Bundesamt für Sicherheit in der Informationstechnik

Leitfaden für die Erstellung und Prüfung formaler Sicherheitsmodelle im Rahmen von ITSEC und Common Criteria

Version 1.0c

23. September 2002

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Autoren:
Heiko Mantel
Werner Stephan
Markus Ullmann
Roland Vogt

Erstellt von

Deutsches Forschungzentrum für Künstliche Intelligenz (DFKI GmbH)

Stuhlsatzenhausweg 3
66123 Saarbrücken

Kontakt:
Roland Vogt
Tel.: 0681-302-4131
email: Roland.Vogt@dfki.de

Im Auftrag von

Bundesamt für Sicherheit in der Informationstechnik (BSI)

Postfach 20 03 63
53133 Bonn

Kontakt:
Markus Ullmann
Ref. VI 4
Tel.: 0228-9582-268
email: ullmann@bsi.de

Inhaltsverzeichnis

Kapitel 1

Zusammenfassung

Im Hinblick auf die Erstellung und Prüfung formaler Spezifikationen gemäß den gültigen Kriterienwerken (ITSEC, CC) bleiben erfahrungsgemäß nach dem Lesen der entsprechenden Passagen viele Fragen offen. Dieser Leitfaden soll – insbesondere für die Erstellung und Prüfung formaler Sicherheitsmodelle (Formal Model of Security Policy, FMSP) – dazu beitragen, Entwicklern und Evaluatoren ein besseres Verständnis dieser Thematik im Kontext der Kriterien zu ermöglichen. Ferner soll er helfen, den Mehrwert formaler Sicherheitsmodelle über die reine Erfüllung der Kriterienanforderungen hinaus zu erkennen.

Hierzu werden zunächst die verschiedenen und an verteilten Stellen der Kriterien definierten Anforderungen differenziert nach ITSEC und CC zitiert. Diese werden mit den Charakteristika formaler Methoden, bekannten und veröffentlichten formalen Sicherheitsmodellen und den Erfahrungen aus der Entwicklung und Prüfung formaler Modelle in Bezug gesetzt. Insbesondere wird deutlich, daß die in den Kriterienwerken zitierten Modelle nach Bell/La Padula und Biba nur jeweils generische Rahmen definieren, in denen verschiedene Sicherheitspolitiken zur Zugriffskontrolle definiert werden können.

Deutlich herausgestellt wird, daß die formalen Sicherheitsmodelle keine isolierten Spezifikationen sind, sondern mit anderen Evaluationsdokumenten im direkten und engen Bezug stehen müssen, wie z. B. mit der funktionalen Spezifikation (siehe CC, Familie ADV_FSP).

So weit es der Verständlichkeit dient, werden die Anforderungen an die Bestandteile formaler Sicherheitsmodelle nicht nur präzisiert, sondern mit kleinen Auszügen aus einem existierenden formalen Modell zu Integrated Circuit Cards mit Signaturfunktion nach DIN V 66291-1 unterlegt und erläutert.

Alle bisherigen Erfahrungen zeigen, daß durch eine formale Modellierung der Sicherheitspolitik – als formales Sicherheitsmodell – ein Zugewinn an Vertrauen in die Sicherheit des nach dieser Sicherheitspolitik arbeitenden Produktes erreicht werden kann. Dies gilt unter anderem deshalb, weil immer bereits Schwachstel-

len und Fehler auf Basis der informellen Sicherheitspolitiken aufgedeckt werden konnten.

Bisher werden diese Ergebnisse insofern immer teuer erkauft, weil die Erstellung formaler Sicherheitsmodelle als isolierte Evaluationsaufgabe nach Ende der Produktentwicklung verstanden wird und nicht bereits Teil der Systementwicklung ist. Wäre dies der Fall, würden viele Fehler und Sicherheitsschwachstellen in Produkten frühzeitig vermieden und bräuchten nicht im Nachhinein mit großem Aufwand beseitigt werden.

Kapitel 2

Einführung

Die *Information Technology Security Evaluation Criteria (ITSEC)* [4, 5] definieren Kriterien für die Evaluierung von IT-Systemen und -Produkten nach den Evaluierungsstufen E1 bis E6, wobei die Stufe E6 die höchsten Anforderungen an die Sicherheit stellt. Analog definieren die *Common Criteria for Information Technology Security Evaluation (CC)* [1, 2] die Evaluierungsstufen EAL1 bis EAL7. IT-Sicherheit (engl. IT security) umfaßt Vertraulichkeit (engl. confidentiality), Integrität (engl. integrity) und Verfügbarkeit (engl. availability). Für die Evaluierungsstufen E4 bis E6 (ITSEC) bzw. EAL5 bis EAL7 (CC) wird gefordert, daß sich die Entwicklung des Evaluationsgegenstands (EVG) auf ein formales Sicherheitsmodell abstützt. Ein formales Sicherheitsmodell ist ein Modell der angestrebten EVG-Sicherheitspolitik, das in einer formalen Notation erstellt ist. Eine solche Notation muß auf mathematischen Konzepten basieren, mit denen die Syntax und die Semantik der Sprache sowie Schlußregeln definiert sind.

Im Rahmen einer Evaluierung können bereits veröffentlichte formale Sicherheitsmodelle als Ganzes oder auch in Teilen verwendet werden, wenn sie für den EVG geeignet sind. Eignet sich keines der bekannten formalen Sicherheitsmodelle, so muß ein solches Sicherheitsmodell erstellt werden. Als Beispiele für veröffentlichte Sicherheitsmodelle werden in den ITSEC das Bell/La Padula Modell, das Modell von Clark und Wilson, das Brewer-Nash Modell, das Eizenberg Modell und das Landwehr Modell angegeben (vgl. [4, Abs. 2.83]). In den CC werden das Bell/La Padula Modell, das Biba Modell und Modelle der Interferenz-Freiheit von Goguen und Meseguer genannt (vgl. [1, Teil 2, Abs. 786]).

Für den EVG und das gewählte Sicherheitsmodell muß der Nachweis erbracht werden, daß die Sicherheitsfunktionen des EVG mit der modellierten Sicherheitspolitik und schließlich mit den funktionalen Anforderungen an den EVG übereinstimmen. Ein formales Sicherheitsmodell stellt damit das Bindeglied zwischen Anforderungs- und Entwurfsspezifikation dar. Es ist damit für die Qualität der Sicherheitsleistung des EVG von zentraler Bedeutung.

2.1 Zielsetzung

Dieses Dokument soll als ein Leitfaden bei der Erstellung und der Evaluierung von Sicherheitsmodellen nach ITSEC bzw. CC dienen. Der Leitfaden soll sowohl den Antragsteller (engl. sponsor) einer Evaluierung, beim Erfüllen der Kriterien, die für ein formales Sicherheitsmodell gefordert sind, als auch den Evaluator bei der Prüfung des EVG unter diesen Kriterien unterstützen.

Der Leitfaden umfaßt eine Übersicht von bekannten Sicherheitsmodellen, die sich bereits in der Evaluierung nach ITSEC bzw. CC bewährt haben bzw. als geeignet erscheinen. Für jedes dieser Modelle wird beschrieben, was die jeweilige Sicherheitspolitik ist, wie die Sicherheitspolitik formal modelliert werden kann und unter welchen Annahmen das Sicherheitsmodell eingesetzt werden kann.

Ob auf ein bestehendes formales Sicherheitsmodell zurückgegriffen werden kann, ist letztlich eine Frage der zu erreichenden Sicherheitsziele und der Sicherheitspolitik des zu prüfenden Produktes. Hier soll der Leitfaden bei der Entscheidung unterstützen, ob ein bekanntes Modell direkt oder leicht modifiziert eingesetzt werden kann oder ob ein neues erstellt werden muß.

2.2 Aufbau des Dokuments

Dieser Leitfaden ist nach den folgenden fünf Fragestellungen gegliedert, die sich bei der Entwicklung und Prüfung von formalen Sicherheitsmodellen in der Praxis ergeben.

- Wie sind die Anforderungen der Kriterienwerke an formale Sicherheitsmodelle im Kontext des Entwicklungsprozesses zu interpretieren?
- Welche Konzepte, Techniken und Werkzeuge formaler Methoden werden für die Erstellung und Prüfung formaler Sicherheitsmodelle benötigt?
- Welche Sicherheitsprinzipien und -charakteristika sind von Bedeutung und wie können sie formal modelliert werden?
- Gibt es allgemein verwendbare formale Sicherheitsmodelle, auf die zurückgegriffen werden kann?
- Wie verläuft die Evaluierung nach ITSEC bzw. CC?

In Kapitel 3 werden zunächst alle Anforderungen an formale Sicherheitsmodelle aus ITSEC und CC zusammengetragen. Anschließend wird ein Anforderungskatalog bestimmt, der sowohl beiden Kriterienwerken als auch dem Stand der Technik im Bereich der formalen Methoden gerecht wird.

Die Grundkonzepte formaler Methoden werden in Kapitel 4 zusammenfassend wiedergegeben. Die Darstellung soll einen Einblick in die Grundlagen und die technischen Möglichkeiten der Verwendung formaler Methoden im Bereich der IT-Sicherheit geben. Sie ist selbstverständlich weder in der Breite noch in der Tiefe als umfassende Einführung in formale Methoden anzusehen.

In Kapitel 5 werden die bisher abstrakt behandelten Begriffe und Anforderungen an dem konkreten Beispiel einer SmartCard-Applikation zur Erzeugung digitaler Signaturen erläutert. Ausgehend von einer informell gegebenen Sicherheitspolitik werden Orientierungshilfen für die Bestimmung sowie Formalisierung von Sicherheitseigenschaften und -merkmalen, für die adäquate Wahl des Abstraktionsgrades und für die Präzisierung von Annahmen an die Einsatzumgebung gegeben. Die Ausführungen werden ergänzt durch Ausschnitte aus dem referenzierten formalen Sicherheitsmodell, das mit Hilfe des *Verification Support Environment (VSE)*, einem vom BSI zugelassenen Werkzeug für die Erstellung formaler Sicherheitsmodelle, spezifiziert und verifiziert worden ist.

Allgemein einsetzbare Sicherheitsmodelle aus der Klasse auf Noninterference basierender Sicherheitsmodelle werden in Kapitel 6 detailliert beschrieben. Im einzelnen werden Noninterference (Abschnitt 6.1), das Modell von Bell/La Padula (Abschnitt 6.2.1) und das Modell von Biba (Abschnitt 6.2.2) behandelt. Diese Klasse von Sicherheitsmodellen kann sowohl zur Modellierung von Anforderungen an die Vertraulichkeit als auch an die Integrität eingesetzt werden. In den Abschnitten 6.4 und 6.5 wird die Verwendbarkeit dieser Modelle in einer konkreten Anwendung behandelt und erläutert, was eine Instantiierung der Modelle für ein System bedeutet.

In Kapitel 7 werden Hinweise und Methoden für die Prüfung formaler Sicherheitsmodelle im Sinne der Anforderungen von ITSEC und CC gegeben. Sowohl Entwickler als auch Evaluatoren sollen dadurch eine Orientierung erhalten, welche Kriterien für die Qualität eines formalen Sicherheitsmodells ausschlaggebend sind und mit welchen Verfahren mögliche Mängel entdeckt werden können.

In Kapitel 8 wird zusammenfassend erläutert, warum der Einsatz formaler Methoden in erster Linie den Entwicklungsprozeß entscheidend verbessert und zu einer deutlichen Qualitätssteigerung führt. Die anschließende Evaluation dient (nur) dazu, die sachgemäße Anwendung dieser Techniken zu überprüfen. Zusätzlich wird auf die Abgrenzung von formalen und semiformalen Methoden eingegangen.

Kapitel 3

Anforderungen an formale Sicherheitsmodelle

Die Common Criteria [1, 2] und die ITSEC [4, 5] formulieren konkrete Anforderungen im Zusammenhang mit formalen Sicherheitsmodellen. Im Falle der ITSEC werden diese durch die Joint Interpretation Library [8] ergänzend erläutert.

Zum Zweck der vergleichenden Darstellung sind in diesem Kapitel zunächst die relevanten Abschnitte aus den Kriterienwerken zitiert. Aus diesem Vergleich werden anschließend geeignete Anforderungen bestimmt, die eine effektive Verwendung formaler Sicherheitsmodelle für Evaluierungen sowohl nach ITSEC als auch nach Common Criteria ermöglichen.

3.1 Anforderungen der ITSEC

In den folgenden Abschnitten sind die Anforderungen der ITSEC an formale Sicherheitsmodelle wiedergegeben. Im Seitenrand sind jeweils die Absatznummern aufgeführt. Sämtliche Anforderungen sind sowohl in englischer als auch in deutscher Sprache zitiert. In Zweifelsfällen ist die englischsprachige Fassung gültig[1].

Zunächst werden die allgemeinen Erläuterungen zu formalen Spezifikationen und formalen Sicherheitmodellen aus [4, 5, Kapitel 2] zitiert. Die konkreten Anforderungen an formale Sicherheitmodelle bzw. an die damit in Beziehung stehenden formalen Spezifikationen der sicherheitsspezifischen Funktionen und des Architekturentwurfs sind aus dem Text der Evaluationsstufe E6 (vgl. [4, 5, Stufe E6, Phase 1 und Phase 2] entnommen. Unterschiede zu den Stufen E4 und E5 sind durch Unterstreichung bzw. **Fettdruck** kenntlich gemacht.

[1] Verbindlicher Anwendungshinweis zitiert aus [9, AIS 2, Version 5 vom 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

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

2.13 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

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.

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

2.76 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.

2.77 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.

2.78 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).

Formale Sicherheitsmodelle

Formal Models of Security Policy

2.81 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.

2.82 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.

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: 2.83

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

[. . .] 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]. E[45**6**].2

	Anforderungen an Nachweise	**Requirements for evidence**
E[4**56**].3	[...] Die informelle Interpretation des formalen Sicherheitsmodells muß beschreiben / **erklären**, auf welche Weise seine zugrundeliegende Sicherheitspolitik durch die Sicherheitsvorgaben erfüllt wird.	[...] The informal interpretation of the formal security policy model shall describe / **explain** how the security target satisfies the underlying security policy.
	Aufgaben des Evaluators	**Evaluator Actions**
E[4**56**].4	[...] Es ist zu überprüfen, ob es Sicherheitsmaßnahmen in den Sicherheitsvorgaben gibt, die zu Konflikten mit der dem Sicherheitsmodell zugrundeliegenden Sicherheitspolitik führen.	[...] Check that there are no security features in the security target that conflict with the underlying security policy.
	Phase 2 — Architekturentwurf	**Phase 2 — Architectural Design**
	Anforderungen an Inhalt und Form	**Requirements for content and presentation**
E[45**6**].5	Eine semiformale / **formale** Notation muß verwendet werden, um einen semiformalen / **formalen** Architekturentwurf zu erstellen. [...]	A semiformal / **formal** notation shall be used in the architectural design to produce a semiformal / **formal** description. [...]
	Anforderungen an Nachweise	**Requirements for evidence**
E[45**6**].6	[...] **Sie [Die Beschreibung der Architektur] muß durch Anwendung einer Mischung von formaler und informeller Technik erklären, wie sie mit dem formalen Sicherheitsmodell übereinstimmt.**	[...] **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.**
	Aufgaben des Evaluators	**Evaluator Actions**
E[45**6**].7	[...] **Es ist zu überprüfen, ob die formalen Argumente gültig sind.**	[...] **Check that formal arguments are valid.**

3.2 Interpretation nach ITSEC JIL

Die ITSEC Joint Interpretation Library (JIL) enthält international abgestimmte, verbindliche Interpretationen zu den Anforderungen der ITSEC betreffend formale Methoden (vgl. [8, Chapter 19]). Die für formale Sicherheitsmodelle relevanten Auszüge aus diesem Kapitel sind in den folgenden Abschnitten wiedergegeben. Im Seitenrand sind jeweils die Absatznummern aufgeführt. Die Zitate erfolgen ausschließlich in englischer Sprache, da eine offizielle Übersetzung nicht existiert.

Background

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

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

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

- 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)

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

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"): 279

- 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.

280 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.

Formal Architectural Design (FAD)

283 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.

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

Proofs

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

287 **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.

290 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.

291 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:

- 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.

292 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.

3.3 Anforderungen der Common Criteria

In den folgenden Abschnitten sind die Anforderungen der Common Criteria an formale Sicherheitsmodelle wiedergegeben. Im Seitenrand sind jeweils die Absatznummern aufgeführt. Sämtliche Anforderungen sind sowohl in englischer als auch in deutscher Sprache zitiert. Die englischsprachigen Zitate stammen aus dem Text der Common Criteria in der gültigen Version 2.1 vom August 1999 (vgl. [1]). Die deutschsprachigen Zitate stammen aus der Übersetzung der Version 2.1 der Common Criteria (vgl. [2]). In Zweifelsfällen ist, analog zu ITSEC, die englischsprachige Originalfassung gültig.

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

Der Begriff „(EVG-)Sicherheitsmodell“ wird innerhalb der Common Criteria [1, Teil 1, Glossar] als strukturierte Darstellung einer (EVG-)Sicherheitspolitik bezeichnet. Es ist deshalb notwendig, die im Folgenden wiedergegebene Charakterisierung des Begriffes „(EVG-)Sicherheitspolitik“ aus dem Teil 2 der Common Criteria [1, Teil 2] in die Betrachtung der Anforderungen an formale Sicherheitmodelle einzubeziehen.

Konzeption der funktionalen Anforderungen

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.

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.

Functional requirements paradigm

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. 14

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. 15

16 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

Strukturell unterscheiden sich die Common Criteria von den ITSEC in vielerlei Hinsicht. Insbesondere sind die Vertrauenswürdigkeitsstufen (Evaluation Assurance Levels – EALs) nicht direkt definiert, sondern als Kombinationen von einzelnen Vertrauenswürdigkeitskomponenten zu verstehen, die zu hierarchisch geordneten Paketen geschnürt sind. Die verschiedenen *Komponenten* (assurance component) sind Bestandteile von *Familien* (assurance family), welche wiederum zu *Klassen* (assurance class), den obersten Konstrukten zur Klassifizierung von Anforderungen an die Vertrauenswürdigkeit, zusammengefaßt sind.

Konkrete Anforderungen an (formale) Sicherheitsmodelle bzw. an die damit in Beziehung stehende funktionale Spezifikation sind in den Familien ADV_FSP (Funktionale Spezifikation) und ADV_SPM (Sicherheitsmodell) der Vertrauenswürdigkeitsklasse ADV (Entwicklung) in Teil 3 der Common Criteria [1, Teil 3] formuliert. Die hier abgebildete Tabelle stellt die Zuordnung der Komponenten dieser beiden Vertrauenswürdigkeitsfamilien zu den Vertrauenswürdigkeitstufen EAL5 bis EAL7 dar.

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

In den folgenden Abschnitten sind die relevanten Auszüge aus der Klasse ADV, speziell aus den Familien ADV_FSP und ADV_SPM, wiedergegeben. Unterschiede zwischen den Komponenten ADV_FSP.3 und ADV_FSP.4 sind durch Unterstreichung bzw. **Fettdruck** kenntlich gemacht.

Weitere Beziehungen bestehen zu den Familien ADV_HLD (Entwurf auf hoher Ebene), ADV_LLD (Entwurf auf niedriger Ebene) und ADV_RCR (Übereinstimmung der Darstellung). Anforderungen aus Komponenten dieser Familien sind hier jedoch nicht zitiert, da der Bezug zu formalen Sicherheitmodellen nur indirekt über die funktionale Spezifikation (Familie ADV_FSP) gegeben ist.

Klasse ADV: Entwicklung

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

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.

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)

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. 98

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. 302

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. [...] 309

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. 99

	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:**
ADV_FSP.[34].1D	Der Entwickler muß eine funktionale Spezifikation bereitstellen.	The developer shall provide a functional specification.
	Elemente zu Inhalt und Form des Nachweises:	**Content and presentation of evidence elements:**
ADV_FSP.[34].1C	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].2C	Die funktionale Spezifikation muß in sich konsistent sein.	The functional specification shall be internally consistent.
ADV_FSP.[34].3C	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].4C	Die funktionale Spezifikation muß die TSF vollständig darstellen.	The functional specification shall completely represent the TSF.
ADV_FSP.[34].5C	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.

Elemente zu Evaluatoraufgaben:

Der Evaluator muß bestätigen, daß die bereitgestellten Informationen alle Anforderungen an Inhalt und Form des Nachweises erfüllen.

Der Evaluator muß feststellen, daß die funktionale Spezifikation eine getreue und vollständige Umsetzung der funktionalen EVG-Sicherheitsanforderungen ist.

Evaluator action elements:

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence. ADV_FSP.[3**4**].1E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the TOE security functional requirements. ADV_FSP.[3**4**].2E

Sicherheitsmodell (ADV_SPM)

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.

Ziele

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.

Security policy modeling (ADV_SPM)

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. 105

Objectives

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. 365

Anwendungsbemerkungen

367 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.

368 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). [. . .]

ADV_SPM.3 Formales EVG-Sicherheitsmodell

Elemente zu Entwickleraufgaben:

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

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

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.

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). [. . .]

ADV_SPM.3 Formal TOE security policy model

Developer action elements:

The developer shall provide a TSP model.

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

Elemente zu Inhalt und Form des Nachweises:	**Content and presentation of evidence elements:**	
Das TSP-Modell muß formal sein.	The TSP model shall be formal.	ADV_SPM.3.1C
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.3.2C
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.3.3C
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.3.4C
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.3.5C
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.	ADV_SPM.3.6C
Elemente zu Evaluatoraufgaben:	**Evaluator action elements:**	
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_SPM.3.1E

3.4 Bestimmung des Anforderungskatalogs

In Übereinstimmung mit den oben zusammengestellten Zitaten werden in diesem Abschnitt die konkreten Anforderungen an formale Sicherheitsmodelle bestimmt. Die Ausprägung der einzelnen Anforderungsbestandteile ist darauf ausgerichtet, den Konzepten und Bedürfnissen sowohl der ITSEC als auch der CC gerecht zu werden. Hierbei wird selbstverständlich davon ausgegangen, daß eine Evaluationsstufe angestrebt wird, die die Bereitstellung eines formalen Sicherheitsmodells erfordert (vgl. [4, Abs. E[456].2] bzw. [1, Teil 3, Abs. 223, 229, 234]).

3.4.1 Terminologie

Zur Bezeichnung der Bestandteile der EVG-Sicherheitspolitik wird in ITSEC und CC eine unterschiedliche Terminologie verwendet. Die korrespondierenden Bestandteile des EVG-Sicherheitsmodells werden weder in ITSEC noch in CC explizit bezeichnet.

Um den begrifflichen Vorgaben beider Kriterienwerke entsprechen zu können, wird in diesem Abschnitt eine Vereinheitlichung der Terminologie vorgeschlagen, die im weiteren Verlauf durchgehend benutzt wird. Die Vereinheitlichung hat zum Ziel, eine mit der Fachliteratur (vgl. Kap. 6) verträgliche Abgrenzung der bezeichneten Objekte zu unterstützen.

Es werden vier Begriffe unterschieden, die jeweils paarweise der EVG-Sicherheitspolitik und dem EVG-Sicherheitsmodell sowie dem Analyse- und dem Entwurfsprozeß im Security Engineering zugeordnet sind (s. Tab. 3.1):

Sicherheitscharakteristika (Security Characteristics) stimmt wörtlich mit der im englischsprachigen Original der CC (vgl. [1, Teil 3, Abs. 368]) verwendeten Bezeichnung für einen Bestandteil der EVG-Sicherheitspolitik überein. Wegen der erforderlichen Abgrenzung von dem Begriff *Sicherheitseigenschaften (Security Properties)* wurde die problematische Übersetzung in der deutschsprachigen Ausgabe der CC (vgl. [2, Teil 3, Abs. 368]) nicht übernommen. Der Begriff *Sicherheitscharakteristika (Security Characteristics)* wird hier als Entsprechung zu der in ITSEC gebrauchten Bezeichnung „Praktiken (practices)“ (vgl. [4, Abs. 2.13]) verwendet. Er ist zusammen mit den *Sicherheitsmerkmalen (Security Features)* dem Entwurfsprozeß zuzuordnen, d. h. er umfaßt alle notwendigen Definitionen, Attribute, Aktionen usw. zur detaillierten Charakterisierung der Funktionalität der EVG-Sicherheitspolitik. Diese Funktionalität ist auf die Umsetzung der in den *Sicherheitsprinzipien (Security Principles)* formulierten Grundsätze der EVG-Sicherheitspolitik ausgerichtet.

	Entwurfsprozeß	Analyseprozeß
Sicherheitspolitik (Security Policy)	*Sicherheitscharakteristika (Security Characteristics)*	*Sicherheitsprinzipien (Security Principles)*
Sicherheitsmodell (Security Policy Model)	*Sicherheitsmerkmale (Security Features)*	*Sicherheitseigenschaften (Security Properties)*

Tabelle 3.1: Terminologie von EVG-Sicherheitspolitik und -modell

Sicherheitsprinzipien (Security Principles) wird in ITSEC (vgl. [4, Abs. 2.81]) und ITSEC JIL (vgl. [8, Abs. 279]) zur Beschreibung der EVG-Sicherheitspolitik verwendet. Wegen der Zweiteilung der EVG-Sicherheitspolitik wird der Begriff hier als Entsprechung zu der in ITSEC gebrauchten Bezeichnung „Gesetze und Regeln (laws and rules)“ (vgl. [4, Abs. 2.13]) bzw. zu der in CC gebrauchten Bezeichnung „Regeln (rules)“ (vgl. [1, Teil 3, Abs. 368]) verwendet. Der Begriff *Sicherheitsprinzipien (Security Principles)* ist zusammen mit den *Sicherheitseigenschaften (Security Properties)* dem Analyseprozeß zuzuordnen, d. h. er umfaßt die aus der Analyse der Sicherheitsumgebung und -ziele abgeleiteten und in den Sicherheitsanforderungen dargelegten Grundsätze der EVG-Sicherheitspolitik. Diese Grundsätze bilden die Rahmenbedingungen für die Formulierung der *Sicherheitscharakteristika (Security Characteristics)* und enthalten somit typischerweise keine detaillierten funktionalen Zusammenhänge.

