UML class diagrams augmented by algebraic specifications. This includes the cardlet, the terminals, and their data.

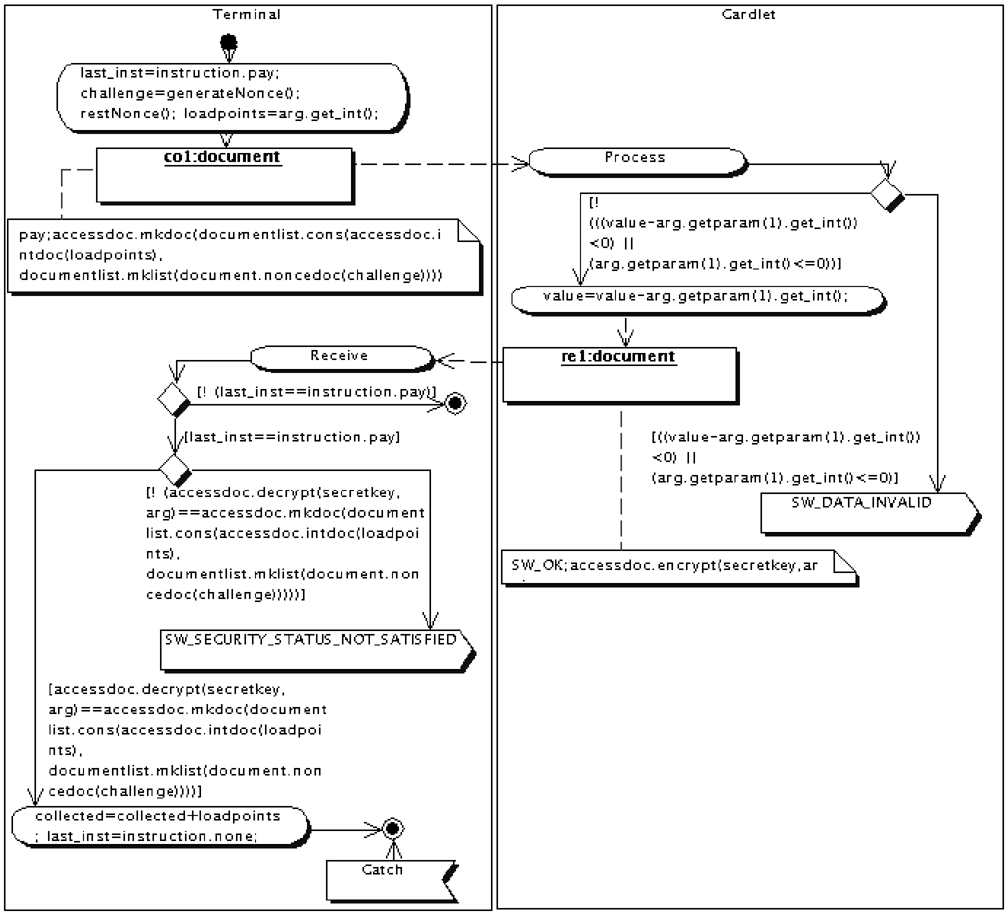
Terminal
Cardlet
last_inst=instruction.pay; challenge=generateNonce(); restNonce(); loadpoints=arg.get_int();
co1:document
pay;accessdoc.mkdoc(documentlist.cons(accessdoc.intdoc(loadpoints), documentlist.mklist(document.noncedoc(challenge))))
Process
[! (((value-arg.getparam(1).get_int()) <0) || (arg.getparam(1).get_int()<=0))]
value=value-arg.getparam(1).get_int();
re1:document
Receive
[! (last_inst==instruction.pay)]
[last_inst==instruction.pay]
[((value-arg.getparam(1).get_int()) <0) || (arg.getparam(1).get_int()<=0)]
SW_DATA_INVALID
SW_OK;accessdoc.encrypt(secretkey,ar
[! (accessdoc.decrypt(secretkey, arg)==accessdoc.mkdoc(documentlist.cons(accessdoc.intdoc(loadpoints), documentlist.mklist(document.noncedoc(challenge)))))]
SW_SECURITY_STATUS_NOT_SATISFIED
[accessdoc.decrypt(secretkey, arg)==accessdoc.mkdoc(documentlist.cons(accessdoc.intdoc(loadpoints), documentlist.mklist(document.noncedoc(challenge))))]
collected=collected+loadpoints ; last_inst=instruction.none;
Catch

Figure 2: Activity diagram of the *pay* protocol

Our focus is on the smartcard in the complete scenario, i.e. the programs on the card and the communication with the terminals. We propose to take the following steps using particular techniques for developing secure smartcard applications:

   (b) Formulate the security properties as class invariants and/or constraints.
   (c) Define the capabilities of an attacker as an algebraic specification.
   (d) Design the communication protocols with UML activity diagrams.
2. **Proving security**
   (e) The UML model is transformed into an

algebraic specification (together with the attacker capabilities); the security properties become proof obligations.

(f) Prove the security properties.

3. **Refinement and verification**
   (g) Refine the abstract data types used in the formal model of the scenario to JavaCard data types (byte arrays etc.).
   (h) Implement the card program in JavaCard.
   (i) Proof obligations are generated from the protocol axioms and the refinement for the correct behaviour of the program.
   (j) Prove the correctness of the refinement and the implementation.
4. **Allowing multiple applications**
   A mandatory security policy for smartcards allows to employ open multiapplicative cards. The application provider must be sure that nobody – not even the card holder – can manipulate or read the card program because it typically contains secret keys [SRS+00, SRS+02].

