

# A Construction Kit for Modeling the Security of M-Commerce Applications

Dominik Haneberg, Wolfgang Reif, Kurt Stenzel

Lehrstuhl für Softwaretechnik und Programmiersprachen  
Institut für Informatik, Universität Augsburg  
86135 Augsburg Germany

**E-Mail:** {haneberg, reif, stenzel}@informatik.uni-augsburg.de

**Abstract.** In this article we present a method to avoid security problems in modern m-commerce applications. The security problems that we are addressing are breaches of security due to erroneous cryptographic protocols. We describe a specification technique that gives way to a formal, and thereby rigorous, treatment of the security protocols used in such applications. Security of communication is important in modern m-commerce applications. As parts of the specification of the security protocols, we describe how to specify the behavior of the agents, how to specify the attacker and how further aspects of the application reflect in the formal specification. The problem is that such formal specifications are difficult to get right, so we propose a construction kit for their development.

## 1 Introduction

### 1.1 Mobile-Commerce

Mobile-commerce applications appear in a large variety of forms. They can be customer-oriented (electronic selling of goods) or they can be intra-organizational (electronic business processes). Although the electronic selling is better known in the society, the mobile reorganization of business processes is getting more and more important. The devices used for m-commerce are manifold. Smart cards for high-secure applications, mobile phones or mobile digital assistants for smart mobile services or notebooks with mobile Internet access for heavyweight applications. Independent of the device used for the service, the requirements of m-commerce services are quite similar: Security of transmitted data, identification of the agents, exclusion of fraud. Yet these goals seem to be quite simple and even though they are claimed in all applications, realizing them properly is quite tricky. Differences in the application design, different possible forms of fraud in different applications and the large differences in the technical features of the mobile devices result in securing an m-commerce application being a challenging task.

## 1.2 Security Protocols for Communication

One important aspect of innovative m-commerce applications is the transformation of business goods into digital data. Examples for such applications are electronic ticketing or electronic purses. It is crucial for such applications that the business goods cannot be manipulated or multiplied because this would permit fraud. For example, someone could increment the amount of money in an electronic purse. Another aspect is that m-commerce applications can require the transfer of customer data that must not be disclosed, e. g. credit card data or other personal information. Communication in m-commerce therefore means a lot of data that must be protected against different threats. The means for protecting data and ensuring authenticity are cryptographic methods. For the different functions of the application, security protocols (also called cryptographic protocols) must be developed that ensure the security of the data. The problem is that these protocols are very error prone [AN95].

There are quite a few well-known security protocols that can be used to ensure many of the more common security demands (e. g. non-disclosure of the transmitted data and authenticity of the agents). One such protocol, often used in WWW-based e-commerce applications, is SSL [FKK96]. The problem is that such standard protocols are not usable in all scenarios because they do not necessarily guarantee the application specific security goals. In smart card based applications, which are the main focus of the work presented here, the additional problem arises that smart cards do not offer the resources necessary to execute such standard protocols.

Our work is a method for the specification and verification of cryptographic protocols for m-commerce applications and we therefore address the security problems arising from erroneously designed cryptographic protocols.

## 1.3 Designing Secure M-Commerce Protocols

Cryptographic protocols are difficult to design. They may contain subtle errors that are not detected for several years (see [BAN90] for examples). It is commonly agreed that formal methods give the highest assurance that the protocols are secure. This means that we can formally prove that a given protocol adheres to all required security properties (which must be given in a suitable mathematical description). As basis of such a formal treatment of security protocols, a specification that unambiguously describes the protocols is needed.

However, creating such specifications is quite complex. The commonly used specification techniques are unsatisfying because they are either too detailed, for example the full EMV standard [EMV00] or they lack a lot of relevant information, e. g. treatment of errors or checks of the data exchanged [BAN90]. The formal proofs are surprisingly difficult, because an informal argumentation necessarily contains holes. And there is a graver problem: If the formal specification does not adequately reflect the real world, a ‘proof’ that a protocol is secure may be possible, but in the real world the protocol is not secure: In [BAN90] the

security of a version of the Needham-Schroeder protocol is ‘proven’, but in fact the protocol was flawed [Low96]. Even an expert in formal methods is at risk.

Therefore we propose a ‘construction kit’ to design formal protocol specifications. While every application requires its own protocols and security properties, experience shows that the remaining parts of a formal specification can be standardized. Using standard components reduces the risk of errors and allows to reuse previous proofs.

