Redefining Security Engineering

Carsten Rudolph and Andreas Fuchs Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstrasse 75, 64295 Darmstadt, Germany
{carsten.rudolph|andreas.fuchs}@sit.fraunhofer.de

***Abstract*—For a long time, security was not in the focus of software engineering and system engineering processes. Only quite recently the situation has changed and security issues are now more and more integrated into concrete steps of the development process. Various approaches exist for the elicitation of security requirements, for threat modeling, for risk analysis, or for security testing. These different approaches are more-and-more adapted for practical use and become integrated parts of software development life-cycles. Nevertheless, they only support isolated steps in the process (e.g. security of code) or concentrate on particular types of requirements (e.g. for access control). The long-term goal for security engineering shall be the establishment of processes supporting all steps of the engineering process in an integrated way and to co-ordinate the contributions by different roles in this process.**

This paper identifies the different tasks of security engineering and discusses what parts of these tasks can be realised by using existing approaches. Further, three embedded scenarios are used to identify some concrete requirements for a security engineering process. This discussion shall show the scope of future research and developments in the area of security engineering and motivate inter-disciplinary approaches to establish security engineering as a research discipline.

***Index Terms*—Security engineering, security requirements, security building blocks.**

I. Introduction

In recent years, the aspect of security in the engineering process of information and communication technology systems gained increasing attention. The efforts of Microsoft's Security Development Lifecycle[1] or IBM's Secure Engineering Framework[2] are examples of industry's increased awareness. The necessity to improve development processes with respect to security at least partly results from the apparent growth in damage above the 100 million dollar mark[3] as well as from the increase in attacker investment. Targeted attack attempts can rely on funding of up to several million dollars[4]. In parallel to this development on the security side, IT systems engineering gets more challenging by increased functional complexity and critical infrastructures in economy and society fully rely on this complex information and communication technology.

Remarkably, already the sole awareness for systems' security has increased the quality of security in systems directly. Developer training, secure coding guidelines and best-practice patterns, code reviews and scanners or even code validation increase the code quality with regard to its security aspects. However, these approaches mostly target the correct implementation of a given specification and they are usually not well-structured and also do not support a more-or-less complete coverage and documentation of security-related issues. The success mainly lies in the prevention of common implementation flaws ending in software weaknesses that can be exploited by attackers in order to execute malicious code on a device.

Obviously, security would strongly benefit from a proper secure engineering process extending various existing development processes. This process would need to start with security requirements elicitation at a very early phase of the process and keep track of requirements and refine them throughout the different engineering steps. Security design decisions then also need to take into account possible attacks and threats and risks associated with them. In principal, such a process seems to be straightforward and some established techniques already implement parts of the process. SDL's STRIDE for example is one useful approach that is focused at a rather low level of abstraction and is oriented towards concrete (security) technologies. Other approaches focus on particular security issues. One example is SecureUML [13] which focuses on access control model-based development. In spite of the large body of work on IT security a generic basis for a more general approach towards secure engineering is still missing. Various formal models for security either focus on particular types of properties (e.g. information flow [8]) or on special areas (e.g. cryptographic protocols [1]). Other approaches require the construction of concrete adversary models [7].

The work that motivated the vision described in this paper concentrates on the domain of systems consisting of embedded systems. Interplay of hardware, software, communication protocols and sometimes sensors and particular physical environments. Furthermore, many embedded systems have other strong non-functional requirements. examples include real-time properties, restricted power consumption, size, cost issues. Security issues for embedded systems become more and more evident when these systems evolve into complex scenarios consisting of large numbers of heterogeneous systems in partly uncontrolled environments. One example of such a system of embedded systems is a future energy network, a so-called smart grid. In a smart grid security is very relevant for various safety requirements and at the same time privacy issues exist e.g. for data on individual energy consumption. Advanced security mechanisms, such as hardware-based security for embedded devices or sophisticated zero-knowledge protocols for privacy can provide many of the security requirements. However, these security mechanism are complex and in some

This work was supported by the EU FP7 project SecFutur

[1]http://www.microsoft.com/security/sdl/default.aspx

[2]http://www.redbooks.ibm.com/abstracts/redp4641.html

[3]compare Playstation Network losses: http://www.cbsnews.com/8301-504083_162-20065621-504083.html

[4]compare Stuxxnet: http://www.securitynewsdaily.com/stuxnet-anniversary-look-ahead-0988/

cases the average developer has difficulties to understand what security requirements these mechanisms can actually fulfil and which underlying assumptions need to be made in order to achieve secure implementations. All these different characteristics make systems with embedded devices a challenging domain for security engineering and no existing approaches provide adequate solutions for these challenges.

