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

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.

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. The method is implemented in the KIV system [Ba00] 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.

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

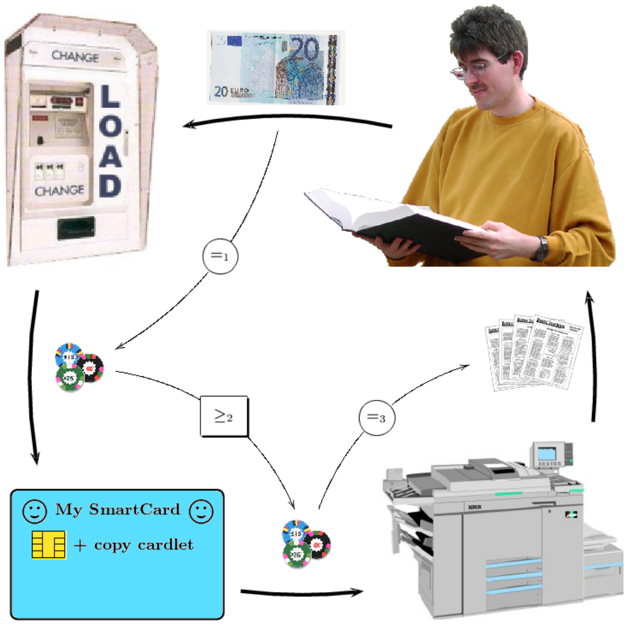

Figure 1: Example scenario

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

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.*

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

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:

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.

   (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 [St01].

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 [Sc00].

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. All the models and the security property are transferred into an algebraic verification logic and proved with the KIV system.

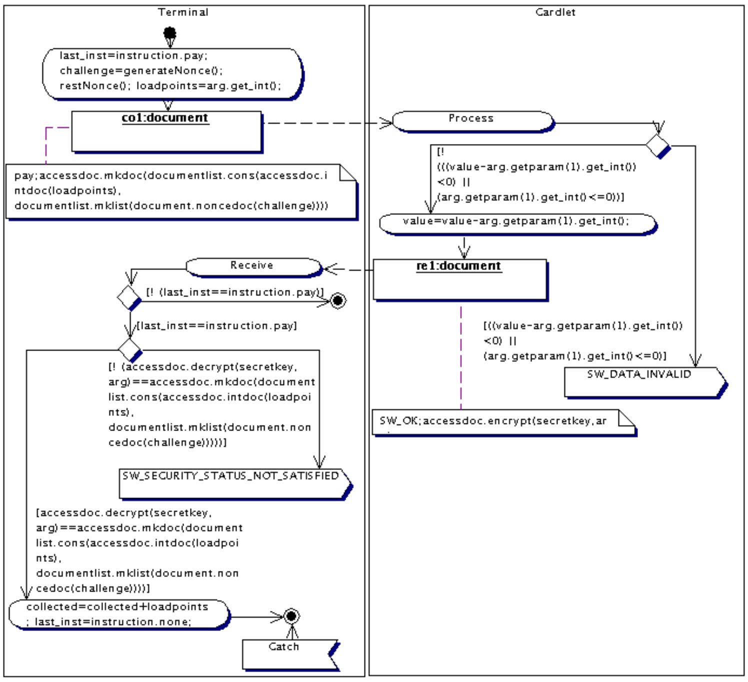
Terminal
last_inst=instruction.pay;
challenge=generateNonce();
restNonce(); loadpoints=arg.get_int();
co1:document
pay;accessdoc.mkdoc(documentlist.cons(accessdoc.intdoc(loadpoints),
documentlist.mklist(document.noncedoc(challenge))))
Receive
[! (last_inst==instruction.pay)]
[last_inst==instruction.pay]
[! (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
Cardlet
Process
[! (((value-arg.getparam(1).get_int()) <0) || (arg.getparam(1).get_int()<=0))]
value=value-arg.getparam(1).get_int();
re1:document
[((value-arg.getparam(1).get_int()) <0) || (arg.getparam(1).get_int()<=0)]
SW_DATA_INVALID
SW_OK;accessdoc.encrypt(secretkey,ar

Figure 2: Activity diagram of the *pay* protocol

5 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. The method applies not only to smartcards, but also to mobile phones, PDAs and other portable devices.

References

[Ba00] Balser, M., Reif, W., Schellhorn, G., Stenzel, K., und Thums, A.: Formal system development with KIV. In: Maibaum, T. (eds.), *Fundamental Approaches to Software Engineering*. Number 1783 in LNCS.

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

[Sc00] Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V., und Toll, D.: 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.

[St01] Stenzel, K.: 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/.