#### 1.4 A Construction Kit for Formal Protocol Specifications

We propose a *construction kit* to design correct formal protocol specifications without much effort. The kit contains the following building blocks:

1. A UML [OMG03] class diagram for the agents in the application, and UML activity diagrams for the protocols (described in section 2).
2. Security properties (described in section 3).  
These can be either class invariants or formulas of the formal specification. Additionally, the formal specification should be validated, i. e. it should fulfill some properties. These properties are: a successful protocol run is possible; and a modified protocol is insecure. The diagrams and security properties are specific for every application and must be designed by the application designer.
3. One of three different attacker models (described in section 4):
  - An attacker that may receive and modify every message.
  - An attacker that receives every message, but cannot modify a message.
  - An attacker that cannot eavesdrop on or modify messages between other agents.
4. The communication structure (described in section 5).  
This includes the number of agents, and the manner of communication. The more realistic this aspect is modeled the more complex the specification and proofs will be. However, a too simple model may be not an adequate model of reality.

Some parts of the building blocks are generated automatically, some are taken from a library and a small part must be added by hand. The building blocks are unified in one large formal specification in which the correctness and security of the cryptographic protocols are proven.

## 2 Modeling Applications

In this section we describe our technique to specify the communication protocols and those parts of the application that contain data relevant for the security (most often the application logic but not the user interface). A specification of an application consists of two parts, the description of the agents and the protocols. The agents are all the entities that play an active role in the application. They are represented by their internal state. Unlike the common descriptions of security

protocols we explicitly model the internal state of the agents. We therefore can describe which data is stored in each agent and when and how an agent modifies its state. Multiple agents of the same type are possible, similar to multiple objects that are different instances of the same class. The communication protocols of an application describe how the different agents work together, which messages are exchanged and which computations take place.

Specifications that are written in a way that they are usable for verification, e. g. algebraic specifications, are hard to develop and require expert knowledge in formal methods. Our goal was to enable the application developer to contribute to the specification by using an UML-based approach. The scenario and the protocols are described in a graphical notation and are automatically converted into an algebraic specification.

## 2.1 An Example

The example we use to illustrate the specification is a simplified electronic purse. A user of this service can use special terminals to load money onto his smart card by inserting the smart card and the money he wants to load. Then the smart card stores the amount of money in the electronic purse. The smart card with the stored money can be used for payments of small amounts, e. g. in a canteen or in public libraries for using the copying machines.

## 2.2 UML Diagrams

We use two kinds of diagrams for the specification of an application. The state of the agents and the methods used to manipulate data are described in a class diagram. Class diagrams are mainly used in a standard manner and not further discussed.

The protocols which describe the communication between the different agents are specified as activity diagrams. In general each function that is to be performed by the application requires its own protocol. The protocols define at what time what message is to be sent by which agent. Also the structure of the messages is defined by the protocols. Most important thereby is how the data is secured, i. e. what cryptographic primitives are used in what way. Figure 1 shows one of the protocols of the electronic purse, the *pay*-function. This function is used if the user wants to pay for goods or services and use the money on his card for it. The example application has another function, the function used to load money onto the card. This function has its own protocol which is omitted here. The diagram for *pay* contains two swimlanes, one for each agent participating in the protocol. The left swimlane represents the terminal, in the case of the *pay*-function this would be a merchant terminal, and the other swimlane stands for the smart card. The protocols all start at the start-node in the terminal swimlane. In smart card applications the communication is always initiated by a terminal, a smart card cannot begin an operation on its own. The course of events in the protocol is described by the control flow of the diagram. Activities stand for calculations being performed and for modifications of the