Sicherheitsmerkmale (Security Features) korrespondieren zu den *Sicherheitscharakteristika (Security Characteristics)*. Der Begriff umfaßt die Modellierung der funktionalen Merkmale der EVG-Sicherheitspolitik. Diese bilden die Grundlage für den Nachweise der Gültigkeit der *Sicherheitseigenschaften (Security Properties)*.

Sicherheitseigenschaften (Security Properties) korrespondieren zu den *Sicherheitsprinzipien (Security Principles)*. Der Begriff umfaßt die Modellierung der grundlegenden Eigenschaften der EVG-Sicherheitspolitik. Diese geben den Rahmen für die Formulierung der *Sicherheitsmerkmale (Security Features)* vor.

3.4.2 Spezifikation

Das formale Sicherheitsmodell muß alle Bestandteile der Sicherheitspolitik des EVG spezifizieren, sofern sie nach dem Stand der Technik formal modellierbar sind (vgl. [4, Abs. E[456].2], [8, Abs. 280] bzw. [1, Teil 3, Abs. 367f und Element ADV_SPM.3.2C]). Unter Berücksichtigung seiner Eingliederung in den

hierarchisch strukturierten Entwurf des EVG soll das formale Sicherheitsmodell mindestens aus den zwei folgenden Komponenten bestehen:

1. *Sicherheitseigenschaften (Security Properties)*

   In diesem Bestandteil des Modells werden die Sicherheitsprinzipien (Security Principles) der Sicherheitspolitik des EVG formal spezifiziert. Die Sicherheitseigenschaften umfassen eine formale Beschreibung der in den Sicherheitsprinzipien (Security Principles) dargelegten Grundsätze der EVG-Sicherheitspolitik.

2. *Sicherheitsmerkmale (Security Features)*

   In diesem Bestandteil des Modells werden die Sicherheitscharakteristika (Security Characteristics) der Sicherheitspolitik des EVG formal spezifiziert. Die Sicherheitsmerkmale umfassen eine formale Beschreibung der notwendigen Definitionen, Attribute, Aktionen usw. zur detaillierten Charakterisierung der in den Sicherheitscharakteristika (Security Characteristics) dargelegten Funktionalität der EVG-Sicherheitspolitik.

Die formale Spezifikation muß für den Nachweis der Übereinstimmung des Sicherheitsmodells mit den Sicherheitsprinzipien und -charakteristika der Sicherheitspolitik (vgl. [4, Abs. E[456].3] bzw. [1, Teil 3, Element ADV_SPM.3.3C]) geeignet sein (s. Abschnitt 3.4.3).

Beide Ebenen der Spezifikation sollen durch einen formalen Verifikationsschritt miteinander verbunden werden (s. Abschnitt 3.4.4), d. h. für die Sicherheitsmerkmale soll die Gültigkeit der Sicherheitseigenschaften bewiesen werden. Zur Erleichterung der Verifikation kann es sinnvoll sein, weitere Modellebenen zwischen Sicherheitseigenschaften und -merkmalen zu spezifizieren.

Die formale Spezifikation muß für den Nachweis der Übereinstimmung des Sicherheitsmodells mit dem Architekturentwurf (vgl. [4, Abs. E6.6]) bzw. der funktionalen Spezifikation (vgl. [1, Teil 3, Element ADV_SPM.3.4C]) geeignet sein (s. Abschnitte 3.4.3 und 3.4.4).

Es ist zulässig, auf ein (bekanntes) generisches formales Sicherheitsmodell (z. B. Noninterference nach Rushby [40]) zu verweisen. Solche Modelle enthalten typischerweise die beiden oben geforderten Ebenen der Spezifikation. Wegen ihres generischen Charakters müssen diese Modelle für den konkreten EVG instantiiert werden. Die Instantiierung muß ebenfalls formal sein (s. Kapitel 6) und den oben genannten Anforderungen genügen.

3.4.3 Interpretation

Die in ITSEC als „informelle Interpretation“ und in CC als „Erklärung“ bezeichnete Erläuterung muß Bestandteil der Dokumentation des formalen Sicherheitsmodells sein. Sie soll den Zusammenhang zwischen der in einer formalen Sprache notierten Spezifikation und den mit dem formalen Sicherheitsmodell in Bezug stehenden Entwicklungsdokumenten herstellen. Die Interpretation soll aus den folgenden Komponenten bestehen:

1. Nachweis der Übereinstimmung des formalen Sicherheitsmodells mit der in den funktionalen Anforderungen formulierten EVG-Sicherheitspolitik (vgl. [4, Abs. E[456].3] bzw. [1, Teil 3, Element ADV_SPM.3.3C]). Entsprechend der Struktur der formalen Spezifikation ergeben sich hieraus Nachverpflichtungen für die

   (a) Übereinstimmung der Sicherheitseigenschaften (Security Properties) mit den Sicherheitsprinzipien (Security Principles).

   (b) Übereinstimmung der Sicherheitsmerkmale (Security Features) mit den Sicherheitscharakteristika (Security Characteristics).

   In ITSEC kommt der Darlegung von funktionalen Sicherheitsanforderungen eine wesentlich geringere Bedeutung zu als in CC. Folgerichtig wird in [8, Abs. 279] ein enger Bezug zwischen der Sicherheitspolitik und den Sicherheitszielen hergestellt. Trotzdem entspricht die hier festgelegte Nachweisverpflichtung sowohl den Anforderungen der CC als auch den Anforderungen der ITSEC. Tatsächlich belegen konkrete Erfahrungen mit Evaluierungen nach ITSEC, daß eine Verfeinerung der Sicherheitsziele zu detaillierten (funktionalen) Anforderungen vorteilhaft ist (s. Abschnitt 5.3). Der hier geforderte Übereinstimmungsnachweis stellt daher bei den ITSEC analog zu den CC einen indirekten Bezug zu den Sicherheitszielen her.

2. Nachweis der Übereinstimmung des formalen Sicherheitsmodells mit der funktionalen Spezifikation (vgl. [1, Teil 3, Element ADV_SPM.3.4C]).

   Obwohl die in [4, Abs. E6.6] hergestellte direkte Beziehung zwischen (formalem) Sicherheitsmodell und (formalem) Architekturentwurf in [8, Chapter 19] bekräftigt wird, entspricht die hier festgelegte Nachweisverpflichtung sowohl den Anforderungen der CC als auch den Anforderungen der ITSEC. Der konkrete Bezug zum Architekturentwurf wird durch den ohnehin notwendigen Nachweis der Übereinstimmung[2] zwischen funktionaler Spezifikation und Architekturentwurf hergestellt.

[2] Für den hier vorliegenden Leitfaden ist die Beziehung zwischen funktionaler Spezifikation und Architekturentwurf von untergeordneter Bedeutung und wird daher nicht näher behandelt.

Falls die funktionale Spezifikation in semiformaler (vgl. [4, Abs. E[45].2] und [1, Teil 3, Komponente ADV_FSP.3]) bzw. formaler (vgl. [4, Abs. E6.2] und [1, Teil 3, Komponente ADV_FSP.4]) Notation vorliegt, soll die informelle Erklärung durch einen semiformalen Nachweis bzw. einen formalen Beweis ergänzt werden (s. Abschnitt 3.4.4).

Die geforderten Erläuterungen sollen eine Brücke zwischen Sicherheitsanforderungen und -funktionen schlagen (vgl. [1, Teil 3, Abs. 314]), um über das formale Sicherheitsmodell zu einer Steigerung des Vertrauens in die Sicherheitspolitik des EVG zu gelangen (vgl. [1, Teil 3, Abs. 309]). Daneben sollen sie durch die Verknüpfung der unterschiedlichen Darstellungsformen die den Sicherheitsbedürfnissen angemessene Qualität des formalen Sicherheitsmodells gewährleisten.

Wird die Spezifikation auf der Grundlage eines (bekannten) generischen formalen Sicherheitsmodell erstellt, so ist die erforderliche Instantiierung Gegenstand der Interpretation, d. h. die Erläuterungen müssen auf das instantiierte Modell Bezug nehmen (s. Kapitel 6).

3.4.4 Verifikation

Entsprechend der geforderten Struktur des formalen Sicherheitsmodells sind die folgenden Beweise zu führen:

1. Die Eignung der formal spezifizierten Sicherheitsmerkmale (Security Features) zur Durchsetzung der formal spezifizierten Sicherheitseigenschaften (Security Properties) ist formal zu verifizieren. Hierzu ist die korrekte und vollständige Übereinstimmung der beiden Ebenen der Spezifikation (s. Abschnitt 3.4.2) formal zu beweisen. Werden zwischen Sicherheitseigenschaften und -merkmalen weitere Ebenen der Sicherheitspolitik modelliert, so ist die Übereinstimmung zwischen allen benachbarten Ebenen zu beweisen.

   Wird die Spezifikation durch Instantiierung eines (bekannten) generischen formalen Sicherheitsmodells erstellt, so liegt der hier geforderte Beweis im generischen Modell häufig bereits vor. In solchen Fällen ergeben sich Beweisverpflichtungen, die sicherstellen, daß die jeweilige Instantiierung innerhalb des durch das generische Modell vorgegebenen Rahmens bleibt (s. Kapitel 6).

   Dieser Teil der Verifikation des formalen Sicherheitsmodells entspricht dem von der ITSEC JIL geforderten Beweis der Invarianz von Eigenschaften (invariance of properties) des formalen Sicherheitsmodells (vgl. [8, Abs. 287]). Er soll auf einer hohen Abstraktionsebene die Qualität der Sicherheitsfunktionalität gemessen an den Sicherheitsanforderungen gewährleisten.

2. Die Widerspruchsfreiheit des formalen Sicherheitmodells ist formal zu beweisen. Wegen seiner prinzipiellen Problematik (s. Abschnitt 4.4) ist dieser Teil der Verifikation nicht vollständig innerhalb des formalen Kalküls, der für die Spezifikation verwendet wird, zu behandeln. Die außerhalb dieses Kalküls liegenden und zum Beweis der Widerspruchsfreiheit notwendigen Konzepte sollen wohlbegründet und auf ein Minimum beschränkt sein.

   Ist die formale Spezifikation in mehrere Ebenen gegliedert, deren paarweise Übereinstimmung dann formal zu beweisen ist, so reicht es typischerweise aus, die Widerspruchsfreiheit der untersten Ebene der Spezifikation, d. h. der Ebene der Sicherheitsmerkmale, zu verifizieren. Darüberhinaus kann der Beweis der Widerspruchsfreiheit auf die Ebene der funktionalen Spezifikation und weiter auf die Ebene des Architekturentwurfs verlagert werden, wenn deren Übereinstimmung mit dem formalen Sicherheitsmodell formal bewiesen ist.

   Wird die Spezifikation durch Instantiierung eines (bekannten) generischen formalen Sicherheitsmodells erstellt, so kann der Beweis häufig auf die Widerspruchsfreiheit der konkreten Instantiierung beschränkt werden. Voraussetzung hierfür ist, daß die Widerspruchsfreiheit des generischen Modells gegeben ist.

   Dieser Teil der Verifikation des formalen Sicherheitmodells entspricht dem von der ITSEC JIL geforderten Beweis der internen Konsistenz (internal consistency) des formalen Sicherheitsmodells (vgl. [8, Abs. 287]). Er soll gewährleisten, daß die formalen Argumente auf einem soliden Fundament ruhen.

In den jeweils höchsten Vertrauenswürdigkeitsstufen der ITSEC bzw. Common Criteria wird eine formale Spezifikation der Sicherheitsfunktionen (vgl. [4, Abs. E6.2] bzw. [1, Teil 3, Komponente ADV_FSP.4]) sowie ein formaler Architekturentwurf (vgl. [4, Abs. E6.5] bzw. [1, Teil 3, Komponente ADV_HLD.5]) gefordert. Zusätzlich zu dem in diesem Fall geforderten Beweis der Übereinstimmung zwischen funktionaler Spezifikation und Architekturentwurf, der nicht Gegenstand dieses Leitfadens ist, muß die Übereinstimmung zwischen formalem Sicherheitsmodell und funktionaler Spezifikation formal verifiziert werden (vgl. [1, Teil 3, Element ADV_SPM.3.6C]). Diese Beweisverpflichtung genügt, wie bereits in Abschnitt 3.4.3 erwähnt, auch den Anforderungen der ITSEC, da die Übereinstimmung mit dem Architekturentwurf indirekt bewiesen wird.

Kapitel 4

Grundkonzepte formaler Methoden

4.1 Zum Begriff Formaler Methoden

Formale Methoden verwenden zunächst eine präzise festgelegte, d. h. etwa durch eine Grammatik *formalisierte Syntax*. Beschreibungen sind damit auf ihre syntaktische Korrektheit hin überprüfbar und maschinenlesbar. Über rein textuelle Repräsentationen hinaus verwenden Werkzeuge häufig auch graphische Darstellungen. In diesen Fällen muß eine Abbildung auf eine mathematisch präzise („interne“) Repräsentation angegeben werden.

Wichtiger als die Festlegung der Syntax von Beschreibungen d ist allerdings die Zuordnung von mathematischen Strukturen $[\![d]\!]$ als *Semantik*. Zielstrukturen solcher Semantikzuordnungen können z. B. Algebren oder Zustandsübergangssysteme, d. h. Relationen auf Zuständen oder Mengen von Zustandsfolgen, sein. Während Beschreibungen *endlich* sind, handelt es sich bei der zugeordneten Semantik im allgemeinen Fall um unendliche Objekte, also z. B. Algebren mit unendlich vielen Elementen und Zustandsübergangssysteme mit unendlich vielen Zuständen.

Diese Begriffsbestimmung schließt auch Programmiersprachen als Beschreibungsmittel grundsätzlich mit ein: da die syntaktische Definition von Programmiersprachen heute Standard ist, kommt es entscheidend darauf an, ob eine Semantik mathematisch präzise festgelegt ist. Obwohl hierfür seit geraumer Zeit mathematisch fundierte Techniken zur Verfügung stehen, ist es bis heute eher die Ausnahme, daß für reale Programmiersprachen in vollem Umfang Semantikzuordnungen existieren. Im Zusammenhang mit formalen Beschreibungstechniken werden daher meistens eingeschränkte und idealisierte programmiersprachliche Notationen verwendet. Diese werden nach Abschluß der formalen Entwicklung oder Analyse in ablauffähige Programme übersetzt. Bei Verwendung nicht formal verifizierter Übersetzer entsteht hierdurch eine Sicherheitslücke.

Neben Programmiersprachen sind eine fast unübersehbare Fülle von Beschreibungstechniken vorgeschlagen worden, einige davon mit sauber definierter Semantik. Wenn man nicht nur an der formalen Beschreibung von Systemen interessiert ist, sondern auch an einer umfassenden Analyse, die nicht auf einige vorab festgelegte Eigenschaften beschränkt ist, benötigt man zusätzlich einen Logikformalismus zum flexiblen Formulieren von *Eigenschaften* (von Systemen).

4.2 Logik- und Spezifikationsformalismen

Logikformalismen stellen eine formale Sprache zur Formulierung von *Aussagen* über einen Gegenstandsbereich zur Verfügung. Die Syntaxbeschreibung gibt vor, nach welchen Regeln aus gewissen sprachlichen Grundelementen Aussagen (Sätze) gebildet werden können. Die Syntaxregeln einer logischen Sprache geben etwa an, wie mit *aussagenlogischen Verknüpfungen*, z. B. $\neg$ (nicht), $\wedge$ (und), $\vee$ (oder) und $\rightarrow$ (impliziert), *Quantoren*, z. B. $\forall$ (für alle) und $\exists$ (es gibt) und (temporallogischen) *Modaloperatoren*, z. B. $\Box$ (immer), $\Diamond$ (irgendwann) komplexe *Formeln* gebildet werden können. Ausgangspunkt sind anwendungsspezifische syntaktische Grundelemente, z. B. *Funktionssymbole* und *Relationssymbole*, die durch eine sogenannte *Signatur* gegeben sind.

Ein *Modell* $\mathcal{M}$ (aus einer vorgegebenen Klasse) zu einer Signatur Sig legt fest, durch welche mathematischen Strukturen die Elemente von Sig *gedeutet* werden. Auf dieser Basis wird dann definiert, wann ein Satz φ, der unter Verwendung der Elemente von Sig und Einhaltung der allgemeinen Syntaxregeln gebildet wurde, in einem Modell „wahr“ ist, $\mathcal{M} \models \varphi$.

Wenn man einen Zusammenhang zwischen der Bedeutung von Systembeschreibungen (s. o.) und Modellen einer logischen Sprache herstellt, bietet sich die Perspektive, nach (implementierbaren) Verfahren zu suchen, die entscheiden, ob eine Aussage φ für ein Modell $\mathcal{M} = \mathit{Mod}([\![d]\!])$, das durch eine Beschreibung d gegeben ist, „wahr“ ist, und damit die Frage „$\mathit{Mod}([\![d]\!]) \models \varphi$?“ beantworten. Insbesondere, aber nicht nur, für endliche Systeme wird dieser Weg mit Erfolg beschritten.

Abgesehen von der Beschränkung auf *praktische* endliche Systeme ist bei dieser Vorgehensweise die Spezifikationstechnik eingeschränkt durch den Typus von System, der durch die verwendete Beschreibungssprache gegeben ist. Eine naheliegende Verallgemeinerung ergibt sich dadurch, daß man die logische Sprache nicht nur zur Formulierung von Eigenschaften, sondern auch zur Spezifikation der Systeme selbst verwendet. Wenn durch logische Aussagen Eigenschaften des Systems *vorgegeben* werden, spricht man von einer *axiomatischen* Spezifikation. Viele logische Formalismen sind so ausdrucksstark, daß sich in ihnen eine Vielzahl von Systemtypen modellieren lassen.

Dabei kann (und wird) es der Fall sein, daß es mehr als ein Modell gibt, das die Beschreibung, d. h. die Axiome, „wahr“ macht (erfüllt). Dieser Effekt ist oft sogar erwünscht: Nebensächliche Eigenschaften können so ignoriert werden. Wenn man im Zusammenhang mit der formalen Programmentwicklung Modelle mit den möglichen Implementierungen assoziiert, bedeutet dies, daß hiermit die Möglichkeit besteht in den frühen Entwurfphasen *Implementierungsentscheidungen* offenzulassen.

Wenn man wie oben im Fall spezieller Systembeschreibungen wieder an der Frage der Gültigkeit von Aussagen interessiert ist, ergibt sich für eine Menge von Axiomen Γ, die die Systemspezifikation bilden, und eine gewünschte Eigenschaft φ die folgende Fragestellung: Ist es der Fall, daß jedes Modell $\mathcal{M}$, welches alle Axiome „wahr“ macht (d. h. $\mathcal{M} \models \psi$ für alle $\psi \in \Gamma$), auch φ „wahr“ macht (d. h. $\mathcal{M} \models \varphi$)? Wenn dies der Fall ist, sagt man: Die Aussage φ *folgt* (logisch) aus Γ (d. h. $\Gamma \models \varphi$). Der Folgerungsbegriff ist das fundamentale Konzept der mathematischen Logik.

Wenn man beispielsweise die Aussage ... *everybody loves my baby, but my baby only loves me* ... durch die prädikatenlogischen Formeln $\forall x.loves(x, baby)$ und $\forall y.(loves(baby, y) \rightarrow y = me)$ repräsentiert, folgt[1] $me = baby$, d. h. ... *I am my baby!* .... Überraschungen wie diese, meistens allerdings unerwünscht, da Sicherheitslücken offenbarend, sind beim Übergang zu formalen Systemspezifikationen häufig.

Der Folgerungsbegriff ist rein semantischer Natur, mathematische Entitäten wie die Klasse aller Modelle, die eine Spezifikation erfüllen, sind höchst *unkonstruktiver* Natur. Man versucht daher, den Folgerungsbegriff rein syntaktisch, d. h. durch die Manipulation von Formeln, zu beschreiben. Sogenannte *Kalküle* (oder Beweisverfahren) legen (syntaktische) Regeln fest, mit denen ausgehend von Axiomen Γ neue Formeln (Aussagen) ψ *abgeleitet* werden können, $\Gamma \vdash \psi$. Ein solcher Kalkül heißt *korrekt*, wenn gilt $\Gamma \vdash \psi \Rightarrow \Gamma \models \psi$. Er heißt *vollständig*, wenn gilt $\Gamma \models \psi \Rightarrow \Gamma \vdash \psi$. Da man im Zusammenhang mit der formalen Programmentwicklung an der *sicheren* Vorhersage von Eigenschaften interessiert ist, ist die Korrektheit der entscheidende Punkt. Vollständigkeit kann bei den meisten Fragestellungen in diesem Bereich nicht erreicht werden. In der Regel ist damit fast immer eine Situation gegeben, wo man im allgemeinen nicht garantieren kann, daß entweder ein Beweis oder eine Widerlegung gefunden wird.

Die Beweise, von denen hier die Rede ist, unterscheiden sich von den Beweisen in mathematischen Abhandlungen dadurch, daß jeder einzelne Schritt nach exakt vorgegebenen Regeln ausgeführt wird. Aus Gründen der *Beherrschbarkeit* aber auch aus Gründen der *Vertrauenswürdigkeit* ist daher bei der Durchführung

[1] In den Formeln ist die gleichzeitige Ersetzung von x und y durch $baby$ zulässig. Die angewendete Folgerungsregel wird als *Modus ponens* bezeichnet.

von Beweisen eine maschinelle Unterstützung unabdingbar.

Implementierte Beweisverfahren erlauben es, Aussagen über *alle* Elemente unendlicher Bereiche, z. B. über unendlich viele Berechnungen nachzuweisen. Bei vielen komplexen, d. h. praktisch unendlichen Systemen ist ihre Anwendung die einzige Möglichkeit, Eigenschaften zuverlässig vorhersagen zu können. Beim Nachweis von Eigenschaften solcher Systeme spielt die Technik des Beweisens *induktiver Theoreme* eine zentrale Rolle. Insbesondere im Zusammenhang mit Induktionsbeweisen sind oft kreative (Beweis-) Entscheidungen notwendig. Die Werkzeugunterstützung in diesem Bereich ist daher in fast allen Fällen so ausgelegt, daß während der Beweissuche Benutzerinteraktionen stattfinden können. Auf der anderen Seite müssen praktisch einsetzbare Deduktionswerkzeuge eine hohe „Eigenintelligenz“ aufweisen, um möglichst viele Routineschritte automatisch abarbeiten zu können und so auch große Beweise tatsächlich durchführbar zu machen.

4.3 Beweise von Sicherheitseigenschaften

Im Kontext von formalen Entwicklungsmethoden ergeben sich *Beweisprobleme* aus bestimmten *Entwicklungschritten*. Typische solche Schritte sind der Nachweis der Modellerhaltung (z. B. zwischen einer funktionalen und einer Anforderungsspezifikation), der Nachweis der Korrektheit von Verfeinerungen und der Nachweis der korrekten Aktualisierung von parametrischen Spezifikationen. Bei Methoden, die auf Transformationen (von Spezifikationen und Programmen) beruhen, treten Beweisprobleme als (semantische) Nebenbedingungen von Transformationsregeln auf.

Bei formalen Sicherheitsmodellen, wie sie von ITSEC und CC gefordert werden, gibt es *methodisch* gesehen zwei Quellen von Beweisproblemen:

1. der Nachweis der Gültigkeit der Sicherheitseigenschaften der modellierten Sicherheitspolitik (s. Abschnitt 3.4.4) und

2. der Nachweis der Übereinstimmung der Sicherheitspolitik mit der funktionalen Spezifikation (auf den Stufen E6 bzw. EAL7).

Technisch gesehen muß sich die Beweisverpflichtung in beiden Fällen nicht unbedingt als Modellerhaltungsproblem darstellen. Es kann z. B. durchaus sein, daß die Sicherheitspolitik eine komplexe generische Struktur hat und sich die Beweisverpflichtung aus der Instantiierung der generischen Begriffe im Rahmen einer Aktualisierung ergibt. Im Zusammenhang mit zustandsbasierten Systemen gibt es auch Beispiele, bei denen Verfeinerungen eingesetzt werden. Ebenso können beim Übergang zur funktionalen Spezifikation verschiedene Nachweistechniken eingesetzt werden.

4.4 Widerspruchsfreiheit

In ITSEC und CC wird gefordert, die *Konsistenz* des formalen Sicherheitsmodells nachzuweisen (s. Abschnitt 3.4.4). Im Zusammenhang mit Logikformalismen bedeutet Konsistenz Widerspruchsfreiheit: es ist nicht möglich, daß für einen Satz φ beides gilt, $\Gamma \vdash \varphi$ und $\Gamma \vdash \neg\varphi$. Konsistenz ist damit ein Begriff, der auf der rein syntaktischen, kalkülmäßigen Ebene angesiedelt ist. Für die meisten Formalismen gilt, daß im Fall einer Inkonsistenz *alle* Formeln ableitbar sind. Damit ist klar: Wenn die Spezifikation der Sicherheitspolitik widersprüchlich ist, wird jeder darauf aufbauende Nachweis einer Sicherheitseigenschaft bedeutungslos.

Ein Problem ergibt sich nun daraus, daß die Konsistenz nicht eine einfache Systemeigenschaft ist, die selbst in dem gegebenen Rahmen (Formalismus + Axiome) nachgewiesen werden kann. Vielmehr benötigt man hierzu eine geeignete und mächtigere Metaebene, für die sich dann natürlich wiederum das Konsistenzproblem stellt. Im Folgenden wird diskutiert, welche Maßnahmen möglich sind, um Konsistenz, soweit wie theoretisch und praktisch möglich, zu gewährleisten.

Um Konsistenz im obengenannten Sinn zu erreichen, muß zunächst der zugrunde gelegte Ableitungsapparat (und dessen Implementierung) korrekt sein. Diese Fragestellung ist von dem individuellen Entwicklungsvorgang unabhängig. Der implementierte Grundformalismus sollte, wenn überhaupt, vom Endbenutzer nur in einer kontrollierten Weise abgeändert werden können, die sicherstellt, daß keine neuen Formeln ableitbar werden und die Korrektheit damit erhalten bleibt. Das Korrektheitsproblem reduziert sich dann darauf, *ein für allemal* den verwendeten Grundformalimus zu „verifizieren“. Dabei kann teilweise formal vorgegangen werden, z. B. indem speziellere Formalismen in einen umfassenderen formalen Rahmen, der dann jedoch vorausgesetzt wird, eingebettet werden. Letztlich ist dieses Problem jedoch nicht vollständig formal lösbar. Wichtig ist in diesem Zusammenhang, daß Grundformalismen verwendet werden, die in der wissenschaftlichen Gemeinschaft verbreitet und anerkannt sind, so daß Fragen der Korrektheit einer „gesellschaftlichen“ Kontrolle unterliegen, wie es in anderen Gebieten der Mathematik auch der Fall ist. Man beachte, daß ja auch für die Methoden anderer Ingenieurdisziplinen die *Grundlagenproblematik der Mathematik* in gleicher Weise besteht.

