Sound Methods and Effective Tools for Model-based Security Engineering with UML

Jan Jürjens*
Competence Center for IT Security, Software & Systems Engineering
Dep. of Informatics, TU Munich, Germany
http://www4.in.tum.de/~juerjens

ABSTRACT

Developing 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.

We present an extensible verification framework for verifying UML models for security requirements. In particular, it includes various plugins performing different security analyses on models of the security extension UMLsec of UML. Here, we concentrate on an automated theorem prover binding to verify security properties of UMLsec models which make use of cryptography (such as cryptographic protocols). The work aims to contribute towards usage of UML for secure systems development in practice by offering automated analysis routines connected to popular CASE tools. We present an example of such an application where our approach found and corrected several serious design flaws in an industrial biometric authentication system.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Design Tools and Techniques—*Computer-aided software engineering (CASE), Object-oriented design methods*; D.2.4 [Software Engineering]: Software/Program Verification

General Terms: Security

Keywords: Unified Modeling Language, UML, security, verification, biometric authentication, cryptographic protocol, verification framework

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 well-being of people and organizations. Due to the widespread interconnection of information systems, attacks can be waged anonymously and from a safe distance. Many security incidents have been reported, sometimes with potentially quite severe consequences [17].

Any support to aid secure systems development is thus dearly needed. 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. Thus, the goal is to develop security-critical systems that are correct by construction, as advocated in [8]. Towards this goal, the security extension UMLsec for the Unified Modeling Language (UML) has been defined in [11, 13]. It allows us to encapsulate knowledge on prudent security engineering and thereby make it available to developers which may not be specialized in security. One can also go further by checking whether the constraints associated with the UMLsec stereotypes are fulfilled.

The work presented here aims to contribute to enabling the use of this approach in industry by providing automated tool-support for the analysis of UMLsec models against security requirements by checking the constraints associated with the UMLsec stereotypes. Besides presenting a general, extensible framework for implementing verification routines for the constraints associated with security-critical UML stereotypes, we focus on a plug-in that utilizes the automated theorem-prover (ATP) SETHEO to verify security properties of UMLsec models which make use of cryptography (such as cryptographic protocols). To do so, the analysis routine extracts information from behavioral UML diagrams that may contain additional specific cryptography-related information. The current work presents a significant advance over the prior work [15] which used a model-checker rather than an automated theorem prover for the security analysis, was restricted to state-charts, and to specific cryptographic primitives (such as RSA encryption, as opposed to general encryption mechanisms here). Our personal experience has been that using ATPs for formal security requirements analysis has a potential for efficiency and the ability to handle relatively large specification documents, by avoiding the state space explosion problem one often faces when having to deal with a non-deterministic adversary (unless applying specialized optimization techniques). This was

*This work was partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author(s).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-58113-963-2/05/0005 ...$5.00.

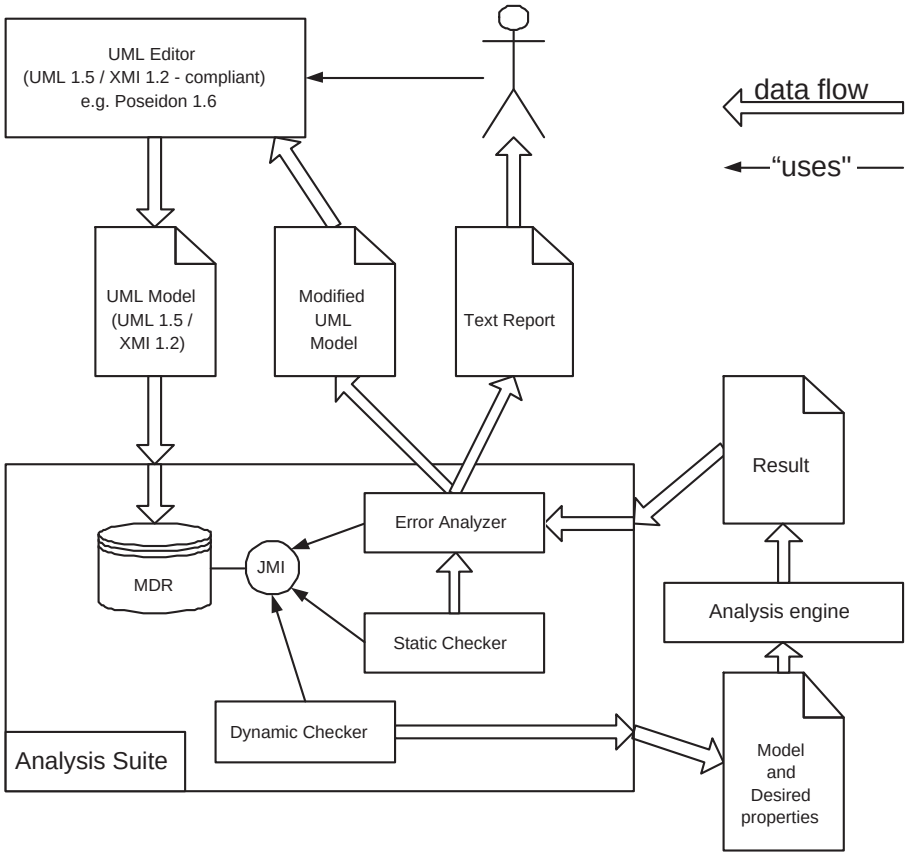

Figure 1: UML verification framework: usage

achieved by devising a completely new translation mapping from UML to first-order logic, rather than to the Promela notation (which now also includes sequence diagrams, which seem particularly useful for protocol specification).

Our goal is thus that the research we present should support the usage of UMLsec in practice by offering analysis routines connected to popular CASE tools using the XMI standard, which allow the automated verification of the constraints associated with the UMLsec stereotypes. As we will explain, our approach has already been useful in industrial application projects identifying several major security flaws in software during its industrial development. Our tool is available as open-source.

Sect. 2 presents the extensible verification framework supporting the construction of automated security requirements analysis tools for UML diagrams and give a short overview over UMLsec model analysis tool plugins for the framework. Section 3 explains how to translate behavioral UML diagrams to first-order logic formulas. In Section 4, we explain the translation at the hand of a variant of the Internet protocol TLS. Section 5 explains how to perform the security analysis using the automated theorem prover. In Sect. 6, we tell our experiences from one of the industrial projects in which our method and tools have been applied. We close with comparisons to related work, a discussion of our work and an outlook on further developments. For some background on the UMLsec extension and on distributed system security, please refer to [13].