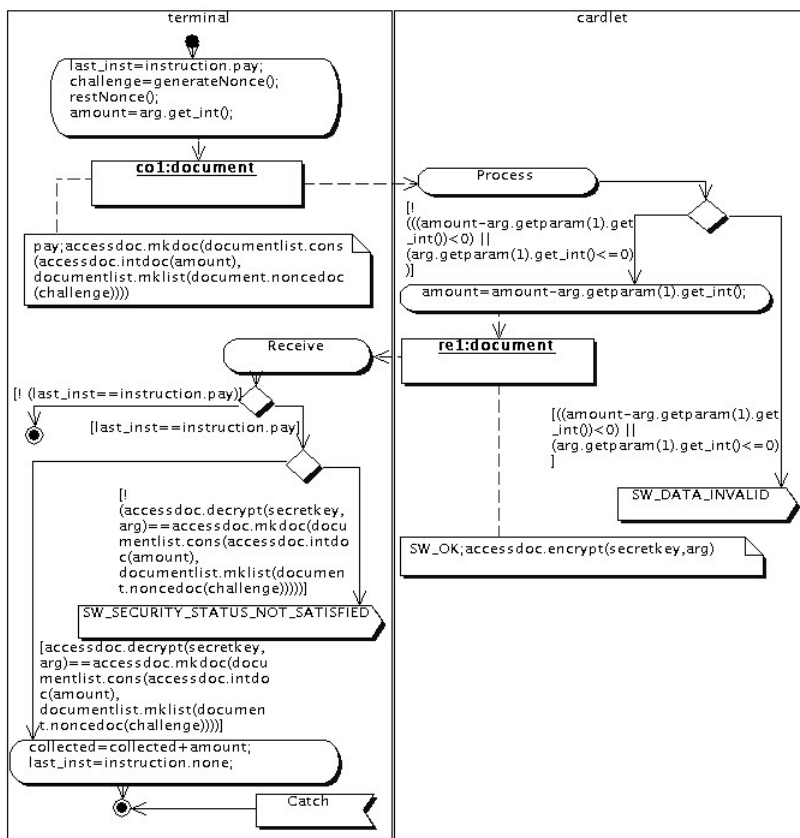


Fig. 1. The Protocol for Payments

state of the agent. A branch node is used for tests that have to be performed, e. g. correctness of received data, object nodes and the notes attached to them describe the exchanged messages. The path from the start node to the last end node describes the intended successful run of the protocol. Signal sending nodes and the other end nodes represent early ends of a protocol run, e. g. because of wrong data.

### 2.3 Algebraic Specifications

The graphical notation described in the last section is good for the specification process, because the notation enables non-experts in formal methods to contribute to the specification. Unfortunately, for the verification we need another description mechanism, one that contains the specification in the language of the theorem prover used. This theorem prover input is, in large parts, automat-

ically generated from the diagrams. In our case we use the KIV theorem prover [BRS<sup>+</sup>00] which works with algebraic specifications [LEW96] and therefore the generation of the formal specification currently targets on algebraic specifications. Of course it would be possible to use other specification techniques, e. g. temporal logics or state machines.

As a result of the automatic transformation process a structured algebraic specification is generated. An algebraic specification consists of a set of data type definitions, functions manipulating the data types and also predicate symbols. Combined with quantifiers we have a powerful formal description mechanism for software systems. The specification created from the diagrams contains the data types for the agents of the application, a document data type describing how messages are built in the communication and a function that describes the agents behavior in the protocols.

Each agent is represented by its internal state, i. e. the value of all fields of the data type. When generating the data type for an agent, the fields are taken from the agents description in the class diagram. This explicit model of the internal state of the agents participating in an application is a significant difference to the other approaches for modeling and verifying security protocols. As we are ultimately aiming at a verified implementation of the application, we think that a model that is close to the actual software is better than the stateless descriptions in other modeling techniques.

The function describing the agents behavior is the central part of the protocol specification. It defines for all agents how they react, i. e. what they answer and how they modify their state, if they receive a certain message. The function

$$\text{agent-says: } (\text{agent}, \text{document}) \rightarrow (\text{agent}, \text{document})$$

takes the current state of an agent and the document the agent receives and returns the state of the agent after the modifications and the document the agent produces as answer. The axioms for this function are generated from the activity diagrams. Note that all the different activity diagrams are brought together in one function. This is due to the fact that each one of the activity diagrams only contains the information how the agents mentioned in the diagram behave in one specific function of the application, but for a description of an agent as a whole all these parts must be aggregated.

A great deal of the algebraic specification can be generated automatically, yet some parts must be added by hand. It is possible to use methods in the activity diagrams describing the protocols (e. g. `generateNonce()` in Figure 1). These methods must be present in the class diagram, but only the signature can be taken from the class diagram and added to the specification. The semantics of these functions has to be added by hand. They are usually functions that deal with the treatment of data, e. g. a function `storeTicket()` to store a ticket in an array.

