Bundesamt für Sicherheit in der Informationstechnik

Guideline for the Development and Evaluation of formal security policy models in the scope of ITSEC and Common Criteria

Version 1.1

17th December 2004

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Authors:
Heiko Mantel
Werner Stephan
Markus Ullmann
Roland Vogt

Prepared by

Deutsches Forschungzentrum für Künstliche Intelligenz (DFKI GmbH)

Stuhlsatzenhausweg 3
66123 Saarbrücken

GERMANY

Contact:
Roland Vogt
phone: +49-681-302-4131
email: roland.vogt@dfki.de

By order of

Bundesamt für Sicherheit in der Informationstechnik (BSI)

Postfach 20 03 63
53133 Bonn

GERMANY

Contact:
Markus Ullmann
Ref. VI 4
phone: +49-228-9582-268
email: ullmann@bsi.bund.de

Contents

Chapter 1

Summary

As to the preparation and verification of formal specifications according to valid criteria (ITSEC, CC), many questions remain open after having read the relevant passages. These guidelines are supposed to help evaluators and developers to reach a deeper understanding on this topic in the context of these criteria, and that in particular for the preparation and verification of so-called *Formal Models of Security Policies* (FMSP). Moreover, it is supposed to help discovering the value of formal security models added to the mere conformance with the criteria requirements.

To this end, first the various requirements that are defined on different parts of the criteria – be it ITSEC or CC – are quoted. These then are related to the characteristics of formal methods, known and published formal security models and the experiences that have been made with the development and the verification of formal models. In particular, this will show that the models by Bell/La Padula and Biba that are mentioned in the criteria merely define a generic framework with which various security policies for access control can be defined.

Special emphasis will be placed on the fact that formal security models are *not* isolated specifications but are closely and directly related to other evaluation documents as, for example, functional specifications (c. f. CC, family ADV_FSP).

In as much as it serves comprehensibility, the requirements for the components of formal safety models are not only made more precise but are also illustrated with the help of small excerpts taken from an existing formal model of Integrated Circuit Cards with signature functionality according to DIN V 66291-1.

Past experiences show that a formal modelling of the security policies – given as a formal security model – may lead to an increase of confidence in the security of the product that obeys these security policies. This is so because it always happened that vulnarabilities and errors were uncovered already on the basis of informal security policies.

Up to now, these results were dearly purchased simply because the preparation

of formal security models is still understood as an isolated evaluation task that is to be carried out towards the end of the product development. However, if it were considered as a part of the system development itself, many errors and security vulnarabilities within products could be omitted at an early stage and would not have to be eliminated ex post with that much of an effort.

Chapter 2

Introduction

The *Information Technology Security Evaluation Criteria (ITSEC)* [4] define criteria for the evaluation of IT-systems and products according to the evaluation levels E1 to E6. The higher the evaluation level the more demanding are the security requirements that have to be fulfilled. In a similar fashion, the *Common Criteria for Information Technology Security Evaluation (CC)* [1] define the evaluation levels EAL1 to EAL7 (with EAL7 being the highest security level). IT security comprises confidentiality, integrity, and availability. For the evaluation levels E4 to E6 (ITSEC), respectively EAL5 to EAL7 (CC), it is required that the development of the target of evaluation (TOE) is supported by a formal security model. A formal security model is a model of the aspired TOE security policy that is prepared on a formal notational basis. Such a notation has to rely on mathematical concepts with which syntax and semantics of the language but also rules of inference are defined.

In the course of an evaluation already published, formal security models may be used as a whole or in parts provided they are appropriate for the TOE. If none of the common security models is appropriate, such a security model has to be newly established. As examples of published security models, the ITSEC mentions the Bell/La Padula model, the model of Clark and Wilson, the Brewer-Nash model, the Eizenberg model, and the Landwehr model (c. f. [4, Par. 2.83]). In the CC, the Bell/La Padula model, the Biba model and models for non-interference by Goguen and Meseguer are mentioned (c. f. [1, Part 2, Par. 786]).

For the TOE and the chosen security model it has to be shown that the TOE's security functions are conformant with both the modelled security policy and with the TOE's functional requirements. A formal security model therefore constitutes a link between the requirements specification and the design specification. It is thus of vital importance for the quality of the TOE's security guarantees.

2.1 Goal

This document is supposed to serve as a guideline for the preparation and the evaluation of security models according to ITSEC and CC, respectively. This guideline supports both the sponsor of an evaluation and the evaluator. The former to fulfil the criteria that are required for a formal security model and the latter for the evaluation of the TOE according to these criteria.

The guideline includes a survey of established security models that proved to be of value or seem appropriate for an evaluation according to ITSEC and CC, respectively. For each of these models it is described which security policy lies underneath, how the security policy can be modelled formally, and under which assumptions this security model can be applied.

Whether or not an existing formal security model can be accessed, is a matter of the security objectives that have to be achieved as well as of the security policy of the product that is to be evaluated. The guideline is supposed to support the decision whether a known model can be applied – be it directly or slightly modified – or whether a new one has to be established.

2.2 Structure of the Document

This guideline is structured along the following five questions that arise during the development and the evaluation of formal security models in practice.

- How are we to interpret the requirements of the criteria with respect to formal security models in the context of a development process?
- Which concepts, techniques, and tools for formal methods are needed for the preparation and the evaluation of formal security models?
- Which security principles and characteristics matter and how can they be modelled formally?
- Are there any general adaptable formal security models that can be accessed?
- How does an evaluation according to ITSEC and CC resp., take course?

In the beginning of chapter 3, all requirements on security models from ITSEC and CC are collected. After that, a requirement catalogue is determined that measures up to both the criteria as well as the state of the art in the area of formal methods.

The base concepts of formal methods are summarized in chapter 4. This is supposed to give some insight into the foundations and the technical possibilities when utilizing formal methods in the area of IT security. Evidently, it is not meant – neither in breadth nor in depth – as a overall comprehensive introduction into formal methods.

In chapter 5 the abstractly outlined notions and requirements are illustrated by a concrete example, namely a smart-card application that generates digital signatures. Starting from an informal security policy assistance is given in various respects. For instance how to determine and to formalize security properties and characteristics. But also how to choose an adequate abstraction level and how to make assumptions about the application environment as precise as possible. These remarks will be supplemented with excerpts taken from the formal security model mentioned above that has been specified and verified with the help of the *Verification Support Environment (VSE)*. VSE is a tool to prepare formal security models. It is authorized by the German Information Security Agency (BSI).

Some generally applicable security models taken from the class of models based on non-interference are described in 6. In detail non-interference (section 6.1), the model of Bell/La Padula (section 6.2.1) and the model of Biba (section 6.2.2) are addressed. This class of security models can be applied to the modelling of both confidentiality and integrity. In sections 6.4 and 6.5, the usability of these models in a concrete application is addressed. Moreover, it is explained what it means for a system to instantiate the models.

In chapter 7, hints and methods for the evaluation of formal security models according to the ITSEC and CC requirements are given. With this, developers as well as evaluators should get an orientation which criteria are crucial for the quality of the formal security models and with which methods potential deficiencies can be uncovered.

In chapter 8, it is summarized why the application of formal methods serves the following two purposes: A vital enhancement of the development process and a noticeable increase of quality. The evaluation afterwards (merely) serves the purpose to check whether these techniques have been applied appropriately. In addition, the issue of differentiating between formal and semi-formal methods is addressed.

Chapter 3

Requirements for Formal Models of Security Policy

In the context of formal models of security policy the Common Criteria [1, 2] and the ITSEC [4, 5] formulate concrete requirements. These are complemented by the Joint Interpretation Library [8] in the case of the ITSEC.

For the purpose of a comparative presentation the relevant sections of the criteria are cited in this chapter first. Based on this comparison appropriate requirements are determined that permit an effective use of formal models of security policy for evaluations according to ITSEC as well as Common Criteria.

3.1 Requirements of ITSEC

In the following sections the requirements of ITSEC on formal models of security policy are reproduced. The paragraph numbers are indicated in the page margins. Any requirements are cited in english as well as german language. In cases of doubt the english version is valid[1].

From [4, 5, Chapter 2] the general explanations on formal specifications and formal models of security policy are cited first. The concrete requirements on formal models of security policy resp. on the related formal specifications of the security enforcing functions and the architectural design are taken from the text of the evaluation level E6 (cf. [4, 5, Level E6, Phase 1 and Phase 2]). Differences w. r.t. the levels E4 and E5 are identified through underlining and **bold face** printing.

[1] Normative application note cited from [9, AIS 2, Version 5 dated 21.08.1998]: „Die ITSEC entstanden im Verlauf eines Harmonisierungsprozesses auf europäischer Ebene. Die abgestimmte und [...] gültige Ausgabe ist die englischsprachige Ausgabe der ITSEC."

System-Sicherheitspolitik

2.13 Die IT-Sicherheitsmaßnahmen einer System-Sicherheitspolitik können vom Rest der System-Sicherheitspolitik getrennt werden und in einem besonderen Dokument festgelegt werden: der sogenannten **„Technischen Sicherheitspolitik."** Sie ist die Menge der Gesetze, Regeln und Praktiken, die die Verarbeitung von sensitiven Informationen und die Nutzung der Betriebsmittel durch die Hard- und Software eines IT-Systems regelt.

System Security Policy

The IT security measures of a System Security Policy may be separated from the remainder of the System Security Policy, and defined in a separate document: a **Technical Security Policy**. This is the set of laws, rules and practices regulating the processing of sensitive information and the use of resources by the hardware and software of an IT system.

Formale Spezifikation

2.76 Eine formale Darstellungsform einer Spezifikation ist in einer formalen Notation geschrieben, die auf wohl begründeten mathematischen Konzepten aufbaut. Diese Konzepte werden dazu benutzt, um die Syntax und die Semantik der Notation und die Prüfregeln zu definieren, die das logische Schließen unterstützen. Formale Spezifikationen müssen aus einer Menge von Axiomen abgeleitet werden können. Die Gültigkeit von Haupteigenschaften, wie z. B. die Erzeugung einer gültigen Ausgabe für alle Eingaben, muß gezeigt werden können. Wenn Spezifikationen hierarchisch aufgebaut sind, muß gezeigt werden können, daß auf jeder Stufe die Eigenschaften der vorhergehenden Stufe erhalten bleiben.

2.77 Die syntaktischen und semantischen Regeln einer formalen Notation, die in Sicherheitsvorgaben verwendet werden, müssen festlegen, wie Konstrukte eindeutig zu erkennen sind und ihre Bedeutung bestimmt werden kann. Wenn Beweisregeln logische Schlüsse unterstützen, muß offensichtlich sein, daß es nicht möglich ist, Widersprüche abzuleiten. Alle Regeln der Notation müssen definiert werden oder es muß darauf hingewiesen werden, wo sie beschrieben sind. Alle Konstrukte, die in einer formalen Spezifikation benutzt werden, müssen vollständig durch die Regeln beschrieben sein. Die formale Notation muß sowohl die Beschreibung der Wirkung einer Funktion als auch aller damit zusammenhängenden Ausnahmen und Fehler erlauben.

Formal Specification

A formal style of specification is written in a formal notation based upon well-established mathematical concepts. The concepts are used to define the syntax and semantics of the notation, and the proof rules supporting logical reasoning. Formal specifications must be capable of being shown to be derivable from a set of stated axioms, and must be capable of showing the validity of key properties such as the delivery of a valid output for all possible inputs. Where hierarchical levels of specification exist, it must be possible to demonstrate that each level maintains the properties established for the previous level.

The syntactic and semantic rules supporting a formal notation used in a security target shall define how to recognise constructs unambiguously and determine their meaning. Where proof rules are used to support logical reasoning, there shall be evidence that it is impossible to derive contradictions. All rules supporting the notation shall be defined or referenced. All constructs used in a formal specification shall be completely described by the supporting rules. The formal notation shall allow the specification of both the effect of a function and all exceptional or error conditions associated with that function.

Beispiele für formale Schreibweisen sind VDM, beschrieben in [SSVDM], Z, beschrieben in [ZRM], die Spezifikationssprache RAISE, beschrieben in [RSL], Ina Jo, beschrieben in [IJRM], die Spezifikationssprache Gipsy, beschrieben in [GIPSY] und die OSI-Sprache zur Spezifikation von Protokollen [LOTOS]. Die Nutzung von Konstrukten der Prädikaten- (oder anderer) Logik und der Mengenlehre als formale Schreibweise ist erlaubt, wenn die Konventionen (Regeln) dokumentiert sind oder ein Verweis auf die Beschreibung (wie bereits oben erwähnt) angegeben ist.

Example formal notations are VDM, described in [SSVDM], Z, described in [ZRM], the RAISE Specification Language, described in [RSL], Ina Jo, described in [IJRM], the Gypsy Specification Language, described in [GYPSY], and the ISO protocol specification language [LOTOS]. The use of constructs from predicate (or other) logic and set theory as a formal notation is acceptable, provided that the conventions (supporting rules) are documented or referenced (as set out above). 2.78

Formale Sicherheitsmodelle

Formal Models of Security Policy

Bei den Evaluationsstufen ab E4 muß dem EVG ein Modell einer Sicherheitspolitik (Sicherheitsmodell) zugrunde liegen, d. h. es muß eine abstrakte Beschreibung der wichtigen Sicherheitsprinzipien vorhanden sein, denen der EVG genügt. Es muß in einer formalen Darstellungsform vorliegen, als ein **formales Sicherheitsmodell**. Auf ein geeignetes veröffentlichtes Modell kann ganz oder teilweise Bezug genommen werden oder es muß ein Modell als Teil der Sicherheitsvorgaben vorhanden sein. Jede der oben beschriebenen Darstellungsformen einer formalen Spezifikation kann benutzt werden, um solch ein Modell zu definieren.

At Evaluation levels E4 and above, a TOE must implement an underlying model of security policy, i. e. there must be an abstract statement of the important principles of Security that the TOE will enforce. This shall be expressed in a formal style, as a **formal model of security policy**. All or part of a suitable published model can be referenced, otherwise a model shall be provided as part of the security target. Any of the formal specification styles identified above may be used to define such a model. 2.81

Das formale Modell muß nicht alle sicherheitsspezifischen Funktionen enthalten, die in den Sicherheitsvorgaben angegeben sind. Jedoch muß eine informelle Interpretation des Modells mit Bezug auf die Aussagen in den Sicherheitsvorgaben vorhanden sein. Es muß gezeigt werden, daß die Sicherheitsvorgaben die zugrunde liegende Sicherheitspolitik umsetzen und keine Funktionen enthalten, die mit der zugrunde liegenden Politik im Widerspruch stehen.

The formal model need not cover all the security enforcing functions specified within the security target. However, an informal interpretation of the model in terms of the Security target shall be provided, and shall show that the security target implements the underlying security policy and contains no functions that conflict with that underlying policy. 2.82

2.83 Beispiele für veröffentlichte formale Sicherheitsmodelle sind:

a) Das Bell-La-Padula-Modell [BLP] — ein Modell für Anforderungen an die Zugriffskontrolle, die für eine nationale Sicherheitspolitik zur Vertraulichkeit von Daten typisch sind.

b) Das Clark und Wilson Modell [CWM] — ein Modell für Integritätsanforderungen an kommerzielle Transaktionssysteme.

c) Das Brewer-Nash-Modell [BNM] — ein Modell für Anforderungen an die Zugriffskontrolle im Hinblick auf Kundenvertraulichkeit; typisch für Finanzinstitutionen.

d) Das Eizenberg-Modell [EZBM] — ein Modell für Zugriffsrechte, die sich mit der Zeit ändern.

e) Das Landwehr-Modell [LWM] — ein Modell für Anforderungen an die Datenübertragung eines Nachrichtenverarbeitungsnetzes.

Examples of published formal models of security policy are:

a) The Bell-La Padula model [BLP] — modelling access control requirements typical of a national security policy for confidentiality.

b) The Clark and Wilson model [CWM] — modelling the integrity requirements of commercial transaction processing systems.

c) The Brewer-Nash model [BNM] — modelling access control requirements for client confidentiality, typical of a financial services institution.

d) The Eizenberg model [EZBM] — modelling access control rights that vary with time.

e) The Landwehr model [LWM] — modelling the data exchange requirements of a message processing network.

Konstruktion — Der Entwicklungsprozeß

Phase 1 — Anforderungen

Anforderungen an Inhalt und Form

E[45**6**].2 [. . .] Ein formales Sicherheitsmodell oder ein Verweis auf ein solches muß zur Verfügung gestellt werden. Darin ist die zugrundeliegende Sicherheitspolitik zu definieren, die vom EVG durchgesetzt werden muß. Eine informelle Interpretation dieses Modells in Bezug zu den Sicherheitsvorgaben muß zur Verfügung gestellt werden. Die in den Sicherheitsvorgaben aufgeführten sicherheitsspezifischen Funktionen müssen sowohl in informeller als auch in semiformaler / **formaler** Notation (siehe [5, Kapitel 2]) spezifiziert werden.

Construction — The Development Process

Phase 1 — Requirements

Requirements for content and presentation

[. . .] A formal model of security policy shall be provided or referenced to define the underlying security policy to be enforced by the TOE. An informal interpretation of this model in terms of the security target shall be provided. The security enforcing functions within the security target shall be specified using both an informal and semiformal / **formal** style as categorised in [4, Chapter 2].

Anforderungen an Nachweise

[...] Die informelle Interpretation des formalen Sicherheitsmodells muß beschreiben / **erklären**, auf welche Weise seine zugrundeliegende Sicherheitspolitik durch die Sicherheitsvorgaben erfüllt wird.

Aufgaben des Evaluators

[...] Es ist zu überprüfen, ob es Sicherheitsmaßnahmen in den Sicherheitsvorgaben gibt, die zu Konflikten mit der dem Sicherheitsmodell zugrundeliegenden Sicherheitspolitik führen.

Requirements for evidence

[...] The informal interpretation of the formal security policy model shall describe / **explain** how the security target satisfies the underlying security policy. E[4**56**].3

Evaluator Actions

[...] Check that there are no security features in the security target that conflict with the underlying security policy. E[4**56**].4

Phase 2 — Architekturentwurf

Anforderungen an Inhalt und Form

Eine semiformale / **formale** Notation muß verwendet werden, um einen semiformalen / **formalen** Architekturentwurf zu erstellen. [...]

Anforderungen an Nachweise

[...] **Sie [Die Beschreibung der Architektur] muß durch Anwendung einer Mischung von formaler und informeller Technik erklären, wie sie mit dem formalen Sicherheitsmodell übereinstimmt.**

Aufgaben des Evaluators

[...] **Es ist zu überprüfen, ob die formalen Argumente gültig sind.**

Phase 2 — Architectural Design

Requirements for content and presentation

A semiformal / **formal** notation shall be used in the architectural design to produce a semiformal / **formal** description. [...] E[45**6**].5

Requirements for evidence

[...] **It [The description of the architecture] shall explain, using a combination of formal and informal techniques, how it is consistent with the formal security policy model of the underlying security policy.** E[45**6**].6

Evaluator Actions

[...] **Check that formal arguments are valid.** E[45**6**].7

3.2 Interpretation of ITSEC JIL

The ITSEC Joint Interpretation Library (JIL) contains internationally agreed normative interpretations w. r. t. the requirements of ITSEC pertaining to formal methods (cf. [8, Chapter 19]). The extractions of this chapter relevant to formal models of secrity policy are reproduced in the following sections. The paragraph numbers are indicated in the page margins. The citations are given in english language only, since an official german translation does not exist.

Background

273 [4, 2.81–2.83] provides explanations about formal model of security policy and [4, 2.76–2.78] about formal specification.

274 [4, 2.78] provides examples of formal notations. Additional notations are CSP, VSE, B. [. . .]

275 The use of formality as applied to the ITSEC deliverables is described as follows:

- At E4 and above, a formal model of security policy is required with an informal interpretation of this model in terms of the security target [4, En.1, En.2 $n > 3$]; referred to within this topic as FMSP and its informal interpretation;
- at E6, a formal description of the architecture of the TOE shall be provided [4, E6.1–E6.5]; referred to within this topic as FAD;
- at E6, a formal specification of security enforcing functions is required [4, E6.1, E6.2]; referred to within this topic as Formal SEFs.

Interpretation

Formal Model of Security Policy (FMSP)

278 The FMSP's aim is to enhance the assurance by formally specifying and proving that the TOE correctly enforces the stated security policy.

279 As described in ITSEM [6, 6.B.25], the system security policy for a system, or the product rationale for a product, should state in the security target the important principles of security (referred to as "the Security Policy"):

- for a system, it corresponds to the security objectives defined in the System Security Policy (SSP) which shall be addressed by a combination of TOE Security Enforcing Functions and personnel, physical or procedural means associated with the system, as described in [4, 2.9];
- for a product, it corresponds to the product rationale which gives an equivalence to the "system security objectives" by identifying the product's security features and all environmental assumptions [6, 6.B.25–6.B.28]. In some cases, a product rationale may specify security objectives.

At E4 and above, part or all the TOE Security Policy of the system or product, known in ITSEC as the Underlying Security Policy, shall be expressed in a formal style in the FMSP. 280

Formal Architectural Design (FAD)

The FSMP and FAD must be separated by a significant design step. Sufficient design steps, described in a formal language, may include the step from abstract behaviour description to a concrete description or flattening a distributed structure into a global structure with constraints. 283

Examples of insufficient design steps include the implementation of trivial constraints or simple data representation changes, such as implementing a set as a sequence. 284

Proofs

The ITSEC requires evidence in order to satisfy requirements. The following proofs shall be presented as evidence. 286

FMSP proofs shall prove evidence for the correctness of the security model. This includes but is not limited to the internal consistency of the security model, in the sense of non-existence of contradictions and invariance (i. e. the impossibility of transition from secure to insecure states) of its properties. 287

A proof must provide evidence that establishes the validity of the subject being proved. It shall be accompanied by a justification of why the proof obligation is a correct formal statement for the subject being proved. 290

Proofs must be formal and independently checkable. It must be possible for someone other than the author to check the correctness of the proof. This may be done in one of four ways: 291

- Manual proof, checked by a different human reviewer,
- Manual proof, checked by an automated proof checker,
- Computer generated proof, checked by a human reviewer,
- Computer generated proof, checked by an automated proof checker.

Proofs to be checked by a human reviewer must be well structured, give intuitive explanations for proof steps, and make good use of lemmas. It is often inappropriate to perform all steps of a proof; however, any steps left out for the reviewer must be obvious and clearly derivable, in that it must not require creative proof work to generate them. Experience has shown that such a level of formality is achievable. 292

3.3 Requirements of Common Criteria

In the following sections the requirements of Common Criteria for formal security policy models are reproduced. The paragraph numbers are indicated in the page margins. Any requirements are cited in english as well as german language. The english citations are taken from the text of the Common Criteria in the valid version 2.1 of August 1999 (cf. [1]). The german citations are taken from the translation of version 2.1 of the Commen Criteria (cf. [2]). Analogously to ITSEC, in cases of doubt the original english version is valid.

3.3.1 Teil 2: Funktionale Sicherheitsanforderungen / Part 2: Security functional requirements

The notion “(TOE) security policy model” is termed within the Common Criteria [1, Part 1, Glossary] as a structured representation of the security policy to be enforced by the TOE. Therefore it is necessary for the assessment of the requirements on formal securiy policy models to include the characterisation of the notion “TOE securiy policy” taken from part 2 of the Common Criteria [1, Teil 2] and reproduced in the sequel .

	Konzeption der funktionalen Anforderungen	**Functional requirements paradigm**
14	Die Prüfung und Bewertung eines TOE (EVG) befaßt sich in erster Linie damit sicherzustellen, daß eine festgelegte ***EVG-Sicherheitspolitik (TSP)*** für alle EVG-Betriebsmittel durchgesetzt wird. Die TSP legt die Regeln fest, nach denen der TOE (EVG) den Zugriff auf seine Betriebsmittel und somit alle durch den TOE (EVG) kontrollierten Informationen und Dienste steuert.	TOE evaluation is concerned primarily with ensuring that a defined ***TOE Security Policy (TSP)*** is enforced over the TOE resources. The TSP defines the rules by which the TOE governs access to its resources, and thus all information and services controlled by the TOE.
15	Die TSP besteht wiederum aus mehreren ***funktionalen Sicherheitspolitiken (SFPs)***. Jede SFP hat einen Anwendungsbereich der Kontrolle, der die durch die SFP kontrollierten Subjekte, Objekte und Operationen festlegt. Die SFP wird von einer ***Sicherheitsfunktion (SF)*** implementiert, deren Mechanismen die Politik durchsetzen und die nötigen Fähigkeiten bereitstellen.	The TSP is, in turn, made up of multiple ***Security Function Policies (SFPs)***. Each SFP has a scope of control, that defines the subjects, objects, and operations controlled under the SFP. The SFP is implemented by a ***Security Function (SF)***, whose mechanisms enforce the policy and provide necessary capabilities.

Die Teile eines TOE (EVG), auf die zur korrekten Durchsetzung der TSP Verlaß sein muß, werden mit dem Sammelbegriff ***EVG-Sicherheitsfunktionen (TSF)*** bezeichnet. Zu den TSF gehört die gesamte Hardware, Software und Firmware eines TOE (EVG), auf die direkt oder indirekt zur Durchsetzung der Sicherheit Verlaß sein muß.

Those portions of a TOE that must be relied on for the correct enforcement of the TSP are collectively referred to as the ***TOE Security Functions (TSF)***. The TSF consists of all hardware, software, and firmware of a TOE that is either directly or indirectly relied upon for security enforcement.

3.3.2 Teil 3: Anforderungen an die Vertrauenswürdigkeit / Part 3: Security assurance requirements

Structurally the Common Criteria differ from the ITSEC in many respects. Specifically the Evaluation Assurance Levels (EALs) are not directly defined but are to be understood as combinations of individual assurance components that are tied up to hierarchically ordered packages. The distinguished assurance *components* are parts of assurance *families* which themselves are pooled to assurance *classes*, the top level construct for the classification of assurance requirements.

Concrete requirements on (formal) securiy policy models resp. on the corresponding functional specification are formulated in the families ADV_FSP (Functional Specification) and ADV_SPM (Security Policy Model) of the assurance class ADV (Development) in part 3 of the Common Criteria [1, Part 3]. The table shown here presents the reference between the components of both assurance families and the evaluation assurance levels EAL5 – EAL7.