Bei einer speziellen Entwicklung stellt sich für den Benutzer einer formalen Entwicklungsmethodik, bzw. eines entsprechenden Werkzeugs, die Frage, ob die von ihm angegebene axiomatische Spezifikation, etwa der Sicherheitspolitik, widerspruchsfrei ist. Man beachte, daß die Konsistenz alleine natürlich nicht sicherstellt, daß eine Spezifikation sinnvoll ist. Eine *formale Reflexion* der Spezifikation durch den Nachweis von Eigenschaften hat jedoch die Konsistenz als unabdingbare Voraussetzung.

Es gibt eine Reihe von Vorgehensweisen zum Nachweis der Konsistenz benut-

zerdefinierter Spezifikationen, die auch miteinander kombiniert werden können.

Für bestimmte Spezifikationstechniken, die einem vorgegebenen *Schema* folgen, ist bekannt, daß sie *immer* zu konsistenten Axiomensystemen führen. Hierzu gehören Spezifikationen (nur) mit Gleichungen oder Hornformeln und Schemata zur Spezifikation von frei erzeugten Datentypen (natürliche Zahlen, Listen, Bäume). In allen diesen Fällen ist kein problemspezifischer Konsistenznachweis notwendig: Wie bei der Korrektheit des Grundformalismus gibt es von der einzelnen Anwendung unabhängige (Meta-)Resultate, für die die oben gemachten Bemerkungen sinngemäß gelten.

Ein grundlegender Ansatz in der Mathematik ist es, existierende (konsistente) Theorien durch neue *Definitionen* zu erweitern. Leider ist es bei vielen Formalismen nur sehr eingeschränkt möglich, neue Begriffe durch Definitionen im engeren Sinn, d. h. solchen, bei denen der neue Begriff in dem definierenden Ausdruck nicht auftaucht, einzuführen. Im Bereich der formalen Programmentwicklung spielen *rekursive* Definitionen eine zentrale Rolle. Diese sind oft auf einer sehr abstrakten Ebene konstruktiv (algorithmisch). Um die Konsistenz zu erhalten, folgen solche Erweiterungen wieder einem vorgegebenen allgemeinen Schema. Zusätzlich ist es jedoch oft notwendig, bestimmte problemspezifische *Beweisverpflichtungen* zu erfüllen. Etwa im Fall rekursiv definierter Funktionen geht es bei diesen Beweisverpflichtungen z. B. um die „Terminierung“ (von abstrakten Algorithmen).

Wenn ein höherer Abstraktionsgrad, als er bei konstruktiven Spezifikationen möglich ist, angestrebt wird, müssen allgemeinere Techniken zur Anwendung kommen. Eine axiomatische Spezifikation Γ ist konsistent, wenn ein Modell $\mathcal{M}$ existiert, das diese Spezifikation erfüllt. Wie oben schon kurz angedeutet wurde, kann man oft *Implementierungen* als Modelle auffassen. Ein Weg die Konsistenz von Γ nachzuweisen besteht dann darin, die Spezifikation Γ solange zu *verfeinern*, bis eine Ebene erreicht wird, die bekanntermaßen (s. o.) konsistent ist. Dieser Weg setzt eine formale Methodik voraus, die einen (geeigneten) Verfeinerungsbegriff kennt. Während der Verfeinerung entstehen Beweisverpflichtungen, die zu erfüllen sind, wenn man die Konsistenz garantieren will. Diese allgemeine Methode erfordert Kreativität bei der Angabe der Verfeinerungsschritte. Allerdings ist eine wirklich effiziente Implementierung hier nicht notwendig: es handelt sich mehr um eine prototypische Implementierung.

Kapitel 5

Erstellung formaler Sicherheitsmodelle

Entsprechend den in Kap. 3.4 herausgearbeiteten Anforderungen werden in den folgenden Abschnitten Hinweise und Empfehlungen zur Erstellung der Bestandteile formaler Sicherheitsmodelle gegeben. Zur Verdeutlichung der Ausführungen werden Beispiele verwendet, die durchgängig auf ein veröffentlichtes formales Sicherheitsmodell Bezug nehmen. Dabei handelt es sich um ein generisches Sicherheitsmodell für die mit dem deutschen Signaturgesetz konforme Erzeugung digitaler Signaturen mit Hilfe einer SmartCard [12]. Es ist zusammen mit generischen Sicherheitsvorgaben [11] für eine Evaluierung[1] der Signaturapplikation von entsprechenden SmartCards auf Initiative des TeleTrusT e. V. erstellt worden. Sein generischer Charakter ist durch die Unabhängigkeit von konkreten SmartCards einschlägiger Hersteller gerechtfertigt. Als gemeinsame Basis von Modellierung und Realisierung diente die standardisierte Schnittstellenspezifikation DIN V 66291-1 [10].

Für die Erstellung des o. g. formalen Sicherheitsmodells wurde das hierfür vom Bundesamt für Sicherheit in der Informationstechnik (BSI) zugelassene *Verification Support Environment* (VSE) verwendet. Es ist ein komplexes Werkzeug für die Anwendung formaler Methoden in der Entwicklung beweisbar korrekter Software. VSE wird vom BSI offiziell für die Software-Entwicklung von IT-Produkten mit den höchsten Qualitätsstufen gemäß ITSEC und Common Criteria empfohlen. In verschiedenen Projekten hat VSE seine Praxistauglichkeit unter Beweis gestellt. Die in den folgenden Abschnitten zitierten Beispiele sind in der von VSE verwendeten Spezifikationssprache formuliert. Weil alle Beispiele aus ihrem jeweiligen Zusammenhang verständlich sind, soll hier lediglich in wenigen Stichworten auf die Grundprinzipien von VSE eingegangen werden.

[1] Angestrebte Evaluationsstufe: ITSEC E4

Das VSE System stellt verschiedene Möglichkeiten zur Strukturierung von Spezifikationen zur Verfügung. Es unterstützt den kompletten Entwicklungsprozeß von der abstrakten Spezifikation der Eigenschaften eines Produktes bis zur automatischen Code-Erzeugung. Formale Entwicklungen mit VSE sind in einer Umgebung gespeichert, die den Benutzer führt und einen konsistenten Zustand des sog. Entwicklungsgraphen aufrecht erhält. Ein integriertes Deduktionssystem stellt die benötigte Beweisunterstützung für die Verifikation der während einer Entwicklung entstehenden Beweisverpflichtungen zur Verfügung. VSE zeichnet sich durch eine ausgereifte Methodologie zur Behandlung verteilter Systeme aus [22, 25]. Seine Spezifikationssprache vereint klassische abstrakte Datentypen auf der Basis von Prädikatenlogik mit einer Variante von TLA (Temporal Logic of Actions) [37].

5.1 Hintergrund des Modellierungsbeispiels

Zur Illustration der Bestandteile formaler Sicherheitsmodelle betrachten wir die Modellierung der Sicherheitspolitik einer betriebsbereiten SmartCard zur Erzeugung digitaler Signaturen [10, 11, 12]. Der Begriff „betriebsbereit“ bedeutet insbesondere, daß der Herstellungs- und Personalisierungsprozeß nicht betrachtet wird, d. h. der integrierte Schaltkreis auf der SmartCard enthält die Signaturapplikation und alle auf den Karteninhaber bezogenen Daten. Es wird angenommen, daß diese Daten von der Personalisierungsstelle sicher verarbeitet wurden und daß die SmartCard auf eine sichere Art und Weise dem rechtmäßigen Besitzer übergeben worden ist. Wir bezeichnen eine solche SmartCard im Folgenden als „Signaturkarte“.

Die digitale Signatur wird innerhalb der Signaturkarte unter Verwendung des geheimen Signaturschlüssels der Karteninhabers erzeugt. Dieser Schlüssel ist innerhalb der Signaturkarte in einem physikalisch geschützten Bereich des integrierten Schaltkreises gespeichert. Auf die Signaturkarte wird mit Hilfe eines Kartenlesers zugegriffen, der z. B. mit einem Personal Computer verbunden ist. Alle Geräte, die ggf. an der Kommunikation mit der Signaturkarte beteiligt sind (Kartenleser, Personal Computer, ...), bezeichnen wir im Folgenden zusammenfassend als „Kartenterminal“.

Die wesentliche Richtlinie der Sicherheitspolitik für Signaturkarten lautet: „Der rechtmäßige Kartenbesitzer ist der einzige autorisierte Benutzer der Signaturanwendung“. Keine andere Person darf, mit oder ohne Verwendung der Signaturkarte, eine dem Kartenbesitzer zurechenbare digitale Signatur erzeugen können. Zu diesem Zweck muß die Vertraulichkeit des geheimen Schlüssels auf der Signaturkarte gewährleistet sein. Der Zugriff auf den Signaturschlüssel darf nur mit der expliziten Zustimmung des Kartenbesitzers möglich sein.

5.2 Bestimmung des Abstraktionsgrades

Die Konstruktion formaler Sicherheitsmodelle verfolgt mehrere Ziele, die teilweise in scheinbar entgegengesetzten Richtungen liegen. Verantwortlich für diese Situation ist die Rolle formaler Sicherheitsmodelle als Bindeglied zwischen den funktionalen Anforderungen einerseits und der funktionalen Spezifikation andererseits. Während die funktionalen Anforderungen festlegen, *was* der EVG leisten soll, beschreibt die funktionale Spezifikation, *wie* diese (Sicherheits-)leistung erbracht wird. Die Herausforderung besteht darin, diese häufig recht große Distanz mit einer tragfähigen Konstruktion des formalen Sicherheitsmodells zu überbrücken.

Gelungene formale Sicherheitsmodelle sind gekennzeichnet von einer ausgewogenen Darstellung der Sicherheitspolitik. Die Qualität der Balance zwischen Sicherheitseigenschaften und -merkmale läßt sich direkt feststellen an den erforderlichen Aufwänden für

1. den Nachweis der Übereinstimmung zwischen Sicherheitsmodell und der in den funktionalen Anforderungen festgelegten Sicherheitspolitik,

2. den formalen Beweis der Gültigkeit der Sicherheitseigenschaften auf der Basis der Sicherheitsmerkmale und

3. den (ggf. formalen) Nachweis der Übereinstimmung zwischen Sicherheitsmodell und der funktionalen Spezifikation.

Die zentrale Aufgabe besteht in der Bestimmung des Abstraktionsgrades von Sicherheitseigenschaften einerseits und Sicherheitsmerkmalen andererseits. Sie ist typischerweise nicht schematisch zu lösen, sondern eher durch einen kreativen Prozeß, in dessen Verlauf die Konstruktion (häufig mehrfach) verändert wird. Dieser Aspekt der Erstellung formaler Sicherheitsmodelle kann im vorliegenden Leitfaden nur unzureichend erfaßt werden. Insbesondere müssen die in diesem Kapitel beschriebenen Bestandteile formaler Sicherheitsmodelle notwendigerweise in einer bestimmten Reihenfolge präsentiert werden. Die gewählte Reihenfolge dient zwar dem Verständnis des gewählten Beispielmodells, soll und kann aber nicht als Empfehlung für einen reinen Top-Down-Entwurfsprozeß verstanden werden.

Neben der Qualität der Balance zwischen Sicherheitseigenschaften und Sicherheitsmerkmalen hat noch ein weiterer Aspekt entscheidenden Einfluß auf die Konstruktion des formalen Sicherheitsmodells, nämlich die Sichtbarkeit der Bedrohungen und Angriffstrategien. Dieser Einfluß äußert sich auf subtile Weise bereits in der formalen Sprache, in der das Modell formuliert ist. Wenn nämlich die der formalen Spezifikation der Sicherheitspolitik zugrundeliegenden Theorien

und Strukturen bereits die Existenz unsicherer Zustände (im Sinne wirksam gewordener Bedrohungen) ausschließen, dann kann das formale Sicherheitsmodell seiner Aufgabe nicht gerecht werden.

Beispiel. *In einer SmartCard mit Signaturapplikation werden verschiedene Daten verarbeitet, die gänzlich unterschiedlichen Zwecken dienen. Dazu gehören zumindest signierte Benutzerdaten, Signaturschlüssel, -zertifikate und (Pseudo-) Zufallszahlen sowie Kommunikationsdaten (Kommandos, Nachrichten, etc.). Typischerweise sind diese Daten ohne ihren jeweiligen Kontext, der durch das Kommunikationsprotokoll und die Struktur der ausgetauschten Datensätze vorgegeben ist, nicht unterscheidbar.*

Da der Kontext bekannt ist, spricht vordergründig nichts dagegen, die verschiedenen Daten durch unterschiedliche abstrakte Datentypen im formalen Modell zu repräsentieren. Allerdings fordern die Sicherheitsprinzipien, wie in Abschnitt 5.3 beschrieben, daß der auf der Karte gespeicherte private Schlüssel nicht preisgegeben wird. Angriffe gegen die Vertraulichkeit des privaten Schlüssels, die auf einer Verwechslung bzw. Identifikation von Signaturschlüsseln und bspw. Zufallszahlen beruhen, wären von einer Modellierung, die diese Datentypen trennt, nicht zu erfassen. Aus diesem Grund sind in [12] alle relevanten Daten in dem abstrakten und unstrukturierten Datentyp `information` *zusammengefaßt.*

Weitere Aspekte der Sichtbarkeit von Bedrohungen, die z. B. bei der Festlegung der Sicherheitsarchitektur und Beschreibung der (Modul-) Schnittstellen auftreten, werden in den Beispielen der folgenden Abschnitte aufgegriffen. Zusammenfassend soll hier betont werden, daß wohldefinierte formale Theorien das Fundament für aussagekräftige formale Sicherheitsmodelle bilden. Typischerweise sind sowohl die Sicherheitseigenschaften als auch die Sicherheitsmerkmale auf der Grundlage der gleichen Theorien formuliert (s. Abb. 5.1). Ihre Auswirkungen sind also weitreichend und ihrem Entwurf sollte deshalb große Aufmerksamkeit gewidmet werden.

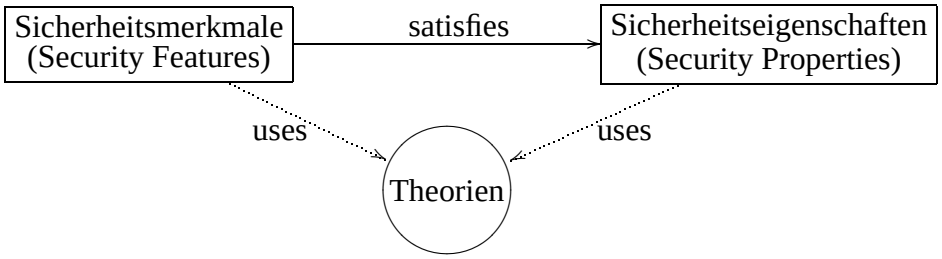


Abbildung 5.1: Typische Grundstruktur eines formalen Sicherheitsmodells

5.3 Sicherheitseigenschaften / Security Properties

Die Sicherheitseigenschaften repräsentieren, wie in Kap. 3.4.2 erläutert, den Teil des formalen Sicherheitsmodells, der mit den Sicherheitsprinzipien der EVG-Sicherheitspolitik übereinstimmen muß. Die funktionalen Anforderungen beeinflussen in entscheidender Weise den Charakter des formalen Sicherheitsmodells. Sie bestimmen die Wahl der wesentlichen Grundelemente und Modellierungskonzepte. Auch wenn es, aus jeweils unterschiedlichen Gründen, nicht immer möglich ist, sollte beim Entwurf der formalen Sicherheitseigenschaften eine möglichst vollständige Abdeckung der in den funktionalen Anforderungen formulierten Sicherheitsprinzipien angestrebt werden.

Beispiel. *Für die beschriebenen Signaturkarten wurden aus den Sicherheitszielen eine Reihe von funktionalen Anforderungen abgeleitet. Eine Auswahl dieser Anforderungen ist hier unter Verwendung der Bezeichnungen aus [11] wiedergegeben:*

(SO1.1) *Die Signaturkarte soll die Extraktion des geheimen Schlüssels des Kartenbesitzers aus dem geschützten Bereich verhindern.*

(SO1.2) *Die Signaturkarte soll die Modifikation des geheimen Schlüssels des Kartenbesitzers im geschützten Bereich verhindern.*

(SO2.1) *Die Signaturkarte soll die Verwendung der Signaturfunktion nur dem Kartenbesitzer nach dessen erfolgreicher Authentisierung durch Wissen erlauben.*

(SO2.2) *Wiederholte Authentisierungsfehler werden von der Signaturkarte als versuchter unautorisierter Zugriff interpretiert und führen zur Sperre der Signaturfunktion.*

(SO2.3) *Die Authentisierungsdaten sind in der Signaturkarte gespeichert und sollen nicht preisgegeben werden.*

(SO7.1) *Die Signaturkarte stellt eine Funktion zur Verfügung, die den geheimen Signaturschlüssel des Kartenbesitzers zur Erzeugung einer mit dem Signaturgesetz konformen digitalen Signatur für die ihr vom Terminal vorgelegten Daten verwendet.*

(SO7.2) *Die Funktion zur Erzeugung einer mit dem Signaturgesetz verträglichen digitalen Signatur arbeitet in einer Weise, daß andere Personen, die nicht in Besitz des geheimen Signaturschlüssels des Kartenbesitzers sind, die digitale Signatur nicht erzeugen können.*

Die Analyse der obigen Anforderungen ergibt folgende Festlegungen für den Entwurf des formalen Sicherheitsmodells:

- *Die Verhinderung von Extraktion des geheimen Signaturschlüssels (SO1.1) und Preisgabe der Authentisierungsdaten (SO2.3) wird durch Forderungen*

an die Modellierung der Schnittstelle der Signaturkarte (Ein- und Ausgabeströme) umgesetzt.

- *Die Verhinderung von Modifikation des geheimen Schlüssels (SO2.1) wird durch Datenkapselung und Zugriffskontrolle modelliert.*
- *Damit die Verwendung der Signaturfunktion nur nach erfolgreicher Authentisierung (SO2.1) möglich ist, wird eine von äußeren Ereignissen beeinflußte Veränderung von Zugriffsrechten modelliert.*
- *Zur (dauerhaften) Sperre der Signaturfunktion (SO2.2) werden persistente Betriebszustände modelliert.*
- *Die Notwendigkeit zum Besitz des Signaturschlüssels für die Erzeugung echter Signaturen (SO7.2) wird durch die Modellierung von Subjekten und ihres (veränderlichen) Wissens umgesetzt.*

Die im Beispiel begonnene Anforderungsanalyse ist typisch für den Entwurf formaler Sicherheitsmodelle. Sie dient dazu, die wesentlichen Begriffe, Elemente und Strukturen für die Modellierung zu erfassen.

Insbesondere wird auf ihrer Grundlage entschieden, ob ein bekanntes generisches Sicherheitsmodell verwendet werden kann. Dies kommt häufig bei Anforderungen aus den Bereichen Zugriffs- oder Informationsflußkontrolle in Frage. Für eine vergleichende Darstellung solcher generischer Sicherheitsmodelle sei auf Kap. 6 verwiesen.

Der häufig weitreichende Einfluß der Anforderungsanalyse soll an der Beispielmodellierung weiter illustriert werden.

Beispiel. *Bei sorgfältiger Analyse der o. g. Sicherheitsanforderungen ist festzustellen, daß SO7.2 eine Anforderung darstellt, die von der Signaturkarte während ihres Betriebes nicht beeinflußt werden kann. Sie formuliert Bedingungen an die Auswahl und Implementierung der kryptographischen Algorithmen und ist daher im engeren Sinne nicht Bestandteil der Sicherheitspolitik des EVG. Dennoch beeinflußt SO7.2 das formale Sicherheitsmodell erheblich, da solche Anforderungen an die Eigenschaften bestimmter Algorithmen auf geeignete Weise axiomatisch erfaßt werden müssen.*

Wie bereits ausgeführt, verlangt diese Bedingung nach Ausdrucksmöglichkeiten für die Modellierung von Subjekten und ihrem (veränderlichen) Besitz an Informationen. Hierzu wird zunächst der abstrakte und unstrukturierte Datentyp `subject` *eingeführt. Mit Hilfe der Funktion* `learns` *und der Prädikate* `knows` *und* `inferable` *(sowie einiger Hilfsfunktionen und -prädikate) wird die Theorie* `TInformation` *konstruiert, die es gestattet, die Anforderung SO7.2 in einem Axiom zu erfassen (vgl.* `TSignature` *in Abschnitt 5.5.2):*

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

Die anschauliche Interpretation dieses Axioms besagt, daß die Herleitbarkeit des Signaturschlüssels `sk` *eine notwendige Voraussetzung für die Herleitbarkeit (Erzeugung) des signierten Dokuments* `sig(i,sk)` *ist. Diese anschauliche Interpretation stimmt nur dann mit der Semantik der formalen Spezifikation überein, wenn die Definition der beteiligten Prädikate die Anschauung widerspiegelt. Dafür ist in der nachfolgend ausschnittweise wiedergegebenen Spezifikation der Theorie* `TInformation` *Sorge getragen.*

```
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
```

Die in der Anforderungsanalyse identifizierten Modellierungskonzepte dienen in einer späteren Phase als Grundlage für die Interpretation [4, Abs. E[456].3] bzw. Erklärung [1, Element ADV_SPM.3.3C]. Als oberste Ebene der Modellierung sind aber zunächst Sicherheitseigenschaften zu spezifizieren, deren Gesamtheit als Formulierung der Sicherheitsprinzipien betrachtet wird. Bei Verwendung generischer Modelle ergeben sich die Sicherheitseigenschaften aus einer geeigneten Instantiierung der generischen Eigenschaften (s. Kap. 6).

Beispiel. *Die Sicherheitseigenschaften für Signaturkarten sind im Rahmen einer temporallogischen Spezifikation formuliert, deren Ein-/Ausgabevariablen* `channelIn` *und* `channelOut` *zu den Standardschnittstellen von SmartCards korrespondieren.*

Neben den Schnittstellenvariablen besteht der Zustandsraum der temporallogischen Spezifikation aus den für Signaturkarten spezifischen Geheimnissen und anderen internen Kartendaten. Hierzu gehören insbesondere der private Schlüssel `skCh` *sowie die Authentisierungsdaten* `pinCh` *des Kartenbesitzers.*

Aufgrund der technischen Standards für die Kommunikation zwischen Terminal und SmartCard ist immer eine eindeutige Zuordnung der Reaktion der Signaturkarte zu einem vorher übertragenen Kommando möglich. Die formalen Sicherheitseigenschaften beruhen daher grundsätzlich auf einem einfachen Konzept: Wenn die Karte eine bestimmte Antwort auf ein Kommando schickt, dann ist die Antwort durch den internen Zustand der SmartCard gerechtfertigt.