### 3 Security Properties

Apart from the specification of the behavior of the agents in an m-commerce application we need two further ingredients in order to prove the security of the communication protocols. At first we need to know what exactly security of the protocols means in a specific application. This is discussed in this section. For the electronic purse the claim of the service provider would be that at no time the sum of the money spent with the cards is higher than the sum of all the money collected in the load stations. In the example, this property is expressed quite elegantly using two fields of the terminal to count the loaded and collected money:

$$\text{term.collected} \leq \text{term.issued}$$

This class invariant is translated into (the additional parts of the formula will be explained later)

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

This theorem guarantees that at no time more money is collected by the point-of-sales terminals than was issued by the load terminals. This rules out fraud because it cannot happen that money is spent that was not previously loaded on a card by a genuine load station. Other applications have completely different security claims. In an electronic shopping scenario, e. g. the customer would demand that his payment information (e. g. credit card data) is not disclosed to anyone other than the merchant.

Finding, respectively determining, the security goals for an m-commerce application is part of the application design. The security demands, most certain at first just as some informal ideas, must be put in rigorous formalization to allow for their formal verification. Therefore the informal demands become logical formulas in the verification process. The theorems representing the security goals are the main object of the protocol correctness proofs.

Besides the actual security goals some additional properties for the application should be proven. They can be seen as ‘sanity’ checks because they validate the specification. Most important are the following:

- It is possible to successfully execute the protocols. This ensures that the specification of the possible traces is not malformed in a way that entirely disables the communication.
- A insecure version of a protocol should be formulated and using this a successful attack should be proven. This ensures that the attacker is not accidentally disabled by the specification of the possible traces.

### 4 Specifying the Attacker

The second building block that has to be added is the description of the threats the application is facing. The threats we are dealing with are not such unwanted

events limiting the availability of the service, e. g. hardware failures. Our focus are the threats resulting from accidental or deliberate misuse by one or more individuals. In the design and analysis of security protocols such individuals are usually called the attackers. To what extent an attacker poses a threat to an application is determined by his abilities. In security protocol analysis in general an instance of the Dolev-Yao attacker [DY81] model is used. This attacker is the most powerful attacker that still permits secure protocols.

The problem is that such a powerful attacker is not realistic in all possible application scenarios. Designing the protocols in a way that they are secure against the Dolev-Yao attacker always guarantees that the application is also secure if the real attacker is less powerful. Therefore we do not have a security problem but we have an economical problem. A more powerful attacker usually means that the security protocols must be more elaborate. They are harder to design and use more advanced cryptographic features. To make things worse, more advanced cryptographic primitives (e. g. asymmetric instead of symmetric cryptography) can necessitate more capable hardware. Additionally the usage of public-key systems could require the build up of a public-key infrastructure, which means a great deal of administrative effort. This all results in increasing costs. For example a smart card that can only perform symmetric cryptography is cheaper than a smart card with support for an asymmetric cryptographic algorithm. All in all we have the problem that an application can become unnecessary complex and expensive if an attacker is selected who is too powerful and therefore inappropriate for the application.

Classical security protocol analysis generally assumes communication over a wide area network such as the Internet. Especially m-commerce scenarios make use of a wider range of possible communication channels:

- Infrared (IrDA),
- radio-based, e. g. Bluetooth for short-range and GSM for long-range communication,
- LANs and WANs,
- but also exclusive point-to-point connections, e. g. a smart card in a terminal.

The different characteristics of the communication channels should reflect in the attacker model. This is another reason why a limitation to the Dolev-Yao attacker is not adequate. We therefore developed several other attackers with different abilities. They represent different levels of realistic adversaries in real life applications.

#### 4.1 Aspects Common to All Attacker Models

Certain aspects of the attacker are common to all the attacker models we use. The central idea of the attacker is that he collects as much information as possible and tries to use the information he gathered to cheat on other agents. The information the attacker has collected is called his knowledge. The attacker's knowledge is a set of documents that initially contains often just the attacker's personal cryptographic keys but it grows when the attacker intercepts new messages.