Assurance Famliy	EAL5	EAL6	EAL7
ADV_FSP	3	3	4
ADV_SPM	3	3	3

In the following sections the relevant extractions of the ADV class, specifically of the families ADV_FSP and ADV_SPM, are reproduced. Differences between the components ADV_FSP.3 and ADV_FSP.4 are identified through underlining and **bold face** printing.

Further relationships exist to the families ADV_HLD (High-level design), ADV_LLD (Low-level design) and ADV_RCR (Representation correspondence). However, requirements from components of these families are not cited here, since the relation to formal security policy models is indirectly given via the funcional specification (family ADV_FSP).

Klasse ADV: Entwicklung

98 Die Vertrauenswürdigkeitsklasse ADV definiert Anforderungen zur schrittweisen Verfeinerung der TSF, angefangen bei der EVG-Übersichtsspezifikation in den ST bis zur tatsächlichen Implementierung. Jede dieser TSF-Darstellungen liefert Informationen, die dem Evaluator helfen festzustellen, ob die funktionalen Anforderungen des TOE (EVG) erfüllt sind.

Anwendungsbemerkungen

302 Die EVG-Sicherheitspolitik (TSP) ist die Menge der Regeln, die bestimmen, wie die Betriebsmittel innerhalb eines TOE (EVG) verwaltet, geschützt und verteilt werden, ausgedrückt durch die funktionalen EVG-Sicherheitsanforderungen. Vom Entwickler wird nicht explizit gefordert, eine TSP bereitzustellen, da die TSP in den funktionalen EVG-Sicherheitsanforderungen mittels einer Kombination aus funktionalen Sicherheitspolitiken (SFP) und den anderen einzelnen Anforderungselementen ausgedrückt ist.

309 Erhebliche Vertrauenswürdigkeit kann dadurch erreicht werden, daß sichergestellt wird, daß die TSF über jede ihrer Darstellungen verfolgt werden können, und daß das TSP-Modell mit der funktionalen Spezifikation übereinstimmt. Die Familie ADV_RCR enthält Anforderungen für entsprechende Übereinstimmungen zwischen den verschiedenen TSF-Darstellungen, und die Familie ADV_SPM enthält Anforderungen an eine Übereinstimmung zwischen dem TSP-Modell und der funktionalen Spezifikation. [. . .]

Funktionale Spezifikation (ADV_FSP)

99 Die funktionale Spezifikation beschreibt die TSF und muß eine vollständige und getreue Umsetzung der Sicherheitsanforderungen an den TOE (EVG) sein. Die funktionale Spezifikation enthält auch Details zur externen Schnittstelle zum TOE (EVG). Von Benutzern des TOE (EVG) wird erwartet, daß sie mit den TSF über diese Schnittstelle interagieren.

Class ADV: Development

Assurance class ADV defines requirements for the stepwise refinement of the TSF from the TOE summary specification in the ST down to the actual implementation. Each of the resulting TSF representations provide information to help the evaluator determine whether the functional requirements of the TOE have been met.

Application notes

The TOE security policy (TSP) is the set of rules that regulate how resources are managed, protected and distributed within a TOE, expressed by the TOE security functional requirements. The developer is not explicitly required to provide a TSP, as the TSP is expressed by the TOE security functional requirements, through a combination of security function policies (SFPs) and the other individual requirement elements.

Significant assurance can be gained by ensuring that the TSF can be traced though each of its representations, and by ensuring that the TSP model corresponds to the functional specification. The ADV_RCR family contains requirements for correspondence mappings between the various TSF representations, and the ADV_SPM family contains requirements for a correspondence mapping between the TSP model and the functional specification. [. . .]

Functional specification (ADV_FSP)

The functional specification describes the TSF, and must be a complete and accurate instantiation of the TOE security functional requirements. The functional specification also details the external interface to the TOE. Users of the TOE are expected to interact with the TSF through this interface.

Ziele	**Objectives**	314
Die funktionale Spezifikation ist eine Beschreibung der für den Benutzer sichtbaren TSF-Schnittstelle und des TSF-Verhaltens auf hoher Ebene. Sie ist eine Umsetzung der funktionalen EVG-Sicherheitsanforderungen. Die funktionale Spezifikation muß zeigen, daß alle EVG-Sicherheitsanforderungen angesprochen sind.	The functional specification is a high-level description of the user-visible interface and behaviour of the TSF. It is an instantiation of the TOE security functional requirements. The functional specification has to show that all the TOE security functional requirements are addressed.	
ADV_FSP.[34] Semiformale / Formale funktionale Spezifikation	**ADV_FSP.[34] Semiformal / Formal functional specification**	
Elemente zu Entwickleraufgaben:	**Developer action elements:**	
Der Entwickler muß eine funktionale Spezifikation bereitstellen.	The developer shall provide a functional specification.	ADV_FSP.[34].1D
Elemente zu Inhalt und Form des Nachweises:	**Content and presentation of evidence elements:**	
Die funktionale Spezifikation muß die TSF und ihre externen Schnittstellen in einem semiformalen / **formalen** Stil beschreiben, der, wo angemessen, von einem informellen, erläuternden Text unterstützt ist.	The functional specification shall describe the TSF and its external interfaces using a semiformal / **formal** style, supported by informal, explanatory text where appropriate.	ADV_FSP.[34].1C
Die funktionale Spezifikation muß in sich konsistent sein.	The functional specification shall be internally consistent.	ADV_FSP.[34].2C
Die funktionale Spezifikation muß den Zweck und die Methode des Gebrauchs aller externen TSF-Schnittstellen beschreiben, einschließlich aller Details sämtlicher Wirkungen, Ausnahmen und Fehlermeldungen.	The functional specification shall describe the purpose and method of use of all external TSF interfaces, providing complete details of all effects, exceptions and error messages.	ADV_FSP.[34].3C
Die funktionale Spezifikation muß die TSF vollständig darstellen.	The functional specification shall completely represent the TSF.	ADV_FSP.[34].4C
Die funktionale Spezifikation muß eine Erklärung enthalten, daß die TSF vollständig dargestellt sind.	The functional specification shall include rationale that the TSF is completely represented.	ADV_FSP.[34].5C

	Elemente zu Evaluatoraufgaben:	**Evaluator action elements:**
ADV_FSP.[3**4**].1E	Der Evaluator muß bestätigen, daß die bereitgestellten Informationen alle Anforderungen an Inhalt und Form des Nachweises erfüllen.	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.
ADV_FSP.[3**4**].2E	Der Evaluator muß feststellen, daß die funktionale Spezifikation eine getreue und vollständige Umsetzung der funktionalen EVG-Sicherheitsanforderungen ist.	The evaluator shall determine that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.
	Sicherheitsmodell (ADV_SPM)	**Security policy modeling (ADV_SPM)**
105	Sicherheitsmodelle sind strukturierte Darstellungen der Sicherheitspolitiken der TSP und dienen der Schaffung einer stärkeren Vertrauenswürdigkeit, daß die funktionale Spezifikation mit den Sicherheitspolitiken der TSP und schließlich mit den funktionalen Anforderungen an den TOE (EVG) übereinstimmt. Dies wird durch entsprechende Übereinstimmungen zwischen der funktionalen Spezifikation, dem Sicherheitsmodell und den Sicherheitspolitiken, die im Modell dargestellt sind, erreicht.	Security policy models are structured representations of security policies of the TSP, and are used to provide increased assurance that the functional specification corresponds to the security policies of the TSP, and ultimately to the TOE security functional requirements. This is achieved via correspondence mappings between the functional specification, the security policy model, and the security policies that are modelled.
	Ziele	**Objectives**
365	Die Zielsetzung dieser Familie besteht in der Schaffung zusätzlicher Vertrauenswürdigkeit, daß die in der funktionalen Spezifikation enthaltenen Sicherheitsfunktionen die Politiken in der TSP durchsetzen. Dies wird erreicht durch die Entwicklung eines Sicherheitsmodells, das auf einer Teilmenge der Politiken der TSP basiert und durch einen Nachweis der Übereinstimmung zwischen der funktionalen Spezifikation, dem Sicherheitsmodell und diesen Politiken der TSP.	It is the objective of this family to provide additional assurance that the security functions in the functional specification enforce the policies in the TSP. This is accomplished via the development of a security policy model that is based on a subset of the policies of the TSP, and establishing a correspondence between the functional specification, the security policy model, and these policies of the TSP.

Anwendungsbemerkungen

Obwohl eine TSP sämtliche Politiken umfassen kann, haben TSP-Modelle bisher nur Teilmengen dieser Politiken dargestellt, da die Erstellung von Modellen für bestimmte Politiken beim gegenwärtigen Stand der Technik nicht möglich ist. Der aktuelle Stand der Technik bestimmt, für welche Politiken Modelle erstellt werden können, und der PP/ST-Verfasser soll die spezifischen Funktionen und die mit diesen verknüpften Politiken, für die Modelle erstellt werden können und daher auch erstellt werden müssen, identifizieren. Zumindest müssen Modelle für Politiken für Zugriffskontrolle und Informationsflußkontrolle erstellt werden (wenn diese Teil der TSP sind), da dies beim aktuellen Stand der Technik möglich ist.

Application notes

While a TSP may include any policies, TSP models have traditionally represented only subsets of those policies, because modeling certain policies is currently beyond the state of the art. The current state of the art determines the policies that can be modeled, and the PP/ST author should identify specific functions and associated policies that can, and thus are required to be, modeled. At the very least, access control and information flow control policies are required to be modeled (if they are part of the TSP) since they are within the state of the art. 367

Für jede Komponente in dieser Familie besteht die Anforderung, die Regeln und Eigenschaften der anwendbaren Politiken der TSP im TSP-Modell zu beschreiben und sicherzustellen, daß das TSP-Modell die entsprechenden Politiken der TSP erfüllt. Bei den „Regeln“ und „Eigenschaften“ eines TSP-Modells ist beabsichtigt, Flexibilität hinsichtlich der Art des Modells, das entwickelt werden kann zu erlauben (zum Beispiel Zustandsübergang, Interferenz-Freiheit). [. . .]

For each of the components within this family, there is a requirement to describe the rules and characteristics of applicable policies of the TSP in the TSP model and to ensure that the TSP model satisfies the corresponding policies of the TSP. The “rules” and “characteristics” of a TSP model are intended to allow flexibility in the type of model that may be developed (e. g. state transition, non-interference). [. . .] 368

ADV_SPM.3 Formales EVG-Sicherheitsmodell

Elemente zu Entwickleraufgaben:

ADV_SPM.3 Formal TOE security policy model

Developer action elements:

Der Entwickler muß ein TSP-Modell bereitstellen. ADV_SPM.3.1D

The developer shall provide a TSP model. ADV_SPM.3.1D

Der Entwickler muß die Übereinstimmung zwischen der funktionalen Spezifikation und dem TSP-Modell nachweisen oder, wie jeweils angemessen, beweisen. ADV_SPM.3.2D

The developer shall demonstrate or prove, as appropriate, correspondence between the functional specification and the TSP model. ADV_SPM.3.2D

	Elemente zu Inhalt und Form des Nachweises:	**Content and presentation of evidence elements:**
ADV_SPM.3.1C	Das TSP-Modell muß formal sein.	The TSP model shall be formal.
ADV_SPM.3.2C	Das TSP-Modell muß die Regeln und Eigenschaften aller derjenigen Politiken der TSP beschreiben, für die ein Modell erstellt werden kann.	The TSP model shall describe the rules and characteristics of all policies of the TSP that can be modeled.
ADV_SPM.3.3C	Das TSP-Modell muß eine Erklärung enthalten, die nachweist, daß dieses in bezug auf alle Politiken der TSP, für die ein Modell erstellt werden kann, konsistent und vollständig ist.	The TSP model shall include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP that can be modeled.
ADV_SPM.3.4C	Der Nachweis der Übereinstimmung zwischen dem TSP-Modell und der funktionalen Spezifikation muß zeigen, daß alle Sicherheitsfunktionen in der funktionalen Spezifikation in bezug auf das TSP-Modell konsistent und vollständig sind.	The demonstration of correspondence between the TSP model and the functional specification shall show that all of the security functions in the functional specification are consistent and complete with respect to the TSP model.
ADV_SPM.3.5C	Wo die funktionale Spezifikation semiformal ist, muß auch der Nachweis der Übereinstimmung zwischen TSP-Modell und funktionaler Spezifikation semiformal sein.	Where the functional specification is semiformal, the demonstration of correspondence between the TSP model and the functional specification shall be semiformal.
ADV_SPM.3.6C	Wo die funktionale Spezifikation formal ist, muß auch der Beweis der Übereinstimmung zwischen TSP-Modell und funktionaler Spezifikation formal sein.	Where the functional specification is formal, the proof of correspondence between the TSP model and the functional specification shall be formal.
	Elemente zu Evaluatoraufgaben:	**Evaluator action elements:**
ADV_SPM.3.1E	Der Evaluator muß bestätigen, daß die bereitgestellten Informationen alle Anforderungen an Inhalt und Form des Nachweises erfüllen.	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

3.4 Determination of the catalog of requirements

In compliance with the citations assembled above the concrete requirements on formal securiy policy models are determined in this chapter. The formation of the individual parts of the requirements is oriented towards a justification of the concepts and needs of ITSEC as well as CC. Naturally, an evaluation assurance level is targeted that requires the provision of a formal security policy model (cf. [4, Par. E[456].2] resp. [1, Part 3, Par. 223, 229, 234]).

3.4.1 Terminology

For describing the constituing parts of the TOE securiy policy a different terminology is used in ITSEC and CC. Neiher in ITSEC nor in CC the corresponding parts of the TOE security policy model are explicitly termed.

In order to respect the terminological constraints of both criteria, a standardisation of the terminology is proposed in this chapter that will consequently be used in the sequel. The goal of this standardisation is to support a demarcation of the identified objects that is consistent with the literature (cf. Chap. 6).

Four distinguished notions are described that are pairwise assigned to the TOE security policy and the TOE security policy model on the one hand resp. the analysis and the design process of software engineering on the other hand (cf. Tbl. 3.1):

Security Characteristics (Sicherheitscharakteristika) literally corresponds to the term used in the english original of the CC (cf. [1, Part 3, Par. 368]) for one part of the TOE security policy. Due to the necessary separation from the notion *Security Properties (Sicherheitseigenschaften)* the problematic translation in the german edition of the CC (cf. [2, Part 3, Par. 368]) was not adopted. The notion *Security Characteristics (Sicherheitscharakteristika)* is going to be used as the analogon of the term “practices (Praktiken)” used in ITSEC (cf. [4, Par. 2.13]). Together with the *Security Features (Sicherheitsmerkmale)* it is assigned to the desing process, i. e. it comprises all definitions, attributes, actions etc. necessary for the detailed characterisation of the functionality of the TOE security policy. This functionality is oriented towards the realisation of the general rules of operation of the TOE security policy formulated in the *Security Principles (Sicherheitsprinzipien)*.

Security Principles (Sicherheitsprinzipien) is used in ITSEC (cf. [4, Par. 2.81]) and ITSEC JIL (cf. [8, Par. 279]) for the description of the TOE security policy. Due to the dichotomy of the TOE security policy the notion is used as the analogon of the terms “laws and rules (Gesetze und Regeln)” used in ITSEC (cf. [4, Par. 2.13]) resp. “rules (Regeln)” used in CC (cf. [1,

	Design process	Analysis process
Security Policy (Sicherheitspolitik)	*Security Characteristics (Sicherheitscharakteristika)*	*Security Principles (Sicherheitsprinzipien)*
Security Policy Model (Sicherheitsmodell)	*Security Features (Sicherheitsmerkmale)*	*Security Properties (Sicherheitseigenschaften)*

Table 3.1: Terminology of TOE security policy and model

Part 3, Par. 368]). Together with the *Security Properties (Sicherheitseigenschaften)* the notion *Security Principles (Sicherheitsprinzipien)* is assigned to the analysis process, i. e. it comprises of the general rules of operation of the TOE security policy derived from the analysis of the security environment and objectes and stated in the security requirements. These rules of operation form the framework for the formulation of the *Security Characteristics (Sicherheitscharakteristika)* and thus typically contain no detailed functional interrelationships.

Security Features (Sicherheitsmerkmale) correspond to the *Security Characteristics (Sicherheitscharakteristika)*. The notion comprises the modeling of the functional features of the TOE security policy. These form the basis for the demonstration of validity of the *Security Properties (Sicherheitseigenschaften)*.

Security Properties (Sicherheitseigenschaften) correspond to the *Security Principles (Sicherheitsprinzipien)*. The notion comprises the modeling of the basic properties of the TOE security policy. These form the framework for the formulation of the *Security Features (Sicherheitsmerkmale)*.

3.4.2 Specification

The formal security policy shall specify all ingridients of the TOE security policy provided that they can be modeled formally according to the state of the art (cf. [4, Par. E[456].2], [8, Par. 280] resp. [1, Part 3, Par. 367 et seq. and Element ADV_SPM.3.2C]). In consideration of its integration in the hierarchically structured design of the TOE the formal security policy model at leat shall consist of the following two components:

1. *Security Properties (Sicherheitseigenschaften)*

   In this part of the model the security principles (Sicherheitsprinzipien) of the TOE security policy are formally specified. The security properties comprise a formal description of general rules of operation of the TOE security policy stated in the security principles (Sicherheitsprinzipien).

2. *Security Features (Sicherheitsmerkmale)*

   In this part of the model the security characteristics (Sicherheitscharakteristika) of the TOE security policy are formally specified. The security features comprise a formal description of the definitions, attributes, actions etc. necessary for the detailed characterisation of the functionality of the TOE security policy stated in the security characteristics (Sicherheitscharakteristika)

The fomal specification shall be appropriate (cf. Sec. 3.4.3) for the correspondance demonstration of the security policy model and the security principles and characteristics of the security policy (cf. [4, Par. E[456].3] resp. [1, Part 3, Element ADV_SPM.3.3C]).

Both levels of the specification shall be linked with each other via a formal verification (cf. Sec. 3.4.4), i. e. the validity of the security properties shall be proved for the security features. In order to facilitate the verification it may be suggestive to specify additional model levels between security properties and features.

The formal specification shall be apprpriate (cf. Sec. 3.4.3 and 3.4.4) for the correspondence demonstration of the securiy policy model and the architectural desing (cf. [4, Par. E6.6]) resp. the functional specification (cf. [1, Part 3, Element ADV_SPM.3.4C]).

It is admissable to refer to a (known) generic formal security policy model (e. g. non-interference according to Rushby [41]) Such models typically contain both levels of specification required above. Due to their generic character these models shall be instantiated for use with the concrete TOE. The instantiation shall also be formal (cf. Chap. 6) and conform to the requirements mentioned above.

3.4.3 Interpretation

The explanation which is called “informal interpretation” in ITSEC and “rationale” in CC shall be part of the documentation of the formal security policy model. It shall provide the connection between the specification, which is given in a formal notation, and the development documents that relate to the formal security policy model. The explanation shall consist of the following components:

1. The correspondence demonstration of the formal security policy model and the TOE security policy formulated in the functional requirements (cf. [4, Par. E[456].3] resp. [1, Part 3, Element ADV_SPM.3.3C]). Corresponding to the structure of the formal specification this results in obligations for demonstrating the

   (a) Correspondence of the security properties (Sicherheitseigenschaften) and the security principles (Sicherheitsprinzipien).

(b) Correspondence of the security features (Sicherheitsmerkmale) and the security characteristics (Sicherheitscharakteristika).

The statement of functional security requirements is of much lower importance in ITSEC than in CC. Consequentially, a relation between security policy and security objectives is established in JIL [8, Par. 279]. Nevertheless, the obligation for demonstration defined here conforms to the requirements of the CC as well as to the requirements of the ITSEC. Concrete experiences with evaluations according to ITSEC actually show that it is beneficial to refine the security objectives in detailed (functional) requirements (cf. Sec. 5.3). For ITSEC as well as CC the required correspondence demonstration therefore establishes an indirect relation to the security objectives.

2. Correspondence demonstration of the formal security policy model and the functional specification (cf. [1, Part 3, Element ADV_SPM.3.4C]).

   Although the direct relation between (formal) security policy model and (formal) architectural design established in ITSEC [4, Par. E6.6] is affirmed in JIL [8, Chapter 19], the obligation for demonstration defined here corresponds to the requirements of the CC as well as to the requirements of ITSEC. The concrete relation with the architectural design will be established via the correspondence demonstration[2] of the functional specification and the architectural design, which is necessary anyway.

   Where the functional specification is given in semiformal (cf. [4, Par. E[45].2] and [1, Part 3, Component ADV_FSP.3]) resp. formal (cf. [4, Par. E6.2] and [1, Part 3, Component ADV_FSP.4]) notation the informal explanation shall be complemented with a semiformal demonstration resp. a formal proof (cf. Sec. 3.4.4).

The required explanations shall bridge the security requirements and functions (cf. [1, Part 3, Par. 314]) in order to arrive at an increase of assurance in the TOE security policy via the formal security poliy model (cf. [1, Part 3, Par. 309]). Beside this, they shall ensure the quality of the formal security policy model w. r. t. the securits needs via the connection of the different presentation formats.

When the specification is built on the basis of a (known) generic formal security policy model the required instantiation is subject of interpretation, i. e. the explanations shall relate to the instantiated model (cf. Chap. 6).

[2] For the guidance at hand the relation between functional specification and architectural design is of subordinate importance and therefore is not elaborated.

3.4.4 Verification

Corresponding to the required structure of the formal security policy model the following proofs shall be provided:

1. The suitability of the formally specified security features (Sicherheitsmerkmale) to enforce the formally specified security properties (Sicherheitseigenschaften) shall be formally verified. For this the correct and complete correspondence of both levels of the specification (s. Abschnitt 3.4.2) shall be formally proved. When additional levels of the security policy are modeled between the security properties and features, the correspondence between all adjacent levels shall be proved.

   Where the specification is built via instantiation of a (known) generic formal security policy model, the required proof frequently exist as a part of the generic model. In such cases arise proof obligations that ensure that the respective instantiation remains within the given framework of the generic model (cf. Chap. 6).

   This part of the verification of the formal security policy model corresponds to the proofs of invariance of properties of the formal security policy model required by ITSEC JIL (vgl. [8, Abs. 287]). It is supposed to ensure at a high abstraction level the quality of the security functionality measured against the security requirements.

2. The absence of contradictions within the formal securit policy model shall be formally proved. Due to fundamental problems (cf. Sec. 4.5) this part of the verification can not be completely adressed within the formal calculus that is used for the specification. The concepts located outside the calculus that are necessary for the proof of absence of contradictions shall be well-founded and restricted to a minimum.

   When the formal specification is structured in several levels, the pairwise correspondence of which is to be formally proved, it is typically sufficient to verify the absence of contradictions for the lowest level of the specifiaction, i. e. for the level of the security features. Moreover, the proof of absence of contradictions is relocatable to the level of the functional specification and further to the level of the architectural design, if their correspondence with the formal security policy model is formally proved.

   Where the specification is built via instantiating a (known) generic formal security policy model, the proof frequently can be constrained to the absence of contradictions within the concrete instatiation. The precondition for this is that the absence of contradictions within the generic model is given.

This part of the verification of the formal security policy model corresponds to the proof of internal consistency of the formal security policy model required by ITSEC JIL (vgl. [8, Abs. 287]). It shall ensure that the formal arguments rest on an solid basis.

The highest assurance levels of ITSEC and Common Critera respectively require a formal specification of the security functions (cf. [4, Par. E6.2] resp. [1, Part 3, Component ADV_FSP.4]) as well as a formal architectural design (cf. [4, Par. E6.5] resp. [1, Part 3, Component ADV_HLD.5]). In this case the formal proof of correspondence between the functional specification and architectural design is required which is not subject of this guidance. However, the correspondence between the formal security policy model and the functional specification shall also be formally verified. (cf. [1, Part 3, Element ADV_SPM.3.6C]). As already noted in Sec. 3.4.3 this proof obligation also conforms to the requirements of ITSEC, since the correspondence with the architectural desing is to be indirectly proved.

Chapter 4

Basic Concepts of Formal Methods

In Information Technology (IT), like in all other disciplines, sufficient guarantees of *security properties* can only be provided if *scientific methods* are applied. Despite its relatively short history software engineering meanwhile offers practically applicable development *methods* whose scientific foundations compare to those of established engineering disciplines and therefore in particular allow for third party evaluations on an *objectified* basis. Thus, the rapidly increasing deployment of systems that are subject to manifold threats and where a failure of protection mechanisms will lead to severe losses becomes justifiable from an economic as well as from a social point of view.

Based on concepts from mathematics and theoretical computer science,and, in particular (symbolic) logic as well as the description and analysis of computational models, so called *Formal Methods* have become a key discipline in computer science.

One of the key issues of Formal Methods is that they rely on fixed and exactly defined languages whose semantics are defined upon mathematical concepts. This means that starting with such *formal descriptions* (specifications) (security) properties and relations between specifications can be formulated and established by *verification methods* that build on the underlying mathematical models. In most cases the methods are *complete*[1] in the sense that opposed to testing they cover *all* computations of a system.

While syntactical correctness – i.e. conformance with the given language – can be checked by help of standard procedures the *reliability* of the verification of properties or relationships remain a critical point. It can only be achieved by using fixed methods that are independent of a particular case and that are implemented in *tools*[2].

[1] The complete coverage of computations should not be confused with the notion of completeness of logical proof systems.

[2] In addition to reliability also the complexity of analysis procedures forces the utilization of

4.1 Formal Methods

For an assessment of formal methods applied in the context of CC evaluations it seems to be necessary to recall the overall aim and the general setting before taking a closer look at particular aspects.

Formal methods apply *mathematical* notions to elements of software engineering. In contrast to the development of mathematical theories this takes place under different conditions and, above all, according to different aims.

Mathematical theory formation takes place within the scientific community. Driving force for the individuals involved is the *publication* of results to prevail on others to take these up in their own work[3]. According to this situation one aims at results of general interest. The public scientific process includes *control functions* that manifest themselves in various institutional forms, like, for example, review boards.