2. THE UML VERIFICATION FRAMEWORK

The analysis routine presented in this paper is embedded in a framework supporting the construction of automated requirements analysis tools for UML diagrams. The framework is connected to industrial CASE tools using data integration with XMI [24] and allows convenient access to this data and to the human user. In this paper, we will, as an example for a usage of this framework, present verification routines to verify the constraints associated with the stereotypes of the security extension UMLsec of UML [11] using automated theorem provers.

The goal is, in particular, 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 a way that allows them to concentrate on the verification logic (rather than on user interface issues).

The usage of the framework as illustrated in Fig. 1 proceeds as follows. The developer creates a model and stores it in the UML 1.4/XMI 1.2 file format.[1] The file is imported by the UML verification framework into the internal MDR repository. MDR is an XMI-specific data-binding library which directly provides a representation of an XMI file on the abstraction level of a UML model through Java in-

[1]This will be updated to UML 2.0 once the corresponding DTD has been officially released.

$$\forall E_1, E_2. \quad \big(\mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2) \Rightarrow \mathsf{knows}(E_1 :: E_2) \wedge \mathsf{knows}(\{E_1\}_{E_2}) \wedge \ \mathsf{knows}(\mathcal{S}ign_{E_2}(E_1))\big)$$
$$\wedge \big(\mathsf{knows}(E_1 :: E_2) \Rightarrow \mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2)\big)$$
$$\wedge \big(\mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(E_2^{-1}) \Rightarrow \mathsf{knows}(E_1)\big)$$
$$\wedge \big(\mathsf{knows}(\mathcal{S}ign_{E_2^{-1}}(E_1)) \wedge \mathsf{knows}(E_2) \Rightarrow \mathsf{knows}(E_1)\big)$$

Figure 2: Structural formulas

terfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, statecharts, and stereotypes. It is part of the Netbeans project [23]. Each plug-in accesses the model through the JMI interfaces generated by the MDR library, they may receive additional textual input, and they may return both a UML model and textual output. The two exemplary analysis plug-ins proceed as follows: The static checker parses the model, verifies its static features, and delivers the results to the error analyzer. The dynamic checker translates the relevant fragments of the UML model into the automated theorem prover input language. The automated theorem prover is spawned by the UML framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static checker and dynamic checker to produce a text report for the developer describing the problems found, and a modified UML model, where the errors found are visualized. Besides the automated theorem prover binding presented in this paper there are other analysis plugins including a model-checker binding [15] and plugins for simulation and test-sequence generation.

The framework is designed to be extensible: advanced users can define stereotypes, tags, and first-order logic constraints which are then automatically translated to the automated theorem prover for verification on a given UML model. Similarly, new adversary models can be defined.

The user webinterface and the source code of the verification framework is accessible at [27].

3. TRANSLATING UMLSEC DIAGRAMS TO FIRST-ORDER LOGIC FORMULAS

We explain the automated translation of UMLsec diagrams to first-order logic (FOL) formulas which then allows automated analysis of the diagrams using automated first-order logic theorem provers. A UMLsec diagram is essentially a UML diagram where security properties and requirements are inserted as stereotypes with tags and constraints, although certain restrictions apply to keep the translation feasible. Details are given in [11, 13], although these are not essential here. The restrictions are motivate by the fact that in a large majority of applications, one can contend with a small fragment of the UML notation. Therefore, we omit modeling elements which are usually not necessary for modeling security-critical systems. In the case of sequence diagrams for example, these include timing constraints, in the case of statecharts for example history states. Also, we do not admit annotations with arbitrary code blocks (for example in statecharts). A precise definition of the fragment of UML considered is laid out in [13].

More precisely, we assume that we are given a UML package containing the following kinds of diagrams: A deployment diagram specifies the physical layer of the system, such as system nodes and communication links, and the level of security it provides, using UMLsec stereotypes, such as «Internet» denoting an Internet communication link. From this, in the security analysis, the adversary model is generated in first-order logic who is able to control certain communication links. Secondly, a class diagram describes the data structure of the system, including the security requirements on the system data, for example using the UMLsec tags {secrecy}, {integrity} and {authenticity} which represent the respective requirements. For the security analysis, from this information the conjecture is derived that is to be checked by the automated theorem prover. The package also contains diagrams specifying the intended behavior of the system, which may include an activity diagram coordinating the components or objects in the package, a sequence diagram specifying interaction between them by message exchange, and statecharts specifying the behavior of single components or objects. The behavioral specifications are compiled to first-order logic axioms giving an abstract interpretation of the system behavior suitable for security analysis. In the following, we explain this translation for sequence diagrams. It works similarly for statecharts and activity diagrams.

We assume a set **Keys** of encryption keys disjointly partitioned in sets of *symmetric* and *asymmetric* keys. 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 term algebra generated from the set $\mathbf{Var} \cup \mathbf{Keys} \cup \mathbf{Data}$ with the operations: $\_ :: \_$ (concatenation), $\mathbf{head}(\_)$ and $\mathbf{tail}(\_)$ (breaking up concatenation), $(\_)^{-1}$ (private keys), $\{\_\}_\_$ (encryption), $\mathcal{D}ec_\_(\_)$ (decryption), $\mathcal{S}ign_\_(\_)$ (signing), and $\mathcal{E}xt_\_(\_)$ (extracting from signature). Here $\_$ denotes an argument place-holder; thus for example $\_ :: \_$ represents a binary operation. In this term algebra, we impose the following equations, formalizing the fact that decrypting with the correct key gives back the initial plaintext, and similarly for extraction of signatures: $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$ (for all $E \in \mathbf{Exp}$ and $K \in \mathbf{Keys}$) and $\mathcal{E}xt_K(\mathcal{S}ign_{K^{-1}}(E)) = E$ (for all $E \in \mathbf{Exp}$ and $K \in \mathbf{Keys}$). We also assume the usual laws regarding concatenation, $\mathbf{head}()$, and $\mathbf{tail}()$, and that $K = K^{-1}$ for any symmetric encryption key K.

Based on this formalization of cryptographic operations, important conditions on security-critical data (such as freshness, secrecy, integrity) can then be formulated. This approach goes back to a long line of logic-based security analysis methods including for example [4, 16, 20].