One common aspect of the attacker models is how they treat the messages they obtain. A document that is received by the attacker is always added to his knowledge. But some documents are made up of sub-documents, e. g. an encrypted document contains the plaintext of the message as sub-document. In case of compound documents special rules apply. Not all sub-documents may go into the attacker's knowledge. The process of adding additional documents to the attacker's knowledge is specified using several functions (similar to [Pau98]), most important:

$$. \mathcal{f}+ . : \text{documentset} \times \text{document} \rightarrow \text{documentset}$$

$\mathcal{f}+$  computes a new set of documents by adding a document to a given set of documents and adding all documents that can be derived. If an attacker, with present knowledge  $\text{docset}$ , receives a document his knowledge is extended to  $\text{docset}_0 = \text{docset} \mathcal{f}+ \text{doc}$ . Some other functions are used in the specification of  $\mathcal{f}+$  and explain how the new knowledge can be computed. In this process, the knowledge of the attacker is extended with all documents derivable from his present knowledge. What the derivable direct sub-documents of a document are depends on the document and the knowledge. For example a document encrypted with key  $key$ ,  $\text{encdoc}(key, \text{doc})$ , contains the plaintext as sub-document but this sub-document is derivable only if the knowledge contains the information necessary to decrypt messages encrypted with  $key$ :

sub-enc-yes :

$$\text{can\_decrypt}(key, \text{docset}) \rightarrow \text{sub-docs}(\text{encdoc}(key, \text{doc}), \text{docset}) = \{\text{doc}\};$$

sub-enc-no :

$$\neg \text{can\_decrypt}(key, \text{docset}) \rightarrow \text{sub-docs}(\text{encdoc}(key, \text{doc}), \text{docset}) = \emptyset;$$

Note that the algebraically specified abstract documents can contain additional information which their real counterparts do not contain, e. g. the  $key$  in an encrypted document. The information that is not contained in the real documents must of course not be a derivable sub-document.

Also common to all attacker models is how they build new messages from the knowledge they have collected so far. They can guess non-confidential data, but they cannot guess confidential data such as nonces or keys. This means they can only use keys and nonces that they acquired by eavesdropping. Otherwise it would be impossible to ensure most of the common security goals. The attacker's ability to derive documents is covered by a special predicate  $\models$  with signature

$$. \models . : \text{documentset} \times \text{document}$$

$\text{docset} \models \text{doc}$  states that the document  $\text{doc}$  can be constructed from the set of documents  $\text{docset}$  which contains the attacker's knowledge. There are axioms for all kind of documents describing if they can be built given a certain set of documents. As mentioned above the attacker cannot guess keys, therefore he can build an encrypted document only if he possesses the key. This is covered by the axiom

$$\begin{aligned} \text{encdoc} : \quad & \text{docset} \models \text{encdoc}(\text{key}, \text{doc}) \\ & \leftrightarrow \text{encdoc}(\text{key}, \text{doc}) \in \text{docset} \\ & \vee \text{docset} \models \text{keydoc}(\text{key}) \wedge \text{docset} \models \text{doc}; \end{aligned}$$

and it states that the attacker can derive an encrypted document if the document is part of his knowledge or if the plaintext of the document and the used key can be derived from his knowledge.

## 4.2 Attacker Models

*The Dolev-Yao Attacker* The Dolev-Yao attacker model is the most powerful attacker model used. This attacker has complete control over the communication, i. e. he can eavesdrop into every data exchange and manipulate all messages traveling through the communication channels. The attacker decomposes all messages he intercepts and decrypts encrypted messages, provided that he has the necessary key. The attacker can arbitrarily generate messages from his knowledge.

It is easy to see that it is not very common that someone has the abilities of this attacker. The attacker must have access to all agents participating in the communication.

*An Attacker with Limited Access to the Communication* This attacker is developed from the Dolev-Yao model by limiting his access to the communication. First of all this attacker can only eavesdrop into the data exchanges but he cannot manipulate the messages sent. This does not mean that he cannot participate in the communication at all. This attacker can still generate his own messages and send them over a communication channel as a regular participant of the service. Another possible limitation is that the attacker cannot eavesdrop on all data exchanges but just a few, i. e. he does not have control over the complete communication infrastructure but on some systems with an installed Trojan horse program.

Prohibiting the attacker to manipulate messages is also necessary in order to prove certain reliability properties. For example it can only be guaranteed that an electronic railway ticket can successfully be presented to the conductor if the attacker cannot simply inhibit all communication by elimination of all messages. Availability of a service cannot be guaranteed with a Dolev-Yao attacker [Mea03].

*Attacker Without Access to the Communication* This is the most restricted attacker we use. This attacker cannot eavesdrop on any communication taking place. What he can do is communicating with his own genuine device, e. g. his own smart card, and he can program a faked device, e. g. using a programmable smart card, that can be used to communicate with real agents and try to trick them into revealing interesting information. This is a quite realistic attacker. Every person with programming knowledge and a smart card reader is an instance of this attacker model. As the attacks that can be performed by this attacker do

not require great effort and are not expensive, attacks of this kind must be expected in every m-commerce application, even if the possible gain of a successful attack is limited.