With respect to scientific progress this kind of *quality assurance* is devoted to the main results. As opposed to that it frequently happens that certain details of the presentation and the proofs contain errors that remain undiscoverd quite a long time. Typically relevant results are worked on for some period with respect to formulation and proofs thereby removing early shortcomings in detail. After that mature mathematical theories present themselves in a form that potentially could be *formalized* in the sense this term is used here. However,such formalizations, as for example in the MISRA project,are actually carried out rarely.

In the area of software engineering one has to distinguish between single developments – in this context this concerns formal artifacts as requested by the CC starting with EAL5 – and the general method which is used for the descriptions and proofs. The increase in reliability with respect to the former follows from the fact that the latter is constructive, can be realized by tools, and has to be proved correct only *once and for all*.

Only for a relatively small community of developers and evaluators (certifiers) single developments are

- of interest
- accessible, and
- understandable.

Quality assurance with respect to highly critical systems has to exclude all errors with the maximum degree of reliability that is conceivable. Formal methods allow for such guarantees. As opposed to mathematical theories these guarantees

tools.

[3]The citation index is an important criterium for assessing the scientific qualification.

do not result from a social process, instead they follow from a rigid application of a method which is known as being correct. The method itself is

- given independently of a particular application,
- public, and
- subject to discussions in the scientific community.

Its safe application becomes feasible (only) through the use of corresponding *tools*.

Following the usage mathematical of logic we speak of particular developments as *object level* entities while methods and tools are dealt with at the so called *meta level*. *Formal* methods are distinguished by the fact that meta level considerations are carried out within a mathematical theory. In particular this means that a mathematical *semantics* is associated with the language constructs given by the method. Based on this semantics (meta level) properties can be established following the usual mathematical practice. Analysis and verification procedures realized in tools heavily rely on these results. In this sense one can speak of the *correctness* of (the meta theory of) a method.

This guide aims at an assessment of individual developments. A *formal* justification[4] of the underlying method is neither mandatory according to the regulations of CC nor available for the methods and tools used today. This does not mean that one should not look for certain general standards with regard to methods and tools. In particular the reliability of tools can be improved without running into a circulus vitiosus.

The given method defines in a more or less restrictive way the framework for modeling systems and their security properties. *How* a concrete instance is treated within this framework is left to the developer. Compared to other engineering disciplines there are comparatively few guidelines or even standards. It is the aim of this guide to improve the current situation with respect to the choice and assessment of formal modeling techniques. From point of view of the authors the problem of *adequacy* of a chosen model will remain the most critical issue. It should be noted that the interpretation and assessment of object level developments requires a certain understanding of the underlying meta level theory.

To sum up we may state the following points:

- There should be a clear distinction between individual object level developments and the general method with its underlying meta theory.
- Methods have to be accessed with respect to scientific foundation of their meta theory. For tools the correct technical realization has to be included.

[4] in the sense we use this term for individual developments

- The reliability of object level results depends on a rigid application of a fixed method (including analysis and verification procedures). This is guaranteed by the use of tools.
- With respect to individual developments the main emphasis has to be on checking the adequacy of the formal model.

4.2 Methods and Tools

Formal methods provide a mathematically grounded framework for

- the description of systems (specifications),
- the formulation of postulated properties and relations, and
- the verification of these postulates.

As before we continue to use the term (formal) *development* for the whole of the resulting artifacts.

First of all formal methods confront the user with a *formal language*. For textual representations the scope of the language, *syntax*, is given by grammars or related formalisms. Today there is an increasing use of graphical representations augmented by textual annotations.

In principle there is no difference: In all cases the *admissible* development objects are uniquely defined, the *syntactical correctness* being *decidable*. Typically one distinguishes between grammatical (context free) correctness and *type correctness*. The latter is concerned with properties that are checked by compilers as part of the so called “semantic analysis”. Most of the grammatical analysis procedures work in linear time, while type checking is more involved. For some systems type correctness is not even decidable which means that type checking needs certain proof obligations to be fulfilled. Strictly speaking this is no longer only a syntactical check.

Editing, visualizing, and syntactical correctness checks are supported by the front-end of tools. Correct syntactical structures tailored for an efficient syntactical analysis are usually transformed into *internal representations* that are used in subsequent steps.

At the meta level structures like these form the starting point for associating *semantics* with language constructs. *Mathematical models* used for semantic definitions have developed into an extremely rich variety of approaches over the last years. As a rough classification we mention

- models of logical languages, i.e. models in a stricter sense,

- algebraic and category theoretic semantics for abstract data types (ADT),
- automata and state transition systems, and
- semantic models for programming languages.

The meta theory of a method provides general assertion about the underlying mathematical models. In particular this concerns verification procedures for establishing postulated properties and relationships as part of developments. The *reliability* of these proofs depends on fundamental requirements for *formal* verification techniques:

- The proof procedure is given in an algorithmic way, i.e. all steps are fixed in a unique and fully detailed way. Basically that proof procedures are given in a way that can lead directly to an implementation.
- The correctness of the verification procedure can be established once and for all. In particular it does not depend on user interactions in a special application.
- The justification (of the correctness) of the procedure is part of a mathematically objectified theory.

As far as their actual content is concerned, methods can be classified according to the extent they support general aspects of software engineering or even application specific techniques.

Systems like Isabelle, [49, 50], and PVS, [51], provide an expressive *logical formalism* as a generic framework for formal developments. In addition to the logical language and the corresponding deductive mechanism they offer support for inductive and recursive definitions as part of a powerful *type system*. Beyond a general mechanism for organizing logical theories there are no means for structuring developments.

Systems of this kind are used in two ways. Firstly, individual developments can be realized directly, either in an ad hoc way or by following a certain scheme. Examples for such schemes in the security area are formal models for information flow properties in [21, 28, 41] and the inductive verification of protocols in [47]. Secondly these systems can be used to support more specific methods, like for example a verification environment for imperative programs in [48]. One speaks of an *embedding* of these methods.

While Isabelle and PVS have to be considered more as logical formalisms than as actual development methods there are a number of systems that support special techniques, among others for the formal treatment of

- data structures and algorithms,

- state transition systems (automata),
- programming languages (like Java and C),
- concurrency (reactive, parallel, and distributed systems), and
- real time.

Specification languages mirror the design paradigms supported by the systems at the syntactical level. The language constructs are mapped to corresponding models by a so called *semantic evaluation*.

Typically developments are *structured* by dividing specifications into components (substructures) and by linking specifications according to relationships like refinement, transformation, simulation, and validity of properties.

Examples for methods that support one or more of the above mentioned aspects and structuring mechanisms are AutoFocus, [52, 53], B, [55], RAISE, [57], VDM, [56], VSE, [23] and Z, [54].

Besides these still fairly general methods a great number of approaches tailored for a special application domain have emerged. Compared to the more general systems special purpose systems allow for a stronger guidance of the user and an increase in efficiency. In the security area systems for the specification and (automatic) analysis of *cryptographic protocols* form a typical example. On the other hand comprehensive developments are concerned with several different aspects and therefore require approaches that integrate special modeling techniques within a sufficiently general framework.

4.3 Verification Techniques

Formal methods lead to a clear distinction between *determinations* which above have been called “specifications” and relations between these. The latter have to be regarded as postulates until their validity has been established. The process that turns postulates into established results will be called *verification*. In the opinion of the authors the verification on an abstract basis, i.e. before a technical realization, constitutes the real strength of formal methods[5]

In the context of ITSEC and CC we have the following basic verification tasks:

1. the validity of security properties of the given security policy (see section 3.4.4) [6],

[5] Of course also formal modeling alone might lead to a deeper insight into security mechanisms.

[6] The verification of (security) *properties* can be regarded as the verification of a “is-satisfied” relation between a system specification and a requirements specification.

2. the correspondence between the functional specification and the security policy, and

3. the correspondence between the high-level design and the functional specification.

The *semantic* models at the meta level uniquely determine whether a postulated relationships is valid or not. However, as discussed in the beginning, for verification purposes it is not enough to use a mathematical argumentation in the usual sense. Instead, one aims at a fixed algorithmic procedure that takes as input the *syntactical* representations of the specifications involved.

Verification techniques used as part of formal development methods can roughly be classified into *model based* and *deductive* approaches. Model based (verification) techniques use a more or less explicit internal representation of the (semantic) models for checking whether they fulfill the required properties. In most cases these properties are given by logical formulae, i.e. we are concerned with a validity problem as discussed in the following section.

Classical techniques for *model checking* reduce the given verification task to a decision problem for certain computation mechanisms, like for example the question whether the language of a (finite) automaton constructed from the input is empty. These techniques are restricted to *finite state* systems. Increasingly systems that build on a *symbolic* representation of the state space are used in this context. Due to their ability to deal with sets of states they are more efficient and can be applied also for decidable problem classes of infinite state systems. Well known model based verification systems are SMV, [58], and SPIN, [60]. In general properties of infinite state systems are not decidable. To use model checking also in these cases *abstraction* techniques have to be applied, [59].

Besides the generic approaches mentioned so far there exists a great variety of systems that are application specific or restricted to particular properties. The latter for example holds for automatic *program analysis*. .

Starting with so called *axioms* which are assumed to be “true” deductive approaches infer certain assertions as *conclusions* . The sequence of inference steps that lead to a conclusion is called a *proof*. The “axiomatic method” has become the main mode of operation – perhaps better presentation – in mathematics.

Formal proofs are carried out within a fixed (formal) language and a fixed *inference system* that precisely defines the admissible proof steps on a purely syntactical basis. In mathematics there are (and were) only a few attempts to actually *carry out* formal proofs in this strict sense. Moreover, today there is a wide agreement that a distinction has to been made between the process of (mathematical) discovery and the presentation of mature theories.

In the context of methods for formal development system specifications are transformed to, or directly given as axioms of a logical language. Verification

tasks give rise to proof obligations which require certain assertions to be inferable (as theorems). The proof obligations are discharged by generating proofs in the given inference system thereby showing that the assertions actually follow from the axioms.

The axiomatic approach is very general since it's only limitations lie in the expressiveness of the underlying logical language. However, in most cases formal proofs require some *user interaction* to guide the proof search. Currently "hybrid" approaches are developed that for example

- combine automatic and interactive proof techniques,
- integrate model based approaches to fulfill certain proof obligations, or the other way round,
- use model based techniques to generate conditions that are then treated deductively.

4.4 Logic

Symbolic logic as it goes back to Gottlob Frege, [61], gives a mathematical precise account of the notion of formal proof. In particular in it's form as *computational logic* it provides the basis for implemented proof systems and their (meta level) analysis.

4.4.1 Basic Notions

Formal inference systems are given by

- a formally defined (logical) language and
- a concept of proof based on rules that operate on syntactical entities.

Formal syntax definitions consist of (formation) rules that describe how the various constructs are composed out of simpler constituents. Logical languages typically distinguish between *terms* to denote objects and *formulas* to formulate propositions. The starting point for building up terms and formulas is given by application specific symbols for *functions* and *relations*. Terms and atomic formulas are obtained by *applying* function and relation symbols to arguments which (in turn) consist of terms. Atomic (unstructured) terms are constants (nullary function symbols) or *variables*. The basic vocabulary is given by a so called *signature*. Formulas in logical languages are structured by propositional *connectives*, like $\neg$

(not), $\wedge$ (and), $\vee$ (or), and $\rightarrow$ (implies), and by *quantification* (on variables), $\forall x$: (for all x) and $\exists x$: (for some x).

The fact that a function denoted by the (unary) symbol f is the *inverse* of a function denoted by the (unary) symbol g is expressed by $\forall x : \ g(f(x)) = x$, where $=$ is a (binary) relation symbol denoting equality. In the area of formal program development one uses mostly *typed* languages where for the function and relation symbols not only a number is given as an *arity* but also the domains from which the respective arguments are taken. For basic domains one often speaks of *sorts*. In typed languages also variables have to be declared to be of some type or sort. Using a sort symbol D the above proposition becomes $\forall x : D.\ g(f(x)) = x$. The injectivity of a function denoted by f is expressed by $\forall x_1 : D.\ \forall x_2 : D.\ (f(x_1) = f(x_2) \rightarrow x_1 = x_2)$.

An interpretation $\mathcal{M}$ associates a concrete mathematical structure with a given signature Sig, thereby giving meaning to the elements of Sig. The semantics for a logical language $\mathcal{L}(Sig)$ is given be defining the *validity* of a proposition (formula) φ constructed from Sig in $\mathcal{M}$, written $\mathcal{M} \models \varphi$. In this case $\mathcal{M}$ is said to be a *model* (in the logical sense) of φ.

The above mentioned techniques for model checking associate interpretations of a logical language with system specifications $spec$. They check whether a proposition φ is valid for the interpretation $\mathcal{M} = \mathit{Mod}([\![spec]\!])$, i.e. whether $\mathit{Mod}([\![spe]\!]) \models \varphi$ holds.

In the *axiomatic* approach the logical language is used not only for expressing properties but also for describing the systems themselves by axioms Γ_{spec}. As opposed to model checking in this case *all* possible models of the axiom system Γ_{spec}. This leads to the central question of symbolic logic: Is each model $\mathcal{M}$ of Γ_{spec}, i.e. $\mathcal{M} \models \psi$ for all $\psi \in \Gamma_{spec}$, also a model of φ, i.e. $\mathcal{M} \models \varphi$? If this is the case φ is said to follow (logically) from Γ_{spec}, written as $\Gamma_{spec} \models \varphi$.

To interpret $\forall x : D.\ g(f(x)) = x$ first of all one has to provide a domain $[\![D]\!]$. The domain $[\![D]\!]$ may consist of data objects like numbers, strings, and sequences of bits. For natural numbers one could chose the semantics of f as $[\![f]\!](n) := n + 1$ and the semantics of g as $[\![g]\!](n) := n \dot{-} 1$, where $\dot{-}$ is the non negative difference. Choosing $[\![D]\!]$ as the residue class modulo $n = p \cdot q$, i.e. as the set $\{0, \ldots, n-1\}$, then f and g could be interpreted as RSA encryption and decryption, respectively. For strings f and g could be interpreted as $compress$ and $uncompress$, respectively. But not only in these concrete cases the function corresponding to f is injective: all models of $\forall x : D.\ g(f(x)) = x$ also are models of $\forall x_1 : D.\ \forall x_2 : D.\ (f(x_1) = f(x_2))$, i.e.

$$\forall x : D.\ g(f(x)) = x \models \forall x_1 : D.\ \forall x_2 : D.\ (f(x_1) = f(x_2) \rightarrow x_1 = x_2)\ .$$

The axiomatic approach admits *abstraction* from certain particuliarities of single models. So $\Gamma_{spec} = \{\forall x : D.\ g(f(x)) = x\}$ can be viewed as an abstract

specification of compress and uncompress functions describing the relevant (functional) behavior for the outside world. The specification is intended to be completely non constructive. *How* the functions will be realized is subject to *design decision* of subsequent development phases. The properties established at the abstract level are preserved provided that the development process takes place within the class of models of Γ_{spec}.

4.4.2 Proof Techniques

The notion of logical consequence as introduced above uses concepts like the class of all models of a (axiomatic) specification Γ_{spec}. On this basis no verification procedures can be defined. One therefore tries to characterize logical consequences on purely *syntactical* grounds by manipulating *proof structures* that are built up from constructs of the underlying logical language. Given axioms Γ and a proposition ψ rules from by a (proof) *calculus* are used to generate a structure that (by definition) represents a proof of ψ from Γ. If such a proof exists one writes $\Gamma \vdash \psi$, i.e. ψ can be *derived* from Γ.

For example, the so called *sequence calculus* uses as proof structures *trees* whose nodes consist of judgments of the form $\tilde{\Gamma} \Rightarrow \tilde{\Delta}$, where $\tilde{\Gamma}$ and $\tilde{\Delta}$ are *Sequences* of formulas. The formulas in $\tilde{\Gamma}$ and $\tilde{\Delta}$ are considered as conjunctions and disjunctions, respectively. The *sequent arrow* has to be read as an implication. For (finite) Γ a proof of ψ is given by a tree with $Seq(\Gamma) \Rightarrow \psi$ as root and leaves $\tilde{\Gamma}_i \rightarrow \tilde{\Delta}_i$ that satisfy $(Set(\tilde{\Gamma}_i) \cap Set(\tilde{\Delta}_i)) \neq \emptyset$. Proof rules define how trees are constructed.

As an example we demonstrate a sequent calculus proof of the injectivity of f under the assumption that g is an inverse of f. The proof in this case is linear, i.e. there are no branching nodes.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a = b \Rightarrow a = b}{g(f(a)) = a, g(f(a)) = b \Rightarrow a = b}\, g(f(a))/a}{g(f(a)) = a, g(f(b)) = b, f(a) = f(b) \Rightarrow a = b}\, f(b)/f(a}{g(f(a)) = a, g(f(b)) = b \Rightarrow (f(a) = f(b) \rightarrow a = b)}\, \rightarrow -E}{\forall x : D.\, g(f(x)) = x, g(f(a)) = a \Rightarrow (f(a) = f(b) \rightarrow a = b)}\, \forall - I(b)}{\forall x : D.\, g(f(x)) = x \Rightarrow (f(a) = f(b) \rightarrow a = b)}\, \forall - I(a)}{\forall x : D.\, g(f(x)) = x \Rightarrow \forall x_2 : D.\, (f(a) = f(x_2) \rightarrow a = x_2)}\, \forall - E(b)}{\forall x : D.\, g(f(x)) = x \Rightarrow \forall x_1 : D. \forall x_2 : D.\, (f(x_1) = f(x_2) \rightarrow x_1 = x_2)}\, \forall - E(a)$$

In a goal oriented way, starting with the proposition at the root and moving up in the tree, there are the following proof steps:

1. In the first two steps the quantifiers on the right side (of $\Rightarrow$) are eliminated by introducing *fresh* variables, sometimes called “parameters”. In words:

"Let a and b be arbitrary but fixed."

2. After that the universal quantifiers on the left hand side are instantiated, the first one by a and the second one by b.

3. In the next step the implication on the right hand side is eliminated by shifting its premise to the left hand side thereby turning it into an additional assumption.

4. In the last two steps equalities on the left hand side are used to replace terms. In the first step $f(b)$ is replaced by $f(a)$ in $g(f(b))$ and after that $g(f(a))$ is replaced by a in $g(f(a))$.

Implicitly we have removed formulas no longer needed.

A calculus is *sound* if $\Gamma \vdash \psi \;\Rightarrow\; \Gamma \models \psi$. In the context of formal program development soundness is the most important meta level issue. Soundness guarantees that properties established by a calculus actually are logical consequences.

A calculus is *complete* if $\Gamma \models \psi \;\Rightarrow\; \Gamma \vdash \psi$. For *first order predicate logic* which was used in the example above complete calculi exist, [73], [76]. However, completeness in this sense does not mean that all (mathematical) "truth" can be established mechanically. By construction of a concrete formula, [74, 75], Gödel has shown that for sufficiently rich axiom systems Γ there exists not even a (constructive) consistent enrichment Γ' such that either $\Gamma' \models \psi$ or $\Gamma' \models \neg\psi$.

For many formalisms used in the area of formal software development there can be no complete calculus. Examples are higher oder logics where quantification is not only over elements of basic domains but also over functions and relations (of arbitrary type). Isabelle and PVS (see above) use formalisms of this kind. Including computational structures in the semantics of certain *modal* extensions of first oder predicate logic, like in the case of *Temporal Logic*, [63, 67], and *Dynamic Logic*, [64], also leads to incompleteness.

Proof procedures organize the generation of proof structures. In most cases this includes *search*. Since in general $\Gamma \vdash \psi$ is not decidable the best one can hope for are systems that actually find a proof in case ψ is derivable but possibly do not terminate otherwise.

In case of the sequent calculus starting with the root one applies rules that *reduce* (proof) goals to lists of subgoals until axioms are reached. If more than one rule is applicable for a given goal, a *choice* has to been made. It might happen that one has to go back on this decision later on by a so called backtracking mechanism. For exhaustive search procedures completeness of the underlying calculus ensures that eventually a proof is found. However, as mentioned above, in general there is no termination criterium that indicates non derivability of the original goal.

Since meta level investigations of calculi and proof procedures are carried out (once and for all) in the (scientific) public the current situation where, with a few exceptions, this work is done “on paper” seems justifiable. As opposed to that for object level proofs the mere request for dependability makes *machine assistance* indispensable. The example presented above is somewhat misleading in this context: Proofs as they occur in real developments consist of hundreds (sometimes thousands) of nodes and thus are far from what humans can carry out manually in all detail.

Implemented proof procedures (“prover”) can be classified into

- decision procedures,
- systems for automated theorem proving, and
- interactive proof systems.

Decision procedures are used for propositional variants of formalismsand special theories,

Automatic theorem provers like SPASS, [65], and SETHEO, [66], automatically carry out a complete proof search for first order formulas based on efficient proof structures. In a first step they generate a *normal form* representation of the given proof obligation.

In the case of interactive systems some guidance concerning proof search has to be provided by the user. They are used for the more expressive formalisms mentioned above and, in particular, for *inductive proofs* that typically require creativity. User interaction in this context is only concerned with the selection of proof steps while the actual (proof) rules remain fixed.

Decision procedures are not necessarily based on calculi of the usual kind. In general, the proof protocol generated by the systems varies much. In the case of model based procedures there are no proofs at all in the sense discussed here. Therefore the request for “formal proofs” should be regarded more as one for “mathematically sound verification procedures”.

4.5 Consistency

Perhaps the most serious problem with the deductive approach is to guarantee *consistency* of axiomatic theories. Consistency means the absence of contradictions: There is no proposition φ such that $\Gamma \vdash \varphi$ *and* $\Gamma \vdash \neg\varphi$ hold at the same time. Thus consistency is a property of the deductive apparatus.

Consistency is a *critical* issue.

- For most deductive systems *all* formulae become derivable in case of inconsistency. If an axiomatic system specification is inconsistent, all proofs based on it therefore become meaningless.

- The problem of consistency concerns individual developments (specifications), but consistency cannot be established within the object level framework given by $(\Gamma, \vdash)$.

ITSEC and CC require a "formal proof" of consistency for the security model, see section 3.4.4. In the following we shall outline object level verification techniques that allow to establish the consistency of axiomatic theories used as part of (fixed) development methods. Like in all other cases these techniques have to be justified by certain meta level results. These guidelines are not concerned with the problem of how to guarantee the correctness of (the meta theory of) methods and their implementation. Possible criteria for methods and tools definitively have to consider the question which parts of the meta theory and its implementation have to be treated formally in the strict sense used here. A comprehensive formal approach will be confronted with the problem of establishing the consistency of the theory that ensures object level consistency.

In the approaches presented here consistency is not formalized (and proved) in a new object level system. Instead the (restricted) *syntactic structure* of certain developments together with *proof obligations* generated from this structure according to fixed rules guarantee consistency due to certain meta level properties.

In most cases one shows *the existence of models*, (satisfiability) for axiomatic specifications Γ_{sec}. This is sufficient provided that the basic calculus is correct. Roughly the following situations can be distinguished:

- The syntactic form of Γ_{spec} already guarantees consistency.

- Γ_{spec} is an extension of Γ_{spec_1}, i.e. $\Gamma_{spec} = \Gamma_{spec_1} \oplus \Gamma_{spec_2}$, where the new part Γ_{spec_2} is of a special form. The consistency of Γ_{spec} follows from the syntactic structure of Γ_{spec_2}, the consistency of Γ_{spec_1}, and proof obligations $\Gamma_{spec_1} \vdash \psi^i_{spec_2}$ generated from Γ_{spec_2}.

- Within a given method for Γ_{spec} a model is constructed by refinement, which in its axiomatic form is entirely based on consistent specifications.

In each case the given method has to grant control over

- the syntactic form of specification units,

- the relationships between these specification units, and

- the proof obligations generated.

Specifications known to be consistent include pure equational theories, sets of Horn formulae, and instances of schemes for freely generated data types, like natural numbers, lists, and trees.

In the area of formal software development *recursive* and *inductive* definitions play a central role. For extensions by recursively or inductively defined concepts schema are used that support the preservation of consistency. In many cases beyond syntactic restrictions problem specific *proof obligations* have to be satisfied. In the case of recursively defined functions these proof obligations are necessary for example to guarantee termination. Mechanisms for consistent extensions can be found in systems like that of Boyer&Moore, INKA, VSE, and, in a more general form, also in Isabelle and PVS.

In all cases considered so far syntactic restrictions were imposed on Γ_{spec}. One could speak of consistency by construction. If, in certain places, unrestricted axiomatic specifications are allowed the uniform methods from above are no longer applicable. In these cases, like in VSE, the consistency of Γ_{spec} can be established by refining the given specification until a level is reached where all imported specification are consistent by construction. To show the correctness of refinements certain proof obligation generated by the system have to be satisfied. Since even refinements following the paradigm of rapid prototyping require some creativity of the user the method of “model construction” is no longer uniform.

Chapter 5

Preparation of Formal Security Models

In view of the requirements that are described in chapter 3.4 the following sections give some instructions and recommendations concerning the preparation of a formal security model and parts of it. Examples that are taken from a published formal security model are used for illustration. This example describes a generic security model for the creation of digital signatures with the help of a smart card according to the German Signaturgesetz [13].

The example together with a generic security target [12] was the basis for a security evaluation[1] of the signature application of corresponding smart cards sponsored by the TeleTrust e. V. It is a rather generic model as it is not based on concrete smartcard specifications. The interface description DIN V 66291-1 [11] was used as the basis for the specification of the model and its realization.

The mentioned formal security model was prepared with the help of the *Verification Support Environment* (VSE) tool which is approved by the German Information Security Agency (BSI). VSE is a well-elaborated tool for the specification and verification of provably correct software. The VSE tool is recommended by the BSI for the development of IT-products according to the highest assurance levels of the ITSEC and CC evaluation criteria. Several projects that involve VSE demonstrate its usability. The examples given in the following sections are described with the help of the VSE specification language (VSE-SL). Since all the examples are self-explanatory we restrict the introduction of the methodology of VSE to some basics.

The VSE tool offers different methods to structure specifications. It supports the complete development process starting with the abstract specification of the desired product properties up to the automatic code generation. Formal develop-

[1] the desired evaluation assurance level was: ITSEC E4.

ments are stored within the VSE system which guarantees a consistent state of the so-called development graph. An integrated deduction unit allows for the verification of proof obligations that arise during the formal development process. The VSE tool is characterized by a mature methodology to deal with distributed systems [23, 26]. Its specification language combines classical abstract datatypes that are based on first-order logic with a variant of the Temporal Logic of Actions (TLA) [38].