The following section discusses the phases of a system life-cycle and identifies possible contributions from security engineering. Then, a closer look on the actual engineering process shows related work and discusses security in the context of the different steps of functional engineering. Then, concrete examples from embedded scenarios are briefly discussed and a sketch of a suitable security engineering process is described. This process shall provide one basis for further work in the area and for discussions on open issues in the research discipline of security engineering [5].

II. Security engineering in the development life-cycle

Security engineering cannot be restricted to the task of building secure software. It needs to be properly integrated into the complete life-cycle. The necessary global view on security in the engineering process needs to consider all the different stages not only of the actual development but also on phases like deployment, usage, maintenance, updates, evolution of systems or de-commissioning.

The term *security engineering* describes various tasks that can support the different phases of engineering and life-cycle. The following paragraph provides lists some of these tasks.

In the first phase of the development process security engineering needs to support the identification of security requirements. This part can be supported by *security modeling* and can be summarized as the essential phase of *security requirements engineering*. Then, *security design decisions* need to be made. These decisions are usually based on *threat modeling* and *risk analysis*. However, recent approaches support the design process with re-usable solutions with positive expressions on security properties. Keywords for these parts of the process are *security patterns* and *security building blocks*. More advanced engineering processes can provide *re-usable solutions*. At several phases of the development process it is useful to support security through *security testing* if possible supported by *model-based testing*. During *update processes* or *maintenance* security engineering can provide support in order to maintain the security properties and security levels of a system. Proper security engineering shall also support the process of *security certification* and *compliance*. Finally, various non-technical tasks shall also be considered by a proper security engineering process. Among others, there is *communication* between developers that should be supported by providing clear semantics for terms describing security properties, and also *education* and *best practices*.

This list of tasks is not complete, but it shows the large scope of the topic of security engineering. clearly, there will be no single approach to support all these different areas. Nevertheless, the research discipline of security engineering shall aim at supporting all relevant phases of the security life-cycle.

[5]One possible platform for co-ordinating further work and discussing security engineering is the Security Engineering Forum (http://www.securityengineeringforum.eu/)

One essential element of security engineering is the phase that deals with security requirements. This phase provides the root of all other parts of the security engineering process. The following list of 10 goals of security requirements engineering highlights the importance of this process.

1) One main goal is to provide exact specifications of security requirements.
2) Communication on security issues requires exact expressions for security requirements.
3) Security design decisions need to be based on clear requirements.
4) Advanced security mechanisms can support complex requirements that can be difficult to be properly understood and integrated.
5) Security mechanisms and underlying assumptions should be related to security requirements and these relations need to be traced through different levels of design decisions.
6) Threat-based approaches and risk analysis can be relocated to more concrete design levels, where better risk estimations can be made.
7) Clear guidance, best-practices and policies for security in the development process can be based on a classification of security requirements. Such a classification can also be relative to particular application domains.
8) Systematic treatment of security requirements or formal refinement with verification can increase assurance.
9) Run-time security processes (monitoring, security information and event management SIEM) can be linked to actual security requirements.
10) Research in security requirements engineering should advance security engineering processes and security certification.

III. Related Work on Security and Revision of relevant Steps in Functional Engineering

In order to illustrate the relationship between functional engineering and security and demonstrate the differentiation between specification and implementation related security issues we utilize the categorization depicted in Figure 1.