Häufig erfordert die Formulierung der Sicherheitseigenschaften die Einbeziehung der Sequenz von Transaktionen, die zum aktuellen Zustand geführt hat. Als Folge davon muß ihre Formalisierung alle relevanten Teile der vollständigen Historie des aktuellen Zustandes beobachten können. Der Zugriff auf diese Historie wird technisch mit Hilfe der Zustandsvariable `channelOutRawH` *zur Verfügung gestellt. Diese Variable enthält alle Ausgaben der SmartCard seit ihrer Initialisierung in chronologischer Reihenfolge.*

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

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

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

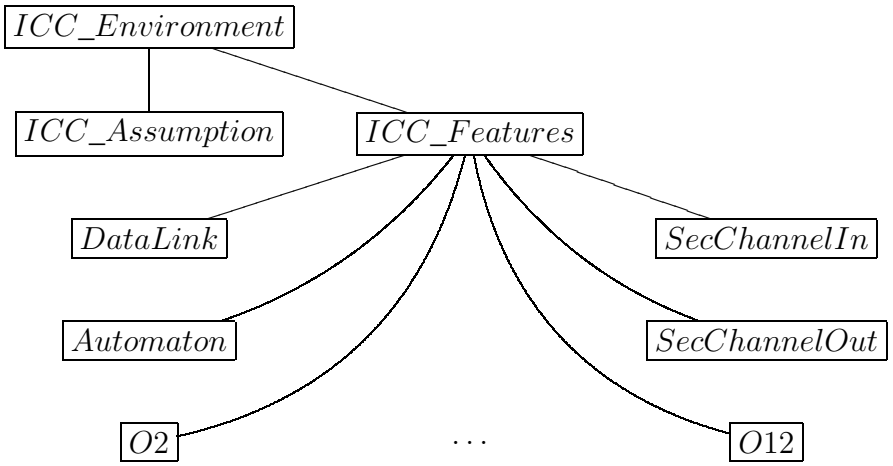
```
/* SO2.1 */
[]  ALL j:
     mInferableWithout(channelOut,
                       def(sig(j,skCh),def(skCh))
     -> histAuthUser(channelOutRawH)

/* 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 Sicherheitsmerkmale / Security Features

Die Sicherheitsmerkmale repräsentieren, wie in Kap. 3.4.2 erläutert, den Teil des formalen Sicherheitsmodells, der mit den Sicherheitscharakteristika der EVG-Sicherheitspolitik übereinstimmen muß. Zur Vorbereitung des Nachweises der Übereinstimmung des formalen Sicherheitsmodells mit der funktionalen Spezifikation empfiehlt sich die Ausrichtung der Spezifikation der Sicherheitsmerkmale auf die dort festgelegten Prozeduren und Schnittstellen der Sicherheitsfunktionen. Wenn ein bekanntes generisches Sicherheitsmodell (vgl. Kap. 6) verwendet wird, muß entschieden werden, auf welche Weise das Modell zu instantiieren ist, um diesem Aspekt gerecht zu werden.

Beispiel. *Die Sicherheitsmerkmale von Signaturkarten beruhen im Wesentlichen auf dem Konzept einer Zustandsmaschine (Modul* `Automaton`*), die den Zugriff auf die zu schützenden Werte innerhalb der SmartCard steuert. Die Werte selbst werden in jeweils eigenständigen Modulen (*`O2 ... O12`*) gekapselt. Weitere Module sind für die interne Kommunikation (*`DataLink`*) zwischen den Modulen bzw. für die Ein-/Ausgabeschnittstellen der SmartCard (*`SecChannelIn`* und* `SecChannelOut`*) zuständig. Alle Module werden in der übergeordneten Spezifikation* `ICC_Features` *miteinander kombiniert (die restlichen Module der Abbil-*

dung werden im weiteren Verlauf erläutert):



Die formale Spezifikation der Sicherheitsmerkmale soll alle für die Gültigkeit der Sicherheitseigenschaften relevanten Aspekte des EVG und seiner Umgebung erfassen. Insbesondere der Sichtbarkeit von Bedrohungen, die die Gültigkeit der Sicherheitseigenschaften in Frage stellen, ist besondere Aufmerksamkeit zu widmen. Wenn die Sicherheitsmerkmale wie im Beispiel auf dem Konzept einer Zustandsmaschine beruhen, können die Bedrohungen auf direkte Weise mit den unsicheren Zuständen der Maschine charakterisiert werden.

Beispiel. *Das Modul* `Automaton` *spezifiziert die der Ausführung von Kommandos zugrundeliegende Bearbeitungsschleife:*

```
WHILE true DO BEGIN
   call(bckSecChannelIn);
   call(bckObjects);
   getEvent;
   stateChange;
   call(bckObjectReset);
   call(bckSecChannelOut)
END
```

Durch die Prozedur `stateChange` *werden die von der Zustandsmaschine ausgeführten Zustandsübergänge spezifiziert. Es ist zu gewährleisten, daß auf diese Weise nur sichere Zustände erreicht werden können. Ein unsicherer und daher nicht erreichbarer Zustand ist z. B. die Freigabe der Signaturfunktion ohne vorherige Benutzerauthentisierung. Zur Spezifikation der Prozedur* `stateChange` *wird die Funktion* `nextstate` *verwendet, die aus dem aktuellen Zustand und dem aktuellen Ereignis (von* `getEvent` *berechnet) den Folgezustand bestimmt. Alle erlaubten Zustandsübergänge sind in Form einer zweidimensionalen Tabelle (Matrix) kodiert (vgl. [11, Tab. 12]).*

Häufig werden zum Beweis der Gültigkeit der Sicherheitseigenschaften auf Basis der Sicherheitsmerkmale Annahmen an die Sicherheitsumgebung des EVG benötigt, die im formalen Sicherheitsmodell explizit spezifiziert werden müssen. Insbesondere im Bereich der Vertraulichkeit ist typischerweise davon auszugehen, daß zu schützende Werte außerhalb des EVG nicht bekannt sind.

Beispiel. *Der private Signaturschlüssel ist außerhalb der Signaturkarte nur bekannt, wenn er aus ihren Ausgaben extrahiert werden kann. Es kann daher insbesondere angenommen werden, daß er nicht in den Eingaben für die Signaturkarte enthalten ist. Diese Annahme ist für den Beweis der Nichtableitbarkeit des privaten Schlüssels aus den Ausgaben der Signaturkarte (SO1.1) essentiell. Sie ist daher im Modul* `ICC_Assumption` *explizit formalisiert.*

Durch die Kombination von `ICC_Assumption` *mit* `ICC_Features` *im Modul* `ICC_Environment` *werden diese Beschränkungen an den Eingabekanal der Signaturkarte gebunden.*

Generell wird empfohlen, solche (expliziten) Annahmen an die Sicherheitsumgebung mit großer Sorgfalt zu spezifizieren. Sollten Sie nämlich nicht zutreffen, ist die Aussagekraft der Beweise für die Gültigkeit der Sicherheitseigenschaften gefährdet.

5.5 Eigenschafts- und Konsistenzbeweise

Mit der Spezifikation von Sicherheitseigenschaften und -merkmale sowie deren Interpretation (ITSEC) bzw. Erklärung (CC) ist die Erstellung eines formalen Sicherheitsmodells noch nicht abgeschlossen. Auf die Notwendigkeit der Verifikation des Modells (vgl. Abschnitt 3.4.4) wurde bereits an verschiedenen Stellen in den vorangegangenen Abschnitten hingewiesen. Die Eigenschafts- und Konsistenzbeweise sind unerläßlich für die Aussagekraft eines formalen Sicherheitmodells, denn erst die Präzision mathematischer Beweise gestattet die Schlußfolgerung auf die Wirksamkeit der modellierten Sicherheitspolitik.

5.5.1 Eigenschaftsbeweise

Die Beweise der Gültigkeit der Sicherheitseigenschaften stellen das charakteristische Merkmal formaler Sicherheitsmodelle dar. Zusammen mit dem (notwendigerweise informellen) Nachweis der Übereinstimmung mit den Sicherheitsprinzipien und -charakteristika bieten sie die Gewähr, daß die dem EVG zugrundeliegende Sicherheitspolitik die gewünschte Qualität erreicht.

Generell wird empfohlen, die erforderlichen Beweise mit einem geeigneten Werkzeug zur (teilautomatischen) Beweisunterstützung durchzuführen. Insbesondere die Erzeugung der Beweisverpflichtungen aus der Spezifikation der Sicherheitseigenschaften und -merkmale sollte vollautomatisch durch die Beweisunterstützung erfolgen. Das vom BSI zugelassene Verification Support Environment (VSE) erfüllt diese Forderung.

Die konkreten Beweisverpflichtungen sind im allgemeinen sehr verschieden. Sie hängen in starkem Maße von den gewählten Spezifikationsformalismen ab. Häufig wird hierbei das Konzept des Beweises durch mathematische Induktion verwendet. Dies gilt insbesondere für die in Kap. 6 behandelten generischen Sicherheitsmodelle. Grundlage für solche Beweise bilden rekursiv definierte Datenstrukturen in prädikatenlogischen bzw. zeitliche Abfolgen von Zuständen in temporallogischen Spezifikationen.

Beispiel. *Zum Beweis der Nichtableitbarkeit des privaten Schlüssels aus den Ausgaben der Signaturkarte müssen sämtliche mögliche Zustandsfolgen betrachtet werden. Für einen beliebigen erreichbaren Zustand ist zu beweisen, daß sich die Gültigkeit der entsprechenden Sicherheitseigenschaft von diesem Zustand auf alle seine durch die Zustandsmaschine spezifizierten (direkten) Folgezustände übertragen läßt.*

Typischerweise sind die ersten Beweisversuche erfolglos, weil die Spezifikation fehlerhaft ist. Die modifizierte Spezifikation führt zu erneuten Beweisversuchen. Durch diese Wechselbeziehung aus (fehlgeschlagener) Verifikation und Spezifikation werden immer mehr Fehler aus dem Sicherheitsmodell eliminiert.

Aber nicht nur mißlungene Beweisversuche liefern Hinweise auf Fehler in der Spezifikation. Insbesondere sehr schnelle Beweiserfolge mit kurzen oder trivialen Beweiszweigen lassen Lücken in der Spezifikation vermuten.

Beispiel. *Eines der Sicherheitsprinzipien für Signaturkarten besagt, daß die Ausgabe der sog.* `DisplayMessage` *nur unter bestimmten Umständen erfolgen darf. Während der ersten Beweisversuche konnte die Gültigkeit der zugehörigen Sicherheitseigenschaft sehr schnell nachgewiesen werden. Der Grund war, daß die* `DisplayMessage` *nie ausgegeben wurde, und die Gültigkeit der Eigenschaft damit auf triviale Weise gegeben war. Es wurde versäumt, hinreichende Bedingungen für die Ausgabe der* `DisplayMessage` *zu verlangen. Das unzureichend spezifizierte Verhalten hatte letztlich seine Ursache in einem Fehler in der Zugriffsmatrix, die im entscheidenden Zustand den (lesenden) Zugriff auf die gespeicherte* `DisplayMessage` *verweigerte.*

Mißerfolge bei Beweisversuchen rühren aber häufig auch von Fehlern in der Sicherheitspolitik selbst her, d. h. die Spezifikation des Sicherheitsmodells ent-

spricht zwar der Intention des Entwicklers, aber die Sicherheitseigenschaften können von den Sicherheitsmerkmalen nicht gewährleistet werden. An solchen Ergebnissen zeigt sich der Wert der Verifikation von formalen Sicherheitsmodellen, denn diese Art von Fehlern sind mit konventionellen Methoden schwer zu entdecken.

Beispiel. *Der Beweis der zu SO2.2 korrespondierenden Sicherheitseigenschaft konnte zunächst nicht geführt werden. Die sorgfältige Analyse des Beweisversuchs ergab, daß die von der Sicherheitspolitik intendierte Sperre der Signaturkarte bei wiederholten Authentisierungsfehlern nicht wirksam war. Ein Angreifer hätte beliebig viele Authentisierungsversuche durchführen können. Dieser gravierende Fehler blieb lange Zeit unentdeckt und wurde erst korrigiert, nachdem die Ursache für das Scheitern des Beweisversuchs aufgeklärt war.*

5.5.2 Konsistenzbeweise

Der Begriff der Konsistenz und die Problematik ihres Nachweises sind schon in Abschnitt 4 angesprochen worden. Die dort gemachten Ausführungen sollen hier systematisiert, vertieft und mit Beispielen unterlegt werden.

Zum Begriff der Konsistenz

Spezifikationen sind unabhängig davon, ob es sich um rein textuelle Darstellungen handelt oder um graphische Eingaben in ein Entwicklungssystem (wie VSE), *syntaktische* Objekte. *Semantisch* gesehen beschreiben sie gewisse mathematische Strukturen (Modelle). Dabei ist es bei axiomatischen Ansätzen oft durchaus intendiert, daß mehrere (essentiell) verschiedene Strukturen als Deutung möglich sind. Hier geht es um die Frage, wann eine syntaktische Beschreibung in sich sinnvoll ist, in dem Sinn, daß überhaupt eine Deutung möglich ist. Die bloße Möglichkeit einer Deutung, bedeutet natürlich nicht, daß auch die Intentionen des Spezfizierenden erfüllt sind.

Eine Spezifikation muß zunächst *syntaktisch zulässig* sein, d. h. bestimmte (syntaktische) Anforderungen erfüllen. Die geforderten Eigenschaften sind dabei immer *entscheidbar*, d.h. sie können von Werkzeugen automatisch überprüft werden. In Bezug auf Texte unterscheidet man in der Regel zwischen der kontextfreien Analyse von Eingaben und dem sogenannten Typcheck. Im ersten Fall wird z.B. geprüft, ob jeder öffnenden Klammer auch eine schließende Klammer entspricht, während beim Typcheck auch geprüft wird, ob die Anzahl und Art der Argumente eines Funktionsaufrufs korrekt ist.

Bei manchen Beschreibungstechniken, insbesondere solchen, die sich auf ausführbare Systeme beschränken, ist mit der syntaktischen Zulässigkeit auch die

Möglichkeit einer semantischen Deutung und sinnvollen Analyse verbunden. Ein gutes Beispiel hierfür sind Programmiersprachen. Ein syntaktisch korrektes Java Programm ist ausführbar und hat damit eine Bedeutung. Dies heißt nicht, daß das betreffende Programm in einem weitergehenden Sinn *vernünftig* ist: Es kann z. B. durchaus sein, daß die Ausführung zu schweren Laufzeitfehlern führt und die Abarbeitung daher abgebrochen wird. Da die Beschreibung aber als solche sinnvoll (deutbar) ist, kann man es aber *analysieren* und dabei erwünschte oder unerwünschte Verhaltenseigenschaften (Laufzeit) herausfinden. Würde man, was durchaus möglich ist, ein solches Programm in eine logische Form übersetzen, wäre die entstehende Formelmenge insbesondere konsistent.

Bei der axiomatischen Beschreibung von Systemen im Rahmen von logischen Formalismen ist die Situation schwieriger. Hier kann man durch syntaktische Überprüfung im allgemeinen Fall nicht sicherstellen, daß eine Spezifikation als Ausgangspunkt für Beweise von intendierten Eigenschaften sinnvoll ist. Auch wenn alle Formeln ψ einer Formelmenge Γ wohlgeformt, d.h. als Sprachobjekte zulässig sind, besteht die Möglichkeit, daß man hat $\Gamma \vdash \varphi$ und $\Gamma \vdash \neg\varphi$. D. h. es sind sowohl φ wie auch $\neg\varphi$ aus Γ *folgerbar*. Die Menge $Gamma$ ist damit *widersprüchlich* oder *inkonsistent*. In den allermeisten Formalismen bedeutet dies, daß dann auch überhaupt *alle* Aussagen ϕ folgerbar sind. Es ist somit sinnlos, eine inkonsistente Formelmenge durch das Herleiten von Aussagen analysieren zu wollen. Für eine inkonsistente Formelmenge Γ kann man zudem keine mathematische Struktur (Modell) als Deutung angeben.

Eine Formelmenge, die (in diesem Sinn) *nicht* widersprüchlich (inkonsistent) ist, heißt *konsistent*. Man beachte, daß die Konsistenz ein Begriff ist, der auf der Kalkülebene ansetzt: Es ist kein Widerspruch *ableitbar*. Die Nichtexistenz von Modellen im Fall der Inkonsistenz folgt aus der (hier unterstellten) *Korrektheit* des verwendeten Formalismus.

In hinlänglich reichhaltigen Formalismen, wie z. B. der Prädikatenlogik, ist die Konsistenz einer Formelmenge im allgemeinen schwierig nachzuweisen. Für konstruktiv gegebene aber sonst beliebige Formelmengen Γ ist die Konsistenz weder auf Grundlage syntaktischer Kriterien entscheidbar, noch kann man sie auf *konstruktive* Weise durch endlich viele Beweisverpflichtungen Δ charakterisieren. Für solche Beweisverpflichtungen würde gelten: $\Gamma \models \phi$ für alle $\phi \in \Delta \quad \Leftrightarrow \quad \Gamma$ ist konsistent. Im allgemeinen Fall ist es nicht einmal möglich, *hinreichende* Bedingungen (Beweisverpflichtungen) zu erzeugen, die die praktisch vorkommenden Fälle auch nur annäherungsweise abdecken. Um die Konsistenz einer Formelmenge kalkülmäßig zu *beweisen*, benötigt man einen echt ausdrucksstärkeren Formalismus, dessen Konsistenz dann natürlich selbst wieder in Frage steht.

Ein gangbarer Weg für den Konsistenznachweis besteht darin, eine gegebene axiomatische Systembeschreibung (Formelmenge) schrittweise auf Formeln einer eingeschränkten Sprachklasse zu reduzieren, für die die Konsistenz immer gege-

ben oder aber durch syntaktische Überprüfungen oder hinreichende Bedingungen leicht zu etablieren ist. Im Laufe dieser Reduktion sind bestimmte Beweisverpflichtungen nachzuweisen. Es handelt sich bei dieser Vorgehensweise um eine prototypische Implementierung oder, aus logischer Sicht, um die Konstruktion eines Modells.

In korrekten Formalismen folgt aus der Existenz von Modellen die Konsistenz: Angenommen Γ ist inkonsistent, d.h. $\Gamma \vdash \varphi$ und $\Gamma \vdash \neg\varphi$. Wegen der Korrektheit des Kalküls gilt dann auch $\Gamma \models \varphi$ und $\Gamma \models \neg\varphi$, d.h. sowohl φ wie auch $\neg\varphi$ folgen aus Γ. Dies bedeutet, daß alle Strukturen $\mathcal{A}$, die (alle Formeln von) Γ erfüllen (wahr machen), auch φ und $\neg\varphi$ erfüllen. Gibt es nun eine solche Struktur (Modell), dann würde in diesem sowohl φ wie auch $\neg\varphi$ wahr sein, was nach dem Begriff des Erfülltseins von Formeln ($\mathcal{A} \models \varphi$) natürlich ausgeschlossen ist.

In der oben skizzierten Vorgehensweise liegt kein Widerspruch zu den Ausführungen des letzten Absatzes. Der Modellbildungsprozeß, der zur Erzeugung der Beweisverpflichtungen führt, erfordert Kreativität und ist nicht uniform, d. h. gemäß einer festen Vorschrift durchführbar. Bevor die Methodik in Unterabschnitt 5.5.2 an Hand von Beispielen erläutert wird, sollen zunächst Axiomatisierungen betrachtet werden, für die die Konsistenz leicht zu garantieren ist.

Konsistente Spezifikationen

Für bestimmte (stark) eingeschränkte Formelmengen ist die Konsistenz a priori gegeben. Hierzu gehören beispielsweise *reine Gleichungssysteme* (keine Ungleichungen) und *Hornformeln* (Regeln und Fakten) wie sie dem logischen Programmieren zugrunde liegen. Der Grund für die Konsistenz dieser Axiomatisierungen besteht vereinfacht gesagt darin, daß nur positive Information angegeben wird.

Ein weiteres typisches Beispiel für konsistente Formelmengen, das hier näher betrachtet werden soll, bilden Axiomatisierungsschemata für *frei erzeugte Datentypen*. Ein solcher Datentyp ist durch die Angabe von *Konstruktor- und Selektorsymbolen* und zusätzlich eventuell *Testprädikaten* gegeben. Zu diesen syntaktischen Bestandteilen kann man auf uniforme Weise Axiome so angeben, daß es (bis auf Isomorphie) genau ein Modell gibt, bei dem jedes (Datenelement durch einen Konstruktorterm dargestellt wird. Diese Darstellung ist obendrein eindeutig, d.h. für jedes Modellelement gibt es *genau* einen Konstruktorterm.

Der Basisdatentyp `NAT` (natürliche Zahlen) etwa ist in VSE durch folgende Spezifikation mit Namen `NATBASIC` gegeben:

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

Konstruktorsymbole sind in diesem Fall `NULL` (das Symbol für die Zahl null) und `SUCC` (das Symbol für die Nachfolgerfunktion). Als Selektorsymbol dient `PRED` (das Symbol für die Vorgängerfunktion). In der obigen Spezifikation steht das Zeichen „|“ für eine *Fallunterscheidung*. Die informelle Lesart der Spezifikation ist daher: *Jede natürliche Zahl ist entweder null oder der Nachfolger einer natürlichen Zahl, die sich durch Anwendung der Vorgängerfunktion ergibt.* Formal hat man die folgenden Formeln als Axiome:

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

Der Datentyp der natürlichen Zahlen ist *rekursiv* in dem Sinn, daß das Argument des Konstruktorsymbols `SUCC` wieder vom Typ `NAT` ist. Damit lassen sich unendlich viele Terme bilden, insbesondere solche der folgenden Bauart:

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

Diese variablenfreien Terme entsprechen eineindeutig den Elementen sogenannter *termerzeugter* Modelle, die für diese Art Spezifikationen immer existieren und aus logischer Sicht untereinander *ununterscheidbar* (isomorph) sind.

Der Aufbau komplexer Theorien in der Mathematik, wie etwa der axiomatischen Mengenlehre, geschieht u.a. dadurch, daß gegebene Strukturen durch die *Definition* neuer Begriffe *erweitert* werden. Hierbei bedient man sich bestimmter *Definitionsprinzipien*. In einfachen Fällen werden *Gleichungen* oder *Äquivalenzen* verwendet, bei denen die zu definierenden Größen auf der rechten Seite nicht vorkommen. Damit ist sichergestellt, daß die neuen Begriffe relativ zu den auf der rechten Seite verwendeten *wohldefiniert* sind (Existenz und Eindeutigkeit).

Diese einfache Vorgehensweise reicht im Zusammenhang mit den von uns betrachteten Spezifikationen, wie etwa der natürlichen Zahlen, nicht aus. Man benötigt *rekursive* Gleichungen (oder Äquivalenzen). Bei diesen kommt das zu definierende Symbol auch auf der rechten Seite vor. Die größere Mächtigkeit dieses Definitionsprinzips wird dadurch erkauft, daß zusätzliche Bedingungen einzuhalten sind, um etwa zirkuläre Definitionen auszuschließen und nach wie vor die Existenz und Eindeutigkeit der Erweiterungen zu garantieren. Im Zusammenhang mit der Frage nach der Konsistenz von Spezifikationen ist dabei nur die Frage nach der Existenz der Erweiterungen relevant.

Die folgenden Axiome beschreiben zusätzliche Funktionen und Prädikate als Erweiterung des oben definierten Datentyps `BASICNAT` (wir betrachten nur Addition (`_ + _`), Subtraktion (`_ - _`) und Multiplikation (`_ * _`)):

```
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
```

Diese Spezifikation erfüllt zwei Zusatzbedingungen:

- Für jeden Fall (`NULL | SUCC(y)`) für die Form des zweiten Arguments der jeweiligen Funktion gibt es genau eine Gleichung.
- Das zweite Argument der Funktionen auf der rechten Seite (also der Term „`y`“) ist im Sinne der oben angedeuteten Termstruktur echt *kleiner* als das entsprechende Argument auf der linken Seite der Gleichungen (also der Term „`SUCC(y)`“).

Sind diese Eigenschaften gewährleistet, so gibt es eine eindeutige Erweiterung der (termerzeugten) Modelle der Ausgangsspezifikationen. Insbesondere bleibt so die Konsistenz erhalten.

Die zweite Eigenschaft kann i. allg. nicht durch bloße syntaktische Inspektion etabliert werden. Vielmehr erzeugt man Aussagen (Beweisverpflichtungen), deren Gültigkeit (in termerzeugten Modellen) die Abnahme der Termkomplexität garantiert. In einer operationalen Lesart der Funktionsdefinition als (funktionale) Programme entspricht dies einem *Terminierungsnachweis*. Zusätzlich liefert eine solche zulässige Spezifikation ein passendes (konsistentes) *Induktionsprinzip* zum Nachweis von Eigenschaften der neu definierten Funktionen.

Die bisher geschilderte Vorgehensweise kann mit entsprechender syntaktischer Unterstützung auch zur konsistenten Einführung benutzerdefinierter Datenstrukturen verwendet werden. Hierauf wird unten im Zusammenhang mit dem dort behandelten Beispiel eingegangen.

Eine besondere Stellung nehmen Beschreibungstechniken ein, die auf der expliziten Angabe von *Zustandsübergangssystemen* beruhen. Abgesehen von der syntaktischen Zulässigkeit der Eingabe, die oft in grafischer Form erfolgt, sind in der Regel keine weiteren Maßnahmen notwendig, um die Konsistenz zu sichern, da ja unmittelbar ein Modell angegeben wird. Diese Techniken werden daher auch als *modellbasiert* bezeichnet.

Eine ähnliche Situation findet man bei zustandsbasierten axiomatischen Ansätzen, wie sie etwa VSE zu Grunde liegen. Auf Grund einer eingeschränkten Spezifikationstechnik, die sich im Wesentlichen auf die Angabe der in einem System möglichen Zustandsübergange (Aktionen) beschränkt, hängt die Konsistenz nur von der Existenz von Anfangszuständen ab, die der Spezifikation genügen. Dieses erfreuliche Ergebnis beruht darauf, daß inkonsistente Aktionsspezifikationen einfach niemals ausgeführt werden können.

Konsistenznachweis durch Modellbildung

Wenn die ursprünglich gegebene Spezifikation (Ausgangsspezifikation) nicht unmittelbar als konsistent nachgewiesen werden kann (etwa durch Überprüfung der Einhaltung der oben beschriebenen Prinzipien), bleibt nur die Möglichkeit der schrittweisen Implementierung (Verfeinerung) solange, bis für jeden Strang der modularen Entwicklung eine garantiert konsistente Spezifikation erreicht ist. Diesen Prozeß kann man als *Modellbildung* betrachten, da ja in jedem solchen Schritt die Korrektheit gezeigt werden muß. Die Gesamtheit der Korrektheitsnachweise bildet zusammen mit den oben diskutierten Konstruktionsprinzipien, denen die Ausgangsspezifikationen folgen, den Konsistenznachweis.

Diese Vorgehensweise soll an einem Beispiel aus dem formalen Sicherheitsmodell für Signaturkarten beschrieben werden. Die folgende mit Kommentaren versehene Spezifikation (VSE-Theorie) beschreibt den Vorgang der *Erzeugung einer Signatur*.

```
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
```

Die Axiomatisierung verwendet die vollständig un(ter)spezifizierte Theorie der *Informationen*. Sie folgt keinem vorgegebenen Prinzip und beschäftigt sich u. a. mit Charakteristika der Signierfunktion, wie der Injektivität, und der Möglichkeit Wissen über signierte Dokumente erlangen zun können. Diese Merkmale werden benötigt, um später die Gültigkeit von Sicherheitseigenschaften nachweisen zu können. So wird z. B. angenommen, daß es unmöglich ist, zufällig und ohne Kenntnis des Schlüssels Signaturen zu erzeugen. Die angegebenen Axiome sind im Gegensatz zu den oben aufgeführten Funktionsdefinitionen nicht *konstruktiv*, es geht um das *was* und nicht um das *wie*. Durch Vermeidung unnötiger Details vereinfachen sich die Beweise von Sicherheitseigenschaften drastisch, praktisch werden sie erst hierdurch möglich.

Auf der anderen Seite bergen Spezifikation wie die oben angegebene in besonderem Maße die Gefahr der Inkonsistenz. Die Forderung der ITSEC und CC nach einem Konsistenznachweis ist daher sehr berechtigt. In Fällen wie dem vorliegenden erfordert dieser allerdings Kreativität in dem Sinn, daß man nicht nach einem vorgegebenen Schema (uniform) vorgehen kann.

Zur Angabe eines Berechnungsmodells ist zunächst die Datenstruktur der Informationen zu konkretisieren. Die folgende Spezifikation führt mittels einer speziellen Syntax einen frei erzeugten Datentyp `iinformation` ein. Gemäß einer Fallunterscheidung in sechs Fälle stehen hierzu Konstruktoren, wie `pKey` (pulic key), und zugehörige Selektoren, wie `pKeyId`, zur Verfügung. Es wird ein außerhalb spezifizierter Datentyp `counter` verwendet. So besteht ein public key im Wesentlichen aus einem Datenobjekt vom Typ `counter`. An den letzten vier Fällen sieht man, daß es sich um einen rekursiven Datentyp handelt, da das erste Argument der Konstruktorfunktionen wieder vom Typ `iinformation` ist. Nach dem Schlüsselwort `WITH` werden Testprädikate eingeführt, die zwischen den sechs möglichen Fällen unterscheiden.