5.1 Background of the Modeling Example

We consider the model of a security policy of an operational smartcard for the creation of digital signatures [11, 12, 13] as an illustrating example for the constituent parts of a formal security model. The notion “operational” means that the production process and the enrollment of personal data will not be considered, i.e. the integrated circuit on the smartcard contains the signature application and all personal data of the card owner. We assume that these data are securely handled within the personalization process and that the smartcard itself is handed over in a secure way to the legal owner. In the following we will call such a smartcard “signature card”.

The digital signature is generated with the help of the secret signature key of the card owner. This secret key is stored in a protected area of the memory of the integrated circuit. Access to the card is realized by a chip card reader that can be connected for example to a personal computer. In the following all the devices that can communicate with the smartcard (chip card reader, personal computer, . . .) are called “card terminal”.

The most important guideline of the security policy for smartcards is: “The legal owner is the only authorized user of the signature application”. It should be impossible that a not authorized person is able to generate a digital signature that is associated with the legal card owner, even if this person possesses the signature card. Thus the confidentiality of the secret key that is stored on the signature card must be ensured. The access to this secret key may only be possible with the explicit compliance of the legal card holder.

5.2 Determination of the Degree of Abstraction

The construction of formal security models is lead by several aims that apparently seem to contradict each other. Responsible for this situation is the role of formal methods as the connector between the functional requirements on the one hand and the functional specification on the other hand. The functional requirements

determine *what* the TOE (Target of Evaluation) should do, whereas the functional specification describes *how* the expected security is achieved. The challenge lies in bridging this gap by a strong formal security model.

Appropriate formal models are characterized by a well-balanced description of the formal security policy. The quality of this balance between the security properties and the security features can be measured by the expenditures for:

1. the correspondence proof between the security model and the security policy that is established by the functional requirements,

2. the formal proof of the validity of the security properties based on the security features and

3. the formal proof (if necessary) of the correspondence between the security model and the functional specification.

The main work consists of the determination of the abstraction degree for the security properties on the one hand and of the security features on the other hand. This cannot be done schematically in general. A creative process is needed during which such a construction is usually adapted several times. This aspect of the preparation of formal security models cannot be considered satisfactory within these guidelines. Furthermore, it is the case that the elements of formal security models as described in this chapter have to be presented in a certain order. The order that is chosen in this guideline improves the understanding of the given example, but it should not be taken as a recommendation for a pure top-down creation process.

Besides the quality of the balance between the security properties and the security features there is another aspect that has a wide influence on the construction of formal security models. This is the visibility of threats and the strategy for attacks. This influence appears in a subtle way already in the formal specification language that is used to formulate the model. If the theories and structures that are based on the formal specification of the security policy exclude the existence of unsafe states (wrt. operative threats) then the formal security model cannot cope with its requirements.

Example. *A smartcard with signature application processes different data that serve completely different purposes. Examples for this data is user data, signature keys, signature certificates (pseudo-)random number generation and communication data (commands, messages, etc.). Typically these data are indistinguishable without the respective context that is given by the communication protocol or by the structure of the exchanged data.*

There is no obvious reason not to represent different data by different abstract datatypes, while the context is clear. But as described in 5.3, the security principles require that the private key that is stored on the card is not compromised. Attacks on the confidentiality of the private key, that are based on the permutation of signature keys and random numbers, cannot not be modeled by different datatypes. This is the reason why all relevant data is collected in an abstract and unstructured datatype `information` *in [13].*

Further aspects of the visibility of threats that arise in the context of the determination of the security architecture and in the context of the description of the (modul-) interfaces are picked up in the examples in the following sections. It should be emphasized that well defined formal theories constitute the grounding for meaningful formal security models. Typically, the security properties as well as the security features are formulated on the basis of the same theory (see figure 5.1). Because of the fact that these theories have a wide influence they should be formulated carefully.

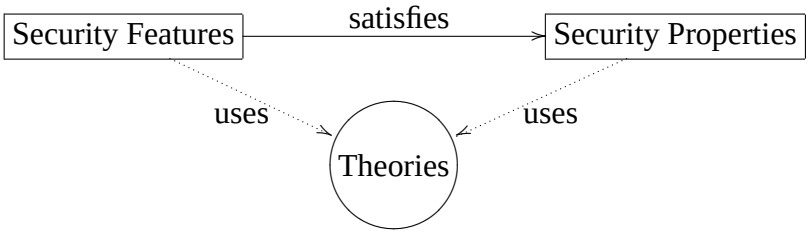


Figure 5.1: Typical basic structure of a formal security model

5.3 Security Properties

As explained in chapter 3.4.2, the security properties represent the part of the formal security model that has to be in accordance with the security principles of the TOE security policy. The functional requirements have a wide influence on the character of the formal security model. They determine the basic elements and the modeling concepts that are used. The design of the formal security properties should reach, whenever possible, a complete coverage of the security principles that are described in the functional requirements.

Example. *A number of functional requirements are derived from the security objectives of the described signature cards. A selection of these requirements is described here using the notation given in [12]:*

The signature card should prevent any kind of extraction of the cardholder's secret key from the ICC. (SO1.1)

The signature card shall prevent any kind of modification of the cardholder's secret key in the ICC. (SO1.2)

The signature card shall allow the use of the digital signature function only to the cardholder after successful authentication by knowledge. (SO2.1)

Successive authentication failures will be interpreted as an attempted unauthorized access and will disable the signature function. (SO2.2)

The authentication data is stored in the signature card and shall not be disclosed. (SO2.3)

The signature card provides a function to generate a SigG digital signature for the data presented by the IFD using the SigG private signature key of the card holder stored in the signature card. (SO7.1)

The function to generate a SigG digital signature works in a way that other individuals not possessing the SigG private signature key of the cardholder cannot generate the signature. (SO7.2)

An analysis of the mentioned requirements results in the following provisions for the development of a formal security model:

- *The prevention of the extraction of the secret signature key (SO1.1) and the disclosure of the authentication data (SO2.3) is realized by the requirements on the design of the interfaces of the signature card (input- and output streams).*
- *The prevention of modifications of the secret key (SO2.1) is modeled by data encapsulation and access control mechanisms.*
- *In order to allow the application of the signature functionality only in case of a successful authentication (SO2.1), there is a modification of access rights modeled that depends on external events.*
- *To allow for a (permanent) lock of the signature functionality (SO2.2) persistent states of the operating system are modeled.*
- *The necessity of the ownership of the signature key in order to generate authentic signatures (SO7.2) is realized in the model with the help of the subjects and their corresponding (mutable) knowledge.*

The requirements analysis that has begun in the example is common for the design of formal security models. It provides the essential notions, elements and structures for the modeling.

After the requirement analysis has been carried out it can be decided whether an already known generic formal security model can be used. Concerning the requirements this is often the case in the area of access- and information flow control. A comparative description of such generic security models is given in chapter 6.

The requirements analysis often has a strong influence as illustrated with the help of an exemplary modeling.

Example. *A careful analysis of the mentioned security requirements concludes that the requirement SO7.2 cannot be influenced by the signature card during its operation. It formulates conditions for the selection and implementation of the cryptographic algorithms and is therefore actually not part of the security policy of the TOE. Nevertheless SO7.2 has a wide influence on the formal security model because of the fact that such requirements concerning the properties of certain algorithms have to be axiomatized conveniently.*

As already mentioned above a necessary prerequisite for such a model is the possibility to specify the subjects and their associated (mutable) knowledge. For this the unstructured datatype `subject` *is introduced. With the help of the function* `learns` *and the predicates* `knows` *and* `inferable` *(as well as some additional help functions and predicates) the theory* `TInformation` *is defined. It allows for the specification of the requirement SO7.2 as an axiom (cp.* `TSignature` *in section 5.5.2):*

```
inferableWithout(i,sig(i,sk),sk)
-> inferable(i,sk)
```

The descriptive interpretation of this axioms expresses that the deducibility of the signature key `sk` *is a necessary prerequisite for the deducibility (generation) of the signed document* `sig(i,sk)`*. This descriptive interpretation corresponds to the semantics of the formal specification if and only if the definition of the used predicates really represent this view. We emphasize on this issue with respect to the theory* `TInformation` *which is partly presented in the sequel.*

```
THEORY TInformation

  PURPOSE
    "The concept of information"

  USING TSubject

  TYPES
    /* Pieces of information (messages):
     * This is the main type of this model.
     * Any information exchanged between the subjects of this
     * model has the type information (or a type derived from
     * type information). This can be documents to be signed,
```

```
     * keys, commands, certificates etc.
     */
    information

  FUNCTIONS
    /* \1 enhances its knowledge with \2 */
    learns: subject, information -> subject;

[...]

  PREDICATES
    /* \1 holds (knows or has in its store) \2 */
    knows : subject, information;

    /* The predicate 'has' is similar to 'knows'.
     * \1 is a place that holds information \2.
     * The difference is the following:
     * While knows is intended to model the fact that a subject
     * can 'compute' or deduce the information from what he has
     * learned, 'has' also is valid if an information can only
     * be obtained by applying some infeasable methods like
     * breaking secure encryptions.
     */
    has: subject, information;

    /* \1 and \2 are the same person/thing, but they may be
     * different in what they know and their internal state but
     * have the same identity. \1 and \2 can be regarded as the
     * same subject watched at different points of time.
     */
    identical: subject, subject;

[...]

    /* Information \1 can be used to infer \2, more precisely:
     * there is a subject that knows \2 after learning \1 even
     * though it did not know \2 before.
     */
    inferable: information, information;

    /* Information \1 can be used to infer \2 even if \3 is unknown,
     * more precisely: there is a subject that knows \2 after
     * learning \1 even though it did not know \2 AND \3 before.
     */
    inferableWithout: information, information, information

[...]

  VARS
    i,j,k: information;
    s,t,u: subject

  AXIOMS

    /* Identity is an equivalence relation */
    identical(s,s);
    identical(s,t) and identical(t,u) -> identical(s,u);
    identical(s,t) -> identical(t,s);

[...]

    /* Learning has no effects on the Identity */
```

```
    identical(s,learns(s,i));

[...]

    /* knows is stronger then has
     * (a subject 'has' what it 'knows').
     */
    knows(s,i) -> has(s,i);

[...]

    /* Definition of inferable */
    /* j is inferable from i, i.e. there is some subject
     * who doesn't has j but can deduce j after learning i
     */
    inferable(i,j) <->
    EX s: NOT has(s,j) AND knows(learns(s,i),j);

    /* Definition of inferableWithout */
    /* j is inferable from i without using k, i.e. there is some subject
     * who neither has j nor k but can deduce j after learning i
     */
    inferableWithout(i,j,k) <->
    EX s: NOT has(s,j) AND NOT has(s,k) AND knows(learns(s,i),j);

[...]

THEORYEND
```

A basis for the interpretation [4, Abs. E[456].3] as well as for the rationale [1, Element ADV_SPM.3.3C] are the modeling concepts as they are identified in the requirements analysis phase. On the highest level of the modeling we first have to specify the security properties. These security properties altogether are regarded as the security principles. In using generic models the security properties result from an appropriate instantiation of the generic properties (see chapter 6).

Example. *The security properties of smartcards are formulated in a temporal logic specification where the interfaces* `channelIn` *and* `channelOut` *belong to the standard interfaces of smartcards.*

The state space of the temporal logic specification consists of specific secrets belonging to signature cards, the interface variables and other internal card data. Especially the private key `skCh` *and the authentication data* `pinCh` *of the card owner belong to this data.*

Because of the technical standards for the communication between terminal and smartcards there is always an unambiguous mapping between the reaction of the signature card and a command which was sent before. Therefore, the formal security properties are in principle based on a simple concept: If the card gives a certain response on a command, then this response is justified by the internal state of the smartcard.

The formulation of the security properties often requires the inclusion of the sequence of transactions that led to the current state. As a consequence of this the

formalization must be able to observe all the relevant parts of the complete history of the current state. The access to this history is technically realized with the help of the state variable `channelOutRawH`*. This variable stores all the output of the smartcard since its initialization.*

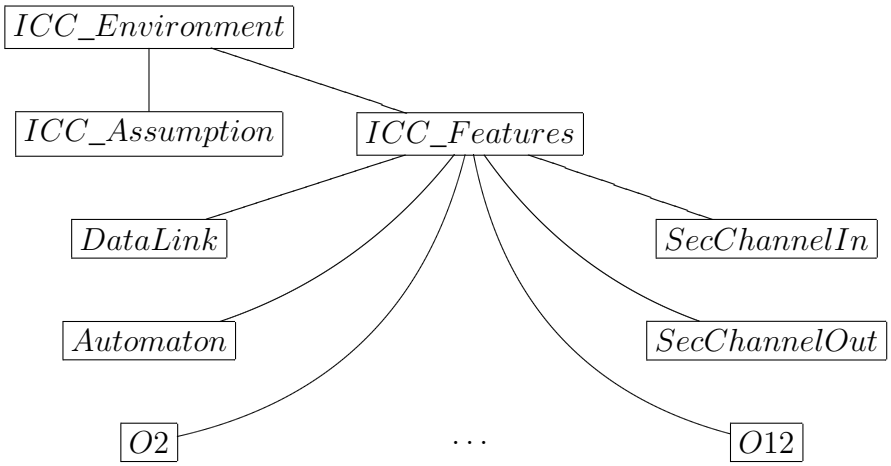
```
/*****************************************************/
/* SECURITY PRINCIPLES                               */
/*****************************************************/

/* SO1.1 */
[] NOT mInferable(channelOut,def(skCh))

/* SO1.2 */
[] skCh = skCh'

/* SO2.1 */
[]  ALL j:
     mInferableWithout(channelOut,
                       def(sig(j,skCh),def(skCh))
     -> histAuthUser(channelOutRawH)

/* SO2.2 */
[]  first(channelOutRawH)
         = value(answerAuthSuccess) AND
    NOT channelOutRawH = nil
    -> NOT maxFailuresExceeded(rest(channelOutRawH))

/* SO2.3*/
[] NOT isSecretInferable(kindPIN,channelOut)

/* SO7.1 */
[]  validpair(isk,ipk) AND
    NOT channelOut' = channelOut AND
    minferable(channelOut',def(sig(j,isk)),def(isk))
    -> isSign(theIfdCommand(channelinDecoded))
```

5.4 Security Features

As described in chapter 3.4.2 the security properties represent that part of a security model that has to correspond to the security features of the TOE-security policy. In order to prove the correspondence between the formal security model

and the functional specification it is recommended to adjust the specification of the security features with respect to the defined procedures and interfaces of the security functions. If a known generic security model (cp. chapter 6) is used, then it must be decided how to instantiate this model in order to accommodate this aspect.

Example. *The security features of signature cards are mainly based on the concept of a state machine (Module* `Automaton`*), which controls the access to data stored on the smartcard that have to be protected. The data itself is encapsulated in self-contained modules (*`02 ...013`*). Further modules are used for the internal communication (*`DataLink`*) between modules and for the input-/output-interfaces (*`SecChannelIn`* and* `SecChannelOut`*). All the modules are then composed to the specification* `ICC_Features` *(the remaining modules are explained in the sequel):*



The formal specification of the security features should contain all information related to the validity of the security properties of the TOE and of the environment. It has to be paid special attention to the visibility of threats which challenge the validity of the security properties. If the security features are based on the concept of state machines as it is the case in our example, then we can characterize the threats in a direct manner with the help of the unsafe states of the corresponding state machine.

Example. *The modul* `Automaton` *specifies a loop for the execution of certain commands:*

```
WHILE true DO BEGIN
   call(bckSecChannelIn);
   call(bckObjects);
```

```
    getEvent;
    stateChange;
    call(bckObjectReset);
    call(bckSecChannelOut)
END
```

In the procedure `stateChange` *the state changes of the state machine are specified. It has to be guaranteed that we can reach only safe state this way. An unsafe state which is not reachable in this model is, for example, the enabling of the signature function without a complete user authentication. The function* `nextstate` *is used for the specification of the procedure* `stateChange`*. It determines the next state with the help of the current state and the current event (which is computed by* `getEvent`*). All allowed state changes are coded in a two dimensional table (matrix) (cp. [12, Tab. 12]).*

Often assumptions about the security environment of the TOE are needed in the proof of the validity of the security properties. These assumptions are explicitly specified in the formal security model. As for confidentiality it is usually the case that data that has to be protected is not known in the environment of the TOE.

Example. *The private signature key is known to the outside of the signature card only if it can be extracted from the output of the signature card. Therefore, it can be assumed that the private signature key is never contained in the input for the signature card. This assumption is essential for the proof of the non-inferability of the private key based on the output of the signature card (SO1.1). Therefore, it is specified explicitly in the modul* `ICC_Assumption`*.*

These restrictions are connected to the input channel of the signature card by the composition of `ICC_Assumption` *and* `ICC_Features` *to the modul* `ICCEnvironment`*.*

We recommend to specify such assumptions on the security environment with great care. If these assumptions are not adequate, then the significance of the proofs of the validity of the security properties is compromised.

5.5 Proofs of Properties and Consistency

The preparation of a formal security model is not yet completed unless the specification of the security properties and the security features together with their corresponding interpretation (ITSEC) and rationale (CC) is finalized. We have already pointed out the necessity to verify the model (cp. section 3.4.4) on different places in the preceding chapters. The proof of the properties of such a model and

the proof of its consistency are necessary for the significance of a formal security model. Only the accuracy of a mathematical proof allows one to deduce the effectiveness of the modeled security policy.

5.5.1 Security Proofs

The proofs of the validity of the security properties represent the characteristic feature of formal security models. Together with the (informal) correspondence proof between the security properties and the security principles and security features, they guarantee the desired quality of the specified security policy of the TOE.

In general it is recommended to undertake the required proofs with the help of a formal development tool that supports the proving process. Especially the generation of the proof obligations that arise from the specification of the security properties should be carried out automatically by such a tool. The Verification Support Environment (VSE) tool meets these requirements.

The concrete proof obligations differ in general. They depend strongly on the chosen specification formalism. Often the concept of mathematical induction is used in such proofs. This holds especially for generic security models as described in chapter 6. The basis for such proofs are recursive datatype definitions in first-order logic specifications and timed state sequences in temporal logic specifications.

Example. *In order to prove that the private key is not inferable from the output of the signature card we have to analyze all possible state sequences. It has to be proved that the validity of the considered security property hold initially and is preserved during state transitions.*

Often first attempts to prove such properties do not succeed since the specification is erroneous. After changing the specification a new proof attempt is started. The correlation between an (abortive) verification and the specification results in the elimination of errors in the security model.

But not only abortive proof attempts give rise to errors in the specification. In particular rather quick proof attempts with short or trivial proof trees point to gaps in the specification.

Example. *One of the security principles of signature cards says that the output of the so-called* `DisplayMessage` *may only occur under certain circumstances. During the first proof attempt the validity of the corresponding security property was established quickly. The reason in this case was that the* `DisplayMessage` *was never displayed so that the security property was vacuously true. It turned out*

that the conditions for the output of `DisplayMessage` *were not sufficient. The reason for the insufficient specified behavior was finally found in an error of the access matrix that refuses the (reading) access to the stored* `DisplayMessage`*.*

Negative proof attempts often follow from errors in the security policy itself. The specification of the security model coincides with the intention of the developer, but the security properties cannot be established using the security features as input. In these cases we can see that the verification of formal models is valuable, since this kind of errors is rather hard to detect using conventional methods.

Example. *Initially the proof of the security properties that corresponds to SO2.2 could not be found. A careful analysis of the proof attempt revealed the fact that the locking of the signature card that has to happen after several unsuccessful authentication attempts was not effective. This locking mechanism did not behave as it was intended in the security policy. An attacker could perform an arbitrary number of authentication attempts. After it has been clarified why the proof attempt failed, this serious error, that remained undetected for a long time, was corrected.*

5.5.2 Consistency Proof

The notion of consistence and the difficulty in proving it are already mentioned in section 4. The explanations that are given there will be analyzed in detail in this chapter and we will give some illustrating examples.

The notion of consistency

Specifications are independent from their representation. They may be given in a textual representation as well as in a graphical representation (as in VSE). They are represented by *syntactical* objects. *Semantically* they describe certain mathematical structures (models). Using an axiomatic methodology it is often intended that several different structures represent possible models. Here we give attention to the question whether such a syntactical description is sensible, i.e. whether there is at least one model. But the pure existence of such a model does not mean that it covers the intention of the person that has prepared the specification.

First a specification has to be *syntactically admissible*, i.e. it has to obey certain (syntactical) requirements. These required (syntactical) properties are always decidable, i.e. they can be checked with the help of automatic tools. We distinguish between the context free analysis of inputs and a so called typecheck. In the first case it is checked for example whether every left parenthesis is related to a corresponding right parenthesis whereas in the second case the typecheck does

also check the correctness of the number of the arguments as well as the type of the arguments of a function call. The possibility of a semantical interpretation and a useful analysis is sometimes connected to the syntactical admissibility. This is especially the case for description techniques that are restricted to executable systems. A good example for such description techniques are programming languages. A syntactically correct Java program is executable and therefore it has a semantical interpretation. This does not mean that such a program is *useful* in any sense: It can be the case that the execution of the program leads to runtime errors which stop this execution. But since the description of the program itself is useful (it can be given a meaning) desired and undesired properties of the program behavior (runtime) can be *analyzed* and detected. It is possible to translate such programs into a logical form. The resulting set of formulas would be consistent.

Concerning logical formalisms the situation is more difficult if the system has an axiomatic description. It cannot syntactically be examined in general whether a specification is a useful basis for the proofs of the intended properties. Even if all the formulas ψ of a set of formulas Γ are well-formed, i.e. they are admissible syntactical objects, there is the possibility that we can conclude $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$. This means that φ as well as $\neg\varphi$ are *deducible* form Γ. We then call this set of formulas Γ *contradictory* or *inconsistent*. For most of the formalisms this means that *all* propositions ϕ are deducible. Therefore, it makes no sense to analyze an inconsistent set of formulas by deducing propositions. Furthermore, it is not possible to give such an inconsistent set of formulas Γ a mathematical structure (model) as interpretation.

A set of formulas that is *not* contradictory (inconsistent) is *consistent*. Note that consistency is a notion that is based on a calculus: A contradiction cannot be *deduced*. The non-existence of models in case of an inconsistent set of formulas follows from the (assumed) *correctness* of the used formalism.

It is in general difficult to proof the consistency of a set of formulas if the corresponding formalism is strong enough as for example first-order logic. It is neither possible for a constructed but arbitrary set of formulas Γ to decide on a syntactical basis whether it is consistent nor can it be characterized in a constructive way by finitely many proof obligations Δ. The following formula would hold for such proof obligations: $\Gamma \models \phi$ for all $\phi \in \Delta \Leftrightarrow \Gamma$ is consistent. In general it is even not possible to generate *sufficient* conditions (proof obligations) which cover the practically occurring cases approximately. In order to *prove* the consistency of a set of formulas with the help of a calculus, we need a more expressive formalism, which then has to be examined for consistency as well.

One way to prove consistency is to reduce a given axiomatic system description (set of formulas) to formulas of a restricted language. The sentences of this language are consistent by definition or the prove of the consistency can be carried out by a syntactical analysis or by proving sufficiency conditions for consistency

that are easy to establish. During this reduction certain proof obligation have to be proved. This approach represents a prototypical implementation or the construction of a model from a logical point of view.

From the existence of a model we can conclude consistency if the underlying formalism is correct: Assume that Γ is inconsistent, i.e. $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$. Since the calculus is correct we have $\Gamma \models \varphi$ and $\Gamma \models \neg\varphi$, i.e. φ as well as $\neg\varphi$ are a consequence of Γ. This means that all structures $\mathcal{A}$ that fulfill (all the formulas of) Γ do also fulfill φ and $\neg\varphi$. If such a structure (model) exists then it would fulfill φ as well as $\neg\varphi$ which is excluded by the notion of satisfiability of formulas ($\mathcal{A} \models \varphi$).

The described approach does not contradict the remarks of the last paragraph. The model preparation process which results in proof obligations requires creativity and it cannot uniformly, i.e. according to certain rules, be executed. We first take a look at specifications for which consistency can be established easily. Afterwards in section 5.5.2 we explain the methodology with the help of some examples.

Consistent specifications

For certain restricted sets of formulas consistency is a priori given. For example *pure systems of equations* (no in-equations) and *horn formulas* (rules and facts) as they are the basis of logic programming. In short the reason for their consistency comes from their axiomatizations since they contain only positive information.

Further typical examples for consistent sets of formulas that are examined in more detail in this work, are the axiom schemata which belong to *freely generated datatypes*. Such a datatype is given by constructor symbols, selector symbols and optionally by some test predicates. We can add uniformly axioms to these syntactical constituents such that there is only one model up to isomorphism in which every element is given by one constructor term. This representation is also unique, i.e. every element of the model is related to *exactly one* constructor term.

The basic datatype NAT (natural numbers) is defined in VSE by the following specification with the name NATBASIC:

```
BASIC NATBASIC
   NAT = NULL   |   SUCC (PRED : NAT)
BASICEND
```

The constructor symbols are NULL (the symbol for the number zero) and SUCC (the symbol for the successor function).

PRED is used as selector symbol (for the predecessor function). In the specification above the character "'|"' is used to designate a *case distinction*. Therefore, the informal reading of the specification is as follows: *Every natural number is*

either zero or it is the successor of a natural number, which itself can be selected by the predecessor function. Formally this is expressed by the following axioms:

```
(x = NULL)  OR  (x = SUCC(y))
NOT (NULL = SUCC(x))
PRED(SUCC(x)) = x
```

The datatype of the natural numbers is a *recursive* one since the argument of the constructor symbol SUCC is again of type NAT. This way infinitely many terms can be constructed and especially terms of the following structure:

```
NULL, SUCC(NULL), SUCC(SUCC(NULL)), ...
```

These variable free terms correspond uniquely to the elements of so called *term generated* models which do always exist for this kind of specification. Furthermore, from a logical point of view they are indistinguishable (isomorphic).