We explain our translation from cryptographic protocols specified as UML sequence diagrams to FOL formulas which can be processed by the automated theorem prover e-SETHEO. The formalization automatically derives an upper bound for the set of knowledge the adversary can gain.

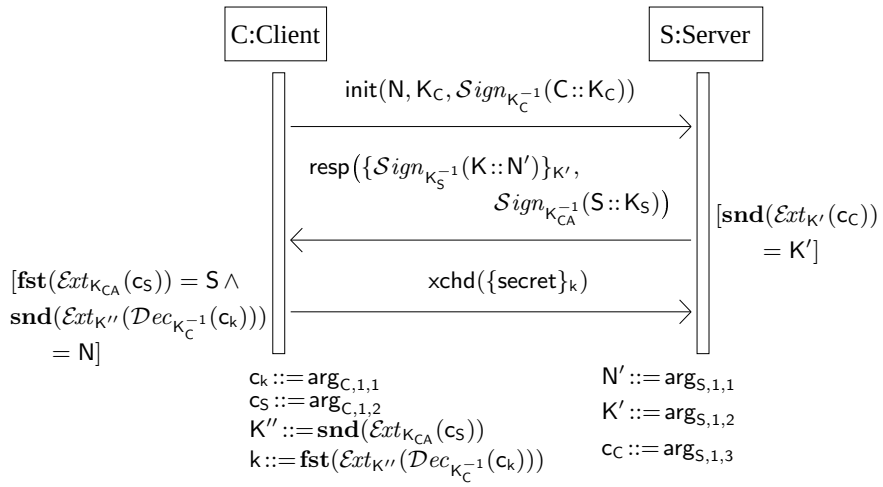

Figure 4: Variant of the TLS handshake

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 supposed to remain secret as specified in the UMLsec model, 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. 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) is captured by the formula in Fig. 2.

For our purposes, a sequence diagram is essentially a sequence of command schemata of the form *await event e – check condition g – output event e'* represented as *connections* in the sequence diagrams. Connections are the arrows from the life-line of a source object to the life-line of a target object which are labeled with a message to be sent from the source to the target and a guard condition that has to be fulfilled.

Suppose we are given a connection $l = (\mathsf{source}(l), \mathsf{guard}(l), \mathsf{msg}(l), \mathsf{target}(l))$ in a sequence diagram with $\mathsf{guard}(l) \equiv cond(arg_1, \ldots, arg_n)$, and $\mathsf{msg}(l) \equiv exp(arg_1, \ldots, arg_n)$, where the parameters arg_i of the guard and the message are variables which store the data values exchanged during the course of the protocol. Suppose that the connection l' is the next connection in the sequence diagram with $\mathsf{source}(l') = \mathsf{source}(l)$. For each such connection l, we define a predicate $\mathsf{PRED}(l)$ as in Fig. 3. If such a connection l' does not exist, $\mathsf{PRED}(l)$ is defined by substituting $\mathsf{PRED}(l')$ with true in Fig. 3.

$$\begin{aligned}&\mathsf{PRED}(l) = \\ &\forall exp_1, \ldots, exp_n.\ \big(\mathsf{knows}(exp_1) \wedge \ldots \wedge \mathsf{knows}(exp_n) \\ &\qquad \wedge\ cond(exp_1, \ldots, exp_n) \\ &\qquad \Rightarrow \mathsf{knows}(exp(exp_1, \ldots, exp_n) \\ &\qquad\quad \wedge\ \mathsf{PRED}(l')\big)\end{aligned}$$

Figure 3: Connection predicate

The formula formalizes the fact that, if the adversary knows expressions $exp_1, \ldots, exp_n$ validating the condition $cond(exp_1, \ldots, exp_n)$, then he can send them to one of the protocol participants to receive the message $exp(exp_1, \ldots, exp_n)$ in exchange, and then the protocol continues. With this formalization, a data value s is said to be kept secret if it is not possible to derive $\mathsf{knows}(s)$ from the formulas defined by a protocol. 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", although this has not happened yet with any real examples. The advantage is that this approach is rather efficient (see Sect. 5 for some performance data).

For each object O in the sequence diagram, this gives a predicate $\mathsf{PRED}(O) = \mathsf{PRED}(l)$ where l is the first connection in the sequence diagram with $\mathsf{source}(l) = O$. The axioms in the overall first-order logic formula for a given sequence diagram are then the conjunction of the formulas representing the publicly known expressions, the formula in Fig. 2, and the conjunction of the formulas $\mathsf{PRED}(O)$ for each object O in the diagram. The conjecture, for which the atp will check whether it is derivable from the axioms, depends on the security requirements contained in the class diagram. For the requirement that the data value s is to be kept secret, the conjecture is $\mathsf{knows}(s)$. An example is given in Sect. 4.

One can define a variation of the formula in Fig. 3 by joining all subformulas $\mathsf{PRED}(l), \mathsf{PRED}(l'), \ldots$ for connections $l, l', \ldots$ in the sequence diagram using the conjunction operator $\wedge$, instead of including the predicate $\mathsf{PRED}(l')$ for next connection l' in the conclusion of the implication in $\mathsf{PRED}(l)$. The effect is that the order of the connections in the sequence diagram is then ignored. This results in a more coarse abstract interpretation of the sequence diagram than that in Fig. 3, which may produce more false positives: allegedly insecure specifications which are in fact secure in reality, because there the order of the connection is in fact observed. However, in particular architectures the order of messages in the sequence diagram is in fact not enforced, and then this variation is useful. For example, this is the

```
input_formula(protocol,axiom,(
  ![ArgS_1_1, ArgS_1_2, ArgS_1_3, ArgC_1_1, ArgC_1_2, ArgS_2_1, DataC_KK, DataC_k, DataC_n] :
  ( % S -> Attacker
    knows(ArgS_1_1) & knows(ArgS_1_2) & knows(ArgS_1_3)
    & ? [X] : equal( sign(conc(X, ArgS_1_2), inv(ArgS_1_2) ), ArgS_1_3 ))
    => knows(conc(enc(sign(conc(kgen(ArgS_1_2), ArgS_1_1), inv(k_s)), ArgS_1_2), sign(conc(s, k_s), inv(k_ca))))
  )
  &
  ( % C -> Attacker
    knows(conc( conc( n,  k_c ),  sign(conc(c, k_c), inv(k_c)) ))
    & ( knows(ArgC_1_1) & knows(ArgC_1_2)
        & equal(sign(conc(s, DataC_KK), inv(k_ca)), ArgC_1_2)
        & equal(enc(sign(conc(DataC_k, DataC_n), inv(DataC_KK) ), k_c), ArgC_1_1 )
        & ( ? [DataC_ks] : equal(sign(conc(s, DataC_ks), inv(k_ca) ), ArgC_1_2 ) )
        & equal(enc(sign(conc(DataC_k, n), inv(DataC_KK) ), k_c), ArgC_1_1 )
        => knows(sym(secret, DataC_k))
      )
  )
)).
```

Figure 5: Protocol part of translation to TPTP

case for the industrial application project which we report on in Sect. 6.