```
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
```

Auf dieser frei erzeugten und somit konsistenten Datenstruktur kann man nun das Signieren konstruktiv, d.h. durch ein Programm, definieren. Das Programm rechnet auf Strukturen vom Typ `iinformation`. Es verwendet den Konstruktor `dInfo`, um zu gewährleisten, daß ein (geheimer) Signierschlüssel als zweites Argument der Funktion vorliegt. Zur Berechnung der Ausgabe vom Typ `iinformation` wird der Konstruktor `sInfo`, der oben als dritter Fall angegeben ist, benutzt. Die übrigen Arten von Information sind in anderen Zusammenhängen, die hier nicht betrachtet werden sollen, relevant.

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

Wie oben diskutiert, erhalten Programme oder (zulässige) Funktionsdefinitionen *immer* die Konsistenz der abstrakten (Import-)Datentypen, auf denen sie rechnen, wie hier `iinformation`. Allerdings waren ja mit der abstrakten Signierfunktion `sig`, die durch das Programm `i_sig` implementiert wird, axiomatische Festlegungen verbunden. Diese müssen als Eigenschaften des Programms `i_sig` nachgewiesen werden (Beweisverpflichtungen). Ein Axiom bezüglich `sig` lautete:

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

Aus diesem Axiom ergibt sich als zu zeigende Programmeigenschaft in der Dynamischen Logik:

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

In der Dynamischen Logik treten Programme als Bestandteile von Formeln auf. Für ein Programm π bedeutet $< \pi > \varphi$: *das Programm π terminiert und anschließend (im Endzustand) gilt φ*. In der Beweisverpflichtung oben wird neben `i_sig` noch ein weiteres Programm `i_validpair` verwendet, das als Implementierung von `validpair` ebenfalls Teil der Modellbildung ist. Damit ist die Beweisverpflichtung folgendermaßen zu lesen: *Wenn das Programm `i_validpair` mit der Ausgabe `t` (true) terminiert, dann kann es nicht sein, daß das Programm `i_sig` mit der Ausgabe `i_sig-res = inf` terminiert.*

Der Konsistenznachweis durch Modellbildung ist also eine Mischung von rein kreativen Schritten bei der Erfindung von Implementierungen und dem kalkülmäßigen Beweis von Aussagen, die vom System (automatisch) als Korrektheitsbedingungen erzeugt werden.

Methodik von Konsistenznachweisen

Beim Lesen der obigen Ausführungen könnte die Frage auftreten: Wenn schon in jedem Fall eine Modellbildung (Implementierung) notwendig ist, warum bewegt man sich dann nicht von Anfang an im Bereich garantiert konsistenter Spezifikationen? Wie ausgeführt, ist ein solcher Ansatz natürlich *technisch* möglich. *Methodisch* gesehen gibt es aber schwerwiegende Einwände gegen eine solche Vorgehensweise.

Da ist zunächst, wie schon angedeutet, die Erkenntnis, daß eine *abstrakte* und oft nicht *konstruktiv* gegebene Ebene, wie in unserem Beispiel die Theorie `TSignature`, als Ausgangspunkt des Requirements Engineering auf jeden Fall notwendig ist. Die abstrakte Ebene sollte auch immer eine eigene Existenz als Spezifikation haben. Somit bleibt nur die Frage nach der *Reihenfolge*: Soll die abstrakte Ebene als erstes entwickelt werden und danach ein Konsistenznachweis durchgeführt werden oder sollen diese Eigenschaften aus einem komplexen, schrittweise aufgebauten (Berechnungs-) Modell abgehoben werden.

Im Bereich der Formalen Programmentwicklung spricht viel für die erstere (top-down) Vorgehensweise. Im Rahmen eines formalen Requirements Egineering sollte eine, i. allg. erst nach mehreren Iterationen erreichte, *stabile* abstrakte Spezifikation erstellt werden. Die dabei vorherrschenden Anwendungsgesichtspunkte sollten dabei nicht mit Aspekten der Realisierung vermischt werden. In diesem Zusammenhang ist zu beachten, daß das oben angedeutete Modell nur eins von vielen möglichen, untereinander signifikant verschiedenen, ist.

In vielen Fällen, wenn auch nicht immer, kann ein zu frühzeitiges modellbasiertes Denken daher in die Irre führen: Man orientiert sich an den Zufälligkeiten eines bestimmten Modells. In der Mathematik hat es eines langen, von vielen hervorragenden Forschern begleiteten Abstraktionsprozesses bedurft, um ausgehend von konkreten Objekten (des Denkens) zu den „richtigen“ Abstraktionen (z. B. Gruppen, Ringe, Körper) zu gelangen.

Die top-down Vorgehensweise paßt auch besser in viele Prozeßmodelle des Software Engineering. Wenn insbesondere eine Entwicklung nach der Stufe E6 (ITSEC) bzw. EAL7 (CC) angestrebt ist, kann der Modellbildungsprozeß zumindest teilweise im Rahmen der dann geforderten formalen Verfeinerung in einen Architekturentwurf (ITSEC) bzw. in eine funktionale Spezifikation (CC) geschehen. Dieser Verfeinerungsschritt ist aber Bestandteil der Gesamtentwicklung und zielt nicht allein auf den Konsistenznachweis.

Allerdings sei darauf hingewiesen, daß es problematisch ist, Konsistenzbeweise zu vernachlässigen. Bleiben nämlich Inkosistenzen zu lange unentdeckt, so müssen, bedingt durch ggf. notwendige Änderungen an grundlegenden Teilen der formalen Spezifikation, große Teile der Beweise von Sicherheitseigenschaften erneut geführt werden.

Kapitel 6

Bekannte formale Sicherheitsmodelle

Sicherheitspolitiken, die auf der Vergabe von Sicherheitsstufen basieren (engl. multi-level security policies), nachfolgend als *MLS-Politiken* abgekürzt, finden sowohl in Systemen mit Menschen als auch in reinen Computersystemen Verwendung. In MLS-Politiken werden den verschiedenen Komponenten eines Systems und eventuell auch den Komponenten der Systemumgebung Sicherheitsstufen zugeordnet. Oft wird dabei zwischen aktiven *Subjekten*, wie z. B. Prozessen oder Benutzer, und passiven *Objekten*, wie z. B. Dateien, unterschieden. Mit dieser Klasse von Sicherheitspolitiken können sowohl Anforderungen an die *Vertraulichkeit* (engl. confidentiality) als auch an die *Integrität* (engl. integrity) definiert werden.

In MLS-Politiken ist die Angabe von Sicherheitsstufen nicht auf lineare Ordnungen beschränkt, wie sie oft im militärischen Bereich verwendet werden (z. B. offen, VS-VERTRAULICH, GEHEIM), sondern es können auch partielle Ordnungen verwendet werden. Die unterliegende mathematische Struktur sind Verbände[1]. Alternativ lassen sich MLS-Politiken durch die Angabe einer Relation ausdrücken, die angibt, welche Subjekte und Objekte durch welche anderen Subjekte und Objekte beeinflußt werden dürfen. Solche Relationen werden als *Interference-Relationen* (engl. für Beeinflussung) bezeichnet [17, 18].

In diesem Abschnitt wird eine Auswahl von etablierten Sicherheitsmodellen beschrieben, die Sicherheitspolitiken aus dieser Klasse formal definieren. Da alle diese Sicherheitsmodelle generisch sind, ergibt sich eine konkrete Sicherheitspolitik erst durch die Instantiierung eines Modells. *Um ein formales Sicherheitsmodell im Sinne von ITSEC oder CC zu erhalten, muß diese Instantiierung formal*

[1] Ein Verband $(L, \bigvee, \bigwedge)$ besteht aus einer Trägermenge L und zwei binären Operationen $\bigvee$ und $\bigwedge$, die zu zwei Elementen die kleinste obere Schranke, das Supremum, bzw. die größte untere Schranke, das Infimum, bestimmen. [23]

erfolgen, und die informelle Interpretation muß sich auf dieses instantiierte Modell beziehen. Für jedes der beschriebenen Modelle wird angegeben, was die Bestandteile einer Instantiierung sind. Außerdem werden die Annahmen der Modelle an die Einsatzumgebung angegeben und bekannte Grenzen der Modelle sowie Probleme mit diesen besprochen.

Im einzelnen werden *Goguen und Meseguers Noninterference*, das *Modell nach Bell und La Padula* und das *Modell nach Biba* beschrieben. Während Noninterference sowohl die Definition von Anforderungen an die Vertraulichkeit als auch an die Integrität erlaubt, ist das Modell nach Bell und La Padula auf Vertraulichkeits- und das von Biba auf Integritätseigenschaften beschränkt. Technisch können die beiden letzteren Modelle als Spezialfälle von Noninterference verstanden werden. Ihre Beliebtheit erklärt sich dadurch, daß sie durch die Verwendung von Referenzmonitoren implementiert werden können. Durch diese Festlegung auf einen konkreten Mechanismus, den der Referenzmonitore, bieten diese Modelle allerdings nur einen beschränkten Rahmen für eine abstrakte Definition von Sicherheitseigenschaften. Noninterference hingegen kann als eine Definition von Sicherheit verstanden werden. Während die Modelle von Bell/La Padula und Biba auf einer Zugriffskontrolle basieren, basiert Noninterference auf der Kontrolle des Informationsflusses.

Aus der Vielfalt der Varianten dieser Sicherheitsmodelle wird im folgenden jeweils eine Variante konkret beschrieben und Alternativen zu dieser Variante werden angesprochen. Alle beschriebenen Modelle basieren auf Zustandsmaschinen. Die zugrundeliegenden Maschinenmodelle unterscheiden sich geringfügig und werden deshalb zu Beginn jedes Abschnitts kurz eingeführt. Basierend darauf wird das jeweilige Sicherheitsmodell beschrieben und Beweistechniken, die den Nachweis von Sicherheit erleichtern, werden angesprochen. Außerdem werden die Bestandteile einer formalen Instantiierung erläutert, die Annahmen an die Einsatzumgebung angegeben und bekannte Grenzen der Modelle besprochen.

6.1 Noninterference

Für die Definition einer Sicherheitspolitik ist es nötig, die Komponenten eines Systems, wie z. B. Daten, Prozesse oder Schnittstellen, und seiner Umgebung, wie z. B. Benutzer oder Benutzergruppen, zu identifizieren. Um von den konkreten Komponenten zu abstrahieren wird der Begriff *Bereich* (engl. domain) benutzt. Ein Bereich kann z. B. ein einzelner Benutzer, eine Datei, aber auch eine Menge von Benutzern und Dateien sein.

Der von Goguen und Meseguer für die Security Gemeinschaft geprägte Begriff *Interference* (engl. für Eingreifen) steht für die Beeinflussung eines Bereichs durch einen anderen. Basierend auf dem dazu komplementären Begriff, *Noninter-*

ference, also der Festlegung, welche Bereiche durch andere Bereiche nicht beeinflußt werden dürfen, lassen sich Sicherheitspolitiken definieren, die sowohl Vertraulichkeit als auch Integrität umfassen. Erstmalig wurde dazu von Goguen und Meseguer in [17, 18] ein Rahmen vorgeschlagen, in dem solche Sicherheitspolitiken definiert werden können. Dieser Rahmen wurde später in verschiedenen Arbeiten modifiziert, erweitert und verallgemeinert, siehe z. B. [40, 42, 29, 38, 33, 34, 45, 39]. Die hier beschriebene Variante orientiert sich an [40], einem Ansatz, der eine Verallgemeinerung auf nicht-transitive Definitionen von Interference ermöglicht und somit nicht auf MLS-Politiken, die im Zentrum dieses Kapitels stehen, beschränkt ist.

Um eine Sicherheitspolitik mit dem Noninterference-Ansatz zu definieren, müssen zunächst die Systemaktionen und die Bereiche bestimmt werden und jeder Aktion muß ein solcher Bereich zugeordnet werden. Durch die Angabe einer *Noninterference-Relation* $\not\rightsquigarrow$ wird definiert, welche Bereiche von anderen nicht beeinflußt werden dürfen. Ist das Komplement $\rightsquigarrow$ von $\not\rightsquigarrow$ *transitiv*, dann entspricht die Zuordnung von Bereichen gemeinsam mit der Angabe der Noninterference-Relation der für MLS-Politiken sonst üblichen Vergabe von Sicherheitsstufen.

Die Unmöglichkeit einer Beeinflussung wird dadurch ausgedrückt, daß für Bereiche d und d' mit $d' \not\rightsquigarrow d$ der Bereich d nicht unterscheiden kann, ob d' eine Aktion unternommen hat oder nicht. Wird kein Bereich von anderen Bereichen gemäß $\not\rightsquigarrow$ unerlaubt beeinflußt, so ist ein System sicher.

Sicherheitsmodelle, die auf einer Noninterference-Relation basieren, besitzen ein breites Anwendungsspektrum, denn *durch die Angabe einer solchen Relation lassen sich sowohl Forderungen an die Vertraulichkeit als auch an die Integrität ausdrücken*. Definiert man $d' \not\rightsquigarrow d$, dann darf d die Aktionen von d' nicht beobachten (Vertraulichkeit) andererseits darf d' auch nicht d beeinflussen (Integrität). Somit deckt Noninterference beide Gebiete für Security ab, für die etablierte Sicherheitsmodelle existieren, die allgemein einsetzbar sind.

6.1.1 Definition des formalen Modells

Die in diesem Abschnitt beschriebene Variante von Noninterference orientiert sich an Rushby [40]. Basierend auf seiner Formulierung wurde der Ansatz von Goguen und Meseguer [17, 18] erweitert, so daß auch Sicherheitspolitiken formuliert werden können, die keine MLS-Politiken sind, d. h. Sicherheitspolitiken, in denen die Interference-Relation nicht transitiv ist. Außerdem wurde für diese Formulierung ein *Unwinding-Theorem* angegeben, dessen Voraussetzungen im Vergleich zu [18] weniger restriktiv sind. Unwinding-Theoreme erleichtern die zum Nachweis von Noninterference notwendigen Beweise.

Das System wird durch eine Zustandsmaschine modelliert. Dazu werden eine Menge S von Zuständen, eine Menge A von Aktionen und eine Menge O

von Ausgaben definiert. Die möglichen Zustandsübergänge werden durch eine Funktion *step* : $S \times A \to S$ modelliert und die Ausgaben durch eine Funktion *output* : $S \times A \to O$. *step*(s, a) definiert den Zustand, der durch Ausführung der Aktion $a \in A$ im Zustand $s \in S$ erreicht wird und *output*(s, a) definiert die Ausgabe, die durch Ausführung von a in s erzeugt wird. Die Ausführung einer Folge α von Aktionen ausgehend von einem Zustand $s_0 \in S$ wird durch eine Funktion *run* : $S \times A^* \to S$ modelliert, die durch die folgenden beiden Gleichungen rekursiv definiert ist.

$$\begin{array}{lcll} \textit{run}(s, \epsilon) & = & s & \text{, für die leere Folge } \epsilon \\ \textit{run}(s, a \circ \alpha) & = & \textit{run}(\textit{step}(s, a), \alpha) & \end{array}$$

Nach Angabe einer Menge D von Sicherheitsdomains wird jeder Aktion mit einer Funktion *dom* : $A \to D$ ein solcher Bereich zugeordnet. Eine Noninterference-Relation $\not\rightsquigarrow$: $D \times D$ ist eine irreflexive Relation, mit der man spezifiziert, welche Bereiche andere nicht beeinflussen dürfen. Die zugehörige reflexive Interference-Relation $\rightsquigarrow$ ergibt sich durch Komplementbildung, d. h. $\rightsquigarrow = (D \times D) \setminus \not\rightsquigarrow$. Gilt $d' \rightsquigarrow d$, so darf d durch d' beeinflußt werden, wenn $d' \not\rightsquigarrow d$ gilt, jedoch nicht. Dem Noninterference-Ansatz liegt folgende Annahme zugrunde: wenn ein Bereich d nicht erkennen kann, ob ein anderer Bereich eine Aktion ausgeführt hat oder nicht, dann wird er auch nicht durch diese Aktion beeinflußt, d. h. es gilt $d' \not\rightsquigarrow d$. Um dieses formal auszudrücken, wird eine Funktion *purge* : $(A^* \times D) \to A^*$ definiert, die aus einer Aktionenfolge alle Aktionen entfernt, die den angegebenen Bereich nicht beeinträchtigen dürfen.

Definition 1. Sei $d \in D$ ein Bereich und $\alpha \in A^*$ eine Aktionenfolge, dann wird *purge*(α, d) durch die folgenden Gleichungen rekursiv definiert.

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

Die Merkmale von Noninterference sind zusammenfassend in Tabelle 6.1 aufgeführt.

Beispiel 2. Sei $D = \{\textit{high}, \textit{low}\}$, $A = \{\textit{hin}, \textit{hout}, \textit{lin}, \textit{lout}\}$, $\textit{dom}(\textit{hin}) = \textit{high} = \textit{dom}(\textit{hout})$, $\textit{dom}(\textit{lin}) = \textit{low} = \textit{dom}(\textit{lout})$ und $\textit{high} \not\rightsquigarrow \textit{low}$. Somit gibt es vier Aktionen, die zwei Sicherheitsstufen zugeordnet sind, wobei die niedrige nicht durch die hohe Sicherheitsstufe beeinflußt werden darf. Für die Aktionenfolge $\alpha = \langle \textit{hin}, \textit{lin}, \textit{hout}, \textit{lout} \rangle$ gelten die folgenden Gleichungen.

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

S	Zustände	
s_0	Anfangszustand	$s_0 \in S$
O	Ausgaben	
A	Aktionen	z. B. $\{hin, hout, lin, lout\}$
D	Bereiche	z. B. $\{high, low\}$
dom	Zuordnung von Bereichen	$dom : A \to D$
$\not\rightsquigarrow$	Noninterference-Relation	$\not\rightsquigarrow \subseteq D \times D$
output	Ausgabefunktion	$output : S \times A \to O$
step	Zustandsübergangsfunktion	$step : S \times A \to S$
run	Ausführung von Aktionenfolgen	$run : S \times A^* \to S$
purge	purge	$purge : A^* \times D \to A^*$

Tabelle 6.1: Merkmale von Noninterference

Mit Hilfe der *purge*-Funktion wird ein Begriff von Sicherheit definiert.

Definition 3 (Sicherheitseigenschaft von Noninterference).
Ein System mit Anfangszustand $s_0 \in S$ ist *sicher* für $\not\rightsquigarrow$: $D \times D$ (gemäß Noninterference), wenn für alle $a \in A$ und $\alpha \in A^*$ folgende Gleichung erfüllt ist.

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

Definition 3 formalisiert, was im Noninterference-Ansatz unter Sicherheit verstanden wird. Intuitiv wird ein System dann als sicher angesehen, wenn kein Bereich erkennen kann, ob eine Aktion aus einem Bereich, von dem er nicht beeinflußt werden darf, ausgeführt wurde oder nicht. Aus technischer Sicht darf sich die Ausgabe nicht ändern, wenn man aus einer Aktionenfolge alle Aktionen löscht (purge), die – gemäß $\not\rightsquigarrow$ – nicht wahrgenommen werden dürfen.

Um die Sicherheit eines Systems gemäß Definition 3 zu beweisen, müssen alle möglichen Aktionenfolgen – und somit unendlich viele – betrachtet werden. Um sich beim Beweis auf einzelne Zustandsübergänge zu beschränken, sind *Unwinding-Theoreme* hilfreich. Um ein solches Theorem zu formulieren, sind zunächst einige weitere Definitionen nötig, die im folgenden eingeführt werden. Diese Definitionen bilden die Grundlage für das am Ende dieses Unterabschnitts angegebene Unwinding-Theorem.

Definition 4. Eine Klasse $\sim = (\overset{u}{\sim})_{u \in D}$ von Äquivalenzrelationen auf S heißt *Sichtenpartitionierung* (engl. view-partition) für eine Menge D von Bereichen. $\sim$ ist *ausgabekonsistent* (engl. output consistent) wenn für alle $s, t \in S$ und alle $a \in A$ gilt:

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

Wenn zwei Zustände bezüglich einer ausgabekonsistenten Sichtenpartitionierung $\sim$ in Relation stehen, dann sind die Ausgaben aller Aktionen auf diesen Zuständen identisch.

Beispiel 5. Für die Wahl einer ausgabenkonsistenten Sichtenpartitionierung gibt es zwei Extremfälle, $\sim_{min} = (\overset{u}{\sim}_{min})_{u \in D}$ und $\sim_{max} = (\overset{u}{\sim}_{max})_{u \in D}$. Zwei Zustände stehen in der Relation $\overset{u}{\sim}_{min}$, wenn sie gleich sind, d. h. wenn

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

gilt. $\sim_{min}$ ist ausgabenkonsistent. Da die $\overset{u}{\sim}$ Äquivalenzrelationen sind, ist $\sim_{min}$ die minimale Sichtenpartitionierung. Die maximale ausgabenkonsistente Sichtenpartitionierung $\sim_{max}$ wird wie folgt definiert:

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

Die Wahl der Sichtenpartitionierung ist von entscheidender Bedeutung bei einer Anwendung des Unwinding-Theorems. Ob es besser ist, $\sim$ möglichst groß oder möglichst klein zu wählen, läßt sich jedoch nicht allgemein sagen.

Lemma 6. *Sei $\not\leadsto$ eine Noninterference-Relation, M ein System und $\sim$ eine ausgabekonsistente Sichtenpartitionierung für M. Wenn für jede Folge α von Aktionen und für jeden Bereich $u \in D$ $\mathit{run}(s_0, \alpha) \overset{u}{\sim} \mathit{run}(s_0, \mathit{purge}(\alpha, u))$ gilt. Dann ist M sicher für $\not\leadsto$.[40]*

Definition 7. Sei M ein System, $\sim$ eine Sichtenpartitionierung für M und $\not\leadsto$ eine Noninterference-Relation. M *respektiert* $\not\leadsto$ *lokal* (engl. locally respects) wenn für alle $a \in A$, alle $u \in D$ und alle $s \in S$ gilt:

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

$\sim$ tritt in dieser Definition nur auf der rechten Seite der Implikation auf. Somit wird der Beweis, daß $\not\leadsto$ durch M lokal respektiert wird, durch eine möglichst große Wahl von $\sim$ erleichtert.

Definition 8. Sei M ein System, $\sim$ eine Sichtenpartitionierung für M und $\not\leadsto$ eine Noninterference-Relation. M ist *schrittkonsistent* (engl. step consistent) wenn für alle $a \in A$, alle $u \in D$ und alle $s, t \in S$ gilt:

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

$\sim$ tritt in dieser Definition sowohl auf der rechten als auch auf der linken Seite der Implikation auf. Somit läßt sich nicht allgemein sagen, ob der Beweis, daß M schrittkonsistent ist, durch eine möglichst große oder kleine Wahl von $\sim$ erleichtert wird.

Das folgende *unwinding-Theorem* basiert auf den Definitionen 4, 7 und 8.

Theorem 9. *Sei $\not\rightsquigarrow$ eine Noninterference-Relation, M ein System und $\sim$ eine Sichtenpartitionierung von M. M ist sicher für $\not\rightsquigarrow$ wenn*

1. *$\sim$ ausgabekonsistent ist,*
2. *M schrittkonsistent ist und*
3. *M lokal $\not\rightsquigarrow$ respektiert.[40]*

6.1.2 Formalisierung einer Sicherheitspolitik

Noninterference bietet einen generischen Rahmen, um konkrete Sicherheitspolitiken zu formalisieren. Im vorhergehenden Abschnitt wurden die Menge S der Zustände, der Anfangszustand s_0, die Menge O der Ausgaben, die Menge A der Aktionen, die Menge der Bereiche D, die Zuordnung *dom*, die Ausgabefunktion *output*, die Funktion *step* und die Noninterference-Relation $\not\rightsquigarrow$ nicht näher spezifiziert. Um ein formales Sicherheitsmodell gemäß ITSEC oder CC für eine konkrete Sicherheitspolitik zu erhalten, müssen diese Parameter instantiiert werden und die Sicherheit des Systems für diese Noninterference-Relation muß formal nachgewiesen werden. *Man beachte, daß die Angabe der Interference- bzw. Noninterference-Relation nur ein Bestandteil dieser Instantiierung ist.* Die zu instantiierenden Parameter von Noninterference sind zusammenfassend in Tabelle 6.2 aufgeführt. Man beachte, daß die Interference-Relation $\rightsquigarrow$ (das Komplement von $\not\rightsquigarrow$) für die hier vorgestellte Variante von Noninterference stets transitiv sein muß, also eine MLS-Politik spezifizieren muß.

Will man beim Nachweis der Sicherheit des Systems (gemäß Noninterference) das Unwinding-Theorem (Theorem 9) einsetzen, so muß eine ausgabenkonsistente Sichtenpartitionierung gewählt werden. In Beispiel 5 werden zwei mögliche Sichtenpartitionierungen vorgestellt.

S	Zustände	
s_0	Anfangszustand	$s_0 \in S$
O	Ausgaben	
A	Aktionen	z. B. $\{hin, hout, lin, lout\}$
D	Bereiche	z. B. $\{high, low\}$
dom	Zuordnung von Bereichen	$dom : A \rightarrow D$
$\not\rightsquigarrow$	Noninterference-Relation	$\not\rightsquigarrow \subseteq D \times D$
output	Ausgabefunktion	$output : S \times A \rightarrow O$
step	Zustandsübergangsfunktion	$step : S \times A \rightarrow S$

Tabelle 6.2: Parameter von Noninterference

6.1.3 Annahmen an die Einsatzumgebung

Bei einer Instantiierung von Noninterference müssen die Parameter angemessen definiert werden. Insbesondere muß die Menge A alle Aktionen und die Menge S alle Zustände umfassen, die Zustandsübergangsfunktion muß die Funktionsweise des Systems angemessen modellieren, und mit der Ausgabefunktion müssen verschiedene Ausgaben des Systems unterschieden werden können.

6.1.4 Bekannte Grenzen des Sicherheitsmodells

Die in den vorhergehenden Unterabschnitten vorgestellte Variante von Noninterference ist auf transitive Interference-Relationen, d. h. auf MLS-Politiken, beschränkt. Außerdem geht das unterliegende Maschinenmodell von einem deterministischen System aus. Beide Beschränkungen treffen auch auf den ursprünglichen Ansatz in [17, 18] zu.