The construction of complex theories as for example the axiomatic set theory in mathematics is carried out inter alia by extending given structures by the *definition* of new concepts. Here certain *principles of definition* are used. In simple cases *equations* or *equivalences* are used, where the defined objects do not occur on the right hand side. This way it is assured that the new objects are *well-defined* (existence and uniqueness) with respect to the objects on the right hand side.

This simple way does not suffice for the considered specifications as for example the natural numbers. *Recursive* equations (or equivalences) are needed. Here the symbol to be defined also occurs on the right hand side of the equation or equivalence. In order to use this more powerful definition principle we have to obey additional conditions to avoid circular definitions and we have to guarantee the existence and the uniqueness of these extensions. In the context of consistency of specifications only the question whether such an extension exists is relevant.

The following axioms describe additional functions and predicates as an extension of the datatype BASICNAT as it is defined above (we consider only addition (_ + _), subtraction (_ - _) and multiplication (_ * _)):

```
THEORY NATURAL
  USING NATBASIC
  FUNCTIONS
    _ + _, _ - _, _ * _, _ DIV _,
    _ MOD _ : NAT, NAT -> NAT
  PREDICATES
    _ < _, _ <= _, _ > _, _ >= _ : NAT, NAT
  VARS
    x, y, z : NAT
```

```
  AXIOMS
    x + NULL = x;
    x + SUCC(y) = SUCC(x + y);
    x - NULL = x;
    x - SUCC(y) = PRED(x - y);
    x * NULL = NULL;
    x * SUCC(y) = (x * y) + x;
    [...]
THEORYEND
```

This specification satisfies two additional conditions:

- For every case (`NULL | SUCC(y)`) of the structure of the second argument there is exactly one equation.
- The second argument of the function on the right hand side (the term "‘`y`"’) is in termini of the above given term structure *less* than the corresponding argument of the left hand side of the equation (the term "‘`SUCC(y)`"’).

If these properties are guaranteed then there is a unique extension of the (term generated) models of the original specification. In particular consistency is preserved this way.

The second property cannot be established in general by a pure syntactical inspection. In fact propositions (proof obligations) are generated whose validity (in term generated models) guarantees the reduction of term complexity. From an operational perspective of the function definition as a (functional) program this corresponds to a *termination proof*. Additionally such an admissible specification provides an appropriate (consistent) *induction principle* which can be used to prove properties of the new defined functions.

The presented method together with an appropriate syntactical support can be used to introduce user defined data structures in a consistent way. This is presented in connection with an example in more detail below.

Description techniques which are based on the explicit declaration of *state transition systems* are somewhat special. Besides the syntactical admissibility of inputs, which is often carried out graphically, there are usually no additional actions needed in order to assure consistency, since a model is given directly. These techniques are referred to as *model based* techniques.

A similar situation is given in case of state oriented axiomatic approaches which are the basis for VSE. Because of a restricted specification technique – one that describes basically the possible state transitions (actions) of a system – the consistency only depends on the initial states that have to satisfy the specification. This result is based on the fact that inconsistent action specifications can never be executed.

Proof of consistency by modeling

If the originally given specification (starting specification) cannot be proven directly to be consistent (by verifying the above given principles), it remains the alternative of a stepwise implementation (refinement). This is done as long as for every branch of the modular development a specification is reached that is guaranteed to be consistent. This process can be regarded as *modeling* since in every such step the correctness has to be proven. All the correctness proofs together with the above discussed construction principles, that are based on the starting specification, constitute the consistency proof.

This methodology is described with the help of an example that is taken from the formal security model of a signature card. The following commented specification (VSE-theory) describes the event of *the generation of a signature*.

```
THEORY TSignature
  PURPOSE
    "Signatures and keys"
  USING TClassification /* includes TInformation */
  FUNCTIONS
    /* The signature function:
       The result is \1 signed with key \2 */
    sig : information, information -> information;
    [...]
    /* The key generation functions:
       The result is the \1-th secret/public key. */
    skeygen: counter -> information;
    pkeygen: counter -> information

  PREDICATES
    /* Valid pair of private and secret key */
    validpair : information, information;
    /* Signature verification: t, iff signature
       is generated with corresponding secret key */
    validSig:   information, information;
    [...]

  VARS
    i,j,k,sk,pk : information;
    s : subject;
    c : counter;
    [...]

  AXIOMS

  [...]

     /* If a piece of information has been signed, the
```

```
      original information still can be inferred from it. */
   validpair(sk,pk) AND knows(s,sig(i,sk)) -> knows(s,i);

   /* If we have a piece of information and a key we
      also know the signed piece of information. */
   knows(s,i) and knows(s,sk) -> knows(s,sig(i,sk));

   /* Signed documents are different if the original
      documents are different */
   sig(i,sk) = sig(j,sk) -> i = j;

   /* Signing a document never generates the secret key */
   validpair(sk,pk) -> sig(i,sk) /= sk;

   /* Signatures must be different from the signed document */
   validpair(sk,pk) -> sig(i,sk) /= i;

   [...]

   /* Signing does not add further information */
   validpair(sk,pk) AND inferable(sig(i,sk),j)
     -> inferable(i,j);

   [...]

   SATISFIES SSignature
THEORYEND
```

The axiomatization uses the theory *Information* which is under-specified but complete. It does not obey a given principle and among others it contains some characteristics of the signature function as for instance injectivity and the possibility to gain knowledge about signed documents. These characteristics are used later to prove the validity of security properties. For example it is assumed that it is impossible to generate signatures accidently and without the knowledge of the corresponding key. The mentioned axioms are not *constructive* as opposed to the above given function definitions. Here we concentrate on *what* the system should do and not *how* it is done. In avoiding unnecessary details the proofs of security properties become drastically simpler. Practically they become possible only by these simplifications.

Otherwise specifications as those mentioned above hold in particular the danger of being inconsistent. Therefore, the demand for consistency proofs in the ITSEC and the CC is fully justified. In cases as considered here such a proof requires creativity since there is no general (uniform) schema that can be used.

In order to give a computational model we first have to concretize the data structure of informations. The following specification introduces a freely generated data type `iinformation` in a special syntax. A case distinction results

in six cases where constructors as `pKey` (public key) and the corresponding selectors as `pKeyId` are available. The externally defined data type `counter` is used. Thus a public key mainly consists of a data object of type `counter`. The last four cases reveal that the used data type is recursive, since the first argument of the constructor term is again of type `iinformation`. After the keyword `WITH` test predicates are introduced which distinguish between the six possible cases.

```
BASIC Sinformation
 USING TConstWord;
       TInformation

 iinformation =

    /* distinguished information */
    dInfo(infoWord : constWord) WITH isDInfo |

    /* public keys */
    pKey(pKeyId : counter) WITH isPKey |

    /* implements sig(i, dInfo(sKey)) */
    sInfo(theSInfo : iinformation) WITH isSInfo |

    /* implements secureMsgEncode(pKey(cNull), dInfo(sKsig), i).
       pKey(cSucc(cNull)) and dInfo(sKdec) are necessary
       to obtain i from this information. */
    scInfo(theSCInfo : iinformation) WITH isSCInfo |

    /* implements certIccKey(information, counter) */
    iKey(iKeyInfo : iinformation, iKeyId : counter)
      WITH isIKey |

    /* implements ifdDoAuthChange(i1,i2) as info pair */
    pInfo(fstInfo : iinformation, sndInfo : iinformation)
      WITH isPInfo
BASICEND
```

Now we can constructively define the signing by a program based on the freely generated and therefore consistent data structure. The program runs its computation on the basis of structures of type `iinformation`. It uses the constructor `dinfo` to guarantee that a secret signature key is given as a second argument of the function. The computation of the output of type `iinformation` uses the constructor `sInfo` which was introduced in the third case above. The remaining kinds of information are only relevant for other contexts that will not be considered here.

```
FUNCTION i_sig
```

```
  PARAMS inf1, inf2 : IN iinformation
  RESULT iinformation
  BODY
    IF inf2 = dInfo(sigSk)
    THEN inf1 := sInfo(inf1)
    FI;
  RETURN inf1
FUNCTIONEND
```

As discussed above programs or (admissible) function definitions *always* preserve the consistency of abstract data types that are used for their computations; as `iinformation` here. But axiomatic provisions are connected with the abstract signature function `sig`, that is implemented by the program `i_sig`. These provisions have to be proved to be properties (proof obligations) of the program `i_sig`. An axiom concerning `sig` is:

```
validpair(sk,pk) -> sig(i,sk) /= sk;
```

The corresponding property to be proved is given in dynamic logic by the following formula:

```
<i_validpair#(inf, inf0, i_validpair-res)>
              i_validpair-res = t
-> NOT <i_sig#(inf1, inf, i_sig-res)>
               i_sig-res = inf
```

In dynamic logic programs occur as constituents of formulas. Let π be a program then the formula $< \pi > \varphi$ says the following: *the program π terminates and afterwards (in the final state) the formula φ holds*. Besides `i_sig` there is another program `i_validpair` used in the proof obligation above, that is also a part of the modeling since it represents the implementation of `validpair`. Thus the reading of the proof obligation is as follows: *If the program `i_valipair` terminates with output `t` (true), then it cannot be the case that the program `i_sig` terminates with the output `i_sig-res = inf`.*

The proof of consistency by modeling is a combination of purely creative steps concerning the invention of implementations on the one hand and the proving of propositions in a calculus that are generated (automatically) by the systems as correctness condition, on the other hand.

Methodology of consistency proofs

While reading the remarks made above the following question could arise: If a modeling (implementation) is needed in every case, why don't we use specifications that are guaranteed to be consistent right from the start? Such an approach

is of course *technically* possible. But there are serious objections against such a method.

First, as already indicated, there is a need for an *abstract* and often not *constructive* level as starting point for the requirements engineering, as the theory `TSignature` in our example. This abstract level should always exist as a specification. It just remains the question of the *order*: Should we construct the abstract level first and prove the consistency hereafter or should these properties be extracted from a complex (computational) model that was constructed step by step?

In the field of the formal program development, there is made a good case for the first (top-down) method. In the context of a formal requirements engineering a *stable* abstract specification should be constructed that needs several iterations in general. Here prevalent application issues should not be mixed with aspects of the realization. In this context it must be pointed out that the drafted model above is only one of a number of possible models that can differ significantly from each other.

Not always but in many cases a model based thinking at an early stage can mislead: One is geared to the contingencies of certain models. In mathematics there was a long abstraction process, that was accompanied by many brilliant researchers, to get the "'right"' abstractions (of for example groups, rings and fields).

The top-down approach fits even better to many process models of software engineering. If in particular a development has to be carried out according to level E6 (ITSEC) respectively EAL7 (CC) the modeling process can partly take place during the required formal refinement to an high level design (ITSEC) respectively to a functional specification (CC). This refinement step is part of the complete development and does not merely aim at the consistency proof.

But it should be noted that one might run into problems if consistency proofs were neglected. If inconsistencies remain undetected for a long time, big parts of the proofs of the security properties must be reestablished since as the case may be these inconsistencies result in changes on basic parts of the formal specification.

Chapter 6

Known Formal Security Models

Security policies that are based on the provision of security levels are called multi-level security policies or *MLS policies* for short. They are used both in systems where humans process information and in computer systems. In MLS policies, the different components of a system and perhaps of the system environment are assigned security levels. Often, a distinction is made between active *subjects* like e.g. processes or users, and passive *objects* like e.g. files. This class of security policies can be used to define requirements with respect to both *confidentiality* and *integrity*.

MLS policies are not restricted to linearly ordered security levels as they are frequently used in the military domain e.g. *unclassified*, *secret*, *top-secret*. It is also possible to use partial orders. The underlying mathematical structure is that of a lattice.[1] Alternatively, MLS policies can be expressed by a relation determining which subjects and objects may be influenced by which other subjects and objects. Such relations are called *interference relations* [18, 19].

In this section a selection of established security models, which formally define security policies from this class, is described. Because all these security models are generic a concrete security policy results from instantiating a model. *To obtain a formal security model in the sense of ITSEC or CC such an instantiation should be formal and the informal interpretation should refer to this instantiated model.* For each of the models that are described here we give the elements of an instantiation. Furthermore, we give the assumptions of these models for the operating environment and known limitations of, and problems with, these models.

In particular, *Goguen and Meseguers's Noninterference*, the *Bell and La Padula model*, and the *Biba model* are described. While noninterference allows for the definition of confidentiality and integrity requirements the Bell and La Padula

[1] A lattice $(L, \bigvee, \bigwedge)$ consists of a domain L and two binary operations $\bigvee$ and $\bigwedge$ that, for every two elements from L, determine the smallest upper bound, the supremum, and the greatest lower bound, the infimum, respectively [24].

model is restricted to confidentiality and the Biba model to integrity properties. Technically, the two latter models can be considered as special cases of noninterference. Their popularity can be explained by the fact that they can be implemented using reference monitors. Due to their fixed concrete mechanism, i.e. that of a reference monitor, they only provide a restricted framework for the abstract definition of security properties. On the other hand, noninterference can be understood as a definition of security. Whereas Bell/La Padula and Biba are based on access control, noninterference is based on the control of information flow.

From the variety of variants of these security models, in the following we describe one variant and briefly discuss alternatives to the respective concrete variant. All models are based on state machines. The underlying machine models are slightly different and are, therefore, introduced at the beginning of each section. Based on this, the respective security model is then described and techniques for simplifying the proof of security are discussed. Furthermore, the elements of a formal instantiation are explained and the assumptions of these models for the operating environment and known limitations of these models are discussed.

6.1 Noninterference

For the definition of a security policy it is necessary to identify the components of a system like e.g. data, processes or interfaces, and of the system's environment like e.g. users or user groups. The notion of a *domain* is used in order to abstract from the concrete components. For instance, a domain can be a single user or a file as well as a group of users and files.

The notion of *interference*, which has been introduced to the security community by Goguen and Meseguer, refers to the interference of one domain by another one. Based on the complementary notion, i.e. that of *noninterference* fixing which domains should not be interfered with by other domains, security policies both for confidentiality and integrity can be defined. A framework in which such security policies can be defined was first suggested by Goguen and Meseguer in [18, 19]. This framework was later modified, extended, and generalized, cf. [41, 43, 30, 39, 34, 35, 46, 40]. The variant that we describe here follows that of [41], an approach which allows for the generalization to non-transitive definitions of interference and, therefore, is not restricted to MLS policies, which are the focus of this chapter.

In order to define a security policy with the approach of noninterference we first need to determine the system actions and the domains, and each action needs to be assigned a domain. Giving a *noninterference relation* $\not\rightsquigarrow$ defines which domains should not be interfered with by others. In case the complement $\rightsquigarrow$ of $\not\rightsquigarrow$ is *transitive* the assignment of domains together with the presentation of the

noninterference relation corresponds to the security levels that are usually given for MLS policies.

The impossibility of an interference is expressed by the statement that for two domains d and d' with $d' \not\rightsquigarrow d$, the domain d cannot distinguish the case in which d' has taken an action from the case in which it has not. A system is secure if there is no unauthorized interference between domains according to $\not\rightsquigarrow$.

Security models based on a noninterference relation enjoy a wide spectrum of possible applications because *such a relation can be used to express requirements concerning both confidentiality and integrity.* In case we have defined $d' \not\rightsquigarrow d$, the domain d is not allowed to observe the actions of d' (confidentiality) and d' is not allowed to influence d (integrity). Therefore, noninterference covers both areas of security for which established and generally usable security models exist.

6.1.1 Definition of the Formal Model

The variant of noninterference described in this section follows Rushby's presentation in [41]. The approach by Goguen and Meseguer [18, 19] is extended to cover security policies that are not MLS policies, i.e. security policies in which the interference relation is not transitive. Moreover, the *unwinding theorem* that is given has less restrictive assumptions when compared to [19]. Unwinding theorems simplify the proofs that are necessary to establish noninterference.

The system is modeled by a state machine. To this end a set S of states, a set A of actions, and a set O of outputs is defined. The allowed system transitions are modeled by a function *step* : $S \times A \rightarrow S$ and the outputs by a function *output* : $S \times A \rightarrow O$. *step*(s, a) defines the state that is reached by the execution of action $a \in A$ in state $s \in S$ and *output*(s, a) defines the output that is generated by the execution of a in s. The execution of a sequence α of actions starting in a state $s_0 \in S$ is modeled by a function *run* : $S \times A^* \rightarrow S$, which is defined recursively by the following two equations.

$$\begin{array}{lcll} \textit{run}(s, \epsilon) & = & s & \text{for the empty sequence } \epsilon \\ \textit{run}(s, a \circ \alpha) & = & \textit{run}(\textit{step}(s, a), \alpha) & \end{array}$$

After giving a set D of security domains, each action is assigned such a domain by a function *dom* : $A \rightarrow D$. A noninterference relation $\not\rightsquigarrow$: $D \times D$ is an irreflexive relation which specifies which domains should not influence others. The associated reflexive interference relation $\rightsquigarrow$ is obtained by taking the complement, i.e. $\rightsquigarrow = (D \times D) \setminus \not\rightsquigarrow$. In case $d' \rightsquigarrow d$, then d may be influenced by d', in case $d' \not\rightsquigarrow d$ it may not. In order to express this formally, a function *purge* : $(A^* \times D) \rightarrow A^*$ is defined which removes all those actions from a sequence of actions that should not influence the given domain.

Definition 1. Let $d \in D$ be a domain and $\alpha \in A^*$ a sequence of actions. Then $\mathit{purge}(\alpha, d)$ is defined recursively by the following equations.

$$\begin{array}{lcll} \mathit{purge}(\epsilon, d) & = & \epsilon & \\ \mathit{purge}(a \circ \alpha, d) & = & a \circ \mathit{purge}(\alpha, d) & \text{in case } \mathit{dom}(a) \rightsquigarrow d \\ \mathit{purge}(a \circ \alpha, d) & = & \mathit{purge}(\alpha, d) & \text{in case } \mathit{dom}(a) \not\rightsquigarrow d \end{array}$$

The characteristics of noninterference are summarized in Table. 6.1.

S	states	
s_0	initial state	$s_0 \in S$
O	outputs	
A	actions	e.g. $\{\mathit{hin}, \mathit{hout}, \mathit{lin}, \mathit{lout}\}$
D	domains	e.g. $\{\mathit{high}, \mathit{low}\}$
dom	domain assignment	$\mathit{dom} : A \to D$
$\not\rightsquigarrow$	noninterference relation	$\not\rightsquigarrow \subseteq D \times D$
output	output function	$\mathit{output} : S \times A \to O$
step	state transition function	$\mathit{step} : S \times A \to S$
run	execution of action sequences	$\mathit{run} : S \times A^* \to S$
purge	purge	$\mathit{purge} : A^* \times D \to A^*$

Table 6.1: Characteristics of Noninterference

Example 2. Let $D = \{\mathit{high}, \mathit{low}\}$, $A = \{\mathit{hin}, \mathit{hout}, \mathit{lin}, \mathit{lout}\}$, $\mathit{dom}(\mathit{hin}) = \mathit{high} = \mathit{dom}(\mathit{hout})$, $\mathit{dom}(\mathit{lin}) = \mathit{low} = \mathit{dom}(\mathit{lout})$, and $\mathit{high} \not\rightsquigarrow \mathit{low}$. Thus, there are four actions that are assigned to two security levels where the low security level should not be influenced by the high security level. For the sequence of actions $\alpha = \langle \mathit{hin}, \mathit{lin}, \mathit{hout}, \mathit{lout} \rangle$ we have

$$\begin{array}{lcl} \mathit{purge}(\alpha, \mathit{low}) & = & \langle \mathit{lin}, \mathit{lout} \rangle \\ \mathit{purge}(\alpha, \mathit{high}) & = & \alpha \ . \end{array}$$

Using the *purge* function a notion of security is defined.

Definition 3 (Security Property of Noninterference). A system with initial state $s_0 \in S$ is *secure* for $\not\rightsquigarrow$: $D \times D$ (according to noninterference) if for all $a \in A$ and all $\alpha \in A^*$ the following equation is satisfied.

$$\mathit{output}(\mathit{run}(s_0, \alpha), a) = \mathit{output}(\mathit{run}(s_0, \mathit{purge}(\alpha, \mathit{dom}(a))), a)$$

Definition 3 formalizes what the notion of security means in the noninterference approach. Intuitively, a system is considered secure if a domain cannot detect whether or not an action of another domain that should not influence the former has been executed. From a technical perspective the output should not change if all actions that should not be observable (according to $\not\rightsquigarrow$) are purged from a sequence of actions.

In order to prove the security of a system according to Definition 3 we need to consider all possible sequences of actions, i.e. infinitely many. *Unwinding theorems* are helpful in order to restrict oneself to separate state transitions. In order to formulate such a theorem several additional definitions are necessary which we introduce now. These definitions form the basis for the unwinding theorem that we give at the end of the section.

Definition 4. A family $\sim = (\overset{u}{\sim})_{u \in D}$ of equivalence relations over S is called a *view-partition* for a set D of domains. $\sim$ is *output consistent* if for all $s, t \in S$ and all $a \in A$ we have

$$s \overset{dom(a)}{\sim} t \quad \Rightarrow \quad \mathit{output}(s, a) = \mathit{output}(t, a) \; .$$

If two states are in relation $\sim$ with respect to an output consistent view-partition, the outputs of all actions from these states are identical.

Example 5. For the choice of an output consistent view-partition there are two extreme cases $\sim_{min} = (\overset{u}{\sim}_{min})_{u \in D}$ and $\sim_{max} = (\overset{u}{\sim}_{max})_{u \in D}$. Two states are in relation $\overset{u}{\sim}_{min}$ if they are equal, i.e. if

$$s \overset{u}{\sim}_{min} t \quad \equiv \quad s = t \; .$$

$\sim_{min}$ is output consistent. Since $\overset{u}{\sim}$ is an equivalence relation $\sim_{min}$ is the minimal view-partition. The maximal output consistent view-partition $\sim_{max}$ is defined as follows:

$$s \overset{u}{\sim}_{max} t \quad \equiv \quad \forall a \in A.\, \mathit{dom}(a) = u \Rightarrow \mathit{output}(s, a) = \mathit{output}(t, a) \; .$$

The choice of the view-partition is of crucial importance for an application of the unwinding theorem. It is not clear in general whether $\sim$ should be chosen as small as possible or as large as possible.

Lemma 6. *Let $\not\rightsquigarrow$ be a non-interference relation, M a system and $\sim$ an output consistent view-partition for M. If for each sequence α of actions and for each domain $u \in D$, $\mathit{run}(s_0, \alpha) \overset{u}{\sim} \mathit{run}(s_0, \mathit{purge}(\alpha, u))$, then M is secure for $\not\rightsquigarrow$ (cf. [41]).*

Definition 7. Let M be a system, let $\sim$ be a view-partition for M, and let $\not\rightsquigarrow$ be a noninterference relation. M *locally respects* $\not\rightsquigarrow$ if for all $a \in A$, all $u \in D$, and all $s \in S$ we have

$$dom(a) \not\rightsquigarrow u \quad \Rightarrow \quad s \overset{u}{\sim} step(s, a) \ .$$

In this definition $\sim$ only occurs on the right hand side of the implication. Thus, the proof that M locally respects $\not\rightsquigarrow$ is simplified by the choice of the maximal possible $\sim$.

Definition 8. Let M be a system, let $\sim$ be a view-partition for M, and let $\not\rightsquigarrow$ be a noninterference relation. M is *step consistent* if for all $a \in A$, all $u \in D$, and all $s, t \in S$ we have

$$s \overset{u}{\sim} t \quad \Rightarrow \quad step(s, a) \overset{u}{\sim} step(t, a) \ .$$

In this definition $\sim$ occurs both on the right and the left hand side of the implication. Thus, it cannot be said in general whether the proof that M is step consistent is simplified by a minimal or maximal possible choice of $\sim$.

The following *unwinding theorem* is based on the definitions 4, 7, and 8.

Theorem 9. *Let M be a system, let $\sim$ be a view-partition for M, and let $\not\rightsquigarrow$ be a noninterference relation. M is secure for $\not\rightsquigarrow$ if*

1. *$\sim$ is output consistent*
2. *M step consistent, and*
3. *M locally respects $\not\rightsquigarrow$*

(cf. [41]).

6.1.2 Formalization of a Security Policy

Noninterference offers a generic framework for formalizing security policies. In the preceding section the set S of states, the initial state s_0, the set O of outputs, the set A of actions, the set D of domains, the domain assignment *dom*, the output function *output*, the step function *step*, and the noninterference relation $\not\rightsquigarrow$ were not further specified. In order to obtain a formal security model according to ITSEC or CC these parameters need to be instantiated and the security of the system needs to be proven for this noninterference relation. *Please note that the statement of the noninterference relation is only one component of this instantiation.* The parameters of noninterference that need to be instantiated are summarized in Table 6.2. Note that the interference relation $\rightsquigarrow$ (the complement of $\not\rightsquigarrow$) should always be transitive for the variant of noninterference that was presented here, i.e. it should specify an MLS policy.

In case one wants to prove the security (with respect to noninterference) using the unwinding theorem (Theorem 9), an output consistent view-partition needs to be chosen. In Example 5, two possible view-partitions are presented.

S	states	
s_0	initial state	$s_0 \in S$
O	outputs	
A	actions	e.g. $\{hin, hout, lin, lout\}$
D	domains	e.g. $\{high, low\}$
dom	domain assignment	*dom* : $A \rightarrow D$
$\not\rightsquigarrow$	noninterference relation	$\not\rightsquigarrow \subseteq D \times D$
output	output function	*output* : $S \times A \rightarrow O$
step	state transition function	*step* : $S \times A \rightarrow S$

Table 6.2: Parameters of Noninterference

6.1.3 Assumptions about the Operation Environment

For an instantiation of noninterference the parameters need to be defined adequately. In particular, the set A needs to include all possible actions, the set S all possible states, the state transition function should adequately model the functioning of the system, and the output function should allow to distinguish different outputs of the system.

6.1.4 Known Limitations of the Security Model

The variant that was presented in the preceding section is restricted to transitive interference relations, i.e. to MLS policies. Furthermore, the underlying machine model assumes a deterministic system. Both limitations are also limitations of the original approach in [18, 19].