From a functional engineering perspective the functional goals of a system are refined during the design phase to a functional specification. The implementation of such a specification aims at providing a system or software that meets this specification in the sense that all specified functionalities are correctly implemented. From a functional perspective, additional behavior is introduced during implementation is not necessarily harmful if additional behavior. For example, for velocity measurements and temperature sensors within a vehicle it might be specified that the velocities of -100 to +300 km/h and temperatures between -100 and +100 $^\circ$C should be reported at the display. If an actual implementation represents these values with a signed 16bit variable, allowing for the

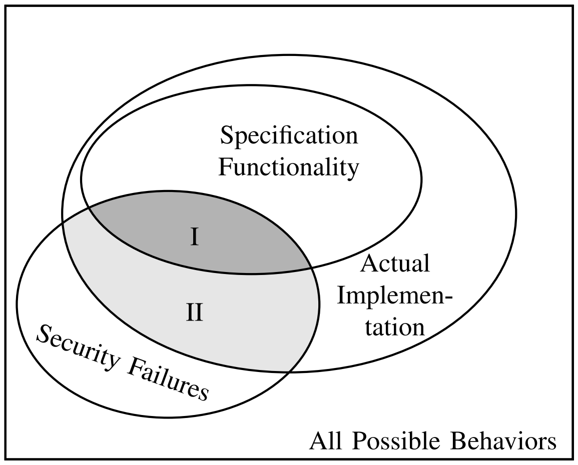

Fig. 1. Relation of Functionality and Security

reporting of -32768 to +32767, the functional goals have been met. Those are the types of testing and quality assurances that are usually met by quality assurance and unit and integration testing in the functional domain.

Adding security creates a more complicated picture. In order to secure these measurements, the functional specification might include digital signatures to be used for the transfer between (trustworthy) sensors and display. Security issues of Type I may now arise if the functional specification is designed in such a way, that security goals are not met. For the example of a vehicle's display of sensor data, this may happen if the security protocols were designed poorly, such that an attacker may manipulate temperature and velocity values.

Security issues of Type II arise if behavior is implemented (possibly unwillingly), that exceed the functional specification and introduces security issues. For the example of a vehicle's display, this might be the possibility to store additional verification keys, such that an attacker may send arbitrary information that the display believes to be authentic from the sensor.

We will now use the categorization of security failures to Type I and Type II in order to discuss possible contributions by existing work.

A. Related Work

The most widely known approach to secure software engineering is Microsoft's Secure Development Lifecycle (SDL) [11] with its DataFlow Diagrams and STRIDE analysis. SDL uses structural models and is focused on security threats. Also the requirements are identified on a low and technical level and their mediation uses a simple mapping to functionalities. Additional requirements resulting from security functionalities are not considered. Also there is no notions of trust relations among entities and according security implications. Finally, security functionalities already included into the (functional) specification are not explicitly considered. Thus, errors in this part of the design are not covered (compare to Type I security failures in Figure 1). A similar role as the STRIDE phase of SDL is provided by the requirements elicitation based on the security modeling framework SeMF [10]. In contrast to STRIDE it is based on behavioral models and focuses on security guarantees rather than on threats is

The Rodin approach [2] is a formal methods based engineering method using the Event-B language. It provides a refinement-driven engineering approach that ends in code generation, where the requirements (in terms of theorems) can be checked through theorem proving on any level of abstraction. The focus of Rodin is therefore to reduce the delta between functional specification and actual system functionality, compare type II in Figure 1, as well as some reduction of Type I through theorem proving. However, handling of the monolithic models of Rodin and Event-B become infeasible for large, complex and interacting systems. Also the approach dictates that engineering is performed in a top-down manner and in the syntax and semantics of Rodin (such as UML-B), which limits its applicability and provides a big overhead.

Other approaches, such as [18] and the project CASENET [3] are focused on the engineering of cryptographic protocols and lack the ability to design large and complex systems or incorporate the behavioral properties of entities in the system and their trust relations.

The approaches of UMLsec [12] and SecureUML [13] focus on the visualization of security properties in UML and also the model-based generation of implementation. SecureUML targets the engineering of access control policies through model refinement. Therefore, the applicability of this rather mature approach is restricted to role-based access control. UMLsec has a similar focus as SDL's STRIDE analysis with a concentration on protecting communication links on more complex scenarios.