4. A VARIANT OF THE TLS PROTOCOL

We will analyze a variant of the handshake protocol of TLS[2] examined in [13] (note that this is not the variant of TLS in common use). To show applicability of our approach, we demonstrate the flaw from [13], suggest a correction, and verify it. The goal of the protocol is to let a client send a secret over an untrusted communication link to a server in a way that provides secrecy and server authentication, by using symmetric session keys.

The central part of the specification of this protocol is shown in Fig. 4.

Parts that have to be left out here are firstly a deployment diagram specifying that the two protocol participants client and server are connected by an Internet connection, using the UMLsec stereotype «Internet». From this, in the security analysis, the adversary model who is able to control this communication link is generated. Secondly, there is a class diagram which includes various security requirements on the protocol data as UMLsec tags {secrecy}, {integrity} and {authenticity}. For the security analysis, from this information the conjecture is derived that is to be checked by the automated theorem prover. Most importantly, the value secret which is exchanged encrypted in the last message of the protocol is required to remain secret.

Depicted in Fig. 4, the protocol proceeds as we explain in the following. Here we assume that the set **Var** contains elements $\mathsf{arg}_{O,l,n}$ for each $O \in \mathsf{Obj}(D)$ and numbers l and n, representing the nth argument of the operation that is supposed to be the lth operation received by O according to the sequence diagram D.

The client C initiates the protocol by sending the message $\mathsf{init}(\mathsf{N}, \mathsf{K_C}, \mathcal{S}ign_{\mathsf{K_C}^{-1}}(\mathsf{C} :: \mathsf{K_C}))$ to the server S. Suppose that the condition $[\mathbf{snd}(\mathcal{E}xt_{\mathsf{K}'}(\mathsf{c_C}))=\mathsf{K}']$ holds, where $K' ::= \mathsf{arg}_{S,1,2}$ and $\mathsf{c_C} ::= \mathsf{arg}_{\mathsf{S},1,3}$. That is, the key K_C contained in the signature matches the one transmitted in the clear. In that case, S sends the message $\mathsf{resp}\big(\{\mathcal{S}ign_{\mathsf{K_S}^{-1}}(\mathsf{K} :: \mathsf{N}')\}_{\mathsf{K}'}, \mathcal{S}ign_{K_{\mathsf{CA}}^{-1}}(S :: K_S)\big)$ back to C (where $\mathsf{N}' ::= \mathsf{arg}_{\mathsf{S},1,1}$). Then if the condition

$$[\mathbf{fst}(\mathcal{E}xt_{\mathsf{K_{CA}}}(\mathsf{c_S}))=\mathsf{S} \wedge \mathbf{snd}(\mathcal{E}xt_{\mathsf{K}''}(\mathcal{D}ec_{\mathsf{K_C}^{-1}}(\mathsf{c_k})))=\mathsf{N}]$$

holds, where $\mathsf{c_k} ::= \mathsf{arg}_{\mathsf{C},1,1}$, $\mathsf{c_S} ::= \mathsf{arg}_{\mathsf{C},1,2}$, and $\mathsf{K}'' ::= \mathbf{snd}(\mathcal{E}xt_{K_{\mathsf{CA}}}(c_S))$ (that is, the certificate is actually for S and the correct nonce is returned), C sends $\mathsf{xchd}(\{\mathsf{s_i}\}_\mathsf{k})$ to S, where $\mathsf{k} ::= \mathbf{fst}(\mathcal{E}xt_{\mathsf{K}''}(\mathcal{D}ec_{\mathsf{K_C}^{-1}}(\mathsf{c_k})))$. If any of the checks fail, the respective protocol participant stops the execution of the protocol.

The main part of the result of the transformation to the e-SETHEO input format TPTP is the protocol definition given in Fig. 5. We have to omit the formulas representing the initial adversary knowledge and the effect of message recombination on the intruder's knowledge predicate knows. The TPTP notation is the de-facto input notation for first-order logic automated theorem provers [26], supported, using existing converters, by a variety of provers including also Otter, SPASS, Vampire, and Waldmeister.

Note that in this notation, conjunction is written as &, and forall resp. exists quantification as $![\mathsf{X1}, \ldots, \mathsf{Xm}]$ resp. $?[\mathsf{X1}, \ldots, \mathsf{Xm}]$, where $\mathsf{X1}, \ldots, \mathsf{Xm}$ are the quantified variables. Also, encryption, signature, and concatenation are represented respectively as binary functions enc, sign, and conc in TPTP. The private key belonging to the public key K is written as inv(K). Constants, such as the nonce N, have to be written in small letters in TPTP. Also, some of the constructs used for readability in sequence diagrams are elminated: any usage of $\mathbf{head}(E)$ is replaced by introducing the condition $E_1 :: E_2 = E$ and by substituting $\mathbf{head}(E)$ by E_1 at each occurrence. In a similar way, $\mathbf{tail}(\_)$, $\mathcal{D}ec\_(\_)$, and $\mathcal{E}xt\_(\_)$ are eliminated.

The protocol itself is expressed by a forall quantification over variables representing the arguments of messages which are transferred over the communication link. Here, the message variables ArgC_11 and ArgC_12 represent the messages received by the client. The message variables ArgS_11, ArgS_12 and ArgS_13 stand for the server receiving messages parts. The protocol example includes three messages (cf. Fig. 4), of which the first and third are sent by the client and the second by the server. Each message is expressed by a single clause of the main conjunction. Therefore three

[2]TLS (transport layer security) is the successor of the Internet security protocol SSL (secure sockets layer).