The specific restriction of information flow between security domains is the basis for the noninterference approach. A system is considered secure if actions of certain domains cannot be observed. The variant of noninterference that was described is restricted to transitive noninterference relations. The transitivity prevents the indirect flow of information from a domain d_1 to a domain d_2 via a third domain unless the direct flow from d_1 to d_2 is also allowed. For MLS policies this restriction is always respected. For some applications, however, a so-called downgrading is required which cannot be formalized with a transitive interference relation. If e.g. a direct flow of information from d_1 to d_2 is forbidden ($d_1 \not\rightsquigarrow d_2$) but an indirect flow of information via the downgrader d_3 is allowed ($d_1 \rightsquigarrow d_3$ and

$d_3 \rightsquigarrow d_2$) then the interference relation is intransitive. Intransitive interference relations are also needed when cryptographic components are employed so that confidential data can be sent over publicly accessible networks after having been encrypted.

This difficulty is discussed extensively by Rushby in [41]. Furthermore, an approach to handle intransitive interference relations is described. Roscoe and Goldsmith [40] argue that this approach does not guarantee a sufficient level of security. They suggest an alternative that builds on a variant of noninterference described in [39]. This variant differs substantially from the one that has been described here since it is based on failure divergence semantics.

The state machines used in Section 6.1 are deterministic since identical inputs (actions) always yield the same outputs. For a generalization of the noninterference approach for non-deterministic systems many variants have been introduced. The common idea of these approaches is that a system is considered secure if from an actual observation of the system from a domain and the knowledge about the functioning of the system, only a set of possible system traces can be deduced, and this set is too big to deduce confidential information from it. The different approaches differ from each other according to the criterion for this set to be big enough where most of the generalizations of noninterference for non-deterministic systems are *possibilistic*, i.e. they require certain system traces to agree with actual observations but ignore the probabilities for these. Examples for possibilistic approaches are *Nondeducibility* [43], *Restrictiveness* [30], *Noninference* [37], *Generalized Noninference* [34], and *Separability* [34]. Uniform frameworks for the analysis of possibilistic approaches have been introduced in [34, 35, 46, 28]. Unwinding theorems for possibilistic generalizations of noninference have been given in [20, 42, 36, 29]. *Probabilistic approaches* like e.g. [21], additionally take into account the probability of different system behaviors. Possibilistic and probabilistic approaches are contrasted in [32, 33].

Non-deterministic machine models are e.g. needed to describe parallel systems with asynchronous communication. Furthermore, non-deterministic descriptions are traditionally used in early phases of the formal software development and the descriptions are then refined stepwise in later phases. For the development of secure systems the refinement paradox needs to be considered. If a non-deterministic description is proven to be secure, this does not imply that a refinement is also secure. There are two basic reasons for the refinement paradox. First, it is possible that an abstract system specification does not provide the descriptive means to specify some threats. The security of a system with respect to such threats can only be investigated on a more concrete level of description. Second, the security of a system can be violated by the restriction of non-determinism that comes with a refinement. This exclusively occurs in the generalizations of noninference for non-deterministic systems which have been

discussed in the preceding paragraph. The difficulty with the refinement paradox does not occur for the development of safe systems and is, therefore, considered a reason for the complexity of developing secure systems. For a discussion of the refinement paradox cf. [25]. Stepwise refinement of complex systems can also be achieved by a bottom-up approach, i.e. by composition of components. For the development of secure systems it is of course desirable that the security of the composed system already follows the security of the components. Of course, this is only possible for certain preconditions. This problem has been investigated e.g. in [30, 34, 35, 45].

6.2 Security by Access Control

Security models that are based on access control allow the definition of security policies that govern the *direct access* of active subjects, e.g. persons, to passive objects, e.g. files. The goal of such a security policy is to protect the content of objects from unauthorized access without needing to trust in the subjects. Subjects access objects via a reference monitor which checks for each access re-

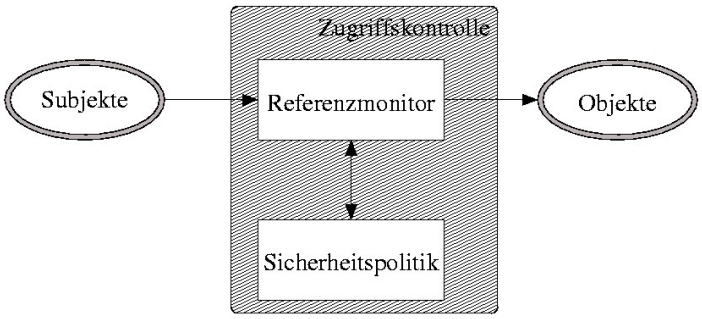


Figure 6.1: Access Control

quest whether it is allowed according to a security policy (cf. Figure 6.1) and only grants authorized access. In order to guarantee a security policy using such an access control scheme, the following prerequisites need to be satisfied.

- *Completeness:* Subjects access objects that should be protected via the access control exclusively.
- *Protection:* The access control is protected against unallowed changes.
- *Correctness:* The access control is implemented correctly, i.e. it guarantees the desired security policy.

Accesses are partitioned into classes according to access modes so that all accesses of one class can be handled uniformly by the access control. In the following, we restrict ourselves to the access modes *read* and *write*. Additional access modes like e.g. *execute* are dealt with by references to the relevant literature where possible.

Within the access control the check whether an access is allowed is usually realized with the aid of an *access control matrix*. An access control matrix includes a row for each subject and a column for each object (cf. Figure 6.2). The entries of the matrix tell which access rights a subject has to an object. A subject is only allowed to read or write an object if the matrix cell in the intersection of the subject's row and the object's column contains the access right *read* or *write*, respectively. The access matrix in Figure 6.2, e.g., allows for read access by s_i to o_j, but no write access. *Access control lists* or *capability lists* are possible technical realizations of an access matrix, cf. [44].

objects subjects	...	o_j	...
. . .		. . .	
s_i	...	read	...
. . .		. . .	

Figure 6.2: Access Control Matrix

Different security models differ in the way how the access control matrix may be initialized, whether or not it may be modified, and if it may be modified which modifications are allowed. There are two fundamental classes of security models based on access control. *Discretionary access control* allows for subjects to delegate their access rights for objects they own to other subjects. Thus, the subjects are responsible for a controlled passing on of rights. Therefore, discretionary access control necessitates trusting the subjects to pass on their rights to other subjects that exercise these rights in a trustworthy way. In computer systems it is difficult to justify this trust. If a subject passes on a right to a program that it starts, the rights that have been passed on can be abused if the program is a Trojan horse (like, e.g., a computer virus). Thus, security models based on a discretionary access control do not offer protection against such attacks, cf. [33]. In contrast, *mandatory access control* does not allow a delegation of access rights. It, therefore, offers a better protection against Trojan horses.

In the following we describe two security models, the Bell/La Padula model and the Biba model, that are based on mandatory access control. The two models are duals, the exact relationship is described in Section 6.3. Because of this relationship similar variants can be built. For the purpose of illustrating the various variants we will describe a variant of Bell/La Padula that allows for dynamic changes of the access control matrix in Section 6.2.1, and a variant of Biba that allows for less changes in Section 6.2.2. With the relationship described in Section 6.3 each variant can easily be adapted to the respective other model.

6.2.1 Bell/La Padula

The model by Bell and La Padula [14, 15, 16], abbreviated by *BLP* in the following, allows for the definition of security policies that govern the access by active subjects like, e.g., processes, to passive objects like, e.g., files. The goal of such a security policy is to protect the content of the objects from forbidden accesses without having to trust the subjects. Starting with [16], several different variants of BLP have been developed, cf. [33]. The variant that we describe here is a simplification relative to the original model and follows [32].

As described above, the access control in BLP is based on reference monitors. Accordingly, the three prerequisites completeness, protection, and correctness need to be guaranteed. The access control matrix can be changed dynamically starting from an initial state in the variant we describe. Which states of the matrix are admissible and which are not is expressed by two properties. The *simple security property* requires that subjects can only read objects which have the same or a lower confidentiality level. Copying data from one object into another one with a lower confidentiality level, i.e. a forbidden downgrading of confidential information, is prevented by the $\star$*-property* (pronounced “star property”). In BLP, a state is only considered secure if both the simple security property and the $\star$-property are satisfied. The combination of both properties guarantees that subjects only get to know the content of objects that are assigned the same or a lower confidentiality level.

In contrast to earlier approaches, e.g. the Lampson Protection Model [27], in which the allowed changes of the access control matrix were defined by a set of rules, in BLP the subjects and objects are assigned confidentiality levels. This approach allows one to prove rules for changing the matrix correct, as is done, e.g., in [15].

Definition of the Formal Model

In the following, we describe a definition of a security model according to Bell and La Padula that is simplified compared to the definition in [16]. Because of this

simplification the access control according to BLP can be formalized by using a single access control matrix. Similar simplifications are made in [32]. Our simplifications are summarized at the end of this section.

Bell and La Padula model the access control system as a deterministic state machine. Such a state machine is defined by a tuple (V, v_0, R, T) where V is a set of states, $v_0 \in V$ an initial state, R a set of possible requests, and $T : (V \times R) \to V$ a state-transition function that transitions the system from one state into another one according to a request. The states of the machine will be described in more detail below.

Subjects and objects are modeled by sets S and O. Subjects can, e.g., be persons or processes. Examples of objects are files or memory areas. A function $F_C : S \cup O \to L$ classifies each subject and each object by assigning them a confidentiality level from a set L. Possible confidentiality levels are, e.g., *unclassified*, *secret*, and *top-secret*. The elements of L are partially ordered by $\leq$[2]and L and $\leq$ form a lattice[3]. The possible access modes are given by the set $A = \{\textit{read}, \textit{write}\}$. The current access rights are fixed in an access matrix $M : S \times O \to P(A)$, where in each cell of the matrix the current set of allowed access modes is entered. The set $\{\textit{write}\}$, e.g., allows for write access by a subject to an object, but not read access.

While the sets S, O, L and A are statically fixed for a concrete security policy, the function F_C and the access control matrix M can be dynamically changed. States of the access control are, therefore, modeled as pairs (F_C, M), such that the set of states for the whole state machine is $V = (S \cup O \to L) \times (S \times O \to P(A))$. The characteristics of BLP are summarized in Table 6.3.

The simple security property is satisfied if subjects can only read objects that have the same or a lower confidentiality level. It, therefore, prevents subjects from a *direct* read access to objects with a higher confidentiality level.

Definition 10. A state satisfies the *simple security property* if and only if

$$\forall s \in S. \forall o \in O. \textit{read} \in M[s, o] \Rightarrow F_C(o) \leq F_C(s) \ .$$

The $\star$-property is satisfied if no subject can read an object o with the confidentiality level $F_C(o)$ and simultaneously write to an object o' with a lower confidentiality level $F_C(o') < F_C(o)$. Otherwise, the subject could copy the contents of the object o with the higher level into an object o' with a lower confidentiality level. This would make the contents directly readable for other subjects even though

[2]I.e. $\leq$ is reflexive ($\forall l \in L. l \leq l$), antisymmetric ($\forall l_1, l_2 \in L. (l_1 \leq l_2 \wedge l_2 \leq l_1) \Rightarrow l_1 = l_2$), and transition ($\forall l_1, l_2, l_3 \in L. (l_1 \leq l_2 \wedge l_2 \leq l_3) \Rightarrow l_1 \leq l_3$).

[3]I.e. for any two elements $l_1, l_2 \in L$ there exists a lower upper bound, the supremum $\bigvee\{l_1, l_2\}$, and a greatest lower bound, the infimum $\bigwedge\{l_1, l_2\}$.

S	subjects	e.g. $\{s_1, s_2, \ldots, s_m\}$
O	objects	e.g. $\{o_1, o_2, \ldots, o_n\}$
L	confidentiality levels	e.g. $\{\textit{unclassified}, \textit{secret}, \ldots\}$
$\leq$	order over L	$\leq \subseteq L \times L$
A	access modes	$A = \{\textit{read}, \textit{write}\}$
F_C	classification	$F_C : S \cup O \rightarrow L$
M	access matrix	$M : S \times O \rightarrow P(A)$
V	states	$V : (S \cup O \rightarrow L) \times (S \times O \rightarrow P(A))$
v_0	initial state	$v_0 \in V$
R	requests	
T	state-transition function	$T : V \times R \rightarrow V$

Table 6.3: Characteristics of BLP

they have a too low confidentiality level. Because in BLP subjects are not trusted generally in order to counter insider attacks and Trojan horses, the intended notion of security can only be reached by a combination of the simple security and the $\star$-property. The $\star$-property thus prevents subjects from indirect read access to the contents of objects with a higher confidentiality level.

Definition 11. A state (F_C, M) satisfies the *$\star$-property* if and only if

$$\forall s \in S. \forall o, o' \in O. (\textit{read} \in M[s, o] \wedge \textit{write} \in M[s, o']) \Rightarrow F_C(o) \leq F_C(o') \ .$$

Remark 12. The following variant of the $\star$-property, which in general is more restrictive than Definition 11, can often be found in the literature. If subjects have a local memory, however, which is assigned the same confidentiality level as the subject itself, then the two variants are equivalent.

$$\forall s \in S. \forall o' \in O. \textit{write} \in M[s, o'] \Rightarrow F_C(s) \leq F_C(o')$$

Because of the modelling as a state machine, an inductive definition of security is possible.

Definition 13 (Security properties of BLP). A state $v = (F_C, M)$ is *secure* (according to BLP), if v satisfies both the simple security property and the $\star$-property. A system (V, v_0, R, T) is *secure* if the initial state v_0 is secure and all states that are reachable from v_0 via the state-transition function T are secure, too.

The following theorem suggests a way in which security according to BLP can be proved by induction. One shows that the initial state is secure and that the state-transition function takes secure states again to secure states. With the aid of this theorem, which is known as *basic security theorem*, one can conclude that the system is secure.

Theorem 14. *A system* (V, v_0, R, T) *is* secure *if and only if* v_0 *is a secure state, and for arbitrary states* $v = (F_C, M)$ *that are reachable from* v_0*,* $v' = (F'_C, M')$ *with* $\exists r \in R.T(v, r) = v'$*, and all* $s \in S$ *and* $o, o_l, o_h \in O$*, all the following conditions hold:*

- *read* $\in M'[s, o]$ *and read* $\notin M[s, o]$ *imply* $F'_C(o) \leq F'_C(s)$
- *read* $\in M[s, o]$ *and* $F'_C(o) \not\leq F'_C(s)$ *imply read* $\notin M'[s, o]$
- *read* $\in M'[s, o_l]$*, write* $\in M'[s, o_h]$ *and read* $\notin M[s, o_l]$ *or write* $\notin M[s, o_h]$ *imply* $F'_C(o_l) \leq F'_C(o_h)$[4]
- *read* $\in M[s, o_l]$*, write* $\in M[s, o_h]$ *and* $F'_C(o_l) \not\leq F'_C(o_h)$ *imply read* $\notin M'[s, o_l]$ *or write* $\notin M'[s, o_h]$*.*

One should note that Theorem 14 is an instance of the usual induction over the computation of state machines, which can be used as a tool to prove security. It does not, however, provide any additional argument for the claim that the definition of security according to BLP itself is adequate. For a detailed discussion on this aspect, cf. [31, 32].

Remark 15 (Comparison to the original BLP model). The original model of BLP in [16] additionally comprised a *need-to-know* principle, which is modeled by a second access matrix. The two matrices are connected conjunctively, i.e. the reference monitor allows for an access only if it is entered in both matrices. This is guaranteed by an additional security property, called ds-property. This makes the model more complex than the variant we have described. Whether the *need-to-know* principle should be taken into account in a model is certainly dependent on the application. It should be noted that need-to-know in [16] is modeled based on *discretionary access control*. In [16], in addition to the access modes *read* and *write*, also *append* and *execute* are defined. Those accesses that both read and write are classified as *append*, and those that neither read nor write are classified as *execute*.

Formalization of a Security Policy

The model according to Bell and La Padula offers a generic framework in which different security policies for access control can be defined. In the preceding section the set S of subjects, the set O of objects, and the lattice $(L, \leq)$ of security levels were not further specified. Also, the concrete definition of the state machine's initial state, the set R of requests, and the state-transition function T were

[4] One should note that "'*read* $\in M'[s, o_l]$, *write* $\in M'[s, o_h]$, *read* $\notin M[s, o_l]$, and *write* $\notin M[s, o_h]$, imply $F'_C(o_l) \leq F'_C(o_h)$"', as required in [32], is not sufficient in this case.

left open. A formal model of a concrete security policy is obtained by an instantiation of these parameters and a proof that the state machine is secure (according to BLP, i.e. the simple security property and the $\star$-property should be satisfied). The parameters that need to be instantiated are summarized in Table 6.4.

S	subjects	e.g. $\{s_1, s_2, \ldots, s_m\}$
O	objects	e.g. $\{o_1, o_2, \ldots, o_n\}$
L	confidentiality levels	e.g. $\{\textit{unclassified}, \textit{secret}, \ldots\}$
$\leq$	order over L	$\leq \subseteq L \times L$
v_0	initial state	$v_0 \in V$
R	requests	
T	state-transition function	$T : V \times R \rightarrow V$

Table 6.4: Parameters of BLP

In [16], an instantiation is given for the definition of the security policy for the MULTICS operating system. The set of subjects, the set of objects, and the confidentiality levels are defined following those defined in MULTICS. The state-transition function is given by a set of rules, where for each rule it is shown that it maps secure states into secure states, and therefore the assumptions of Theorem 14 are satisfied. The rules abstract from the real MULTICS kernel functions. The correspondence between kernel function and rule is given in [16].

Assumptions about the Operation Environment

Security policies defined with the aid of the BLP model require that the access control cannot be circumvented. Furthermore, all subjects and objects that are relevant for the access control need to be modeled in S and O. In particular, local memory of processes should be modeled by elements of O. If local memory is not modeled, it can be used as an intermediate memory to copy contents of high-level objects into low-level ones. This downgrading, however, compromises the security of the system.

Known Limitations of the Security Model

The BLP model is accepted and is widely used. However, it allows for the definition of security policies that, intuitively, do not provide an adequate definition of security. This difficulty is exemplified by system Z, an example taken from [33]. The initial state of Z is an arbitrary secure state, and the transitions of Z change F_C in a way that assigns the lowest confidentiality level to all subjects and objects if any access is requested. Thus, each access request is successful. System Z satisfies the definition of security according to BLP but it does not protect the objects

from any access. This shows that it is often necessary to restrict the way in which F_C and M can be changed over and above the requirements of the BLP model. A generic framework for this is described in [32]

If changes to F_C are forbidden completely, the confidentiality levels of objects or their contents cannot be lowered but only increased. In practice, this requirement is often too restrictive and incompatible with other system requirements. To circumvent this, most of the time trusted processes are introduced. A *trusted process* may bypass the security policy and thus downgrade objects. Of course, the security policy of the whole system can then only be guaranteed modulo the trustworthiness of such processes. Trustworthiness here means that one is forced to trust the processes. Whether they can in fact be trusted is often an exceedingly difficult question for the concrete application. There is further need for research in this direction.

While the general notion of security and BLP have been used as synonyms, in particular in the 1970s, today BLP is no longer seen as a definition of security but more as a technical realization which can be used to effect security (e.g. in the sense of information flow control).

Variants of Bell/La Padula

Compared with the version of Bell/La Padula we presented, two possibilities for variation can be distinguished. First, the combination of mandatory access control with discretionary access control using an additional access control matrix. Access is only granted if the respective access mode is present in both access matrices. An example is the original variant of Bell/La Padula, which has already been discussed in Remark 15. Second, the malleability of F_C and M with state transitions can be restricted. E.g. it is possible to change the $\star$-property according to Remark 12. The resulting requirement is somewhat more restrictive than the $\star$-property given in Definition 11. In Section 6.2.2, this variant is illustrated for the security model following Biba. In [32], a framework is presented which allows for the definition of different variants of BLP differing in how F_C can be changed dynamically. The spectrum ranges from arbitrary allowed changes of F_C to a static F_C that is not changed at all, via restrictions that only allow certain subjects to change F_C.

6.2.2 Biba

The Biba model [17] allows for the definition of security policies that govern changes by active subjects like, e.g., processes, to passive objects like, e.g., files. The goal of such a security policy is to protect the content of the objects from

forbidden changes without having to trust the subjects. We present the original model but use the terminology introduced in Section 6.2.1.

Access control in Biba is based on reference monitors like Bell/La Padula. Thus, the three prerequisites completeness, protection, and correctness need to be guaranteed. The access control matrix can, in the variant described here, be changed dynamically, but with tighter restrictions than those given in Section 6.2.1. The access matrix is required to satisfy two properties that correspond to the simple security property and the $\star$-property of BLP. The first requires write access of subjects to be restricted to objects that have the same or a lower integrity level. Unallowed upgrading of information is forbidden by the second property. The combination of both properties guarantees that subjects only change the contents of objects that are assigned the same or a lower integrity level.

In contrast to previous approaches like the Lampson Protection Model [27], in which the allowed changes of the access control matrix were defined by a set of rules, in Biba the subjects and objects are assigned confidentiality levels. Like in Section 6.2.1, this approach allows one to prove rules for changing the matrix correct.

Definition of the Formal Model

In the following a definition of the security model according to Biba [16] is described. For uniformity, the same terms and notions are used as in Section 6.2.1. In [16], several security policies are discussed, and the one that is today known as the Biba security policy is called *strict integrity policy* there.

The access control system is modeled as a deterministic state machine. A state machine is defined by a tuple (V, v_0, R, T) where V is a set of states, $v_0 \in V$ an initial state, R a set of possible requests, and $T : (V \times R) \rightarrow V$ a state-transition function that transitions the system from one state into another one according to a request. The states of the machine will be described in more detail below.

Subjects and objects are modeled by sets S and O. Subjects can, e.g., be persons or processes. Examples of objects are files or memory areas. A function $F_I : S \cup O \rightarrow I$ classifies each subject and each object by assigning them an integrity level from a set I. Possible integrity levels are, e.g., *low* and *high*. The elements of I are partially ordered by $\leq$[5]and I and $\leq$ form a lattice[6]. The possible access modes are given by the set $A = \{\textit{read}, \textit{write}\}$. The current access rights are fixed in an access matrix $M : S \times O \rightarrow P(A)$, where in each cell of the matrix

[5]I.e. $\leq$ is reflexive ($\forall i \in I.i \leq i$), antisymmetric ($\forall i_1, i_2 \in I.(i_1 \leq i_2 \wedge i_2 \leq i_1) \Rightarrow i_1 = i_2$), and transition ($\forall i_1, i_2, i_3 \in I.(i_1 \leq i_2 \wedge i_2 \leq i_3) \Rightarrow i_1 \leq i_3$).

[6]I.e. for any two elements $i_1, i_2 \in I$ there exists a lower upper bound, the supremum $\bigvee\{i_1, i_2\}$, and a greatest lower bound, the infimum $\bigwedge\{i_1, i_2\}$.

the current set of allowed access modes is entered. The set $\{write\}$, e.g., allows for write access by a subject to an object, but not for read access.

While the sets S, O, L and A are statically fixed for a concrete security policy, the function F_I and the access control matrix M can be dynamically changed. States of the access control are, therefore, modeled as pairs (F_I, M), fixing the set of states for the whole state machine. The characteristics of BLP are summarized in Table 6.5.

S	subjects	e.g. $\{s_1, s_2, \ldots, s_m\}$
O	objects	e.g. $\{o_1, o_2, \ldots, o_n\}$
I	integrity levels	e.g. $\{low, high, \ldots\}$
$\leq$	order over I	$\leq \subseteq I \times I$
A	access modes	$A = \{read, write\}$
F_C	classification	$F_I : S \cup O \rightarrow L$
M	access matrix	$M : S \times O \rightarrow P(A)$
V	states	$V : (S \cup O \rightarrow I) \times (S \times O \rightarrow P(A))$
v_0	initial state	$v_0 \in V$
R	requests	
T	state-transition function	$T : V \times R \rightarrow V$

Table 6.5: Characteristics of Biba

Because of the state machine mode, an inductive definition of security is possible.

Definition 16 (Security properties of Biba). A state $v = (F_I, M)$ is *secure* (according to Biba), if v satisfies both of the following conditions:

1. $\forall s \in S. \forall o \in O. write \in M[s, o] \Rightarrow F_I(o) \leq F_I(s)$
2. $\forall s \in S. \forall o \in O. read \in M[s, o] \Rightarrow F_I(s) \leq F_I(o)$

A system (V, v_0, R, T) is *secure* if the initial state v_0 is secure and all states that are reachable from v_0 via the state-transition function T are secure, too.

The first condition corresponds to the simple security property (cf. Definition 10) and the second to the $\star$-property (cf. Remark 12) in BLP. We, therefore, call these conditions simply *dual simple security property* and *dual $\star$-property*.

The dual $\star$-property given here is logically dual to the $\star$-property given in Remark 12, but not dual to that given in Definition 11. Because the property that is logically dual to Definition 11 can easily be constructed, we have presented another variant to illustrate the variety of variants. In the dual given here, whether a *write* entry is allowed in the access control matrix only depends on the integrity

levels but not on other entries in the access control matrix (in contrast to Definition 11).

Remark 17 (Comparison to the original Biba model). The original Biba model [17] comprises, in addition to the access modes *read* and *write*, the dynamic creation of new subjects *invoke*. With this extension, the definition of a secure system (Definition 16) additionally needs to require that a newly created subject has an integrity level that is no higher than the level of the subject that created it.

Formalization of a Security Policy

The Biba model offers a generic framework in which different security policies for access control can be defined. In the preceding section, the set S of subjects, the set O of objects, and the lattice $(I, \leq)$ of integrity levels were not further specified. Also, the concrete definition of the state machine's initial state, the set R of requests, and the state-transition function T were left open. A formal model of a concrete security policy is obtained by an instantiation of these parameters and a proof that the state machine is secure (according to Biba, i.e. the conditions from Definition 16 should be satisfied). The parameters that need to be instantiated are summarized in Table 6.6.