Die gezielte Einschränkung von Informationsfluß zwischen Sicherheitsbereichen ist die Grundlage für den Noninterference-Ansatz. Ein System wird als sicher betrachtet, wenn Aktionen aus bestimmten Bereichen nicht wahrgenommen werden können. Die beschriebene Variante von Noninterference ist auf transitive Interference-Relationen beschränkt. Die Transitivität verhindert, daß Informationen, die nicht direkt von einem Bereich d_1 zu einem Bereich d_2 fließen dürfen, auch nicht über einen Umweg von d_1 nach d_2 gelangen. Für MLS-Politiken trifft diese Einschränkung stets zu. Für manche Anwendungen wird jedoch ein sogenanntes *downgrading* benötigt, d. h. die Herabstufung von vertraulichen Informationen, das mit einer transitiven Interference-Relation nicht formalisiert werden kann. Ist z. B. ein direkter Informationsfluß von d_1 nach d_2 unzulässig, also $d_1 \not\rightsquigarrow d_2$, ein Informationsfluß über den Umweg d_3 (downgrader) jedoch erlaubt, also $d_1 \rightsquigarrow d_3$ und $d_3 \rightsquigarrow d_2$, so ergibt sich eine intransitive Interference-Relation. Nicht-transitive Interference-Relationen werden ebenfalls benötigt, wenn Kryptokomponenten eingesetzt werden, so daß vertrauliche Daten nach einer Verschlüsselung über offen zugängliche Netze versandt werden.

Diese Problematik wird von Rushby in [40] ausführlich diskutiert. Außerdem wird ein Ansatz zur Behandlung von nicht-transitiven Interference-Relationen beschrieben. Roscoe und Goldsmith argumentieren, daß dieser Ansatz keine ausreichende Sicherheit bietet [39]. Sie schlagen eine Alternative vor, die auf einer in [38] beschriebenen Variante von Noninterference aufbaut. Diese Variante unterscheidet sich von der hier beschriebenen stark, da sie auf der failure-divergence Semantik von CSP [21] basiert.

Die in Abschnitt 6.1 verwendeten Zustandsmaschinen sind deterministisch, da identische Eingaben (Aktionen) stets dieselben Ausgaben erzeugen. Für eine Verallgemeinerung des Noninterference Ansatzes für nicht-deterministische

Systeme wurde eine Vielzahl von Varianten vorgeschlagen. Die diesen gemeinsame Idee ist, daß ein System dann als sicher angesehen wird, wenn man aus einer aktuellen Beobachtung des Systems aus einer Domäne und der Kenntnis der Funktionsweise des Systems nur auf eine Menge von möglichen Systemverhalten schließen kann, die zu groß ist, um vertrauliche Informationen zu deduzieren. Die verschiedenen Ansätze unterscheiden sich darin, wann diese Menge als groß genug angesehen wird, wobei die meisten Verallgemeinerungen von Noninterference für nicht-deterministische Systeme *possibilistisch* (engl. possibilistic) sind, also verlangen, daß bestimmte Systemverhalten mit aktuellen Beobachtungen konform sind, Wahrscheinlichkeiten für diese jedoch nicht berücksichtigen. Beispiele für possibilistische Ansätze sind *Nondeducibility* [42], *Restrictiveness* [29], *Noninference* [36], *Generalized Noninference* [33], und *Separability* [33]. Einheitliche Rahmen für die Untersuchung von possibilistischen Ansätzen wurden in [33, 34, 45, 27] vorgestellt. Unwinding-Theoreme für possibilistische Verallgemeinerungen von Noninterference wurden in [19, 41, 35, 28] angegeben. *Probabilistische Ansätze* (engl. probabilistic), wie z. B. [20], berücksichtigen zusätzlich die Wahrscheinlichkeit der verschiedenen Systemverhalten. Possibilistische und probabilistische Ansätze werden in [31, 32] gegenübergestellt.

Nicht-deterministische Maschinenmodelle werden z. B. für die Beschreibung paralleler Systeme mit asynchroner Kommunikation benötigt. Außerdem werden nicht-deterministische Beschreibungen traditionell für abstrakte Spezifikationen in frühen Phasen der formalen Softwareentwicklung eingesetzt, die dann in späteren Phasen schrittweise verfeinert werden. Für die Entwicklung sicherer (im Sinne von engl. secure) Systeme ist jedoch das Verfeinerungsparadox (engl. refinement paradox) zu beachten. Wurde eine nicht-deterministische Beschreibung als sicher (im Sinne von engl. secure) bewiesen, so kann im allgemeinen nicht geschlossen werden, daß eine Verfeinerung ebenfalls sicher ist. Das Verfeinerungsparadox hat im wesentlichen zwei Ursachen. Zum einen kann es sein, daß eine abstrakte Systemspezifikation nicht die Beschreibungsmittel bietet, die zur Spezifikation mancher Bedrohungen benötigt werden. Die Sicherheit eines Systems gegenüber solchen Bedrohungen kann dann erst auf einer konkreteren Beschreibungsebene untersucht werden. Zum anderen kann die Sicherheit eines Systems durch die Einschränkung von Nichtdeterminismus bei der Verfeinerung verletzt werden. Letzteres tritt ausschließlich bei den Verallgemeinerungen von Noninterference für nicht-deterministische Systeme auf, die im vorhergehenden Absatz angesprochen wurden. Die durch das Verfeinerungsparadox verursachte Problematik tritt bei der Entwicklung sicherer Systeme (im Sinne von engl. safe systems) nicht auf und wird deshalb als eine Ursache für die Komplexität bei der Entwicklung sicherer Systeme (im Sinne von engl. secure systems) angesehen. Für eine Diskussion des Verfeinerungsparadox siehe auch [24]. Die schrittweise Entwicklung komplexer Systeme kann auch mit einem Bottom-up Ansatz erfolgen, d. h. durch die Kom-

position von Komponenten. Für die Entwicklung sicherer Systeme (im Sinne von engl. secure systems), ist es natürlich wünschenswert, daß die Sicherheit des zusammengesetzten Systems bereits aus der Sicherheit der Komponenten gefolgert werden kann. Natürlich ist dieses nur unter bestimmten Voraussetzungen möglich. Diese Problematik wurde z. B. in [29, 33, 34, 44] untersucht.

6.2 Sicherheit durch Zugriffskontrolle

Sicherheitsmodelle, die auf einer Zugriffskontrolle basieren (engl. access control models), erlauben die Definition von Sicherheitspolitiken, durch die der *direkte Zugriff* von aktiven Subjekten, wie z. B. Personen, auf passive Objekte, wie z. B. Dateien, geregelt wird. Das Ziel einer solchen Sicherheitspolitik ist es, den Inhalt der Objekte vor unerlaubten Zugriffen zu schützen, ohne dabei den Subjekten vertrauen zu müssen.

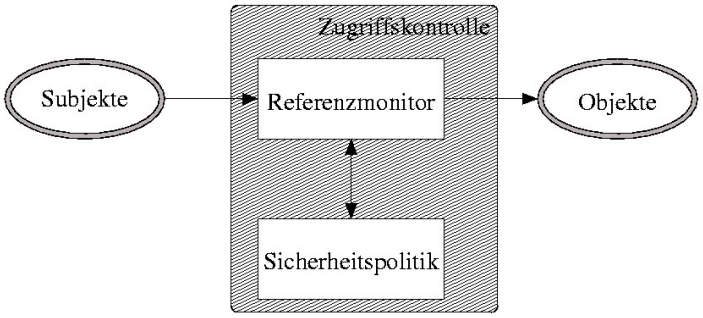


Abbildung 6.1: Zugriffskontrolle

Zugriffe der Subjekte auf die Objekte erfolgen über einen Referenzmonitor, der jeden Zugriffsversuch auf seine Zulässigkeit bzgl. einer Sicherheitspolitik überprüft (vergleiche Abbildung 6.1) und nur zulässige Zugriffe gewährt. Um mit einer solchen Zugriffskontrolle die Durchsetzung der Sicherheitspolitik zu garantieren, müssen folgende Voraussetzungen erfüllt sein.

- *Vollständigkeit:* Alle Zugriffe der Subjekte auf die zu schützenden Objekte erfolgen über die Zugriffskontrolle.
- *Geschütztheit:* Die Zugriffskontrolle ist gegenüber unerlaubten Veränderungen geschützt.
- *Korrektheit:* Die Zugriffskontrolle ist korrekt implementiert, d. h. garantiert die gewünschte Sicherheitspolitik.

Um die einzelnen Zugriffe zu klassifizieren, werden Zugriffsarten identifiziert, so daß Zugriffe, die zu einer Klasse gehören, von der Zugriffskontrolle einheitlich behandelt werden können. Wir beschränken uns im folgenden auf die Zugriffsarten *Lesen* (engl. read) und *Schreiben* (engl. write). Weitere Zugriffsarten, wie z. B. *Ausführen* (engl. execute) werden, soweit möglich, durch Verweise auf entsprechende Literatur abgehandelt.

Innerhalb der Zugriffskontrolle wird die Überprüfung der Zulässigkeit von Zugriffen bezüglich der Sicherheitspolitik, meist mit Hilfe einer *Zugriffsmatrix* (engl. access control matrix) realisiert. Eine Zugriffsmatrix enthält für jedes Subjekt eine Reihe und für jedes Objekt eine Spalte (siehe Abbildung 6.2). Die Einträge in der Matrix geben an, welche Zugriffsrechte die Subjekte auf die jeweiligen Objekte haben. Ein Subjekt darf nur dann auf ein Objekt lesend bzw. schreibend zugreifen, wenn in der Zelle der Matrix, die in der Reihe des Subjekts und in der Spalte des Objektes liegt, das Zugriffsrecht *read* bzw. *write* eingetragen ist. Die Zugriffsmatrix in Abbildung 6.2 z. B. erlaubt dem Subjekt s_i lesende, aber keine schreibenden Zugriffe auf das Objekt o_j. *Zugriffslisten* (engl. access control lists) oder *Befähigungslisten* (engl. capability lists) sind mögliche technische Realisierungen einer Zugriffsmatrix (siehe auch [43]).

Objekte Subjekte	...	o_j	...
. . .		. . .	
s_i	...	read	...
. . .		. . .	

Abbildung 6.2: Zugriffsmatrix

Die verschiedenen Sicherheitsmodelle unterscheiden sich darin, wie die Zugriffsmatrix initialisiert werden kann, ob sie verändert werden darf und, wenn ja, welche Veränderungen erlaubt sind. Grundsätzlich unterscheidet man zwei Arten von Sicherheitsmodellen, die auf Zugriffskontrolle basieren. *Discretionary access control* erlaubt Subjekten, ihre Zugriffsrechte für Objekte, die sie besitzen, an andere Subjekte weiterzureichen. Somit sind die Subjekte für eine kontrollierte Weitergabe von Rechten verantwortlich. Demzufolge muß bei discretionary access control den Subjekten vertraut werden, daß sie ihre Rechte nur an solche Subjekte weitergeben, die vertrauenswürdig damit umgehen. In Computersystemen, ist es jedoch schwierig, ein solches Vertrauen zu rechtfertigen. Gibt ein Subjekt beim Start eines Programmes seine Rechte an dieses weiter, so können die wei-

tergegebenen Rechte mißbraucht werden, wenn das Programm ein trojanisches Pferd (wie z. B. ein Computervirus) ist. Somit bieten Sicherheitsmodelle, die auf discretionary access control basieren, keinen Schutz gegenüber solchen Attacken. Dazu siehe auch [32]. *Mandatory access control* hingegen erlaubt eine Weitergabe von Zugriffsrechten nicht. Daher bietet mandatory access control einen besseren Schutz gegen trojanische Pferde.

Im folgenden werden mit dem Bell/La Padula Modell und dem Modell nach Biba zwei Sicherheitsmodelle beschrieben, die auf Zugriffskontrolle im Sinne von mandatory access control basieren. Bell/La Padula und Biba sind dual zueinander, die genaue Beziehung wird in Abschnitt 6.3 erläutert. Wegen dieser Beziehung lassen sich ähnliche Varianten bilden. Zur Illustration der Variantenvielfalt beschreiben wir in Abschnitt 6.2.1 eine Variante von Bell/La Padula, die dynamische Änderungen der Zugriffsmatrix zuläßt und in 6.2.2 eine Variante von Biba, die weniger Änderungen erlaubt. Mit der in Abschnitt 6.3 beschriebenen Beziehung lassen sich diese Varianten für das jeweils andere Sicherheitsmodell leicht anpassen.

6.2.1 Bell/La Padula

Das Modell von Bell und La Padula [13, 14, 15], nachfolgend mit *BLP* abgekürzt, erlaubt die Definition von Sicherheitspolitiken, durch die der Zugriff von aktiven Subjekten, wie z. B. Prozessen, auf passive Objekte, wie z. B. Dateien, geregelt wird. Das Ziel einer solchen Sicherheitspolitik ist es, den Inhalt der Objekte vor unerlaubten Zugriffen zu schützen, ohne dabei den Subjekten vertrauen zu müssen. Ausgehend von [15] wurden verschiedene Varianten von BLP entwickelt [32]. Die hier beschriebene Variante ist eine Vereinfachung gegenüber dem Originalmodell und orientiert sich an [31].

Die Zugriffskontrolle in BLP basiert, wie zuvor beschrieben, auf Referenzmonitoren. Entsprechend müssen die drei Voraussetzungen Vollständigkeit, Geschütztheit und Korrektheit sichergestellt werden. Die Zugriffsmatrix kann in der hier beschriebenen Variante ausgehend von einem Initialzustand dynamisch verändert werden. Welche Zustände der Matrix zulässig sind und welche nicht, wird durch zwei Eigenschaften ausgedrückt. Die *einfache Sicherheitseigenschaft* (engl. simple security property) verlangt, daß Subjekte nur solche Objekte lesen können, welche die gleiche oder eine niedrigere Vertraulichkeitsstufe haben. Das Kopieren von Daten von einem Objekt in ein anderes Objekt mit einer niedrigeren Vertraulichkeitsstufe, also ein unerlaubtes Herabstufen vertrauenswürdiger Informationen, wird durch die $\star$-*Eigenschaft* (gesprochen „Stern-Eigenschaft“; engl. $\star$-property, gesprochen “star-property”) verhindert. In BLP wird ein Zustand nur dann als sicher angesehen, wenn sowohl die einfache Sicherheitseigenschaft als auch die $\star$-Eigenschaft erfüllt ist. Die Kombination beider Eigenschaften garan-

tiert, daß Subjekte nur den Inhalt von Objekten erfahren, denen die gleiche oder eine niedrigere Vertraulichkeitsstufe zugeordnet ist.

Im Unterschied zu vorausgehenden Ansätzen, wie dem Lampson Protection Model [26], in denen die zulässigen Änderungen der Zugriffsmatrix über eine Menge von Regeln definiert ist, werden in BLP den Subjekten und Objekten Vertraulichkeitsstufen zugeordnet. Dieser Ansatz erlaubt es, Regeln zur Änderung der Matrix als korrekt nachzuweisen, wie z. B. in [14].

Definition des formalen Modells

Nachfolgend wird eine Definition des Sicherheitsmodells nach Bell und La Padula beschrieben, die gegenüber der Definition in [15] vereinfacht ist. Durch diese Vereinfachung kann die Zugriffskontrolle gemäß BLP mit einer einzelnen Zugriffsmatrix formalisiert werden. Ähnliche Vereinfachungen werden in [31] gemacht. Unsere Vereinfachungen sind am Ende dieses Abschnittes zusammengefaßt.

Bell und La Padula modellieren das System zur Zugriffskontrolle als deterministische Zustandsmaschine. Eine solche Zustandsmaschine wird durch ein Tupel (V, v_0, R, T) definiert, wobei V eine Menge von Zuständen, $v_0 \in V$ ein Initialzustand, R (für engl. request) eine Menge möglicher Anfragen und $T : (V \times R) \rightarrow V$ eine Zustandsübergangsfunktion (für engl. transition) ist, die das System auf Anfrage von einem Zustand in einen anderen überführt. Die Zustände der Maschine werden im folgenden noch näher beschrieben.

Subjekte und Objekte werden durch Mengen S und O modelliert. Subjekte können z. B. Personen oder Prozesse sein. Beispiele für Objekte sind Dateien oder Speicherbereiche. Eine Funktion $F_C : S \cup O \rightarrow L$ klassifiziert jedes Subjekt und Objekt, indem es ihnen eine Vertraulichkeitsstufe aus einer Menge L (für engl. level) zuordnet. Mögliche Vertraulichkeitsstufen sind z. B. „offen“, „VS-VERTRAULICH“ oder „GEHEIM“. Auf den Elementen von L existiert eine partielle Ordnung $\leq$[2], mit der L eine verbandsgeordnete Menge[3] bildet. Die möglichen Zugriffsarten von Subjekten auf Objekte werden durch die Menge $A = \{\textit{read}, \textit{write}\}$ (für engl. access) angegeben. Die aktuellen Zugriffsrechte werden in einer Zugriffsmatrix $M : S \times O \rightarrow P(A)$ festgehalten, wobei in jeder Zelle der Matrix die aktuelle Menge der erlaubten Zugriffsarten eingetragen ist. Die Menge $\{\textit{write}\}$ z. B. erlaubt einem Subjekt schreibende, aber keine lesenden Zugriffe auf ein Objekt.

Während die Mengen S, O, L und A für eine konkrete Sicherheitspolitik statisch festgelegt werden, können die Funktion F_C und die Zugriffsmatrix M dyna-

[2] D. h. $\leq$ ist reflexiv ($\forall l \in L. l \leq l$), antisymmetrisch ($\forall l_1, l_2 \in L. (l_1 \leq l_2 \wedge l_2 \leq l_1) \Rightarrow l_1 = l_2$) und transitiv ($\forall l_1, l_2, l_3 \in L. (l_1 \leq l_2 \wedge l_2 \leq l_3) \Rightarrow l_1 \leq l_3$).

[3] D. h. zu je zwei Elementen $l_1, l_2 \in L$ existieren eine kleinste obere Schranke, das Supremum $\bigvee\{l_1, l_2\}$, und eine größte untere Schranke, das Infimum $\bigwedge\{l_1, l_2\}$.

misch verändert werden. Zustände der Zugriffskontrolle werden daher als Paare (F_C, M) modelliert, woraus sich die Menge $V = (S \cup O \rightarrow L) \times (S \times O \rightarrow P(A))$ der Zustände der Zustandsmaschine ergibt. Die Merkmale von BLP sind zusammenfassend in Tabelle 6.3 aufgeführt.

S	Subjekte	z. B. $\{s_1, s_2, \ldots, s_m\}$
O	Objekte	z. B. $\{o_1, o_2, \ldots, o_n\}$
L	Vertraulichkeitsstufen	z. B. {*offen, GEHEIM,...*}
$\leq$	Ordnung auf L	$\leq \subseteq L \times L$
A	Zugriffsarten	$A = \{$*read, write*$\}$
F_C	Klassifikation	$F_C : S \cup O \rightarrow L$
M	Zugriffsmatrix	$M : S \times O \rightarrow P(A)$
V	Zustände	$V : (S \cup O \rightarrow L) \times (S \times O \rightarrow P(A))$
v_0	Initialzustand	$v_0 \in V$
R	Anfragen	
T	Zustandsübergangsfunktion	$T : V \times R \rightarrow V$

Tabelle 6.3: Merkmale von BLP

Die einfache Sicherheitseigenschaft ist erfüllt, wenn Subjekte nur solche Objekte lesen können, welche die gleiche oder eine niedrigere Vertraulichkeitsstufe haben. Sie verhindert also, daß Subjekte *direkt* auf Objekte mit einer höheren Vertraulichkeitsstufe lesend zugreifen.

Definition 10. Ein Zustand (F_C, M) erfüllt die *einfache Sicherheitseigenschaft* (engl. simple security property) genau dann, wenn

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

Die $\star$-Eigenschaft ist erfüllt, wenn kein Subjekt Objekte einer Vertraulichkeitsstufe $F_C(o)$ lesen kann und gleichzeitig ein Objekt o' mit einer niedrigeren Vertraulichkeitsstufe $F_C(o') < F_C(o)$ schreiben kann. Ansonsten könnte das Subjekt den Inhalt eines höher klassifizierten Objektes o in ein Objekt o' mit einer niedrigeren Vertraulichkeitsstufe kopieren. Dadurch würden die Inhalte für andere Subjekte direkt lesbar werden, obwohl diese eine zu niedrige Vertraulichkeitsstufe haben. Da in BLP angenommen wird, daß Subjekten nicht vertraut werden kann, und damit die Möglichkeiten von Innentätern und Trojanischen Pferden ausgeschlossen werden, wird der angestrebte Sicherheitsbegriff erst durch die Kombination der einfachen Sicherheitseigenschaft mit der $\star$-Eigenschaft erreicht. Die $\star$-Eigenschaft verhindert somit, daß Subjekte die Inhalte von Objekten mit einer höheren Vertraulichkeitsstufe *indirekt* lesen können.

Definition 11. Ein Zustand (F_C, M) erfüllt die $\star$-*Eigenschaft* (engl. $\star$-property) genau dann, wenn

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

Bemerkung 12. Folgende Variante der $\star$-Eigenschaft, die im allgemeinen restriktiver als Definition 11 ist, findet man oft in der Literatur. Haben Subjekte jedoch einen lokalen Speicher, dem dieselbe Vertraulichkeitsstufe wie dem Subjekt zugeordnet ist, dann sind die beiden Varianten der $\star$-Eigenschaft logisch äquivalent.

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

Durch die Modellierung als Zustandsmaschine wird eine induktive Definition der Sicherheit von Systemen möglich.

Definition 13 (Sicherheitseigenschaften von BLP).
Ein Zustand $v = (F_C, M)$ ist *sicher* (gemäß BLP), wenn v sowohl die einfache Sicherheitseigenschaft als auch die $\star$-Eigenschaft erfüllt. Ein System (V, v_0, R, T) ist *sicher*, wenn der Initialzustand v_0 sicher ist und alle Zustände, die von v_0 durch die Zustandsübergangsfunktion T erreichbar sind, sicher sind.

Das nachfolgende Theorem legt ein Vorgehen nahe, mit dem Sicherheit gemäß BLP durch Induktion nachgewiesen werden kann. Dabei zeigt man, daß der Anfangszustand eines Systems sicher ist und daß sichere Zustände durch die Zustandsübergangsfunktion in Zustände überführt werden, die wiederum sicher sind. Mit Hilfe des Theorems, das als *grundlegendes Theorem der Sicherheit* (engl. *Basic Security Theorem*) bezeichnet wird, kann man dann auf die Sicherheit des Systems schließen.

Theorem 14. *Ein System (V, v_0, R, T) ist genau dann* sicher *wenn v_0 ein sicherer Zustand ist und für beliebige von v_0 erreichbare Zustände $v = (F_C, M)$, $v' = (F'_C, M')$ mit $\exists r \in R. T(v,r) = v'$, für alle $s \in S$ und für alle $o, o_l, o_h \in O$ die folgenden Bedingungen gelten:*

- *wenn* $\textit{read} \in M'[s,o]$ *und* $\textit{read} \notin M[s,o]$ *dann* $F'_C(o) \leq F'_C(s)$,
- *wenn* $\textit{read} \in M[s,o]$ *und* $F'_C(o) \not\leq F'_C(s)$ *dann* $\textit{read} \notin M'[s,o]$,
- *wenn* $\textit{read} \in M'[s,o_l]$, $\textit{write} \in M'[s,o_h]$ *und* $\textit{read} \notin M[s,o_l]$ *oder* $\textit{write} \notin M[s,o_h]$ *dann* $F'_C(o_l) \leq F'_C(o_h)$[4] *und*
- *wenn* $\textit{read} \in M[s,o_l]$, $\textit{write} \in M[s,o_h]$ *und* $F'_C(o_l) \not\leq F'_C(o_h)$ *dann* $\textit{read} \notin M'[s,o_l]$ *oder* $\textit{write} \notin M'[s,o_h]$.

[4] Man beachte, daß „wenn $\textit{read} \in M'[s,o_l]$, $\textit{write} \in M'[s,o_h]$, $\textit{read} \notin M[s,o_l]$, und $\textit{write} \notin M[s,o_h]$, dann $F'_C(o_l) \leq F'_C(o_h)$“, wie in [31] gefordert, für diesen Fall nicht ausreicht.

Man beachte, daß Theorem 14 eine Instantiierung der üblichen Berechnungsinduktion für Zustandsmaschinen ist, die als Hilfsmittel beim Nachweis der Sicherheit eingesetzt werden kann. Es bietet jedoch kein zusätzliches Argument dafür, daß die Definition von Sicherheit gemäß BLP angemessen ist. Für eine ausführlichere Diskussion dieses Aspektes siehe auch [30, 31].

Bemerkung 15 (Vergleich zum Originalmodell von BLP). Das ursprüngliche Modell von BLP in [15] umfaßt neben den zuvor beschriebenen Aspekten ein *need-to-know* Prinzip, das durch eine zweite Zugriffsmatrix modelliert wird. Die beiden Zugriffsmatrizen sind konjunktiv verknüpft, d. h. der Referenzmonitor gewährt Zugriffe nur dann, wenn sie in beiden Zugriffsmatrizen eingetragen sind. Dieses wird durch eine weitere Sicherheitseigenschaft, die ds-property, sichergestellt. Dadurch wird die Modellierung komplexer als die hier beschriebene Variante. Ob das need-to-know Prinzip in einem Sicherheitsmodell berücksichtigt werden sollte, hängt sicherlich von der jeweiligen Anwendung ab. Zu beachten ist, daß die Modellierung von need-to-know in [15] auf *discretionary access control* basiert. In [15] wird neben den Zugriffsarten *read* und *write* auch *append* und *execute* behandelt. Zugriffe, die sowohl lesend als auch schreibend sind, werden als *append* klassifiziert und solche, die weder lesend noch schreibend sind, als *execute*.

Formalisierung einer Sicherheitspolitik

Das Modell nach Bell und La Padula bietet einen generischen Rahmen, in dem verschiedene Sicherheitspolitiken zur Zugriffskontrolle definiert werden können. Im vorigen Abschnitt wurden die Menge S der Subjekte, die Menge O der Objekte und die verbandsgeordnete Menge $(L, \leq)$ der Vertraulichkeitsstufen nicht näher spezifiziert. Ebenso wurden für die Zustandsmaschine die konkreten Definitionen des Anfangszustands v_0, der Menge R der Anfragen und der Zustandsübergangsfunktion T offengelassen. Durch eine Instantiierung dieser Parameter und dem Nachweis der Sicherheit der Zustandsmaschine (gemäß BLP, d. h. einfache Sicherheits- und $\star$-Eigenschaft müssen erfüllt sein) erhält man ein formales Modell einer konkreten Sicherheitspolitik. Die zu instantiierenden Parameter sind zusammenfassend in Tabelle 6.4 angegeben.

In [15] wird eine solche Instantiierung durchgeführt, um eine Sicherheitspolitik für das MULTICS-Betriebssystem zu definieren. Die Menge der Subjekte, der Objekte und der Vertraulichkeitsstufen werden in Anlehnung an MULTICS instantiiert. Die Zustandsübergangsfunktion wird durch eine Menge von Regeln definiert, wobei für jede Regel bewiesen wird, daß sie einen sicheren Zustand in einen Zustand überführt, der wiederum sicher ist, und somit die Voraussetzungen von Theorem 14 erfüllt sind. Die Regeln abstrahieren von den eigentlichen MULTICS-Kernelfunktionen. Die Korrespondenzen zwischen Kernelfunktionen und Regeln sind in [15] angegeben.

S	Subjekte	z. B. $\{s_1, s_2, \ldots, s_m\}$
O	Objekte	z. B. $\{o_1, o_2, \ldots, o_n\}$
L	Vertraulichkeitsstufen	z. B. {*offen, GEHEIM,...*}
$\leq$	Ordnung auf L	$\leq \subseteq L \times L$
v_0	Initialzustand	$v_0 \in V$
R	Anfragen	
T	Zustandsübergangsfunktion	$T : V \times R \rightarrow V$

Tabelle 6.4: Parameter von BLP

Annahmen an die Einsatzumgebung

Mit dem BLP Modell definierte Sicherheitspolitiken setzen voraus, daß ein Umgehen der Zugriffskontrolle unmöglich ist. Außerdem müssen alle für die Zugriffskontrolle relevanten Subjekte und Objekte durch Elemente in S und O modelliert sein. Insbesondere müssen lokale Speicher von Prozessen durch Elemente von O modelliert werden. Werden solche lokalen Speicher nicht modelliert, so können diese als Zwischenspeicher benutzt werden, mit deren Hilfe Inhalte von Objekten mit hoher Vertraulichkeitsstufe in Objekte mit niedriger Vertraulichkeitsstufe kopiert werden. Ein solches downgrading der Vertraulichkeitsstufen verletzt jedoch die Sicherheit des Systems.

Bekannte Grenzen des Sicherheitsmodells