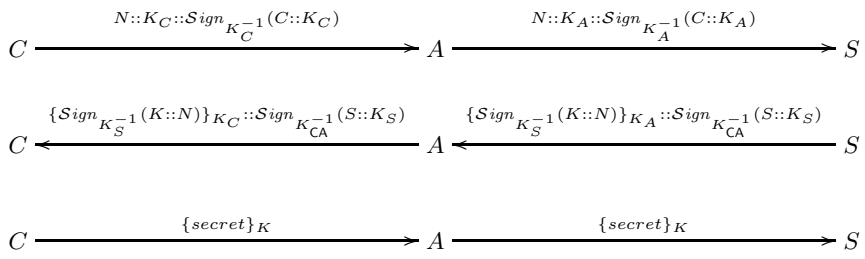

Figure 6: Attack Visualization: Man-in-the-middle

clauses occur in Fig. 5. The first,

$$\mathsf{knows(n)} \,\&\, \mathsf{knows(k\_c)} \,\&\, \mathsf{knows(sign(conc(c, k\_c), inv(k\_c)))},$$

is the message sent from the client to the server. It has no preconditions because it is sent unconditionally without previous receipt of any other message. The second and third protocol messages are specified by implications. In the preconditions of the implications the expected pieces of input message are expressed by knows(Arg_C11), knows(ArgC_12), and knows(ArgS_11), knows(ArgS_12), knows(ArgS_13) respectively. Conditional parts of the received messages are expressed by equations. For instance, the checking if the key of the signed message equals the transmitted key k_c is expressed by

$$?[\mathsf{X}] : \mathsf{equal(sign(conc(X, ArgS\_12), inv(ArgS\_12)), ArgS\_13)}.$$

The conditional equations use the binary function equal(a, b) which is a predefined expression of TPTP syntax and represents the equality relation. In order to express extractions of parts of the messages a pattern matching approach is chosen. One example is the third line of the third protocol message:

$$\mathsf{equal(sign(conc(s,\ DataC\_KK),\ inv(k\_ca)),\ ArgC\_12\ )}$$

The value which is determined by pattern matching is DataC_KK the server's public key for the signature verification. The implication's postconditions include the messages send over the communication channel like for example

$$\mathsf{knows(enc(secret,\ DataC\_k))}$$

in the last message from the client. The specification of fresh keys as it is used in the second message from the server is expressed by encapsulating it within an unary function kgen. For instance the key from the first message of the client within an unary function named kgen, e.g. kgen(ArgS_12).

5. PROTOCOL ANALYSIS WITH ATPS

The prover SETHEO is an efficient automated theorem prover for first oder logic in clausal normal form, extended to the prover e-SETHEO for reasoning about equality properties [22]. We use e-SETHEO for verifying security protocols as a "black box": A TPTP input file is presented to the ATP and an output from the ATP is observed. No internal properties of or information from e-SETHEO is used. This allows one to use e-SETHEO interchangingly with any other ATP accepting TPTP as an input format (such as SPASS, Vampire and Waldmeister) when it may seem fit.

With respect to the security analysis described in Sect. 3, the results of the theorem prover have to be interpreted as follows: If the conjecture stating for example that the adversary may get to know the secret can be derived from the axioms which formalize the adversary model and the protocol specification, this means that there may be an attack against the protocol. We then use an attack generation machine programmed in Prolog to then construct the attack. This program can be downloaded from [27]. If the conjecture cannot be derived from the axioms, this constitutes a proof that the protocol is secure with respect to the security requirement formalized as the negation of the conjecture, because logical derivation is sound and complete with respect to semantic validity for first-order logic. Note that since first-order logic in general is undecidable, it can happen that the atp is not able to decide whether a given conjecture can be derived from a given set of axioms. However, experience has shown that for a reasonable set of protocols and security requirements, our approach is in fact decidable.

In our example, e-SETHEO returns as an output that the conjection knows(secret) can be derived from the defined rules (within three seconds[3]). For this example the attack tracking tool needs around 20 seconds to produce the attack. The derived message flow diagram corresponding to a man-in-the-middle attack is depicted in Fig. 6.

We can fix this problem by substituting $K :: N$ in the second message (server to client) by $K :: N :: K_C$ and by including a check regarding this new message part at the client. Now the new version with the additional signature information about the client key k_c can be verified by the automated theorem prover approach. When e-SETHEO runs on the fixed version of the protocol it now gives back the result that the conjecture knows(secret) cannot be derived from the axioms formalizing the protocol. Note that this result, which was delivered within 5 seconds, means that the actually exists no such derivation, not just that the theorem prover is not able to find it. This means in particular that the attacker cannot gain the secret knowledge anymore. Note that this statement of course in itself is bound to the particular execution model and the formalizations of the security requirements used here. A formal proof that this model and the axiomatizations are sound is sought in future work within the Verisoft project [29]. We do not initially strive of completeness of the axiomatizations in so far as the security analysis may falsely claim that there may be an attack against the specified system, because of the optimizing abstractions used. This, however, has not so far surfaced as a limitation in practical applications.

[3]On a SunFire 3800 (4 prozessors, 6 GByte RAM, Solaris 9).