Many approaches exist for the formal analysis of systems and protocols, ranging from logics such as BAN [5] or Temporal Logics [16] to formal languages and automaton analysis, such as SHVT [15], AVISPA [4] and AVANTSSAR[6]. This latter category typically uses the properties of Safety/Lifeness and Hypersafety/Hyperlifeness [6] to describe properties within a dynamic system. These properties however have a behavioral background rather than a security related motivation. The bases for SeMFgineering, namely the Security Modeling Framework SeMF, relies on formal language theory as well, but with an additional set of concepts for local view and initial knowledge to support the expression of security properties. Also, many of these approaches require for and explicit attacker model and security properties is defined as "anti-patterns". In the security modeling framework SeMF [9], security properties can be defined as "patterns" fulfilled by the system without the modeling of attacker capabilities, but rather through security guarantees.

B. Analysis of Engineering Concepts

Security engineering processes need to extend steps taken in the design phases of existing functional engineering processes. The following paragraphs briefly introduce the characteristics of different functional engineering concepts and identify the common steps taken in each of them.

[6]http://www.avantssar.eu

1) Engineering Processes Models: The functional engineering concepts of **Waterfall** [17] and **V-Model**are well-known industrial engineering processes. In their (theoretical) application of strict top-down engineering, they use the concept of *refinement* to step from high level requirements to implementation details. Throughout these refinement steps, the correctness of requirements and properties of the system must however be ensured property preserving homomorphisms. Also many *design decisions* have to be made, as to which solution path to follow.

More advanced interpretations of Waterfall and V-Model include the notion of feedback loops towards previous design phases. Within these feedbacks and for documentation purposes ("design rational") it is necessary to support the *backtracing* of component functionalities and properties through the design decisions made up to the requirements that motivated them. This is even more important during deployment of a system, if some requirements targeted by a given software are not important for a given scenario.

The concept of **model-driven engineering** and the OASIS' "Model Driven Architecture" [14] are refinement driven approaches. Starting from a Computational Independent Model (CIM) – often referred to as domain model – a transformation to a Platform Independent Model (PIM) is made and finally a transformation to a Platform Specific Model (PSM) using a Platform Model's mapping rules. Within each of these steps, a (more or less automatic) *refinement* of the previous model is performed using templates and patterns.

During the refinement from CIM to PIM many design choices have to be made. This is the place for security engineering to provide guidance on design choices and security trade offs. Further, the refinement from CIM to PIM can include the generation of several components or modules within the PIM that collaborate in order to fulfill a challenge of the CIM. These *decompositions* introduce a complexity into the system that needs to be mitigated.

During the mapping from PIM to PSM, the PIM's requirements against, for example, service qualities of a given platform, need to be states. Here, a vocabulary is necessary on which the compatibility of a platform to a given PIM can be derived – or hints on how to refine the PIM further before mapping. These can be expressed as so-called *security contracts*. Examples of security contracts of a platform might be communication-related, such as the confidentiality of data within a certain channel (such as WebService-Security) or device related, such as the confidentiality of cryptographic key material within a given device (such as an embedded smart card or Trusted Platform Module). Theoretically, automated model transformations should take care of all design decisions. However, for the definition of transformation rules and with semi-automated transformations, manual *design decisions* have to be performed.

Within the family of **Incremental**, **Iterative** and **Agile** engineering concepts [19], many of the same steps are implemented but within a more dynamic fashion. The most important and common goal of all these approaches is not to build a "Big Design Up Front". Within Scrum Sprints for example, only a certain set of the overall requirements are targeted within every 2-4 weeks period and a working version of the software is expected at each sprint's end. Within each of these iterations, *refinement* and refactoring play a major role. Refactoring however can again be broken down into the more basic tasks of *generalization* directly followed by a *specialization* in order to e.g. reuse code in several methods, *decomposition* directly followed by *composition* in order to e.g. reuse classes and components in several locations.

Due to this dynamic nature the definition of functionalities of components must be stated even more precisely.Also even though more responsibility is given to the programmers (esp. in Extreme Programming), *design decisions* still have to be made and the possibility to *backtrace* the decision made during a previous sprint is highly important.

2) Common Steps in Engineering Processes: From these analysis of engineering concepts we can now derive the steps and tasks that need to be supported by security engineering.