Das BLP-Modell ist anerkannt und seine Verwendung ist weit verbreitet. Trotzdem läßt es die Definition von Sicherheitspolitiken zu, die von einem intuitiven Gesichtspunkt keine angemessene Definition von Sicherheit bieten. Diese Problematik wird am System Z, einem Beispiel aus [32], deutlich. Der Anfangszustand von Z ist ein beliebiger sicherer Zustand und die Transitionen von Z verändern F_C, so daß alle Subjekte und Objekte die niedrigste Vertraulichkeitsstufe erhalten, wenn ein Zugriff versucht wird. Auf diese Weise ist jeder versuchte Zugriff erfolgreich. Das System Z erfüllt die Definition von Sicherheit gemäß BLP, bietet jedoch keinen Schutz für die Objekte vor Zugriffen. Hier zeigt sich, daß es oft nötig ist, die möglichen Veränderungen von F_C und M auf eine Weise einzuschränken, die über BLP hinaus geht. Ein generischer Rahmen für solche Einschränkungen wird in [31] beschrieben.

Werden Veränderungen von F_C völlig verboten, so können die Sicherheitsstufen von Objekten, bzw. deren Inhalt, nicht verringert, sondern nur erhöht werden. In der Praxis ist dieses oft eine zu scharfe Forderung, die mit anderen Systemanforderungen inkompatibel ist. Um dieses Problem zu umgehen, werden meist vertrauenswürdige Prozesse eingesetzt. Ein *vertrauenswürdiger Prozeß* darf

die Sicherheitspolitik umgehen und somit auch Objekte herabstufen (engl. downgrading). Natürlich kann die Sicherheitspolitik für das gesamte System dann nur modulo der Vertrauenswürdigkeit solcher Prozesse garantiert werden. Vertrauenswürdigkeit heißt hier also, daß dem Prozeß vertraut werden *muß*. Ob den jeweiligen Prozessen tatsächlich vertraut werden *kann*, ist für die jeweilige Anwendung oft eine äußerst schwierige Frage. In diesem Bereich gibt es noch einigen Forschungsbedarf.

Während vor allem in den siebziger Jahren der allgemeine Sicherheitsbegriff und das Bell/La Padula synonym verwandt wurden, wird BLP heute nicht mehr als eine Definition von Sicherheit, sondern eher als eine technische Realisierung verstanden, mit der Sicherheit (z. B. im Sinne von Informationsflußkontrolle) erreicht werden kann.

Varianten von Bell/La Padula

Im Vergleich zur hier vorgestellten Version von Bell/La Padula lassen sich zwei Möglichkeiten zur Bildung von Varianten unterscheiden. Erstens, die Kombination von mandatory access control mit discretionary access control, die durch eine weitere Zugriffsmatrix realisiert wird. Zugriffe werden nur dann gewährt, wenn die entsprechende Zugriffsart in beiden Zugriffsmatrizen eingetragen ist. Ein Beispiel hierfür ist die Originalvariante von Bell/La Padula, auf die bereits in Bemerkung 15 eingegangen wurde. Zweitens kann die Veränderbarkeit von F_C und M bei den Zustandsübergängen eingeschränkt werden. Z.B. kann die $\star$-property gemäß Bemerkung 12 abgeändert werden. Die resultierende Forderung ist etwas stärker als die $\star$-Eigenschaft aus Definition 11. In Abschnitt 6.2.2 wird diese Variante für das Sicherheitsmodell nach Biba illustriert. In [31] wird ein genereller Rahmen angegeben, der die Definition verschiedener Varianten von BLP erlaubt, die sich darin unterscheiden, wie F_C dynamisch verändert werden kann. Das Spektrum reicht von beliebigen Veränderungen von F_C über eine Einschränkung der Veränderung, so daß nur bestimmte Subjekte F_C verändern können, bis zu einem völligen Verbot einer Veränderung von F_C.

6.2.2 Biba

Das Modell von Biba [16] erlaubt die Definition von Sicherheitspolitiken, durch die die Veränderung von passiven Objekte, wie z. B. Dateien, durch aktive Subjekte, wie z. B. Prozessen, geregelt wird. Das Ziel einer solchen Sicherheitspolitik ist es, den Inhalt der Objekte vor unerlaubter Veränderung zu schützen, ohne dabei den Subjekten vertrauen zu müssen. Die hier beschriebene Variante entspricht dem Originalmodell, benutzt aber die Begriffe aus Abschnitt 6.2.1.

Wie im Modell nach Bell/La Padula basiert auch die Zugriffskontrolle in Biba auf Referenzmonitoren. Entsprechend müssen die drei Voraussetzungen Vollständigkeit, Geschütztheit und Korrektheit sichergestellt werden. Die Zugriffsmatrix kann in der hier beschriebenen Variante ausgehend von einem Initialzustand dynamisch verändert werden, wobei die möglichen Veränderungen stärker eingeschränkt werden als in Abschnitt 6.2.1. Die Zugriffsmatrix muß zwei Eigenschaften erfüllen, die zur einfachen Sicherheitseigenschaft und zur $\star$-Eigenschaft von BLP korrespondieren. Die erste Eigenschaft verlangt, daß Subjekte nur solche Objekte schreiben können, welche die gleiche oder eine niedrigere Integritätsstufe haben. Ein unerlaubtes Heraufstufen von Informationen wird durch die zweite Eigenschaft verboten. Die Kombination beider Eigenschaften garantiert, daß Subjekte nur den Inhalt von Objekten verändern, denen die gleiche oder eine niedrigere Integritätsstufe zugeordnet ist.

Im Unterschied zu vorausgehenden Ansätzen, wie dem Lampson Protection Model [26], in denen die zulässigen Änderungen der Zugriffsmatrix über eine Menge von Regeln definiert ist, werden in Biba den Subjekten und Objekten Integritätsstufen zugeordnet. Wie auch in Abschnitt 6.2.1 erlaubt es dieser Ansatz, Regeln zur Änderung der Matrix als korrekt nachzuweisen.

Definition des formalen Modells

Nachfolgend wird eine Definition des Sicherheitsmodells nach Biba [15] beschrieben. Zur Vereinheitlichung der Darstellung werden dieselben Begriffe wie in Abschnitt 6.2.1 verwendet. In [15] werden verschiedene Sicherheitspolitiken diskutiert und die heute als Sicherheitspolitik nach Biba bekannte wird dort als *strikte Integritätspolitik* (engl. strict integrity policy) bezeichnet.

Das System zur Zugriffskontrolle wird als deterministische Zustandsmaschine modelliert. Eine Zustandsmaschine wird durch ein Tupel (V, v_0, R, T) definiert, wobei V eine Menge von Zuständen, $v_0 \in V$ ein Initialzustand, R (für engl. request) eine Menge möglicher Anfragen und $T : (V \times R) \rightarrow V$ eine Zustandsübergangsfunktion (für engl. transition) ist, die das System auf Anfrage von einem Zustand in einen anderen überführt. Die Zustände der Maschine werden im folgenden noch näher beschrieben.

Subjekte und Objekte werden durch Mengen S und O modelliert. Subjekte können z. B. Personen oder Prozesse sein. Beispiele für Objekte sind Dateien oder Speicherbereiche. Eine Funktion $F_I : S \cup O \rightarrow I$ klassifiziert jedes Subjekt und Objekt, indem es ihnen eine Integritätsstufe aus einer Menge I (für engl. integrity) zuordnet. Mögliche Integritätsstufen sind z. B. „offen“ oder „GEHEIM“. Auf den Elementen von I existiert eine partielle Ordnung $\leq$[5], mit der I eine verbandsge-

[5] D. h. $\leq$ ist reflexiv ($\forall i \in I . i \leq i$), antisymmetrisch ($\forall i_1, i_2 \in I . (i_1 \leq i_2 \wedge i_2 \leq i_1) \Rightarrow i_1 =$

ordnete Menge[6] bildet. Die möglichen Zugriffsarten von Subjekten auf Objekte werden durch die Menge $A = \{\textit{read}, \textit{write}\}$ (für engl. access) angegeben. Die aktuellen Zugriffsrechte werden in einer Zugriffsmatrix $M : S \times O \rightarrow P(A)$ festgehalten, wobei in jeder Zelle der Matrix die aktuelle Menge der erlaubten Zugriffsarten eingetragen ist. Die Menge $\{\textit{write}\}$ z. B. erlaubt einem Subjekt schreibende, aber keine lesenden Zugriffe auf ein Objekt.

Während die Mengen S, O, I und A für eine konkrete Sicherheitspolitik statisch festgelegt werden, können die Funktion F_I und die Zugriffsmatrix M dynamisch verändert werden. Zustände der Zugriffskontrolle werden daher als Paare (F_I, M) modelliert, woraus sich die Menge V der Zustände der Zustandsmaschine ergibt. Die Merkmale von Biba sind zusammenfassend in Tabelle 6.5 aufgeführt.

S	Subjekte	z. B. $\{s_1, s_2, \ldots, s_m\}$
O	Objekte	z. B. $\{o_1, o_2, \ldots, o_n\}$
I	Integritätsstufen	z. B. $\{$*offen*,*GEHEIM*,$\ldots\}$
$\leq$	Ordnung auf I	$\leq \subseteq\ I \times I$
A	Zugriffsarten	$A = \{\textit{read}, \textit{write}\}$
F_I	Klassifikation	$F_I : S \cup O \rightarrow I$
M	Zugriffsmatrix	$M : S \times O \rightarrow P(A)$
V	Zustände	$V : (S \cup O \rightarrow I) \times (S \times O \rightarrow P(A))$
v_0	Initialzustand	$v_0 \in V$
R	Anfragen	
T	Zustandsübergangsfunktion	$T : V \times R \rightarrow V$

Tabelle 6.5: Merkmale von Biba

Durch die Modellierung als Zustandsmaschine wird eine induktive Definition der Sicherheit von Systemen möglich.

Definition 16 (Sicherheitseigenschaften von Biba).
Ein Zustand $v = (F_I, M)$ ist *sicher* (gemäß Biba), wenn v die folgenden beiden Eigenschaften erfüllt:

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

Ein System (V, v_0, R, T) ist *sicher*, wenn der Initialzustand v_0 sicher ist und alle Zustände, die von v_0 durch die Zustandsübergangsfunktion T erreichbar sind, wiederum sicher sind.

i_2) und transitiv ($\forall i_1, i_2, i_3 \in i. (i_1 \leq i_2 \wedge i_2 \leq i_3) \Rightarrow i_1 \leq i_3$).

[6] D. h. zu je zwei Elementen $i_1, i_2 \in I$ existieren eine kleinste obere Schranke, das Supremum $\bigvee\{i_1, i_2\}$, und eine größte untere Schranke, das Infimum $\bigwedge\{i_1, i_2\}$.

Die erste dieser beiden Bedingungen korrespondiert zur einfachen Sicherheitseigenschaft (siehe Definition 10) und die zweite zur $\star$-Eigenschaft (siehe Bemerkung 12) in BLP. Wir bezeichnen die Bedingungen deshalb auch als *duale einfache Sicherheitseigenschaft* und als *duale $\star$-Eigenschaft*.

Die hier angegebene duale $\star$-Eigenschaft ist logisch dual zur $\star$-Eigenschaft in Bemerkung 12, jedoch nicht zu der in Definition 11. Da sich die zu Definition 11 logisch duale Eigenschaft einfach konstruieren läßt, wurde hier zur Illustration der Variantenvielfalt eine andere Variante vorgestellt. Ob ein *write*-Eintrag in der Zugriffsmatrix zulässig ist, hängt bei der hier angegebenen dualen $\star$-Eigenschaft nur von den Integritätsstufen, aber nicht von den sonstigen Einträgen in der Zugriffsmatrix ab (im Unterschied zu Definition 11).

Bemerkung 17 (Vergleich zum Originalmodell von Biba). Das ursprüngliche Modell nach Biba [16] umfaßt neben Zugriffen *read* und *write* auch noch die dynamische Erzeugung von neuen Subjekten *invoke*. Mit dieser Erweiterung muß in der Definition eines sicheren Systems (Definition 16) zusätzlich gefordert werden, daß ein neu erzeugtes Subjekt keine höhere Integritätsstufe besitzt, als das Subjekt, durch das es erzeugt wurde.

Formalisierung einer Sicherheitspolitik

Das Modell nach Biba bietet einen generischen Rahmen, in dem verschiedene Sicherheitspolitiken zur Zugriffskontrolle definiert werden können. Im vorigen Abschnitt wurden die Menge S der Subjekte, die Menge O der Objekte und die verbandsgeordnete Menge $(I, \leq)$ der Integritätsstufen nicht näher spezifiziert. Ebenso wurden für die Zustandsmaschine die konkreten Definitionen des Anfangszustands v_0, der Menge R der Anfragen und der Zustandsübergangsfunktion T offengelassen. Durch eine Instantiierung dieser Parameter und dem Nachweis der Sicherheit der Zustandsmaschine (gemäß Biba, d. h. die Bedingungen aus Definition 16 müssen erfüllt sein) erhält man ein formales Modell einer konkreten Sicherheitspolitik. Die zu instantiierenden Parameter sind zusammenfassend in Tabelle 6.6 angegeben.

Die Klassifikation der Integrität von Subjekten entspricht oft der Klassifikation der Vertraulichkeit. Die Zuverlässigkeit von Subjekten unterscheidet sich meist nicht in Bezug auf Vertraulichkeit und Integrität. Ein Subjekt, das vertrauliche Daten geheim hält, wird im allgemeinen ein System auch nicht sabotieren. Die Klassifikation der Integrität von Objekten kann sich hingegen stark unterscheiden. Während Vertraulichkeit eine unerwünschte Weitergabe von vertraulicher Informationen verhindern soll, verhindert Integrität eine Sabotage von Informationen. Integritätsstufen in I können den Vertraulichkeitsstufen L in BLP entsprechen, können aber auch anders gewählt werden.

S	Subjekte	z. B. $\{s_1, s_2, \ldots, s_m\}$
O	Objekte	z. B. $\{o_1, o_2, \ldots, o_n\}$
I	Integritätsstufen	z. B. $\{$*offen, GEHEIM,...*$\}$
$\leq$	Ordnung auf I	$\leq \subseteq I \times I$
v_0	Initialzustand	$v_0 \in V$
R	Anfragen	
T	Zustandsübergangsfunktion	$T : V \times R \rightarrow V$

Tabelle 6.6: Parameter von Biba

Annahmen an die Einsatzumgebung

Mit dem Modell nach Biba definierte Sicherheitspolitiken setzen voraus, daß ein Umgehen der Zugriffskontrolle unmöglich ist. Außerdem müssen alle für die Zugriffskontrolle relevanten Subjekte und Objekte durch Elemente in S und O modelliert sein. Insbesondere müssen lokale Speicher von Prozessen durch Elemente von O modelliert werden. Werden solche lokalen Speicher nicht modelliert, so können diese als Zwischenspeicher benutzt werden, mit deren Hilfe Inhalte von Objekten mit niedriger Integritätsstufe in Objekte mit hoher Integritätsstufe kopiert werden. Ein solches upgrading der Integritätsstufen verletzt jedoch die Sicherheit des Systems.

Bekannte Grenzen des Sicherheitsmodells

Wegen der starken konzeptionellen Analogie zwischen den Sicherheitsmodellen nach Bell/La Padula und Biba (siehe auch Abschnitt 6.3.1) gelten für Biba dieselben Grenzen wie für BLP. Wir verweisen deshalb auf Abschnitt 6.2.1.

Varianten von Biba

Unter Verwendung der engen Korrespondenz zwischen den Sicherheitsmodellen nach Bell/La Padula und Biba (siehe auch Abschnitt 6.3) lassen sich Varianten, die für BLP entwickelt wurden, leicht auf Biba anpassen. Ebenso können Varianten von Biba auf BLP angepaßt werden. Wir verweisen daher auf die Beschreibung der Varianten von BLP in Abschnitt 6.2.1.

6.3 Beziehung zwischen den Modellen

Zwischen MLS-Sicherheitspolitiken und transitiven Interference-Relationen gibt es eine enge Korrespondenz. Jede MLS-Sicherheitspolitik entspricht einer transitiven Interference-Relation und umgekehrt entspricht jede transitive Interference-Relation einer MLS-Politik [40]. Diese Korrespondenz bildet gewissermaßen die Grundlage für unsere Darstellung von bekannten Sicherheitsmodellen, mit denen sich MLS-Politiken formal darstellen lassen. Offensichtlich lassen sich solche Politiken mit BLP und Biba formalisieren. In diesen Modellen werden den Subjekten ja direkt Sicherheitsstufen (bzw. Integritätsstufen) zugeordnet. Eine Formalisierung von MLS-Politiken mit dem Noninterference-Ansatz wird mit der zuvor angesprochenen Korrespondenz möglich.

Es stellt sich die Frage, in welcher Beziehung die beschriebenen Sicherheitsmodelle zueinander stehen. BLP und Biba sind duale Sicherheitsmodelle, was eine wechselseitige Übertragung von Konzepten und theoretischen Resultaten ermöglicht. Im Vergleich zu Noninterference können BLP und Biba als Implementierungen angesehen werden.

6.3.1 Korrespondenz zwischen Bell/La Padula und Biba

Die Sicherheitsmodelle nach Bell/La Padula und Biba sind dual zueinander. Dieses wird an der Beziehung zwischen der einfachen Sicherheitseigenschaft und der $\star$-Eigenschaft in BLP mit den korrespondierenden Sicherheitseigenschaften in Biba deutlich.

Während die einfache Sicherheitseigenschaft in BLP verbietet, daß ein Subjekt auf ein Objekt mit einer höheren Vertraulichkeitsstufe lesend zugreifen kann (verhindert direkten Verlust der Vertraulichkeit), verbietet die duale Eigenschaft in Biba, daß ein Subjekt ein Objekt mit einer höheren Integritätsstufe schreiben kann (verhindert direkten Verlust der Integrität).

Während die $\star$-Eigenschaft in BLP verbietet, daß ein Subjekt auf ein Objekt schreibend zugreifen kann, wenn es gleichzeitig ein anderes Objekt mit einer höheren Sicherheitsstufe lesen kann (verhindert indirekten Verlust der Vertraulichkeit durch Kopieren), verbietet die duale Eigenschaft in Biba, daß ein Subjekt ein Objekt mit einer Integritätsstufe schreiben kann, wenn es gleichzeitig ein Objekt mit einer niedrigeren Integritätsstufe lesen kann (verhindert indirekten Verlust der Integrität durch Kopieren).[7]

Anhand dieser Korrespondenz ist eine Übertragung von Varianten des einen Sicherheitsmodells auf das andere leicht möglich. Ebenso können theoretische

[7] Diese Eigenschaft weicht nur deshalb von der entsprechenden Eigenschaft in Abschnitt 6.2.2 ab, weil sich die Varianten unterscheiden.

Resultate übertragen werden, so daß z. B. ein grundlegendes Theorem der Sicherheit (engl. *Basic Security Theorem*) auch für Biba gilt.

6.3.2 Zugriffskontrolle und Noninterference

Das Sicherheitsmodell nach Bell/La Padula kann als Spezialisierung von Noninterference verstanden werden [40]. Entsprechend der zuvor beschriebenen Dualität der Sicherheitsmodelle ist es ebenfalls möglich, das Modell nach Biba als eine Spezialisierung von Noninterference zu verstehen. Daher werden die Modelle nach BLP und Biba auch als *Implementierungen* von Noninterference bezeichnet.

Die Grundlage hierfür ist die enge Verwandtschaft zwischen MLS-Politiken und transitiven Interference-Relationen. Für BLP erhält man eine Interference-Relation durch die Festlegung

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

wobei d_1 und d_2 im einen Fall (in $d_1 \rightsquigarrow d_2$) beliebige Sicherheitsdomänen und im anderen Fall (in $F_C(d_1) \leq F_C(d_2)$) die korrespondierenden Subjekte oder Objekte bezeichnen. Entsprechendes gilt für Biba, wobei die Interference Relation allerdings durch $d_1 \rightsquigarrow d_2$ gdw. $F_I(d_2) \leq F_I(d_1)$ definiert wird.

Ein technisches Problem in dieser Definition ergibt sich durch die unterschiedliche Handhabung der Noninterference-Relation $\not\rightsquigarrow$ in Abschnitt 6.1 und der Klassifikationsfunktionen F_C und F_I in den Abschnitten 6.2.1 und 6.2.2. Während F_C und F_I Bestandteile des Zustands sind, und damit dynamisch verändert werden können, ist die Noninterference-Relation $\not\rightsquigarrow$ nicht dynamisch veränderbar. Entsprechend der oben angegebenen Korrespondenz zwischen $\rightsquigarrow$ und F_C wird in [40] (Korollar 1) für den Nachweis, daß BLP eine Spezialisierung von Noninterference ist, angenommen, daß F_C nicht dynamisch veränderbar ist. Um allgemein BLP als Spezialisierung von Noninterference nachzuweisen (d. h. mit dynamischer Veränderung von F_C), müßte eine Variante von Noninterference konstruiert werden, die eine dynamische Veränderung der Noninterference-Relation zuläßt.

6.4 Verwendbarkeit klassischer Modelle

Noninterference und die Modelle nach Bell/La Padula und Biba sind generische Sicherheitsmodelle, mit denen Anforderungen an die Vertraulichkeit und an die Integrität definiert werden können (siehe Tabelle 6.7). Jedes dieser generischen Modelle steht für eine Klasse von Sicherheitsmodellen, wobei man ein konkretes Modell durch Instantiierung erhält (dazu siehe auch Abschnitt 6.5).

	Vertraulichkeit	Integrität
BLP	√	
Biba		√
Noninterference	√	√

Tabelle 6.7: Verwendbarkeit von BLP, Biba und Noninterference

Während BLP und Biba Modelle für Politiken zur Zugriffskontrolle sind, ist Noninterference ein Modell für Politiken zur Informationsflußkontrolle. Sicherheitsmodelle, die auf einer Zugriffskontrolle basieren, kontrollieren den Zugriff auf Objekte, welche zu schützenden Daten enthalten. Die Vertraulichkeit (bzw. die Integrität) der Daten wird also nicht direkt durch eine Zugriffsbeschränkung auf die Daten, sondern durch eine Zugriffsbeschränkung auf die Informationscontainer erreicht. Im allgemeinen kann mit solchen Zugriffskontrollen, die Existenz von verdeckten Kanälen (engl. covert channels) nicht ausgeschlossen werden. Zeitabhängige verdeckte Kanäle (engl. timing channels) entstehen z. B. wenn ein trojanisches Pferd, das Verschicken einer Nachricht, die keine vertraulichen Informationen erhält, in Abhängigkeit von geheimen Daten verzögert. Aus dieser Verzögerung kann ein Angreifer dann die geheimen Daten rekonstruieren, auf die er sonst (wegen der Zugriffskontrolle) nicht zugreifen könnte. Verdeckte Kanäle können nicht nur durch zeitabhängiges Verhalten sondern z. B. auch durch den unterschiedlich starken Gebrauch gemeinsamer Resourcen, wie Speicherbedarf, entstehen. Um die Möglichkeit verdeckter Kanäle auszuschließen reicht es nicht aus, den Zugriff auf Informationscontainer einzuschränken. Vielmehr muß der Zugriff auf die darin enthaltenen Daten eingeschränkt werden. Dieses wird durch Politiken zur Informationsflußkontrolle, wie sie mit Noninterference formal modelliert werden können, erreicht. Dadurch kann mit Noninterference die Existenz von verdeckten Kanälen ausgeschlossen werden.

Daß die formale Modellierung von Politiken, die auf Zugriffs- oder Informationsflußkontrolle basieren, beim aktuellen Stand der Technik möglich ist, wird in der CC festgehalten. Die formale Modellierung für solche Politiken wird explizit gefordert:

> [...] 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. [...] [1, Teil 3, Abs. 367] (vergleiche auch Abschnitt 3.3.2)

6.5 Instantiierung klassischer Modelle

Die zuvor beschriebenen Sicherheitsmodelle, Noninterference, BLP und Biba, sind alle generisch. Dadurch ergibt sich ein großes Anwendungsspektrum für jeden einzelnen dieser Ansätze.

Die Konstruktion eines Sicherheitsmodells kann somit in zwei Schritten erfolgen. Im ersten Schritt wird ein generisches Sicherheitsmodell, wie z. B. Noninterference, BLP oder Biba, ausgewählt und im zweiten Schritt werden dann alle Parameter des Modells instantiiert.

Zu beachten ist, daß sich ein konkretes Sicherheitsmodell erst durch die Instantiierung der Parameter ergibt. Eine Aussage wie z. B. „Wir benutzen das Modell nach Bell/La Padula.“, gibt nur die Entscheidung für eine bestimmte Klasse von Sicherheitsmodellen an. Da diese Klasse extrem groß ist, kann die intendierte Instantiierung der Parameter im allgemeinen nicht offensichtlich sein, sondern sie muß explizit angegeben werden. Für ein formales Sicherheitsmodell (wie von ITSEC oder CC gefordert) muß die Instantiierung selbstverständlich formal angegeben werden. Eine informelle Beschreibung der Instantiierung kann eine formale Instantiierung nicht angemessen ersetzen.

Als Hilfestellung für eine solche formale Instantiierung wurden alle Parameter der verschiedenen Sicherheitsmodelle in den jeweiligen Abschnitten angegeben. Außerdem wurde die Bedeutung der einzelnen Parameter erläutert. Dazu siehe jeweils den Absatz „Formalisierung einer Sicherheitspolitik“.

Kapitel 7

Prüfung formaler Sicherheitsmodelle

Weil (formale) Sicherheitsmodelle zusätzliche Vertrauenswürdigkeit schaffen sollen, daß die in der funktionalen Spezifikation enthaltenen Sicherheitsfunktionen die EVG-Sicherheitspolitiken durchsetzen (vgl. [1, Teil 3, Abs. 365]), kommt ihrer Prüfung eine besondere Bedeutung zu. Das Ziel der Prüfung formaler Sicherheitsmodelle besteht in der Bestätigung der geforderten Übereinstimmung zwischen funktionaler Spezifikation und in den funktionalen Anforderungen enthaltenen Sicherheitspolitiken.

Die erforderlichen Aktivitäten des Evaluators beziehen sich im Wesentlichen auf drei Bereiche:

- Die Übereinstimmung zwischen den funktionalen Anforderungen und den Sicherheitsprinzipien (Security Principles)
- Die Übereinstimmung zwischen den Sicherheitsprinzipien (Security Principles) und den Sicherheitscharakteristika (Security Characteristics)
- Die Übereinstimmung zwischen den Sicherheitscharakteristika (Security Charakteristics) und der funktionalen Spezifikation

In den nachfolgenden Abschnitten werden diese Schwerpunkte erläutert. Appendix A enthält eine Zusammenfassung in Form einer an die Art der Darstellung von [3] angelehnten Subaktivität. Die Ausführungen in diesem Kapitel dienen als Begründung und zusätzliche Erläuterung der dort vorgeschlagenen Work Units.

7.1 Sicherheitspolitik vs. Sicherheitsmodell

Die Übereinstimmung mit der in den funktionalen Anforderungen formulierten Sicherheitspolitik des EVG stellt die Verbindung zwischen den Sicherheitsvorgaben (ST) und dem formalen Sicherheitsmodell her. Dieser Bereich stellt die höchsten Anforderungen an die Kompetenz des Evaluators, denn hier ist ein komplexer Übergang zwischen informell beschriebenen Anforderungen und formal spezifizierten Sicherheitseigenschaften und -merkmalen zu bewerten:

1. Der Evaluator muß ggf. in der Lage sein, zu beurteilen, ob die nicht formal spezifizierten Sicherheitspolitiken nach dem Stand der Technik tatsächlich nicht formal modelliert werden können (vgl. Work Units ADV_SPM-3 und ADV_SPM-4).