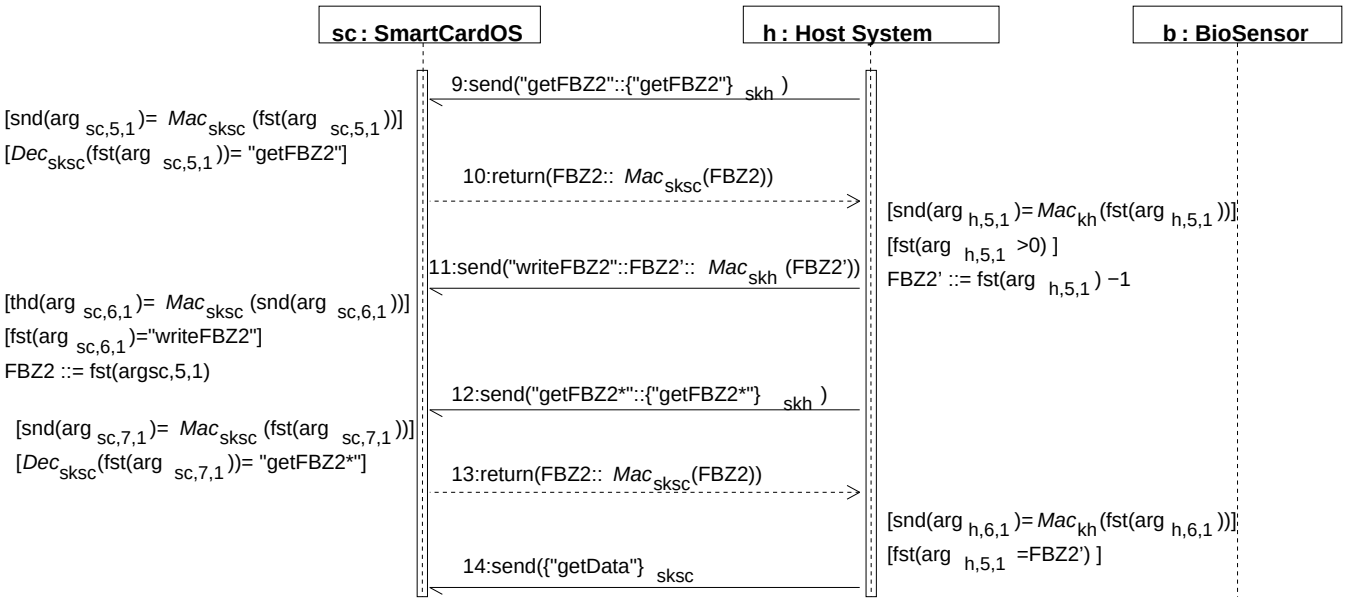

Figure 7: Excerpt from biometric authentication protocol

6. INDUSTRIAL CASE-STUDY: BIOMETRIC AUTHENTICATION

We applied our methods and tools in an industrial application project with a major German company. The goal of the project was the correct development of a security-critical biometric authentication system which is supposed to control access to a protected resource, for example by opening a door or letting someone log into a computer system. In this system, a user carries his biometric reference data on a personal smart-card. To gain access, he inserts the smart-card in the card reader and delivers a fresh biometric sample at the biometric sensor, for example a finger-print reader. Since the communication links between the host system (containing the bio-sensor), the card reader, and the smart-card are physically vulnerable, the system needs to make use of a cryptographic protocol to protect this communication. Because the correct design of such protocols and the correct use within the surrounding system is very difficult, our method was chosen to support the development of the biometric authentication system.

Within the project, the system was specified using UML diagrams: a deployment diagram describing the architecture, a class diagram defining the data structure, an activity diagram specifying the general workflow, and a sequence diagram giving a detailed specification of the cryptographic protocol. A fragment of the sequence diagram is shown in Fig. 7.

In the next step, this specification was enriched with security-relevant information, according to the UMLsec extension. This includes specifying the level of security provided by the physical layer of the system in the deployment diagram, and formulating security goals on the execution of the system and on the protection of particular data values in the activity and class diagrams.

Then the security of the protocol was analyzed using the automated tool support described in the previous sections. The analysis is done with respect to the threat model which is derived from a deployment diagram of the system and the security goals contained in the class diagram, as explained in the previous sections. This way, it turned out that the protocol in fact contains a vital flaw. To prevent an attack where an attacker simply repeatedly tries to match a forged biometric sample, for example, using an artificial finger, with a forged or stolen smart-card, the protocol contains a misuse counter which is decreased from an initial value of 3 to 0, when the card will be disabled. The attack which was found using our tools enables the attacker to prevent the misuse counter from being updated, thereby enabling a brute-force attack.

The relevant part of the attack is displayed in Fig. 8. The attacker is assumed to control the communication between the smart-card and the host system, which is realistic since it is not protected by physical means. He chooses to act as a relay between the smart-card and the host system, until the host system signals the smart-card to decrease its misuse counter $\mathsf{FBZ2}$ by sending it the message $\mathsf{writeFBZ2}$ which contains the new, decreased value $\mathsf{FBZ2}'$ that the smart-card should assign to its counter. This message is simply dropped by the attacker. Note that it is possible to simply drop the message although the integrity of the message is protected using a Message Authentication Code (MAC) in its third argument $\mathsf{Mac}_{\mathsf{skh}}(\mathsf{FBZ2})$. Here skh is a secret key of the host system, which in a correct protocol is supposed to be equal to the secret key sksc of the smart-card. One should note here that the smart-card does not keep an internal state of the protocol execution history. This means that it accepts any of the messages in the protocol at any point. Therefore, after dropping the message telling the smart-card to decrease its misuse counter, the protocol can simply proceed with the next message from the host system, which is again forwarded by the adversary to the smart-card. This problem had already been detected by our tools at an earlier version of the protocol. To fix it, the protocol was extended with the message $\mathsf{getFBZ2}$ by which the host system tries to make sure that the misuse counter has actually

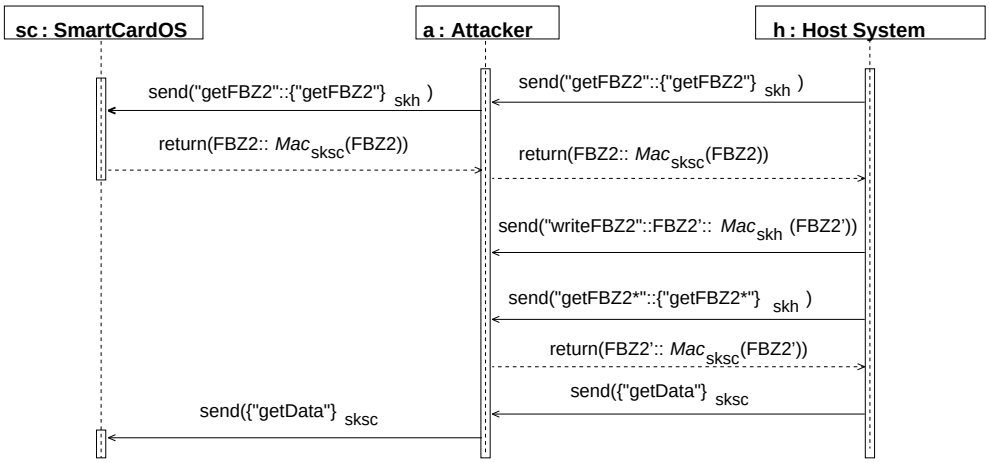

Figure 8: Attack against biometric authentication protocol

been decreased, as shown in Fig. 7. The return value then expected by the host system from the smart-card is the misuse counter FBZ2, protected in its integrity by also sending the MAC $\mathsf{MAC}_{\mathsf{sksc}}(\mathsf{FBZ2})$, which is supposed to be correctly decreased to give the value FBZ'. Unfortunately, this value had already been sent in the previous message writeFBZ2, since the keys skh and sksc are supposed to be the same, so the adversary only needs to replay the value from that message to the host system.