- Refinement: The derivation of a system representation from another system representation that is more concrete than this previous one.
  $\rightarrow$ The consistency of security requirements and assumptions need to be preserved during refinement.
- Design decisions: The decision to solve a given requirement in a certain way.
  $\rightarrow$ The security wise consequences need to be clear and can guide the decision making.
- Abstraction: The representation of a system with less detail e.g. for documentation or interface description.
  $\rightarrow$ The consistency of security requirements and assumptions needs to be preserved.
- Composition: The utilization of two or more (previously given) components in order to fulfill a given functionality.
  $\rightarrow$ The description of components' provided and required security properties through e.g. contracts.
- Decomposition: The separation of a single component into two or more components that perform the same functionality, for the purpose of reuse and/or complexity reduction.
  $\rightarrow$ The description of components' provided and required security properties through e.g. contracts.
- Generalization: The extension of functionality of a given component in order to be applicable to a bigger set of problems.
  $\rightarrow$ The consistency of security requirements and assumptions needs to be preserved.
- Specialization: The reduction of functionality of a given component for a given problem.
  $\rightarrow$ The consistency of security requirements and assumptions needs to be preserved.
- Backtracing: The search for the requirement and the design decisions that led to a given functionality. $\rightarrow$ Supporting the traceability of security aspects of a given design.

C. Three use-cases with embedded components

This section discusses three different embedded scenarios and their particular requirements on security engineering. For each scenario a brief description is given, characteristics shown and demands on security engineering processes are identified.

1) Secure smart grid gateway:

a) Description of the scenario: A so-called *smart grid gateway* is part of future energy distribution networks. It provides the technical interface between the information technology and communication part of the energy network and the end-user perimeter. A smart grid gateway provides interfaces to smart meter, for communication to energy network operator, energy provider and end-user. In addition, provider of additional value-added services can also become relevant. Energy metering information is processed and distributed via the smart grid gateway. Thus, main security requirements are motivated by privacy requirements. Smart gateways can also provide the interface or main controller of a *smart home* where many functionalities can be automated and remote-controlled. Such a smart home obviously imposes additional security requirements.

b) Characteristics: An integrated smart grid gateway can be one single device with clearly defined interfaces. Other implementation also consider several gateways for communication with the different stake-holders. In all cases, the devices will have a restricted functionality and not many dynamic changes. Smart grid gateways need to support the different processes in a smart grid. Nevertheless, these processes are mostly not directly controlled by the smart grid gateway, but by other components of the system.

c) Demands on security engineering: A smart grid gateway is a completely new device for a new and advanced infrastructure. Thus, threats and risks need to be identified and analysed in the complete view of the infrastructure and cannot be restricted to the device itself. Consequently, security requirements engineering needs to support the identification of requirements on processes in the smart grid infrastructure. Nevertheless, for the actual development, security requirements need then to be refined in order to identified them for clearly defined interfaces. Furthermore, various stake-holders need to be considered and standardization also for security protocols is necessary.

2) Multi-service home-gateway:

a) Description of the scenario: The device is in principle similar to an IP-based set-top box for TV. However, instead of just providing one type of service it can support multiple services by several different provides. The device is typically owned by a telecommunication company and other service providers can rent so-called *slices* on the box. These slices can provide space for virtual machines with software provided by the service providers. Thus, each service provider can install software and provide different types of services using the interfaces of the device. Typical services can be multimedia distribution or streaming, but also peer-to-peer overlay networks for various applications can be imagined.

b) Characteristics: The multi-service home gateway is a single device like the smart grid gateway, but with different characteristics. there is one single owner which is not the end user, but different stake-holders (end-user, owner of device, service providers) have access. Thus, requirements of different stake-holders need to be considered. Security requirements of the different stake-holders might be conflicting. Furthermore, different levels need to be considered. First the gateway and its connection to the end-user network and to the network provider is relevant and second, also the security requirements for slices on the device and for services installed by clients need to be considered.

c) Demands on security engineering: It must be possible to consider the different life-cycles of gateway itself and of slices on the gateway. Furthermore, security perimeters are not clearly defined. Although there are clear technical interfaces between the platform, the virtual machines in the slices and the networks connected to the box, the logical boundaries are not that clear. Privacy issues can further increase the complexity of the security requirements. Thus, security requirements need probably to be defined for processes on the gateway.