2. Der Evaluator muß in der Lage sein, die Bedeutung der formalen Spezifikation der Sicherheitseigenschaften und -merkmale vor dem Hintergrund der Sicherheitsvorgaben unabhängig von den möglicherweise fehlerhaften oder unvollständigen Erläuterungen des Entwicklers zu erfassen (vgl. Work Unit ADV_SPM-2).

3. Der Evaluator muß die Sicherheitseigenschaften und -merkmale systematisch analysieren mit dem Ziel, zu bestätigen, daß sowohl alle Sicherheitsziele (z. B. als sichere Zustände) als auch alle Bedrohungen (z. B. als unsichere Zustände) in der Modellierung deutlich erkennbar sind (vgl. Work Unit ADV_SPM-5). Hierzu gehört ebenfalls die Überprüfung der übrigen Sicherheitsumgebung (Annahmen und organisatorische Sicherheitspolitiken) auf ihre angemessene Berücksichtigung.

4. Der Evaluator muß unter Berücksichtigung der Erklärung/Interpretation des Entwicklers systematisch nach Lücken in der formalen Spezifikation der Sicherheitseigenschaften und -merkmale suchen mit dem Ziel, die vollständige Abdeckung der in den funktionalen Sicherheitsanforderungen formulierten Sicherheitsprinzipien und -charakteristika zu bestätigen (vgl. Work Units ADV_SPM-3, ADV_SPM-4 und ADV_SPM-9).

5. Der Evaluator muß unter Berücksichtigung der Erklärung/Interpretation des Entwicklers systematisch nach Abweichungen von den in den funktionalen Sicherheitsanforderungen formulierten Sicherheitsprinzipien und -charakteristika suchen mit dem Ziel, die Konsistenz mit den Sicherheitseigenschaften und -merkmalen zu bestätigen (vgl. Work Units ADV_SPM-3, ADV_SPM-4 und ADV_SPM-8).

7.2 Sicherheitseigenschaften vs. -merkmale

Die Übereinstimmung zwischen den Sicherheitseigenschaften (Security Properties) und den Sicherheitsmerkmalen (Security Features) verknüpft die beiden Teile des formalen Sicherheitsmodells mit einem formalen Beweis. Dieser Beweis stellt das zentrale Bindeglied zwischen der Anforderungs- und der Entwurfsspezifikation dar. Hier ist die Gültigkeit der Eigenschafts- und Konsistenzbeiweise zu bewerten.

1. Der Evaluator muß das formale Sicherheitsmodell systematisch analysieren mit dem Ziel, zu bestätigen, daß alle Angriffsmöglichkeiten in der Modellierung deutlich erkennbar sind (vgl. Work Unit ADV_SPM-5). Hierbei sind z. B. die Wahl von Datenstrukturen oder die Spezifikation von Kommunikationskanälen zu berücksichtigen.

2. Der Evaluator muß den formalen Beweis der Übereinstimmung zwischen Sicherheitseigenschaften und Sicherheitsmerkmalen im Hinblick auf die Eignung und Vollständigkeit der Beweisverpflichtungen sowie im Hinblick auf die Gültigkeit der verwendeten Argumente bewerten (vgl. Work Unit ADV_SPM-6). Alle Teilbeweise müssen vollständig geführt sein. Dies muß vom Entwickler auf geeignete Weise nachgewiesen und vom Evaluator mindestens durch zufällige Stichproben überprüft werden.

3. Der Evaluator muß die interne Konsistenz des formalen Sicherheitmodells bestätigen. Hierzu ist die Gültigkeit der verwendeten Argumente zu bewerten (vgl. Work Unit ADV_SPM-7).

7.3 Sicherheitsmodell vs. -funktionen

Die Übereinstimmung mit der funktionalen Spezifikation des EVG stellt die Verbindung zwischen dem formalen Sicherheitsmodell und der obersten Ebene der Darstellung der Implementierung her. Sie sorgt dafür, daß die Entwicklung auf der Basis einer gegen die Anforderungen validierten Beschreibung der Funktionalität des EVG aufsetzt. 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. Der Evaluator muß unter Berücksichtigung des Übereinstimmungsnachweises des Entwicklers systematisch nach Lücken in der funktionalen Spezifikation suchen mit dem Ziel, die vollständige Abdeckung der Sicherheitseigenschaften und -merkmale zu bestätigen (vgl. Work Units ADV_SPM-10,

ADV_SPM-12 und ADV_SPM-13). Insbesondere im Fall einer nicht formal dargestellten funktionalen Spezifikation ist hierbei die Gültigkeit der verwendeten Argumente zu bewerten.

2. Der Evaluator muß unter Berücksichtigung des Übereinstimmungsnachweises des Entwicklers systematisch nach Differenzen zu den Sicherheitseigenschaften und -merkmalen suchen mit dem Ziel, die Konsistenz mit der funktionalen Spezifikation zu bestätigen (vgl. Work Units ADV_SPM-11, ADV_SPM-12 und ADV_SPM-13). Insbesondere im Fall einer nicht formal dargestellten funktionalen Spezifikation ist hierbei die Gültigkeit der verwendeten Argumente zu bewerten.

Kapitel 8

Nutzen formaler Sicherheitsmodelle für IT-Hersteller

Ziel der Verwendung formaler Methoden bei der Erstellung von Sicherheitsmodellen ist die Erhöhung des Grades an *Vertrauenswürdigkeit*, die der Sicherheit eines Systems oder Produkts entgegengebracht werden kann. Deutliche Qualitätsverbesserungen ergeben sich in diesem Zusammenhang vor allem aus der

- Eindeutigkeit und damit intersubjektiven Verbindlichkeit von Festlegungen,
- Widerspruchsfreiheit (Konsistenz) der Konzepte untereinander und
- Gültigkeit der postulierten Zusammenhänge.

Die Verwendung von *Beschreibungstechniken*, die mathematisch fundiert sind und die auf dieser Grundlage rechnerunterstützte Analyseverfahren zulassen, entspricht der Vorgehensweise in anderen Ingenieurdisziplinen, wenn Eigenschaften mit einem hohen Maß an Zuverlässigkeit vorhergesagt werden sollen.

In dem formal behandelten Bereich ist die *Bedeutung* der verwendeten Konzepte wie auch die *Folgerbarkeit* mit mathematischer *Absolutheit*[1] festgelegt. Damit gibt es keinen (subjektiven) Interpretationsspielraum in Bezug auf Fragen wie „Was bedeutet diese Aussage?“ und „Welche Folgerungen ergeben sich aus diesen Festlegungen?“. Insbesondere lassen sich auf dieser Grundlage sowohl die Widerspruchsfreiheit der zugrundegelegten Konzepte wie auch die Gültigkeit der zunächst nur unterstellten (oder geforderten) Zusammenhänge (Sicherheitseigenschaften) nachweisen.

Wie in Kapitel 3 im Einzelnen dargestellt, werden formale Methoden von ITSEC und CC für die frühen Phasen der Entwicklung gefordert. Dies entspricht

[1] Die grundsätzlich verbleibende Unsicherheit ergibt sich aus der „Grundlagenproblematik der Mathematik“.

der vielfach geteilten Einschätzung, daß entscheidende, oft *die* entscheidenden Fehler in der Phase des Requirements Engineering unterlaufen. Hier geht es darum, Anforderungen und Lösungskonzepte *abstrakt*, d. h. unabhängig von den Spezifika technischer Realisierungen zu betrachten. Insbesondere dann, wenn (mathematisch) präzise Abstraktionen benötigt werden, gibt es zur Verwendung von formalen Methoden keine Alternative.

Wenn Formale Methoden in *kritischen* Teilbereichen von Gesamtentwicklungen angewendet werden, ergibt sich insgesamt ein deutlicher Zugewinn an Sicherheit, Präzision (der Beschreibung) sowie Nachvollziehbarkeit (von Begründungen) und damit eine erhebliche *Qualitätsverbesserung* bei dem Entwicklungsergebnis. Letztere ist die eigentliche Motivation für die Verwendung von Verfahren, die zunächst einen zusätzlichen Aufwand bedeuten. Die Prüfung und Bewertung weist die erhöhte Vertrauenswürdigkeit nur nach.

Im Folgenden sollen einige der angesprochenen Aspekte näher beleuchtet werden. Wir beginnen mit eine kurzen Einordnung von Formalen Methoden in den Entwicklungsprozeß allgemein.

8.1 Entwicklungsmethodik

Formale Methoden zielen auf eine Verbesserung der *Entwicklungsmethodik*. Je nach Entwicklungsphase ergeben sich dabei unterschiedliche Problemstellungen. Allgemein betrachtet man als Phasen die *Anforderungsdefinition*, die *Konzeption* von Lösungen und die technische *Realisierung*.

Übergeordneter Leitgedanke bei der Gestaltung von Entwicklungsmethodiken sollte die *Beherrschung der Komplexität* sein. Hierzu dienen zwei grundsätzliche Maßnahmen: *Strukturierung* und *Abstraktion*. Während die Strukturierung in möglichst unabhängige Einheiten letztlich die Architektur des Entwicklungsgegenstandes selbst betrifft, geht es bei der Abstraktion um angemessene (bezogen auf die verschiedenen Phasen) Sichtweisen auf diesen Gegenstand.

Statistischen Erhebungen zufolge unterlaufen kritische Fehler am haeufigsten bereits in den frühen Entwicklungsphasen. Im Security-Bereich betrifft dies die Formulierung der Sicherheitseigenschaften (Anforderungsdefinition), die Spezifikation der grundlegenden Sicherheitsfunktionen (Konzeption, abstraktes Systemmodell) und den Architekturentwurf. Gerade aber in diesen Phasen ist eine saubere Strukturierung und vor allem ein angemessener Abstraktionsgrad entscheidend. Unübersichtliche Spezifikationen und eine Vermischung von konzeptuellen Fragen (was) mit technischen Implementierungsdetails (wie) machen eine sinnvolle Analyse unmöglich.

Grundvoraussetzung für ein erfolgreiches Requirements Engineering sind damit Beschreibungsmittel, die Abstraktheit und eine modulare Vorgehensweise mit

einer semantisch fundierten *Analyse* verbinden. *Programmiersprachliche* Konzepte sind zwar überwiegend wohlverstanden und semantisch fundierbar, aber auf dieser Stufe wegen mangelnder Abstraktionsmöglichkeit oft nicht geeignet. Natürlichsprachliche Beschreibungen verwenden weder fixierte Sprachmittel noch ist die jeweils unterstellte Semantik präzisiert. Sie bieten daher auch kaum Ansatzpunkte für weitergehende Analysen. Dies gilt auch dann, wenn die natürliche Sprache begrifflich eingeschränkt wird und zusätzlich graphische Darstellungen zur Strukturierung verwendet werden. Ein häufig festzustellender (methodischer) Fehler besteht darin, von einer verbalen Beschreibung, die zwar eventuell ausreicht, mit Außenstehenden über den Entwicklungsgegenstand zu kommunizieren, jedoch keinerlei technische Reflexion zuläßt, direkt zu implementierungsorientierten Konzepten überzugehen.

Formale Beschreibungstechniken ermöglichen *präzise Abstraktionen*, die als „Missing Link“ zwischen informeller Anforderungsspezifikation und technischer Realisierung dienen. Für sogenannte semiformale Ansätze gilt dies in z. T. erheblich eingeschränkter Weise. Im folgenden Abschnitt wird etwas eingehender auf eine Charakterisierung und Abgrenzung eingegangen.

Neben der genannten Verwendung in den frühen Entwicklungsphasen sind Formale Methoden ebensogut geeignet, abstrakte Spezifikationen *korrekt* in technische Realisierungen zu überführen. Typischerweise wird dabei in verschiedenen Schritten vorgegangen. Generell ist bei der Auswahl des formal modellierten Bereichs wie auch der (formalen) Modellierungstiefe die *Kritikalität* entscheidend. Wenn bei der weiteren technischen Realisierung kritische Designschritte, etwa in Form komplexer Algorithmen und Datenstrukturen, auftreten, ergibt sich aus der formalen Behandlung auf tieferen Stufen ein zusätzlicher Sicherheitsgewinn, auch wenn von ITSEC und CC (auf den Stufen E6 bzw. EAL7) nur die formale Behandlung des ersten Verfeinerungsschritts (Architekturentwurf) gefordert wird.

8.2 Semiformale Beschreibungstechniken

Sogenannte *semiformale* Beschreibungstechniken unterscheiden sich von rein informellen Vorgehensweisen durch die Verwendung exakt definierter *Sprachmittel*. Die in diesem Zusammenhang verbreiteten graphischen Darstellungen, etwa Zustandsübergangssysteme, lassen sich mindestens prinzipiell als *formale Sprache* auffassen.

Formale Sprachen schränken die Ausdrucksmöglichkeiten gegenüber der natürlichen Sprache ein und strukturieren sie. Durch die verbindliche Festlegung der Syntax wird der gedankliche Umgang mit Entwicklungsobjekten vorgeformt. Allein schon durch die Verwendung solcher Denkmuster kann bei sachgemäßer Verwendung ein beträchtlicher Zugewinn an Qualität entstehen.

Darüberhinaus lassen sich auf Grundlage einer fixierten Syntax gewisse Eigenschaften, wie etwa die (syntaktische) Konsistenz von Schnittstellendefinitionen, überprüfen. Eine entsprechende Werkzeugunterstützung unterstellt, können so gewisse Designfehler schon in den frühen Phasen erkannt (und behoben) werden.

Die Beschränkung auf *syntaktische Eigenschaften*, d. h. solche, die sich allein unter Bezug auf syntaktische Definitionen präzise formulieren lassen, ist allerdings gravierend. Sicherheitseigenschaften werden in der Regel eine Bezugnahme auf eine mathematisch definierte *Semantik* notwendig machen.

Die Beschränkung auf rein syntaktische Elemente, wie etwa *Signaturen* von Datentypen, Prozeduren, Funktionen oder Methoden, ist nur ein Spezialfall der Einschränkung der *Beschreibungstiefe*. In letzter Zeit sind Beschreibungstechniken populär geworden, die zwar über die reine Syntax hinausgehen und dabei auch semantisch präzisiert sind (oder zumindest ohne Schwierigkeiten präzisiert werden könnten), die aber dennoch auf ganz bestimmte *Aspekte* beschränkt sind. Ein Beispiel hierfür sind Objektdiagramme, wie sie bei objektorientierten Ansätzen verwendet werden. Die dabei betrachteten Eigenschaften von Relationen lassen sich, etwa im Rahmen sogenannter terminologischer Logiken, mathematisch präzise fassen. Problematisch und nicht gelöst ist es, diese Aspekte mit anderen „Sichten“, die etwa die verwendeten Datenstrukturen, den Informationsfluß und das Zeitverhalten betreffen, semantisch in Verbindung zu setzen.

8.3 Formale Methoden

Formale Methoden sind durch eine mathematisch präzisierte Syntax und *Semantik* gekennzeichnet. Da die syntaktischen Konzepte semantisch unterlegt sind, trägt allein schon die Strukturierung des Denkens viel stärker noch als im Fall der semiformalen Methoden zur Qualitätsverbesserung bei. Die Erfahrung zeigt, daß eine nicht vollständig formale Vorgehensweise es immer erlaubt, konzeptionelle Unklarheiten und Fehler, wie z. B. implizite Annahmen und mögliche Ausnahmefälle, unter Ausnutzung der nicht vollständig erklärten Semantik zu „verstecken“.

Bei der formalen Vorgehensweise kann jedoch über eine mathematisch disziplinierte Denkweise hinausgehend die Reflexion über Systeme vollständig objektiviert werden. Aufbauend auf die festgelegte Semantik gibt es (Rechen-) Verfahren, um die Gültigkeit von Aussagen über formal spezifizierte Systeme zu etablieren. Im Allgemeinen geschieht dies durch (maschinell unterstützte) mathematische Beweise, für spezielle Problemklassen aber auch durch vollständig automatisierte Entscheidungsverfahren (z. B. Model Checking). Die Möglichkeit, Systemspezifikationen formal reflektieren und so Eigenschaften wirklich *garantieren* zu können, ist der entscheidende Vorteil dieses Ansatzes. Abgesehen von

der Art der verwendeten mathematischen Methoden, entspricht dies der Situation in (heute) etablierten Ingenieurdisziplinen.

Alle gängigen Formalismen erlauben eine Modellierungstiefe, die es gestattet, relevante Sicherheitseigenschaften und die ihnen entsprechenden Funktionen abstrakt und gleichzeitig mathematisch präzise zu behandeln. Die axiomatische Methode erlaubt dabei einen hohen Abstraktheitsgrad durch die Verwendung mathematischer Strukturen, nichtkonstruktive Festlegungen (was, nicht wie), Indeterminismus und das kontrollierte Offenlassen von Einzelheiten.

Wie oben bereits erwähnt, ist es weiter möglich, den Verfeinerungsprozeß von der Anforderungsspezifikation bis zur Implementierung durch Modellierung entsprechender softwaretechnischer Lösungen formal zu unterstützen. Dies betrifft z. B. Algorithmen, Datenstrukturen und die technische Realisierung der Kommunikation zwischen verteilten und nebenläufigen Komponenten.

Auch wenn etwa beim Übergang von informellen Anforderungen zu formalen Sicherheitsmodellen und auch bei der anschließenden (nichtformalen) technischen Realisierung nach wie vor *zusätzliche* Irrtümer und Fehler auftreten können, so resultiert aus einer adäquaten Verwendung formaler Techniken ein deutlicher Zugewinn an Qualität, der mit anderen Maßnahmen nicht erreicht werden kann. Eine solche sachgemäße und nutzbringende (gemessen an der erreichbaren Qualitätsverbesserung) Anwendung formaler Entwicklungsmethoden zu fördern, ist Anliegen dieses Leitfadens.

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 formally specified security properties and security features and whether this description corresponds with the description of security functions in the functional specification.

A.2 Application notes

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. The part of the TSP model corresponding to “principles” (resp. “characteristics”) is denoted by “properties” (resp. “features”).

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

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

Those policies that can be modeled neither formally nor semiformally should be modeled informally. If none of the security policies of the TOE can be (semi-) formally modeled, 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;

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

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

It is recognized that not all policies (see work units for ADV_SPM.[123].2C) can be (semi-) formally modeled for all TOEs. Where formal security policy models are required but not possible, the policy must be provided in semiformal form. Where semiformal security policy models are required but not possible, the policy must be provided in an informal form.

For any security policy provided in an informal (resp. semiformal) form where semiformal (resp. formal) security policy models are required, the security policy model must contain a demonstration that these security policies cannot be semi-formally (resp. formally) modeled. If none of the TOE's security policies can be formally modeled, ADV_SPM.3 cannot be met. If none of the TOE's security policies can be (semi-) formally modeled, ADV_SPM.2 cannot be met.

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

If the entire security policy model is informal, this work unit is not applicable and is therefore considered to be satisfied.

Supporting narrative descriptions are necessary for those portions of the security policy model that are difficult to understand only from the semiformal or formal description (for example, to make clear the meaning of any formal notation).

ADV_SPM.[123].2C

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

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 (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.

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

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 security 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 security properties and security features describe the security posture of the TOE. It is likely that such a description would be contained within an evaluated and certified ST. In order to be considered a clear articulation, such a description

should define the notion of security for the TOE, identify the security attributes of the entities controlled by the TOE and identify the TOE actions which change those attributes. For example, if a policy attempts to address data integrity concerns, the security policy model would:

a) define the notion of integrity for that TOE;

b) identify the types of data for which the TOE would maintain integrity;

c) identify the entities that could modify that data;

d) identify the rules that potential modifiers must follow to modify data.

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

The demonstration or proof, as appropriate, shall show that the security features enforce the security properties. To determine the enforcement, the evaluator considers 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 security policy model to determine that it demonstrates or proves, as appropriate, its internal consistency.

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. Where this is not completely achievable, e. g. due to fundamental barriers of the specification language, the consistency demonstration should use semiformal (tool) support to the largest possible extent.

If the security policy model is formal, the proof of its internal consistency shall be formal. Where this is not completely achievable, e. g. due to fundamental barriers of the specification language, the consistency proof should use formal (tool) support to the largest possible extent.

For guidance on consistency analysis see Annex B.3.

ADV_SPM.[123].3C

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

In determining consistency, the evaluator verifies that the rationale shows that each security property resp. security feature description in the security policy 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 security policy 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 Annex B.3.

The evaluator ***shall examine*** the 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 security policy model and maps those security principles and security 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 security policy model.

ADV_SPM.[123].4C

The evaluator ***shall examine*** the functional specification correspondence demonstration of the 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

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

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

descriptions in the functional specification.

To demonstrate 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 security policy model identify the same attributes and security features of the security policy model and enforce the same security properties as the security policy 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 Annex B.3.

ADV_SPM.[23].5C

ADV_SPM-12 The evaluator ***shall check*** the functional specification correspondence demonstration of the security policy model to determine that it is semiformal.

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

Where both 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.

ADV_SPM.3.6C

ADV_SPM-13 The evaluator ***shall check*** the functional specification correspondence demonstration of the security policy model to determine that it is formal.

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

Anhang B

Korrespondenz zu CEM

Die aktuelle Fassung der *Common Evaluation Methodology for Information Technology Security Evaluation (CEM)* [3] beschreibt die Evaluationsaktivitäten für die Evaluationsstufen EAL1 bis EAL4. Zwar wird in keiner dieser Stufen ein formales Sicherheitsmodell gefordert, doch enthält die Stufe EAL4 die Komponente ADV_SPM.1, also ein informelles Sicherheitsmodell.

In [3, Sec. 8.6.7] ist die Subaktivität zur Evaluierung des informellen Sicherheitsmodells als Bestandteil von EAL4 beschrieben. Die work units der in Anhang A vorgeschlagenen Subaktivität korrespondieren eng mit den work units nach CEM. Die Korrespondenz ist in Tab. B.1 dargelegt.

Anhang A	[3, Chap. 8.6.7]
ADV_SPM-1	—
ADV_SPM-2	4:ADV_SPM.1-1
ADV_SPM-3	4:ADV_SPM.1-2
ADV_SPM-4	4:ADV_SPM.1-3
ADV_SPM-5	4:ADV_SPM.1-4
ADV_SPM-6	—
ADV_SPM-7	—
ADV_SPM-8	4:ADV_SPM.1-5
ADV_SPM-9	4:ADV_SPM.1-6
ADV_SPM-10	4:ADV_SPM.1-7
ADV_SPM-11	4:ADV_SPM.1-8
ADV_SPM-12	—
ADV_SPM-13	—

Tabelle B.1: Korrespondenz von Subaktivität ADV_SPM und [3, Chap. 8.6.7]

Literaturverzeichnis

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

[11] 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.

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

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

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

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

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

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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.

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

[29] 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.

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

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

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

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

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

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

[44] 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.

[45] 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.