S	subjects	e.g. $\{s_1, s_2, \ldots, s_m\}$
O	objects	e.g. $\{o_1, o_2, \ldots, o_n\}$
I	integrity levels	e.g. $\{\mathit{low}, \mathit{high}, \ldots\}$
$\leq$	order over I	$\leq \subseteq I \times I$
v_0	initial state	$v_0 \in V$
R	requests	
T	state-transition function	$T : V \times R \to V$

Table 6.6: Parameters of Biba

A subject's integrity classification is often the same as its confidentiality classification. Usually, there is no difference between the reliability of subjects with respect to confidentiality and integrity. A subject that keeps confidential data secret will in general not sabotage a system either. An object's integrity classification can be very different, however. While confidentiality is supposed to prevent undesirable leakage of confidential information, integrity prevents sabotage of information. Integrity levels in I can correspond to the confidentiality levels in L, or they can be chosen differently.

Assumptions about the Operation Environment

Security policies defined with the aid of the Biba model require that the access control cannot be circumvented. Furthermore, all subjects and objects that are relevant for the access control need to be modeled in S and O. In particular, local memory of processes should be modeled by elements of O. If local memory is not modeled, it can be used as an intermediate memory to copy contents of low-integrity objects into high-integrity ones. This downgrading of integrity levels, however, compromises the security of the system.

Known Limitations of the Security Model

Because of the strong conceptual analogy between the Bell/La Padula and Biba security models (cf. Section 6.3.1), the same limitations apply for the BLP and the Biba model. We therefore refer to Section 6.2.1.

Variants of Biba

Using the close correspondence between the Bell/La Padula and the Biba security models (cf. Section 6.3), the variants that have been developed for BLP can easily be adjusted to Biba. Similarly, variants of Biba can be adjusted to BLP. Therefore, we refer to the description of BLP variants in Section 6.2.1.

6.3 Relationship between the Models

There is a close correspondence between MLS policies and transitive interference relation. Each MLS policy corresponds to a transitive interference relation and vice versa [41]. In a way, this correspondence is the basis for our representation of known security models, with which MLS policies can be represented formally. Obviously, it is possible to formalize those policies with BLP and Biba. In these models, the subjects are directly assigned to security levels (or integrity levels, respectively). A formalization of MLS policies with the noninterference approach becomes possible because of the correspondence addressed above.

There is a question about the relationship between the described security models. BLP and Biba are dual security models. This fact provides a mutual transfer of concepts and theoretic results. Compared to noninterference, BLP, and Biba can be considered as implementations.

6.3.1 Correspondence between Bell/La Padula and Biba

The security models according to Bell/La Padula and Biba are dual. This is clarified in the relationship between the simple security property or the $\star$-property in BLP and the corresponding security properties in Biba.

While the simple security property in BLP prohibits that a subject can get read access to an object with a higher confidential level (it prevents direct loss of confidentiality), the dual property in Biba prohibits that a subject can get write access to an object with a higher integrity level (it prevents direct loss of integrity).

While the $\star$-property in BLP prohibits that a subject can get write access to an object if it has read access to an object with a higher security level at the same time (it prevents indirect loss of confidentiality by copying), the dual property in Biba prohibits that a subject can get write access to an object with an integrity level if it has read access to an object with a lower integrity level (it prevents indirect loss of integrity by copying)[7].

By means of this correspondence, it is easy to transfer one variant of a security model to the other model. In the same way it is possible to transfer theoretic results, thus e.g. a *Basic Security Theorem* is also valid for Biba.

6.3.2 Access Control and Noninterference

The Bell/La Padula security model can be considered as a specialization of noninterference [41]. According to the duality described above between the security models, it is also possible to consider the Biba model as a specialization of noninterference. Hence the BLP and Biba models are called *implementations* of noninterference.

The basis for this is the close relation between MLS policies and transitive interference relations. For BLP the interference relation is obtained by the fixing

$$d_1 \rightsquigarrow d_2 \text{ iff } F_C(d_1) \leq F_C(d_2),$$

where d_1 and d_2 are arbitrary security domains in one case (in $d_1 \rightsquigarrow d_2$) and the corresponding subjects or objects in the other case (in $F_C(d_1) \leq F_C(d_2)$). Corresponding statements are valid for Biba, where the interference relation, however, is admittedly defined by $d_1 \rightsquigarrow d_2$ iff $F_I(d_2) \leq F_I(d_1)$.

A technical problem in this definition arises from the different use of the noninterference relation $\not\rightsquigarrow$ in Section 6.1 and the classifications F_C and F_I in Section 6.2.1 and 6.2.2. While F_C and F_I are state components, and thus can be modified dynamically, the noninterference relation $\not\rightsquigarrow$ cannot be modified dynamically. According to the correspondence given above between $\rightsquigarrow$ and F_C, the proof

[7] The reasons for the difference to the corresponding property in Section 6.2.2 are the different variants.

in [41] (corollary 1) that BLP is a specialization of noninterference assumes that F_C cannot be modified dynamically. To prove in general that BLP is a specialization of noninterference (i.e. with dynamic modification of F_C) it is necessary to construct a variant of noninterference that allows a dynamic modification of the noninterference relation.

6.4 Usability of Classical Models

Noninterference and the Bell/La Padula and Biba models are generic security models with which confidentiality and integrity requirements can be defined (see Table 6.7). Each of these generic models represents a class of security models and a concrete model is obtained by initialization (cf. Section 6.5).

	Confidentiality	Integrity
BLP	$\surd$	
Biba		$\surd$
noninterference	$\surd$	$\surd$

Table 6.7: Usability of BLP, Biba and noninterference

While BLP and Biba are models for access control policies, noninterference is a model for information flow control policies. Security models that are based on access control access to objects that contain the data that should be protected. Thus, the confidentiality (and the integrity, respectively) of data is not achieved directly by access restriction to the data but by access restriction to the information container. In general, using such access controls, it is not possible to exclude the existence of covert channels. Timing channels result e.g., if a Trojan horse delays the transmission of one message without confidential content subject to secret data. An attacker is now able to reconstruct those secret data, to which he otherwise does not have access (because of the access control), from this delay. Covert channels cannot only arise by time-dependent behavior but, e.g., a differung intensity of the usage of shared resources like memory requirements. To exclude the possibility of covert channels, it is not sufficient to restrict the access to the information containers. In fact, the access to the data contained therein must be restricted. This can be achived by using information flow control policies which can be formally modeled by noninterference. Thus, the existence of covert channels can be excluded by using noninterference.

The CC stress that policies based on access or information flow control can be modeled formally given the state of the art. Formal modeling for these policies is demanded explicitly:

> [...] At the very least, access control and information flow control policies are required to be modeled (if they are part of the TSP) since they are within the state of the art. [...] [1, part 3, para. 367] (also compare Section 3.3.2)

6.5 Instantiation of Classical Models

The security models that described above noninterference, BLP, and Biba, are all generic. Thus, there is a wide application range for each of these approaches.

So the construction of a security model can be carried out in two steps. In the first step, one of the generic security models e.g. noninterference, BLP or Biba is chosen and in the second step, all model parameters are instantiated.

It must be pointed out that a concrete security model does not arise until the parameter instantiation is done. A statement like, e.g., "'We use the Bell/La Padula model."' only expresses the decision for one specific class of security models. Given that this class is extremely huge, the intended parameter instantiation will in general not be obvious, rather it must be indicated explicitly. Certainly, in case of a formal security model (as demanded by ITSEC or CC), the instantiation must be indicated formally. An informal instantiation description cannot adequately replace a formal instantiation.

As support for such a formal instantiation, all parameters of the different security models are given in the particular sections. Furthermore, the meaning of each parameter is explained. Refere to the respective section "'Formalization of a Security Policy"'.

Chapter 7

Evaluation of Formal Security Policy Models

Since (formal) security policy models shall create additional assurance that the security functions contained in the functional specification enforce the TOE security policy (cf. [1, Part 3, Par. 365]), their evaluation is of particular importance. The goal of the evaluation of formal security policy models is in the confirmation of the required correspondence between the functional specification and the security policies contained in the functional requirements.

The required evaluator activities essentially are related to three areas:

- The correspondence between the functional requirements and the security principles (Sicherheitsprinzipien)
- The correspondence between the security principles (Sicherheitsprinzipien) and the security characteristics (Sicherheitscharakteristika)
- The correspondence between the security characteristics (Sicherheitscharakteristika) and the functional specification

In the following sections these key aspects will be explained. Appendix A contains a summary in form of a sub-activity adapted to the presentation style of [3]. The explanations in this chapter serve as justification and additional remarks of the Work Units proposed there.

7.1 Security Policy vs. Security Policy Model

The correspondece with the TOE security policy formulated in the functional requirements provides the link between the security target (ST) and the formal security policy model. This area demands for the highest requirements on evalu-

ator competence since a complex transformation between informally described requirements and formally specified security properties and features is to be assessed:

1. Where appropriate, the evaluator has to be able to decide if the security policies that are not formally specified could actually not be formally modeled according to the state of the art (cf. Work Units ADV_SPM-3 and ADV_SPM-4).

2. The evaluator has to be able to comprehend the meaning of the formal specification of the secuirty properties and features against the background of the security target and independent of the possibly erroneous or incomplete explanations of the developer (cf. Work Unit ADV_SPM-2).

3. The evaluator has to be able to systematically analyse the security properties und features with the goal to confirm that all security objectives (e. g. as secure states) as well as all threats (e. g. as insecure states) are clearly distinguishable in the model (vgl. Work Unit ADV_SPM-5). This also includes the assessment of the remaining security environment (assumptions and organisational security policies) for appropriate consideration.

4. In consideratoin of the rationale/interpretation of the developer the evaluator has to systematically search for gaps in the formal specification of the security properties and features with the goal to confirm the complete coverage of the security principles and characteristics formulated in the functional security requirements (cf. Work Units ADV_SPM-3, ADV_SPM-4 and ADV_SPM-9).

5. In consideration of the rationale/interpretation of the developer the evaluator has to systematically search for deviations from the security principles and characteristics formulated in the functional security requirements with the goal to confirm the consistency with the security properties and features (cf. Work Units ADV_SPM-3, ADV_SPM-4 and ADV_SPM-8).

7.2 Security Properties vs. Features

The correspondence between the security properties (Sicherheitseigenschaften) and the security features (Sicherheitsmerkmale) connects both parts of the formal security policy model via a formal proof. This proof forms the central link between the requirements specification and the design specification. Here, the validity of the proofs of properties and consistency are to be assessed.

1. The evaluator has to systematically analyse the formal security policy model with the goal to confirm that all attacking possibilites are clearly distinguishable in the model (cf. Work Unit ADV_SPM-5). During this task are e. g. the choice of data structure or the specification of communication channels to be considered.

2. The evaluator has to assess the formal proof of correspondence between security properties and security features in view of suitability and completeness of the proof obligations as well as in view of the validity of the arguments used (cf. Work Unit ADV_SPM-6). All partial proofs must be completely processed. For this the developer has to provide evidence and the evaluator has to assess at least a random sample of the evidence.

3. The evaluator has to confirm the internal consistency of the formal security policy model. For this the validity of the arguments user has to be assessed (cf. Work Unit ADV_SPM-7).

7.3 Security Policy Model vs. Security Functions

The correspondence with the functional specification of the TOE establishes the connection between the formal security policy model and the highest level of the presentation of the implementation. It arranges that the development erects at the basis of a description of the TOEs functionality that is validated against the requirements. Hier ist der fehlerfreie Übergang einer häufig von komplexen Zusammenhängen geprägten Darstellung der Sicherheitseigenschaften und -merkmale in eine nach funktionalen Gesichtspunkten gegliederte Spezifikation zu bewerten:

1. In consideration of the correspondence demonstration of the developer the evaluator has to systematically seach for gaps in the functional specification with the goal to confirm the complete coverage of the security properties and features (vgl. Work Units ADV_SPM-10, ADV_SPM-12 und ADV_SPM-13). In particular when the functional specification is not formally presented the validity of the arguments used has to be assessed.

2. In consideration of the correspondence demonstration of the developer the evaluator has to systematically seach for differences with respect to the security properties und features with the goal to confirm the consistency with the functional specification (cf. Work Units ADV_SPM-11, ADV_SPM-12 and ADV_SPM-13). In particular when the functional specification is not formally presented the validity of the arguments used has to be assessed.

Chapter 8

How IT Manufacturers Benefit from Formal Security Models

Main objective of using formal methods in the generation of security models is to achieve a higher degree of *confidentiality* in the security of a system or product. Definite quality improvement essentialy results from

- agreements that are unambiguous and therefore free of subjective interpretations,
- pairwise consistency of the concepts, and
- validity of the postulated interrelations.

The utilization of mathematically established *description techniques* admit computer supported analysis methods. This corresponds to approaches known from other engineering disciplines where properties are to be predicted with a high extent of confidentiality.

In a formal setting the *meaning* of the concepts as well as the *derivability* are fixed with mathematical *absoluteness*.[1] This therefore does not leave any interpretational scope in questions like “What is the meaning of this statement?” and “What can we derive from these definitions?”. On this basis the consistency of the basic concepts as well as the validity of initially assumed (or demanded) relationships (security properties) can be proved.

As mentioned in chapter 3 ITSEC and CC require formal methods already in the early phases of a development. This corresponds to the commonly shared position that the *most crucial* errors creep in during the requirements engineering phase. Here, requirements and solutions should be considered on an *abstract*

[1] Remaining vagueness results from more fundamental problems within the foundations of mathematics.

level, i. e. independent of detailled technical realizations. In particular, as soon as (mathematically) precise abstractions are needed, there is no alternative to the utilization of formal methods.

Whenever formal methods are applied in *critical* parts of a broad development a noticeable gain in security, precision (of the description), as well as justification (of the statements made) arises. This all leads to a considerable *quality improvement* concerning the result of the development. And the latter is the real motivation for the utilization of methods which, in the first instance, require some additional effort. The evaluation and the validation merely demonstrates the increase in confidentiality.

In the following some light will be shed on some of the aspects mentioned above. We shall start with a short classification of formal methods in view of a development process in general.

8.1 Development Methodology

Formal methods aim at an improvement of the *development methodology*. Depending on the respective development phase different kind of problems arise. In general we consider the following phases: the *requirements definition*, the *conception* of solutions, and the technical *realization*.

Guiding idea for the formation of development methodologies should be *controlling the complexity*. Two basic measures serve this purpose: *structuring* and *abstraction*. Structuring results in possibly independent units and therefore affects essentially the architecture of the development artifact. In contrast, abstraction aims at appropriate (according to the different phases) views on this artifact.

As a result of data collections critical flaws mostly slip in during the early development phases. In the area of security this apllies to the formulation of security properties (requirements definition), the specification of the basic security functions (conception, abstract system model), and the architectural description. But just during these phases clean structuring and in particular appropriate levels of abstraction are crucial. A reasonable analysis is ruled out by non-concise specifications and by mixing up conceptual questions (what) with technical implementation details (how).

Basic prerequisit for a successful requirements engineering therefore are description means that combine abstractness and modular approaches with a semantically established *analysis*. Concepts on the *programming language* level are fairly well understood and could more or less easily be established semantically. However, they are often inappropriate because of their lack of abstraction possibilities. Natural language descriptions neither use a fixed formalism nor is the underlying semantics clearly stated. Therefore they do not offer a suitable starting-point

for further analyses. This holds even if the natural language is artificially restricted to a selective set of notions or, if graphical representations are added that somehow support structuring. Verbal descriptions may possibly suffice if we want to talk to others about the development artifact, although they do not allow for any technical reflection. A common (methodical) mistake, however, is to directly pass over from verbal descriptions to implementation oriented concepts.

Formal description techniques allow for *precise abstractions* that serve as a "missing link" between informal requirements specifications and technical realizations. For so-called semi-formal approaches the same applies at best in a rather restricted way. The following section is concerned with their characterization and scope.

In addition to the mentioned application during the early development phases, formal methods are equally well suited to *correctly* carry over abstract specifications to technical realizations. To this end several different steps have to be taken. In general, the given *criticality* is decisive concerning the choice which area has to be modelled formally and to what depth. Whenever critical design steps arise – be it in form of complex algorithms or data structures – an additional security benefit can be gained from the formal treatment of lower levels. And that even though ITSEC and CC (for levels E6 and EAL7 respectively) only demand a formal treatment of the first refinement step (architectural design).

8.2 Semi-formal Description Techniques

So-called *semi-formal* description techniques differ from purely informal approaches by using exactly defined *formalisms*. In this context commonly used graphical representations like state transition systems can be viewed – at least in principle – as a *formal language*.

Formal languages restrict the expressiveness of natural language and give it some kind of structure. The binding agreement on its syntax somehow predetermines the mental handling of development objects. Assuming appropriate utilization, such patterns of thought already might considerably improve the quality.

Moreover, certain properties – as, for instance, the (syntactical) consistency of interface definitions – can be checked on the basis of a fixed syntax. Presuming appropriate tool support certain design flaws can be recognized (and corrected) already in early phases.

The restriction to *syntactical properties*, i. e. those properties that can be precisely formulated solely in terms of syntactic definitions, is indeed serious. As a rule, security properties will necessarily refer to mathematically defined *semantics*.

The restriction to purely syntactical elements – for instance *signatures* of data types, procedures, functions or methods – is just a special case of restricting the *description level*. Recently, description techniques stepping well beyond pure syntax became popular. They are even semantically specified, or at least can be so quite easily. Nevertheless, they are restricted to certain special *aspects*. As an example consider object diagrams as they are used in object-oriented approaches. There the examined relation properties can be conceptualized in a mathematically precise fashion, for example with the help of the so-called terminological logics. It remains a problem how other "views" – whether concerned with the used data structures, the information flow or the temporal behavior – can possibly be semantically connected with these aspects.

8.3 Formal Methods

Formal methods are characterized by a mathematically precise syntax and *semantics*. Since semantics is added to the syntactic concepts the contribution of structured reasoning to quality improvement is even stronger than in the case of semi-formal methods. As experience shows, conceptual vagueness and even flaws – as for example implicit assumptions and potential exceptions – can easily be "hidden" behind incompletly defined semantics.

Beyond the mathematically disciplined way of thinking, however, formal approaches allow one to completely objectify the reflection on systems. Based on the constituted semantics there are (calculation-) methods that establish the validity of assertions via formally specified systems. In general, this happens through (machine-supported) mathematical proofs. For special problem classes, however, completely automated decision procedures (e. g. model checking) are known. The major advantage of such a formal approach lies in both the possibility to formally examine system specifications and to *guarantee* desired properties. Except for the form of the used mathematical methods this corresponds to the situation in (todays) established engineering disciplines.

All well-established formalisms allow for a modelling depth that admits an abstract and at the same time mathematically precise treatment of the relevant security properties together with their respective functions. Thereby, the axiomatic approach allows for a high abstraction degree by utilizing mathematical structures, non-constructive specifications (what, not how), non-determinism, and by omitting details in a controlled manner.

As already mentioned earlier, it is furthermore possible to formally support the refinement process from the requirement specification down to the implementation by modelling appropriate software engineering solutions. This concerns, for example, algorithms, data structures, and the technical realization of the com-

munication between distributed and concurrent components.

Additional mistakes and flaws might possibly arise during the transition from informal requirements to formal security models as well as during the (non-formal) technical realization. Nevertheless, an adequate utilization of formal techniques results in a considerable gain in quality, more than possibly being reached with other measures. The main concern of this guideline is to bring forward such an appropriate and profitable (as gain in quality improvement) application of formal development methods.

Appendix A

Evaluation of security policy modeling (ADV_SPM)

A.1 Objectives

The objectives of this sub-activity are to determine whether the security policy model clearly and consistently describes the security principles and security characteristics of the security policies in terms of security properties and security features as their formal counterparts, and whether this description corresponds with the description of security functions in the functional specification.

A.2 Application notes

This sub-activity applies to cases where the developer has modeled the security policies of the TOE in an informal, semiformal of formal style.

In element ADV_SPM.[123].2C the components of the TSP are denoted by the terms “rules” and “characteristics”. Throughout this sub-activity, the term “(security) principles” is used instead of “rules”. Both terms are considered synonymous.

A TOE security policy model is a representation of the principles and characteristics of security policies in mathematical terms. Their counterparts are called security properties and security features, respectively. The representation includes but is not limited to . . . algebraic specifications, finite state machines and logic formalisms strong enough to infer the properties from the features. The TSP model is accompanied by an informal interpretation as part of the rationale explaining how the principles and characteristics are mapped to the respective properties and features.

It is recognized that not all policies (see work units for ADV_SPM.[123].2C) can be modeled for all TOEs. This is because either the state of the art is insufficient to model a given policy in a (semi-) formal style, or because the nature of the TOE renders impossible the modeling of policies that would otherwise be possible to model.

Hence the possibility to (semi-) formally model some security policy depends on the very nature of the TOE enforcing it. While access control policies and information flow control policies have both been formally modeled successfully, the possibility of modeling other policies is based on a case by case decision. Abstention from (semi-) formally modeling security policies requires justification and rests the burden of proof entirely on the developer's side.

Those policies that cannot be modeled formally should be modeled semiformally, if possible. If none of the TOE security policies can be formally modeled, ADV_SPM.3 cannot be met.

Those policies that can be modeled neither formally nor semiformally must be modeled informally. If all TOE security policies are modeled in an informal style, ADV_SPM.2 cannot be met.

A.3 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the TOE security policy model (TSP model);

d) the user guidance;

e) the administrator guidance.

A.4 Evaluator Actions

This sub-activity comprises one CC Part 3 evaluator action element:

a) ADV_SPM.[123].1E.

A.4.1 Action ADV_SPM.[123].1E

ADV_SPM.[123].1C ***The TSP model shall be [1: informal; 2: semiformal; 3: formal]***

ADV_SPM-1 The evaluator ***shall examine*** the TOE security policy model to determine that it is presented in the degree of formality (informal, semiformal or formal) actually required by the component level.

The evaluator identifies the formal framework upon which the TOE security policy model is based and ensures that it is founded on well established mathematical concepts. He also identifies the security properties and features addressed in the application notes and ensures the formalization of at least one security policy.

ADV_SPM-2 The evaluator ***shall examine*** the TOE security policy model to determine that it contains all necessary informal explanatory text.

Supporting narrative descriptions are necessary for all parts of the TSP model (for example, to make clear the meaning and use of any notation) including the security principles and security properties.

ADV_SPM.[123].2C ***The TSP model shall describe the rules [i. e. principles; see application notes] and characteristics of all policies of the TSP that can be modeled.***

ADV_SPM-3 The evaluator ***shall check*** the TOE security policy model to determine that all security policies that are explicitly included in the ST are modeled.

The security policy is expressed by the collection of the functional security requirements in the ST. Therefore, to determine the nature of the security policy (and hence what policies must be modeled), the evaluator analyses the ST functional requirements for those policies explicitly called for (e. g. by FDP_ACC and FDP_IFC, if included in the ST).

If the ST contains no explicit policies (because neither FDP_ACC nor FDP_IFC are included in the ST), this work unit is not applicable and is therefore considered to be satisfied.

ADV_SPM-4 The evaluator ***shall examine*** the TOE security policy model to determine that all security policies represented by the security functional requirements claimed in the ST are modeled.

In addition to the explicitly-listed policies (see work unit ADV_SPM-3), the evaluator analyses the ST functional requirements for those policies implied by the other functional security requirement classes. For example, inclusion of FDP requirements (other than FDP_ACC and FDP_IFC) would need a description of the Data Protection policy being enforced; inclusion of any FIA requirements would necessitate that a description of the Identification and Authentication policies be present in the security policy model; inclusion of FAU requirements need

a description of the Audit policies; etc. While the other functional requirement families are not typically associated with what are commonly referred to as security policies, they nevertheless do enforce security policies (e. g. non-repudiation, reference mediation, privacy, etc.) that must be included in the security policy model.

If the ST contains no such implicit policies, this work unit is not applicable and is therefore considered to be satisfied.

The evaluator ***shall examine*** the TOEsecurity properties and security features of the security policy model to determine that the modeled security behaviour of the TOE is clearly articulated. ADV_SPM-5

The TSP model's properties describe the TOE's behaviour in enforcing the principles of the policy. For example, a policy that is modeled on the basis of state transitions would include principles of its states, identify its initial state, and define what it means to be a secure state.

The TSP model's features describe the attributes and conditions of the TOE that come into consideration when enforcing its policy's characteristics. For example, a policy that is modeled on the basis of state transitions would describe the necessary conditions to transform the TOE from one state to the next.

Together the security principles and security characteristics describe the entire security posture of the TOE security policy.

In the context of a TOE security policy model the security behaviour is considered to be clearly articulated only if an adequate mapping from principles and characteristics to their properties and features as their respective counterparts has been given. The mapping is considered to be adequate if the level of abstraction from the TOE's realization is detailed enough to allow for correct identification of all security objectives and the relation to the security environment.

The above condition for clear articulation is necessary but not sufficient. An informal interpretation of all concepts (including attributes, predicates and variables, if available) must be provided in order to make clear their intended meaning.

The TSP model shall include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP that can be modeled. ADV_SPM.[123].3C

The evaluator ***shall examine*** the TOE security policy model rationale to determine that it demonstrates or proves, as appropriate, the correspondance between the security properties and the security features. ADV_SPM-6

The demonstration or proof, as appropriate, shall show that the security features enforce the security properties. To determine the enforcement, the evaluator con-

siders the security properties and the security features and verifies that the arguments used in the demonstration or proof, as appropriate, are valid.

If the security policy model is semiformal, the demonstration of correspondence between the security properties and the security features shall be semiformal.

If the security policy model is formal, the proof of correspondence between the security properties and the security features shall be formal.

ADV_SPM-7 The evaluator ***shall examine*** the TOE security policy model rationale to determine that it demonstrates or proves, as appropriate, the internal consistency of the TSP model.

The demonstration or proof, as appropriate, shall show the absence of contradictions within the security policy model. In determining the absence of contradictions, the evaluator verifies that the arguments used in the demonstration or proof, as appropriate, are valid.

If the security policy model is semiformal, the demonstration of its internal consistency shall be semiformal.

If the security policy model is formal, the proof of its internal consistency shall be formal.

It is recognized that a complete semiformal demonstration resp. formal proof of the internal consistency of the TSP model usually is not possible due to the fundamental nature of (semi-) formal frameworks. Generally, it is sufficient to generate evidence using semiformal demonstrations resp. formal proofs based on the specific TSP model that shows the internal consistency by means of a combination with generic arguments of the (semi-) formal framework.

For guidance on consistency analysis see [CEM, Annex B.3].