Based on our findings, the protocol was corrected by using a different one of the coding modes suggested in the specification that makes sure that the return message from the getFBZ2 message cannot be a replay of earlier messages, using a freshly generated random value. This corrected version of the protocol is currently subject to ongoing analysis using our tools.

Since UML was used in the development of this system anyhow, the only extra effort needed was to extend the UML diagrams with the security-critical information describing the level of physical security, and the security goals to be achieved, as explained above. Considering the gain from using our methods, namely detecting several mistakes in various versions of the protocol, and making sure that the final version is correct, this modest extra effort seems to be worthwhile. Since the developers involved at our industrial partner were already familiar with the UML notation, they only had to be trained in using the relevant part of the UMLsec notation, and in using our tool-support. Both could be done without too much effort. In conclusion, experiences from this industrial application have been quite positive.

7. RELATED WORK

There are several tools for automated verification of UML models. The vUML Tool [19] analyzes the behavior of a set of interacting objects, defined in a similar way. The tool can verify various properties of the system, including deadlock freeness and liveliness, and find problems like entering a forbidden state or sending a message to a terminated object. [32] presents an approach for consistency-checking of UML specifications involving a knowledge base engine and a tool mapping UML specifications in XMI format to the knowledge base. [2] describes an approach for automated structural and behavioral analyses of UML diagrams including consistency checks, and simulation and model checking for behavioral analysis of UML diagrams. The results are visually interpreted using the original UML diagrams.

We could not make direct use of these tools because they do not support the special features for describing the security mechanisms and properties of the system being modeled, such as cryptographic algorithms. In particular, automated verification of the constraints associated with stereotypes defined within the UMLsec extension (such as the data security requirements), which is vital to support the UMLsec approach, does not seem to be supported by other tools. Also, to our knowledge this is the first work analyzing UML diagrams with an automated theorem prover. When analyzing systems with respect to a highly non-deterministic adversary, we have made the experience that using automated theorem provers allows us to perform a more abstract analysis compared to the use of model-checking, where one might to have to construct the state space of all possible adversary behaviors. Compared to our earlier work in [15], it turned out that the approach presented here is much more efficient and less restricted.

Other approaches for using UML for secure systems development can for example be found in the workshop series [14]. In other approaches to automated software engineering for security, [18] uses the Software Cost Reduction method (SCR) to analyze a cryptographic system called for various security properties. The paper uses SCR tools such as a consistency checker, simulator, model checker, invariant generator, theorem prover, and validity checker to verify that the specification satisfies certain security properties and discusses issues of usability, cost-effectiveness, scalability, and a process for translating an informal specification into a formal notation and for applying a formal method. Also, [9] uses the specification modeling and validation tools of the Interactive Specification Acquisition Tools (ISAT) suite to analyze network security gateway products. [5] uses the first-order formal specification language Alloy to provide a formal basis for security architecural styles.

There is too much work on verifying cryptographic protocols to give a complete overview. Overviews of applications of formal methods to security protocols can be found

for example in [1, 21]. Close to our work is for example [30]. It describes research on a translation from e-commerce protocols into formal models that are then validated using model checking. This approach is applied to the design of the e-commerce protocol NetBill. Other approaches to using first-order logic ATPs for cryptoprotocol analysis include the following: [25] formalizes the well-known BAN logic in first-order logic and uses the atp SETHEO to proof statements in the BAN logic. It is different from our approach which is based on the knowledge of the adversary, instead of the beliefs of the protocol participants. [31] analyzes the Neuman-Stubblebine key exchange protocol using first-order monadic Horn formulas and the atp Spass. This approach differs from ours for example in that in general we also admit non-monadic Horn formulas (and even non-Horn formulas), to be able to consider unbounded state when necessary to express a security property. [3] uses first-order invariants to verify cryptographic protocols against safety properties. The approach is supported by the atp TAPS. Compared to our approach, the method does not generate counter-examples (that is, attacks) in case a protocol is found to be insecure.

With respect to security requirements, there are related approaches using goal-oriented development techniques. [12] integrates the usage of goal-trees in a UMLsec-based development process to be able to systematically consider security aspects through the different stages of the requirements engineering process. Goal-oriented techniques are used to establish threats against the security goals, which can be analyzed in threat threes that are built up systematically. Security requirements can then be analyzed using formal analysis tools to establish whether they are provided in presence of the threats identified in the vulnerability analysis. [28] introduces an approach for goal-oriented security requirements analysis which is based on the KAOS framework for requirements analysis. It allows a systematic consideration of security aspects during the requirements engineering process using a constructive approach to the modeling, specification and analysis of application-specific security requirements. Threats (called anti-goals) by attackers against the security goals can be analyzed and threat trees built systematically through anti-goal refinement. One can also use a formal epistemic specification constructs and patterns for a formal derivation and analysis process.

8. CONCLUSION AND FUTURE WORK

We presented work to support model-based security engineering using UML by providing tool-support for the analysis of UML models against security requirements. We described a UML verification framework supporting the construction of automated security analysis tools for UML diagrams which is connected to industrial CASE tools using XMI and allows convenient access to this data and to the human user. As an example, we presented an analysis plug-in that uses the automated theorem prover SETHEO to verify for example cryptographic protocols.

The method is automatic and, for practical purposes, sufficiently efficient and powerful. Our approach translates behavioral UMLsec diagrams to formulas in first-order logic with equality (more specifically, Horn formulas). The resulting formulas are rather intuitive and compact. They are then input into any automated theorem prover supporting the TPTP input notation. If the analysis reveals that there could be an attack against the protocol, the theorem prover is again called using an attack generator written in Prolog to produce an attack scenario. The protocol can then be corrected by the designer, and the process repeated. We explained our approach at the hand of a variant of the Internet protocol TLS.