### 4.3 Further Aspects

What kind of attacker should be used in the verification of the application is a design decision that must be taken in an early step of the application development because it has a direct impact on the protocols and probably on the hardware selection. Some factors must be observed when trying to decide on the most realistic threat. In general one must anticipate more elaborate attacks if the result of a success is more severe. For example a smart card based credit card will attract more and technologically more advanced attacks than for example a loyalty card.

There is another aspect of the attackers that must be kept in mind. It has no direct consequence for the formalization of the attackers but for the design of the security protocols. It is generally assumed that an attacker is trying to achieve a personal benefit by attacking a service. This is not necessarily true. When designing a service one should keep in mind that the service may have to deal with attacks whose only purpose is to annoy others, see e. g. denial of service attacks on the Internet. Preparing for such attacks is especially important if the availability of the service is crucial.

### 4.4 In Brief

The attacker is a very important concept in our treatment of security protocols. There are different aspects to be considered. In brief the following things are important:

- How can the attacker decompose messages and build new ones.
- Which communication channels can be eavesdropped by the attacker, which cannot.
- Can the attacker replace messages in transfer by his own messages.
- Does the attacker participate as a normal user of the service. (This is the case in most scenarios.)
- Does the attacker use faked devices, e. g. a self-programmed smart card.
- Does one attacker suffice or should a set of attackers be considered.
- Is an ‘annoy only’ attacker relevant?

## 5 The Communication Structure

After modeling the protocols and the agents, and after selecting the attacker the communication structure remains to be modeled. This requires further choices:

1. The number of agents.

It may be sufficient to consider only one attacker, one customer, and one merchant. This reduces the complexity of the specification and proofs considerably compared to one that considers an arbitrary number of agents. So this choice is desirable. On the other hand, there may be attacks against the protocol involving several agents, that are not possible otherwise. In this case the formal specification must consider an arbitrary number of agents. Otherwise the specification is not an adequate model of reality.

2. The manner of communication.

This can be either broadcast (one to all for radio based communication) or one to one, and may contain one channel or several. For example, there may be an 'insecure' radio based channel like WAP and a 'secure' channel based on GSM. Again, the choice influences the complexity of the specification and proofs, and how adequate the formal model is.

3. Interleaving of operations.

Is it adequate to assume that every software component immediately answers to a received message? Or is it necessary to assume that things can happen interleaved or even in parallel? Since we are dealing only with the logical properties of the protocols the first possibility is sufficient. However, one agent may have exclusive access to a communication partner. (E. g. a terminal to a smart card if an attacker without access to the communication is used.)

4. Occurrence of state changes.

Every agent has an internal state. This is important when the specification is refined into an implementation. If interleaving is used an agent receives a message and some time later issues an answer. Usually, the internal state is modified. The question is when? When a message is received, when an answer is issued, or sometimes between?

With these choices the formal specification can be completed. An *environment* contains all agents with their internal state, an *event* consists of an environment and a communication between two or more agents, and a *trace* is a sequence of events. A trace is *admissible* if all agents adhere to their specified behavior. In the electronic purse example this means that the smart card, the load stations and the pay stations follow their protocols, and that the attacker sends only messages that are derivable from his knowledge. An admissible trace can be viewed as a sequence of events that can happen in the real world. Consequently the security properties must be proven for all admissible traces (see `sec_glob` in section 3). The definition of admissible can be divided into several parts that are to some degree independent of each other:

$$\begin{aligned} & \text{admissible}(\text{trace}) \\ \leftrightarrow & \text{attacker\_admissible}(\text{trace}) \\ & \wedge \text{protocol\_admissible}(\text{trace}) \\ & \wedge \text{state\_admissible}(\text{trace}) \\ & \wedge \dots \end{aligned}$$

For example, `attacker_admissible(trace)` may look like:

$$\begin{aligned}
& \text{attacker\_admissible}([\text{ev}, \text{ev}_0, \text{trace}]) \\
\leftrightarrow & (\text{ev.from} = \text{attacker} \rightarrow \text{ev.env.known} \models \text{ev.doc}) \\
& \wedge (\text{ev}_0.\text{to} = \text{attacker} \rightarrow \text{ev.env.known} = \text{ev}_0.\text{env.known} \text{ } f+ \text{ev}_0.\text{doc}) \\
& \wedge (\text{ev}_0.\text{to} \neq \text{attacker} \rightarrow \text{ev.env.known} = \text{ev}_0.\text{env.known}) \\
& \wedge \text{attacker\_admissible}([\text{ev}_0, \text{trace}])
\end{aligned}$$