ADV_SPM-8 The evaluator ***shall examine*** the TOE security policy model rationale to determine that the behaviour modeled is consistent with respect to principles and characteristics described by the security policies (i. e. as articulated by the functional requirements in the ST).

In determining consistency, the evaluator verifies that the rationale shows that each description of security properties resp. security features in the TSP model accurately reflects the intent of the corresponding security principle resp. security characteristic. For example, if a policy stated that access control was necessary to the granularity of a single individual, then a TSP model describing the security behaviour of a TOE in the context of controlling groups of users would not be consistent. Likewise, if the policy stated that access control for groups of users was necessary, then a security policy model describing the security behaviour of a TOE in the context of controlling individual users would also not be consistent.

For guidance on consistency analysis see [CEM, Annex B.3].

The evaluator ***shall examine*** the TOE security policy model rationale to determine that the behaviour modeled is complete with respect to the principles and characteristics described by the security policy (i. e. as articulated by the functional requirements in the ST). ADV_SPM-9

In determining completeness of this rationale, the evaluator considers the security properties and security features of the TSP model and maps those properties and features to explicit policy statements (i. e. functional requirements). The rationale should show that all security principles and security characteristics of the policies that are required to be modeled have an associated security property resp. security feature description in the TSP model.

Abstention from (semi-) formally modeling policy statements where this is required by the component level always calls for justification on the developer's side (also confer the application notes).

The demonstration of correspondence between the TSP model and the functional specification shall show that all of the security functions in the functional specification are consistent and complete with respect to the TSP model. ADV_SPM.[123].4C

The evaluator ***shall examine*** the functional specification correspondence demonstration of the TOE security policy model to determine that it identifies all security functions described in the functional specification that implement a portion of the policy. ADV_SPM-10

The degree of formality of the correspondence depends on the presentation style of the functional specification and the TSP model. If the functional specification and the TSP model are at least semiformal, the correspondence must be at least semiformal (cf. ADV_SPM-12). If the functional specification and the TSP model are formal, the correspondence must be formal (cf. ADV_SPM-13).

In determining completeness, the evaluator reviews the functional specification, identifies which functions directly support the TSP model and verifies that these functions are present in the functional specification correspondence demonstration of the TSP model.

The evaluator ***shall examine*** the functional specification correspondence demonstration of the TOE security policy model to determine that the descriptions of the functions identified as implementing the TSP model are consistent with the descriptions in the functional specification. ADV_SPM-11

To determine consistency, the evaluator verifies that the functional specification correspondence shows that the functional description in the functional specification of the functions identified as implementing the policy described in the TSP model correspond to the associated security features of the TSP model and therefore enforce the same security properties as the TSP model.

In cases where a security policy is enforced differently for untrusted users and administrators, the policies for each are described consistently with the respective behaviour descriptions in the user and administrator guidance. For example, the "identification and authentication" policy enforced upon remote untrusted users might be more stringent than that enforced upon administrators whose only point of access is within a physically-protected area; the differences in authentication should correspond to the differences in the descriptions of authentication within the user and administrator guidance.

For guidance on consistency analysis see [CEM, Annex B.3].

ADV_SPM.[23].5C ***Where the functional specification is [2: at least] semiformal, the demonstration of correspondence between the TSP model and the functional specification shall be semiformal.***

ADV_SPM-12 The evaluator ***shall examine*** the functional specification correspondence demonstration of the TOE security policy model to determine that it is presented in a semiformal style.

Where the functional specification is presented in an informal style, this work unit is not applicable and is therefore considered to be satisfied.

Where the functional specification and the security policy model are presented in a formal style, this work unit is not applicable and is therefore considered to be satisfied. In this case, ADV_SPM-13 applies.

The functional specification correspondence demonstration (cf. ADV_SPM-10 and ADV_SPM-11) has to be semiformal.

A semiformal correspondence is one that results from a structured approach with a substantial degree of rigor (in terms of completeness and correctness). Such a semiformal correspondence limits the subjective interpretations of its terms, and so it provides less ambiguity than would exist in an informal correspondence.

ADV_SPM.3.6C ***Where the functional specification is formal, the proof of correspondence between the TSP model and the functional specification shall be formal.***

ADV_SPM-13 The evaluator ***shall examine*** the functional specification correspondence demonstration of the TOE security policy model to determine that it is presented in a formal style.

Where the functional specification is not presented in a formal style, this work unit is not applicable and is therefore considered to be satisfied.

The functional specification correspondence demonstration (cf. ADV_SPM-10 and ADV_SPM-11) has to be a formal proof.

The formal proof of correspondence removes all subjective interpretations of its terms by enlisting well-established mathematical concepts to define the syntax and semantics of the formal notation and the proof rules that support logical reasoning. The security features within the TOE (which are identified in the formal TSP model) are expressed in a formal specification language and proved to be satisfied by the formal functional specification.

Appendix B

Correspondence with CEM

The effective versino of the *Common Evaluation Methodology for Information Technology Security Evaluation (CEM)* [3] describes the evaluation activities for the evaluation assurance levels EAL1 – EAL4. In fact in none of these levels a formal security policy model is required but, the level EAL4 contains the component ADV_SPM.1, thus an informal securiy policy model.

In CEM [3, Sec. 8.6.7] the sub-activity for the evaluation of an informal security policy model is described as a part of EAL4. In AIS-34 [10, Sec. 1.6.8] the sub-activity for the evaluation of a formal security policy model is described as a part of EAL5. The work units of the sub-activity proposed in Appendix A closely correspond with the work units of CEM [3] and AIS 34 [10]. The correspondence is exposed in Tbl. B.1.

Appendix A	CEM [3, Sec. 8.6.7]	AIS-34 [10, Sec. 1.6.8]
ADV_SPM-1	—	5:ADV_SPM.3-1
ADV_SPM-2	4:ADV_SPM.1-1	5:ADV_SPM.3-2
ADV_SPM-3	4:ADV_SPM.1-2	5:ADV_SPM.3-3
ADV_SPM-4	4:ADV_SPM.1-3	5:ADV_SPM.3-4
ADV_SPM-5	4:ADV_SPM.1-4	5:ADV_SPM.3-5
ADV_SPM-6	—	5:ADV_SPM.3-6
ADV_SPM-7	—	5:ADV_SPM.3-7
ADV_SPM-8	4:ADV_SPM.1-5	5:ADV_SPM.3-8
ADV_SPM-9	4:ADV_SPM.1-6	5:ADV_SPM.3-9
ADV_SPM-10	4:ADV_SPM.1-7	5:ADV_SPM.3-10
ADV_SPM-11	4:ADV_SPM.1-8	5:ADV_SPM.3-11
ADV_SPM-12	—	5:ADV_SPM.3-12
ADV_SPM-13	—	5:ADV_SPM.3-13

Table B.1: Correspondence of sub-activity ADV_SPM with CEM and AIS-34

Appendix C

Best Practices

The following notes address possible problems of FSPM evaluations which might arise as a consequence of ambiguous interpretations of CEM [3]. In order to minimize the scope of ambiguity the following suggestions (TBDs[1]) are made. They are based on experience with present evaluations.

Evaluators can be positive about the evaluation if the developers adhere to the suggestions. It is recommended to support the guidance of SPM presented in this document by these additional suggestions.

The suggestions are intended as a summary and are not intended to be conclusive. In the following the notions of the building blocks of security policy models are sometimes termed differently. Security Properties are also known as Invariants whereas Security Characteristics should be identified with their ITSEC counterpart "'Practices"'. Security Features translate into the German "Merkmale".

Apart from initial security conditions and assumptions the security features also incorporate the definition of datatypes, attributes, subjects, objects, etc. Note that the formal part must be strong enough to allow for logical rules of inference. In addition to the usual requirements the developer provides sufficient information to render all required demonstrations and proofs comprehensible to the evaluator.

Common Criteria [1, Part 3, Sec. 10.7] states that

> "While a TSP may include any policies, TSP models have traditionally represented only subsets of those policies, because modeling certain policies is currently beyond the state of the art. The current state of the art determines the policies that can be modeled, and the PP/ST author should identify specific functions and associated policies that can, and thus are required to be, modeled. At the very least, access control and information flow control policies are required to

[1] "To Be Done": Action required to be performed by the developer

be modeled (if they are part of the TSP) since they are within the state of the art."

The definition of TSP models is accompanied by the Common Evaluation Methodology [3] for informal TSP models in ADV_SPM.1:

> "In cases where the security policy model presentation is informal, all security policies can be modeled (i. e. described), and so must be included."

Because the security policy is represented as a collection of security requirements, all security requirements must be addressed at the informal level. The developer has to argue if formal and/or semiformal modeling is impossible for certain requirements. However, the argument does not excuse him from informally modeling the security policy being addressed by the requirement.

TBD1

The developer is required to explicitly provide an interpretation as part of the formal security policy model. The following paragraphs provide guidance on how this should be done.

According to work unit ADV_SPM-6 it has to be formally shown that the security features enforce the security properties. As long as the evaluator is not able to identify the security principles and security characteristics the work unit remains inconclusive at best. The use of tables like Tab. C.1 is suggested where all missing information should be provided by the developer in the respective boxes. It is included as an example of the degree of detail the developer is required to provide. All security functional requirements (SFRs) of the ST and PP (if claimed) have to be addressed.

All SFRs which cannot be modeledmust nevertheless be addressed by the developer in order to provide sufficient arguments why this cannot be done. They always have to be informally modeled. The evaluator examines the arguments, he does not provide them. Failure of identification renders the work unit inconclusive:

- If some of the security functional requirements are not applicable (n/a) the developer has to provide convincing arguments why they cannot be covered by the model. As an example the evaluator would expect something like the following:

  Security characteristic char1 is informally described in section sec1 of [ref1] and maps to security feature feat3 described formally in section sec3 of [ref3]. Security principle prin2 is informally described in section sec2 of [ref1] and maps to security property prop4 described formally in section sec4 of [ref3].

TSF	SFR	TOE Security Policy		TSP Model	
		Characteristic	Principle	Feature	Property
TSF1	FMT_LIM.1.1				
	FMT_LIM.2.1	char1 in sec1	prin2 in sec2	fest3 in sec3	prop4 in sec4
	FPT_SEP.1.1				
	FPT_SEP.1.2				
TSF2	FDP_ACC.1.1				
	FDP_ACF.1.1				
	FDP_ACF.1.2				
	FDP_ACF.1.3				
	FDP_ACF.1.4				
	FMT_MSA.3.1				
	FMT_MSA.3.2	char1 in sec1	char2 in sec2	n/a	n/a
	FMT_MSA.1.1				
	FMT_SMF.1.1				
TSF3	FPT_FLS.1.1				
	FRU_FLT.2.1				

Table C.1: Relation of TSFs and SFRs to the TSP model

- Characteristics, principles, features and properties need not be different for each box (although they may), but it has to be indicated that equal characteristics (like Memory Access Control) and Principles (like restricted user access) map to features and properties. If features and properties are the same for different boxes it has to be argued how they support different security requirements.

  Formal theorem *thm1* (e. g. *feat3* |- *prop4*) has been proven in section sec of reference [ref], showing the invariant *prop4* under the assumption *feat3*.

  Logically speaking theory *feat3* could be made up of non-logical axioms *ax1*, *ax2*, etc. using the rules of inference and logical axioms of some logic implicit in the deduction operator "|-"(e. g. HOL in Isabelle or some sequent logic in Gentzen style or some intuitionistic logic or some temporal logic in VSE, or . . .).

  thm1 supports FMT_LIM.2.1 because . . . (developer explanation required). The non-logical axioms making up the theory *feat3* in which the proof was given are: *ax1*, *ax2*, etc. They have the following interpretation and were chosen for the following reason: . . . (developer argument).

- Explanations like the following: "FDP_ACC.1: is met because the model describes access control, relating subjects, objects and operations." are insufficient and unacceptable for several reasons: It is not clear which theorem supports FDP_ACC.1 and this has to be

identified by the developer. It is necessary to support the above claim by convincing arguments. It is also necessary to explain how the model describes access control by referencing the feature and property of the invariant the theorem addresses, see above.

Important note: The developer has to provide informal interpretations of the theorems being proved. The evaluator then examines the arguments provided by the developer for completeness and consistency and verifies that the formal proofs are correct. To support this, complete formal proof scripts must be provided by the developer.

TBD2

The informal description must contain a level of detail enabling the evaluator not only to see why the description is made but rather how the model should be interpreted. This includes a detailed informal description of security principles and characteristics and a mapping onto their respective formal counterparts “Security Features” and “Security Properties”, needed to state and proof the invariants of the model. It is necessary to explain the invariants informally, map them to their formal counterparts and show how they contribute to the security requirements, see TBD1.

TBD3

Work unit ADV_SPM-7 requires the evaluator to examine that the formal security policy model contains a proof of its internal consistency. The developer should argue for consistency of his model, i. e. absence of formal contradictions. It is well known that formal consistency proofs are possible as soon as a finite model of all of the axioms making up the theory can be given. If for example the FSPM models interacting state machines, a concrete finite state machine which provably satisfies all of the axioms could serve as a proof of consistency.

TBD4

The SPM sometimes includes security requiremenets (e. g. FRU_FLT.2 and FPT_FLS.1) supported by different security functions. If so, the developer has to provide formal support on each instance or otherwise argue why this ist not possible. The coverage of the security policy is not considered to be complete unless the developer argues for each and every instance of occurrence.

Bibliography

[1] Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation (CC)*, Version 2.1, August 1999. Also available as *ISO/IEC 15408: IT – Security techniques – Evaluation criteria for IT security*.

[2] Bundesamt für Sicherheit in der Informationstechnik. *Gemeinsame Kriterien für die Prüfung und Bewertung der Sicherheit von Informationstechnik*, Version 2.1, August 1999. Übersetzung des englischsprachigen Originals [1].

[3] Common Criteria Project Sponsoring Organisations. *Common Methodology for Information Technology Security Evaluation*. Part 2: Evaluation Methodology, Version 1.0, August 1999.

[4] Office for Official Publications of the European Communities. *Information Technology Security Evaluation Criteria (ITSEC)*, Version 1.2, June 1991.

[5] Amt für amtliche Veröffentlichungen der Europäischen Gemeinschaften. *Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik*, Version 1.2, Juni 1991. Übersetzung des englischsprachigen Originals [4].

[6] Office for Official Publications of the European Communities. *Information Technology Security Evaluation Manual (ITSEM)*, Version 1.0, September 1993.

[7] Amt für amtliche Veröffentlichungen der Europäischen Gemeinschaften. *Handbuch für die Bewertung der Sicherheit von Systemen der Informationstechnik*, Version 1.0, September 1993. Übersetzung des englischsprachigen Originals [6].

[8] Joint Interpretation Working Group (JIWG). *ITSEC Joint Interpretation Library (ITSEC JIL)*, Version 2.0, 1998.

[9] Bundesamt für Sicherheit in der Informationstechnik. *Anwendungshinweise und Interpretationen zum Schema (AIS)*. Stand: 6. August 2002.

[10] Bundesamt für Sicherheit in der Informationstechnik. *Anwendungshinweise und Interpretationen zum Schema (AIS)*. Evaluation Methodology for CC Assurance Classes for EAL5+ Version: 1.00, June 1, 2004.

[11] DIN V 66291-1, Ausgabe: 2000-04. *Chipkarten mit Digitaler Signatur-Anwendung / Funktion nach SigG und SigV – Teil 1: Anwendungsschnittstelle*. Beuth-Verlag, April 2000.

[12] TeleTrusT Deutschland e. V. *Generic Security Target for ICC embedded software for Signature Creation conforming with German SigG, SigV and DIN V 66291-1*, Version 1.0, September 12th, 2000.

[13] Bundesamt für Sicherheit in der Informationstechnik. *Generic Formal Model of Security Policy for a SigG compliant ICC*, Version 1.2, November 27th, 2001.

[14] D. E. Bell, L. La Padula. Secure Computer Systems: Mathematical Foundations. MITRE Technical Report 2547, Volume I, March 1973.

[15] D. E. Bell, L. La Padula. Secure Computer Systems: A Mathematical Model. MITRE Technical Report 2547, Volume II, May 1973.

[16] D. E. Bell, L. La Padula. Secure Computer Systems: Unified Exposition and Multics Interpretation. MITRE Technical Report 2997, March 1976.

[17] K. J. Biba Integrity Considerations for Secure Computer Systems. MITRE Technical Report MTR-3153, April 1977.

[18] J. A. Goguen, J. Meseguer. Security Policies and Security Models. In Proceedings of the Symposium on Security and Privacy, IEEE Computer Society, pp. 11–20, Oakland, CA, April 1982.

[19] J. A. Goguen, J. Meseguer. Inference Control and Unwinding. In Proceedings of the Symposium on Security and Privacy, IEEE Computer Society, pp. 75–86, Oakland, CA, April 1984.

[20] J. Graham-Cumming and J.W. Sanders. On the Refinement of Non-interference. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 35–42, 1991.

[21] J. Gray. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, Vol. 1, no. 3–4, pp. 255–294, 1992.

[22] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[23] D. Hutter, H. Mantel, G. Rock, W. Stephan, A. Wolpers, M. Balser, W. Reif, G. Schellhorn, and K. Stenzel. VSE: Controlling the Complexity in Formal Software Development. In *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, Springer LNCS 1641, 1999.

[24] T. Ihringer. *Diskrete Mathematik*. B. G. Teubner, Leitfäden der Informatik, Stuttgart, 1994.

[25] J. Jacob. On the Derivation of Secure Components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, Oakland, CA, 1.–3. Mai, 1989.

[26] F. Koob, M. Ullmann, S. Wittmann. The New Topicality of Using Formal Models of Security Policy within the Security Engineering Process. In *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, Springer LNCS 1641, 1999.

[27] B. W. Lampson. Protection, Proceedings of the Fifth Annual Princeton Conference on Information Science Systems, pp. 437-443, 1971.

[28] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit –. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, July 3–5 2000.

[29] H. Mantel. Unwinding Possibilistic Security Properties. In *European Symposium on Research in Computer Security, ESORICS 2000*, LNCS, Toulouse, France, October 4-6 2000. Springer.

[30] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 161–166, Oakland, CA, 27.–29. April, 1987.

[31] J. McLean A Comment on the “Basic Security Theorem” of Bell and La Padula Inform ation Processing Letters 20, pp. 67–70, 1985.

[32] J. McLean. The Specification and Modeling of Computer Security. Computer, Vol. 23, no. 1, Jan. 1990.

[33] J. McLean. Security Models. Encyclopedia of Software Engineering, Ed. John Marciniak, Wiley & Sons, Inc., 1994.

[34] J. McLean. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. IEEE Symposium on Security and Privacy, pp. 79–93, IEEE Press, 1994.

[35] J. McLean. A General Theory of Composition for a Class of "Possibilistic" Properties. *IEEE Transactions on Software Engineering*, January 1996, Volume 22 Number 1, pp. 53–67.

[36] J. K. Millen. Unwinding Forward Correctability. In *Proceedings of the Computer Security Foundations Workshop*, pages 2–10, 1994.

[37] C. O'Halloran. A Calculus of Information Flow. In *Proceedings of the European Symposium on Research in Computer Security*, Touloute, France, 1990.

[38] G. Rock, W. Stephan, and A. Wolpers. Modular Reasoning about Structured TLA Specifications. In R. Berghammer and Y. Lakhnech (Eds.), *Tool Support for System Specification, Development and Verification*, pp. 217–229, Advances in Computing Science, Springer, 1999.

[39] A.W. Roscoe, J.C.P. Woodcock, L. Wulf. Non-Interference through Determinism. *Journal of Computer Security*, 4(1), 1996. Revised from European Symposium on Research in Computer Security (ESORICS), LNCS 875, pp. 33–53, Springer, 1994.

[40] A.W. Roscoe, M.H. Goldsmith. What is Intransitive Noninterference? In Proceedings of the 12th IEEE Computer Security Foundations Workshop, Mordano, Italien, IEEE Society Press, June 28–30, 1999.

[41] J. Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical Report SRI-CSL-92-02, Computer Science Laboratory, SRI International, Menlo Park, CA, October 1992.

[42] P.Y.A. Ryan. A CSP Formulation of Non-Interference and Unwinding. *Cipher*, pages 19–30, Winter 1991.

[43] D. Sutherland. A Model of Information. In *9th National Computer Security Conference*, September 1986.

[44] A. S. Tanenbaum *Modern Operating Systems*. International Editions, Prentice Hall, Englewood Cliffs, NJ, USA, 1992.

[45] A. Zakinthinos and E. Lee. The Composability of Non-Interference. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 2–8, Kenmare, Ireland, 1995.

[46] A. Zakinthinos and E.S. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, May 4–7 1997.

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

[48] Tobias Nipkow. Hoare Logics in Isabelle/HOL. In *Proof and System-Reliability*, H. Schwichtenberg and R. Steinbrüggen,editors. Kluwer, 2002, pages 341-367.

[49] L. C. Paulson. Isabelle: the next seven hundred theorem provers (system abstract). In: E. Lusk and R. Overbeek (editors), 9th International Conf. on Automated Deduction (Springer LNCS 310, 1988), pages 772 - 773.

[50] Tobias Nipkow and L. C. Paulson. Isabelle-91 (system abstract). In: D. Kapur (editor), 11th International Conf. on Automated Deduction (Springer LNAI 607, 1992), pages 673 - 676.

[51] S. Owre and J. M. Rushby and and N. Shankar. PVS: A Prototype Verification System. In: D. Kapur (editor), 11th International Conference on Automated Deduction (CADE), 1992, Lecture Notes in Artificial Intelligence, volume 607, pages 748–752, Springer

[52] Franz Huber and Bernhard Schätz and Alexander Schmidt and Katharina Spies. AutoFocus - A Tool for Distributed Systems Specification. In: Proceedings FTRTFT'96 - Formal Techniques in Real-Time and Fault-Tolerant Systems. 1996. p. 467-470. Bengt Jonsson, Joachim Parrow (ed.). LNCS 1135, Springer Verlag.

[53] O. Slotosch. Modelling and Validation: AutoFocus and Quest. Formal Aspects of Computing 12(4):225-227, 2000

[54] J. M. Spivey. The Z Notation: A reference Manual. International Series in Computer Science. Prentice-Hall, New-York, second edition, 1992

[55] Jean-Raymond Abrial and M. K. O. Lee and D. S. Neilson and P. N. Scharbach and I. H. Sorensen, The B-method (software development). BP Res., Sunbury Res. Centre, Sunbury-on-Thames, UK, VDM 91. Formal Software Development Methods. 4th International Symposium of VDM Europe Proceedings.", volume 2, pages 398–405, W. J. Prehn, S.; Toetenel (editors), Springer-Verlag, Berlin, Germany.

[56] J. Bicarregui (ed.). Proof in VDM: Case Studies. Springer-Verlag, FACIT series, March'98.

[57] The RAISE Method Group. The RAISE Development Method. In BCS Practitioner Series. Prentice Hall, 1995.

[58] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10E20 states and beyond. In LICS, 1990.

[59] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Programming Languages, January 1992.

[60] G. Holzmann. SPIN Model Checker, The: Primer and Reference Manual. AT&T Bell Labs Murray Hill New Jersey, ISBN: 0-321-22862-6, Addison Wesley Professional, 2004

[61] Friedrich Ludwig Gottlob Frege. Begriffsschrift (Concept Script), eine der arithmetischen nachgebildete Formelsprache des reinen Denkens, Halle a. S., 1879

[62] Flemming Nielson and Hanne Riis Nielson and Chris Hankin. Principles of Program Analysis. Springer-Verlag, Berlin - Heidelberg - New York, 1999

[63] Leslie Lamport. Specifying Systems. The TLA+ Language and Tools for Hardware and Software Engineers. Addison Wesley, 2003

[64] David Harel and Dexter Kozen and Jerzy Tiuryn. Dynamic Logic. Foundation of Computing Series, MIT Press, 2000

[65] Christoph Weidenbach and Bernd Gaede and Georg Rock SPASS & FLOTTER Version 0.42. Lecture Notes In Computer Science, proceedings of the 13th International Conference on Automated Deduction, pages 141 - 145, Springer Verlag London, 1996

[66] Gernot Stenz and Andreas Wolf. E-SETHEO: Design, Configuration and Use of a Parallel Automated Theorem Prover. Proceedings, 12th Australian Joint Conference on Artificial Intelligence, LNAI, Norman Foo (editor), volume 1747, Sydney, Australia, Springer Berlin 1999

[67] Zohar Manna and Amir Pnueli. The temporal logic of reactive and concurrent systems. Springer-Verlag New York,1992

[68] Rober S. Boyer and J. Strother Moore and Matt Kaufmann. The Boyer-Moore Theorem Prover and Its Interactive Enhancement. Computers and Mathematics with Applications, 29(2), pages. 27-62, 1995

[69] Dieter Hutter and Claus Sengler, INKA - The Next Generation. In Proceedings of 13th International Conference on Automated Deduction, CADE-13, Mc Robbie and J. Slaney (editors), LNAI, volume 1104, pages 288 - 292, Springer-Verlag, New Brunswick, USA, 1996

[70] Serge Autexier and Dieter Hutter and Heiko Mantel and Axel Schairer. System Description: INKA 5.0 - A Logical Voyager. In Proceedings 16th International Conference on Automated Deduction, CADE-16, H. Ganzinger (editor), LNAI, volume 1632, Trento, Italy, Springer-Verlag, 1999

[71] Dieter Hutter. Annotated Reasoning. Annals of Mathematics and Artificial Intelligence (AMAI). Special Issue on Strategies in Automated Deduction, volume 29, pages 183 - 222, Kluwer Academic Publishers, 2000

[72] Alan Bundy and David Basin and Dieter Hutter and Andrew Ireland. Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge University Press, December, 2004

[73] H.-D. Ebbinghaus and J. Flum and W. Thomas. Einführung in die mathematische Logik. Darmstadt, 1978.

[74] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatsheft für Math. und Physik 38, 1931, S.173-198.

[75] Kurt Gödel. Diskussion zur Grundlegung der Mathematik, Erkenntnis 2. Monatsheft für Math. und Physik, 1931-32, S.147-148.

[76] Melvin Fitting. First-Order Logic and Automated Theorem Proving. Springer-Verlag, 1990. Second edition, 1996.