4 Modelling and verification of m-commerce protocols

The protocols of a smartcard scenario are modelled with UML activity diagrams. Figure 2 shows the *pay* protocol of the copycard example. The objects and their internal state are specified in a UML class diagram. Every actor of a protocol has a swimlane in the activity diagram. Following an activity diagram from the start to the end node presents the intended flow of the protocol, the faultless completion of the protocol. Error handling is treated by the send and receipt nodes. The communication steps can be identified by looking at the border of the swimlanes. Each object flow link crossing a border represents a data exchange. As we need a protocol description in the algebraic specification we have to take the step from the graphical notation to the specification language of the KIV system. Therefore we implemented a transformation that parses the activity diagrams, follows the the flow of the protocol and generates axioms that describe the behavior of the components. Two aspects must be considered:

- the answer a component generates upon receiving data and
- the modifications to the internal state of a component that result from the processing of received data.

For each component a specific function is generated that deals with these two aspects. E. g. the function

$$\text{send} : \text{cardlet} \times \text{document} \rightarrow \text{cardlet} \times \text{document}$$

is responsible for the cardlet. It describes how the current state of the cardlet is modified as result of certain input and what the result of the cardlet is. The axioms for these functions are directly generated from the acitivity diagram. In our example the axiom describing an successfull payment by the card is:

$$\begin{aligned}
&\text{pay\_cardlet\_1:}\\
&\qquad \neg\,(\text{cl .value} - \text{get\_int(getparam(1, doc))} < 0\\
&\qquad\quad \vee\ \text{get\_int(getparam(1, doc))} \leq 0)\\
&\quad \wedge\ \text{cl}_1 = \text{set\_value(cl, cl .value} -\\
&\qquad\qquad\qquad \text{get\_int(getparam(1, doc)))}\\
&\rightarrow \text{send(cl, commanddoc(pay, doc))} =\\
&\quad \text{cl}_1 \times \text{responsedoc(encdoc(cl}_1\text{ .secretkey, doc),}\\
&\qquad\qquad\qquad \text{SW\_OK)}
\end{aligned}$$

This axiom corresponds to the left successor of the branch node in the cardlet swimlane of figure 2.

5 Proving security properties

The specification of the protocols and the components is enriched with an attacker model which describes the specific abilities of the attacker in the given application. Parameters of the attacker specification are his abilities to decrypt and compose messages, which communication he can eavesdrop on and what data he can guess. The attacker model is an algebraic specification that is joined with the protocol specification.

Proving security properties is done by reasoning about lists of events (so-called traces). Every event represents the transfer of a data set. It includes the transferred data, the sender, the receiver and the current state of all components. The proofs are generally done by induction. The approach to the verification is similar to Paulson's work [Pau98].

For the verification the local security property local-sec is turned into the global security property

$$\begin{aligned}
\text{sec\_glob:}\quad & \text{init(tr)} \wedge \text{admissible(tr)}\\
&\rightarrow \forall n.\ \text{tr}[n]\text{.env.term.collected} \leq\\
&\qquad \text{tr}[n]\text{.env.term.issued}
\end{aligned}$$

by adding quantification over every event in the trace and adding conditions that ensure that only well-formed traces have to be considered.

Given the overall specification

$$\mathbf{SPEC} = \text{classes} + \text{protocols} + \text{attacker}$$

the correctness of the security property (sec_glob) can now be established by proving that

$$\mathbf{SPEC} \models \text{sec\_glob}$$

6 Verification of the implementation

The security of the application is only guaranteed if the protocols, proven to be correct, are implemented without flaws in the terminal as well as in the cardlet. Nevertheless the correctness of the cardlet is more crucial. It is nearly impossible to correct an error after the distribution of the application. We therefore suggest the verification of at least the card program. We focus only on cards programmable in JavaCard (a sub-language of Java) tailored to the limited ressources of smartcards. JavaCard does not support certain complex aspects of full Java (e. g. garbage collection and threads) and is hence accessible to formal treatment. The verification of the implementation is done using a dynamic logic [Har79] calculus for JavaCard implemented in KIV [Ste01].

7 Conclusion

We sketched a methodology and a framework based on formal methods to solve security concerns in innovative e- and m-commerce applications. It is possible to prove that the designed protocols and the implementation guarantee the claimed security properties. Although our method was originally tailored to deal with smartcard applications it applies as well for a wider variety of m-commerce scenarios. It applies not only to smartcards, but also to mobile phones, PDAs and other portable devices.

References

[AN95] R. Anderson and R. Needham. Programming satan's computer. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*. Springer LNCS 1000, 1995.

[BRS⁺00] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, number 1783 in LNCS. Springer, 2000.

[Har79] D. Harel. *First Order Dynamic Logic*. LNCS 68. Springer, Berlin, 1979.

[HRS02] D. Haneberg, W. Reif, and K. Stenzel. A Method for Secure Smartcard Applications. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology and Software Technology, Proceedings AMAST 2002*, LNCS 2422, Berlin, 2002. Springer.

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

[SRS⁺00] G. Schellhorn, W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a formal security model for multiapplicative smart cards. In *Proc. of the* 6th *European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 17–36. Springer, 2000.

[SRS⁺02] G. Schellhorn, W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a formal security model for multiapplicative smart cards. *Journal of Computer Security, special issue on ESORICS 2000*, 10(4):339 – 367, 2002.

[Ste01] Kurt Stenzel. Verification of JavaCard Programs. Technical report 2001-5, Institut für Informatik, Universität Augsburg, Germany, 2001. Available at http://www.Informatik.Uni-Augsburg.DE/swt/fmg/papers/.