In order to provide a sufficient level of security it might be necessary to consider advanced security mechanisms in the development process. Privacy could be provided by advanced cryptographic protocols while security of the gateway itself might need to be based on hardware security mechanisms. Although the question of identities is complex. Thus, security engineering need to provide expressiveness for complex security requirements and these security requirements need to be mapped to suitable security building blocks.

3) Secure mobile ad-hoc networks (MANET):

a) Description of the scenario: Mobile ad-hoc networks consist of devices that construct a network through direct communication between each device. Thus, all devices become network nodes. MANETs are highly dynamic and routing information needs to be frequently updated. Nodes can move and location information can be relevant. the situation in MANETs is very different from mobile networks (e.g. for mobile phones), where devices move, but network nodes remain mainly static.

b) Characteristics: Security in MANETs is different to other static networks with hierarchical structure or centralized control. Each device is responsible for central network functionalities. Therefore, security requirements can be identified already for low technical levels. One example is routing in the network which can be easily attacked in an insecure MANET.

c) Demands on security engineering: Security requirements need to be defined for dynamic scenarios with various components. Technical solutions cannot rely on trusted entities or central control components. Lifecycle processes are relevant and differ for MANET scenarios.

IV. Conclusions

The large variety of tasks that can be summarized in the process of security engineering shows that there will be no single approach to cover all different aspects. Furthermore, the concrete requirements on security engineering processes identified for the three examples from different embedded scenarios motivate domain-specific instantiations of security engineering processes. Application domains will have their particular sets of requirements, ontologies and meta-models for security, sets of suitable security building blocks and also their own best practice solutions. A security engineering process shall support all these variations.

The observations described in this paper can provide some guidelines for future research topics in the area of security engineering. It is immediately clear that interdisciplinary co-operation is necessary. Mainly three disciplines need to co-operate, namely IT security experts, software and systems engineering experts and for each application domain there need to be contributions by the particular domain experts. The establishment of proper security engineering needs co-ordinated activities from all these disciplines.

Painting the picture of security engineering also provides a good view of the open issues and research tasks to be done. The scope of the research issues motivates a development to consider security engineering as its own research discipline with strong relations to engineering and IT security. It needs to make IT security expert knowledge available to engineers and at the same time provide better ways to bring advanced security solutions into practical applications.

References

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 62–76, 2005.

[2] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.

[3] I. Agudo, S. Gürgens, and J. Lopez. CASENET One Year Later. In *18th IFIP International Information Security Conference 2003*, Athens, Greece, May 2003.

[4] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P. Héam, O. Kouchnarenko, J. Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285. Springer, 2005.

[5] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8, 1990.

[6] M. Clarkson and F. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[7] A. Datta, J. Franklin, D. Garg, L. Jia, and D. Kaynar. On adversary models and compositional security. *Security Privacy, IEEE*, 9(3):26 – 32, 2011.

[8] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), 1976.

[9] A. Fuchs, S. Gürgens, and C. Rudolph. Formal Notions of Trust and Confidentiality - Enabling Reasoning about System Security. *Journal of Information Processing*, 19:274–291, 2011.

[10] A. Fuchs and R. Rieke. Identification of authenticity requirements in systems of systems by functional security analysis. In *Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplementary Volume*, 2009.

[11] M. Howard and S. Lipner. *The security development lifecycle*, volume 8. Microsoft Press, 2006.

[12] J. Jürjens. Umlsec: Extending uml for secure systems development. *UML 2002 The Unified Modeling Language*, pages 1–9, 2002.

[13] T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. *UML 2002 The Unified Modeling Language*, pages 426–441, 2002.

[14] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group, 2003.

[15] P. Ochsenschläger, J. Repp, and R. Rieke. *Simple Homomorphism Verification Tool – Tutorial*. Fraunhofer Institute for Secure Telecooperation SIT, Darmstadt, 2002.

[16] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[17] W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON*. Los Angeles, 1970.

[18] C. Rudolph. *A Model for Secure Protocols and its Application to Systematic Design of Cryptographic Protocols*. PhD thesis, Queensland University of Technology, 2001.

[19] K. Schwaber and M. Beedle. *Agile software development with Scrum*, volume 18. Prentice Hall PTR Upper Saddle Riverˆ eNJ NJ, 2002.