Here, a trace consisting of a last event `ev`, a previous event `ev0`, and a remaining trace is considered. (For verification purposes the first element in a trace is the last event that occurred.) The first part of the axiom specifies that if a document is sent by the attacker (`ev.from = attacker`), he must be able to generate it from his knowledge (`ev.env.known  $\models$  ev.doc`); the second part specifies that if he received a document (`ev0.to = attacker`) in the next to last event it is added to his knowledge and all derivable documents as well (`ev.env.known = ev0.env.known  $f+$  ev0.doc`) so that the knowledge is present in the last event; the third part specifies that if the document is not sent to the attacker his knowledge remains unchanged. This implies an attacker without access to the communication.

`protocol_admissible(trace)` specifies that if a normal agent issues a message in event `ev` it is the answer to a received document in some event `ev0` and follows the protocol as defined by `agent-says`:

$$\begin{aligned}
& \text{protocol\_admissible}([\text{ev}, \text{trace}]) \\
\leftrightarrow & (\text{ev.from} \neq \text{attacker} \\
& \rightarrow \exists \text{ev}_0 \in \text{trace}: \text{agent-says}(\text{ev}_0.\text{agent}, \text{ev}_0.\text{doc}) = (\text{ev.agent}, \text{ev.doc}) \\
& \quad \wedge \text{ev}_0 = \text{matching\_event}([\text{ev}, \text{trace}])) \\
& \wedge \text{protocol\_admissible}(\text{trace})
\end{aligned}$$

If interleaving is used event `ev0` is not necessarily the next event, but can occur somewhere in the trace. This must be specified with `matching_event`. Furthermore, the axiom does not define the internal state of the agent between the two events, or when the state changes. The predicate `state_admissible` specifies that the state of all agents that do not participate in a communication remains unchanged. Further predicates are needed to specify the communication structure exactly.

The problem is that the different building blocks described so far cannot be specified completely separate. Actually, they are interwoven and may contain subtle interactions. This means it is very easy to introduce errors if the specification is written ‘by hand’. A faulty specification is not an adequate model of the real world. In the best case it is not possible to prove that a protocol is indeed secure. In the worst case a ‘proof’ is possible for an insecure protocol. For example, the axioms used to describe admissible traces may be inadvertently contradictory (one axiom may say that a state change must occur, another that a state change may not occur). In this case there are no admissible traces (in the formal model) and every security property is trivially fulfilled. Especially the combination of interleaving and exclusive access to an agent is very difficult to specify correctly.

## 6 Conclusion

In this paper we discuss questions concerning the security of m-commerce applications. We introduce a method to avoid security problems by formally analyzing the application. The problem is that building a formal model is difficult and it is easy to introduce errors. Therefore we propose a ‘construction kit’ that allows an automatic generation of large parts of the specification. The kit consists of several parts: We describe how security protocols can be modeled. We also describe different attackers relevant for m-commerce applications. It is described what different abilities these attackers possess and how some of these abilities are represented in an algebraic specification. We also discuss further aspects that influence the modeling of the application and especially how these factors influence which traces (list of events) are possible and which not.

## 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.
- [BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), Feb 1990.
- [BRS<sup>+</sup>00] M. Balsler, 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, pages 363–366. Springer-Verlag, 2000.
- [DY81] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22th IEEE Symposium on Foundations of Computer Science*, pages 350–357. IEEE, 1981.
- [EMV00] EMVCo LLC. *EMV 4.0 Specifications Book 1 – Application independent ICC to Terminal Interface requirements*, December 2000. <http://www.emvco.com/documents/specification/view/book1.pdf>.
- [FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. *The SSL Protocol Version 3.0*. Netscape Communications, November 1996. <http://wp.netscape.com/eng/ssl3/>.
- [LEW96] J. Loeckx, H. Ehrlich, and M. Wolf. *Specification of Abstract Data Types*. Wiley-Teubner, 1996.
- [Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
- [Mea03] Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.
- [OMG03] The Object Management Group (OMG). *OMG Unified Modeling Language Specification Version 1.5*, 2003. <http://www.omg.org/technology/documents/formal/uml.htm>.
- [Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.