The tools we presented have been used in industrial projects in Germany with companies including a major car manufacturer, bank, and telecommunications company. Experiences have been very positive. Several security design weaknesses could be demonstrated which have lead to changes in the designs of the systems that are being developed. As a representative example, we reported on our experiences with the use of our methods and tools in an application project with a major German telecommunications company with the aim to develop a biometric authentication system. Several security flaws were found and corrected.

The verification framework has been instantiated with a number of analysis plug-ins so far. It has proven to be sufficiently flexible and expressive to support plug-ins for a variety of property checks (not only in security-critical systems, but for example also for safety checks), while not being overly complicated. The connection to the CASE tool and the human user work nicely (apart from the usual XMI compatibility issues which have to be solved with suitable converters as usual). In particular, the tool-binding to the automated theorem prover presented here works fine.

Since UML is widely known, and since it allows subtle security requirements to be packaged up into pre-defined stereotypes which can be used by developers without much background in formal methods, this should contribute to the further uptake of model-based development of secure systems in practice.

The tools presented here can be downloaded from [27] as open-source. A mailinglist for their users and external contributors is available.

In future work, we aim to investigate how to combine the UMLsec approach and its tools with approaches for security requirements elicitation such as [6, 7].

Acknowledgements. Fruitful collaborations with Robert Schmidt and Thomas Kuhn and helpful discussions with Gernot Stenz and Matthias Schwan are gratefully acknowledged, as well as interesting discussions with Carlo Montangero about constructing UML security tools.

9. REFERENCES

[1] M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation*, pages 39–60. IOS Press, Amsterdam, 2000. 20th International Summer School, Marktoberdorf, Germany.

[2] L. Campbell, B. Cheng, W. McUmber, and K. Stirewalt. Automatically detecting and visualising errors in UML diagrams. *Requir. Eng.*, 7(4):264–287, 2002.

[3] E. Cohen. First-order verification of cryptographic protocols. *Journal of Computer Security*, 11(2):189–216, 2003.

[4] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, 1983.

[5] P. Fenkam, H. Gall, M. Jazayeri, and C. Krügel. DPS: An architectural style for development of secure software. In G. Davida, Y. Frankel, and O. Rees, editors, *International Conference on Infrastructure Security (InfraSec 2002)*, volume 2437 of *LNCS*, pages 180–198. Springer, 2002.

[6] P. Giorgini, F. Massacci, and J. Mylopoulos. Requirement engineering meets security: A case study on modelling secure electronic transactions by VISA and Mastercard. In I.-Y. Song, S. Liddle, T. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *LNCS*, pages 263–276. Springer, 2003.

[7] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. The effect of trust assumptions on the elaboration of security requirements. In *12th IEEE International Conference on Requirements Engineering (RE 2004)*. IEEE Computer Society, 2004.

[8] A. Hall and R. Chapman. Correctness by construction: Developing a commercial secure system. *IEEE Software*, 19(1):18–25, 2002.

[9] R. Hall. Specification modeling and validation applied to a family of network security products. In ASE01 [10], pages 71–80.

[10] IEEE Computer Society. *16th IEEE International Conference on Automated Software Engineering (ASE 2001)*, 2001.

[11] J. Jürjens. UMLsec: Extending UML for secure systems development. In J.-M. Jézéquel, H. Hußmann, and S. Cook, editors, *UML 2002 – The Unified Modeling Language*, volume 2460 of *LNCS*, pages 412–425. Springer, 2002.

[12] J. Jürjens. Using UMLsec and goal-trees for secure systems development. In G. Lamont, H. Haddad, G. Papadopoulos, and B. Panda, editors, *Proceedings of the 2002 Symposium of Applied Computing (SAC)*, pages 1026–1031. ACM Press, 2002.

[13] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.

[14] J. Jürjens, E. Fernandez, R. France, and B. Rumpe, editors. *Critical Systems Development with UML (CSDUML 2004)*, TU München Technical Report, 2004. UML 2004 satellite workshop proceedings.

[15] J. Jürjens and P. Shabalin. Automated verification of UMLsec models for security requirements. In J.-M. Jézéquel, H. Hußmann, and S. Cook, editors, *UML 2004 – The Unified Modeling Language*, volume 2460 of *LNCS*, pages 412–425. Springer, 2004.

[16] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[17] R. Kemmerer. Cybersecurity. In *25th International Conference on Software Engineering (ICSE 2003)*, pages 705–717. IEEE Computer Society, 2003.

[18] 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 1999)*, pages 81–88. IEEE Computer Society, 1999.

[19] J. Lilius and I. Porres. Formalising UML state machines for model checking. In R. France and B. Rumpe, editors, *The Unified Modeling Language (UML 1999)*, volume 1723 of *LNCS*, pages 430–445. Springer, 1999.

[20] C. Meadows. A system for the specification and analysis of key management protocols. In *IEEE Symposium on Security and Privacy*, pages 182–195, 1991.

[21] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 237–250. IEEE Computer Society, 2000.

[22] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO – The CADE-13 Systems. *Journal of Automated Reasoning (JAR)*, 18(2):237–246, 1997.

[23] Netbeans project. Open source. Available from http://mdr.netbeans.org, 2003.

[24] Object Management Group. *OMG XML Metadata Interchange (XMI) Specification*, Jan. 2002.

[25] J. Schumann. Automatic verification of cryptographic protocols with SETHEO. In W. McCune, editor, *14th International Conference on Automated Deduction (CADE-14)*, volume 1249 of *LNCS*, pages 87–100. Springer, 1997.

[26] G. Sutcliffe and C. Suttner. The TPTP problem library for automated theorem proving, 2001. Available at http://www.tptp.org.

[27] UMLsec tool, 2002-04. Open-source. Accessible at http://www.umlsec.org.

[28] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *ICSE*, pages 148–157. IEEE Computer Society, 2004.

[29] Verisoft project webpages. http://www.verisoft.de, 2003.

[30] J. Wei, S. Cheung, and X. Wang. Exploiting automatic analysis of e-commerce protocols. In *25th Annual International Computer Software and Applications Conference (COMPSAC 2001)*, pages 55–62, 2001.

[31] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNCS*, pages 314–328, 1999.

[32] A. Zisman and A. Kozlenkov. Knowledge base approach to consistency management of UML specification. In ASE01 [10], pages 359–363.