COMMITTEE DRAFT **ISO/IEC CD 15408-3, revision**	Reference document: **SC 27 N18805**
Date: **2019-01-08**	Supersedes document: N18752

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 27 Information technology - Security techniques Secretariat: Germany (DIN)	Circulated to P- and O-members, and to technical committees and organizations in liaison for comments by: **2019-03-05** Please submit your comments via the online balloting application by the due date indicated.

ISO/IEC CD 15408-3, revision

Title: IT Security techniques – Evaluation criteria for IT security — Part 3: Security assurance components

Project: 1.27.16.03 (ISO/IEC 15408-3, revision)

Explanatory Report

Status	**SC 27 Decision**	**Reference documents**	
		Input	**Output**
For details regarding previous development stages refer to 2nd page of this explanatory report.			

ISO/IEC 15408-3 1st WD	54th WG 3 meeting, April 2017, Recommendations 5,10 11, 14 (N17041 = WG 3 N1413).	SoV (N17027).	Liaisons to:
			CCDB (WG 3 N1391);
			The Open Group (WG 3 N1394);
			ISO/TC 22/SC 32 (N17373);
			Text f. 1st WD (WG 3 N1437).

ISO/IEC 15408-3 2nd WD	55th WG 3 meeting, Oct. 2017, Recommendations 8, 10 (N17666 = WG 3 N1494).	SoCom (WG 3 N1467);
		Draft DoC (WG 3 N1501).
			Editor's report (WG 3 N1465);
			Liaisons to:
			CCDB (WG 3 N1455);
			ISO/TC 22/SC 32 (N18103);
			DoC (WG 3 N1462);
			Text f. 2nd WD (WG 3 N1469).

ISO/IEC 15408-3 1st CD	56th WG 3 meeting, April 2018, Recommendations 8, 10 (N18471 = WG 3 N1557) / 30th SC 27 Plenary, April 2018, Resolution 6 (N18710).	SoCom (WG 3 N1530);
		Late Com (WG 3 N1564).
			Liaison to:
			CCDB (WG 3 N1521);
			DoC (WG 3 N1527);
			Text f. 1st CD (N18752).

ISO/IEC 15408-3 2nd CD	57th WG 3 meeting / CRM, Sep / Oct 2018, Recommendations11, 14 (N18820 = WG 3 N1610) .	SoV (N18853).	Liaison to:
			CCDB (WG 3 N1619);
			DoC (N18802);
			Text f. 2nd CD (N18805).

2nd CD Consideration

In accordance with Recommendation 14 (see SC 27 N18820 = WG 3 N1610) of the 57th SC 27/WG 3 meeting held in Gjøvik, Norway, 2018-09-30/10-04 the hereby attached document is being circulated for a 8-week 2nd CD letter ballot closing by

2019-03-05

Medium: http://isotc.iso.org/livelink/livelink/open/jtc1sc27

No. of pages: 2 + 220

Explanatory Report			
Status	**SC 27 Decision**	**Reference documents**	
		Input	**Output**
Study Period IT security testing, evaluation and assurance standards and techniques	51st WG 3 meeting, Oct. 2015, Recommendations 5, 6 (N15594 = WG 3 N1251).		Terms of Reference (WG 5 N1258); 1st /2nd call f. contr. (WG 3 N1259 /1317)..

	52nd WG 3 meeting, April 2016, Recommendation 5, 7 (N16026 = WG 3 N1296).	Expert contr. (WG 3 N1299, 1301).	3rd call f. contr. (WG 3 N1377);
			Rapporteur's report (WG 3 N1320);
			Liaison to:
			CCDB (WG 3 N1266).

ISO/IEC NP 15408-3 (revision) Evaluation criteria for IT security -- Part 3 NWIP	53rd WG 3 meeting, Oct. 2016, Recommendations 6, 15 (N16800 = WG 5 N600).	Expert contr. (WG 3 N1368, N1371, N1373).	SP report (WG 3 N1363);
			Call f. editor (WG 3 N1387 = N16886);
			Liaisons to:
			CCDB (WG 3 N1330);
			The Open Group (WG 3 N1332);
			Text f. NWIP (N16965 [replaces N16883]).

ISO/IEC 15408-3:XXXX(E)

ISO/IEC JTC 1/SC 27/WG 3 N1654

Secretariat: DIN

IT security techniques — Evaluation criteria for IT security — Part 3: Security assurance components

CD stage

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 15408-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

This fourth edition cancels and replaces the third edition (ISO/IEC 15408-3:2008), which has been technically revised.

ISO/IEC 15408 consists of the following parts, under the general title *IT security techniques -- Evaluation criteria for IT security*:

— *Part 1: Introduction and general model*

— *Part 2: Security functional components*

— *Part 3: Security assurance components*

— *Part 4: Framework for the specification of evaluation methods and activities*

— *Part 5: Pre-defined packages of security requirements*

Editors note: The following para will be updated when the new content of this part ist stable.

This corrected version of ISO/IEC 15408-3:XXXX incorporates miscellaneous editorial corrections mainly related to EAL4 and EAL6 assurance components, ADV_FSP, ADV_TDS, ATE_DPT.2, ATE_IND, and ALC.

Legal Notice

Australia/New Zealand:	The Defence Signals Directorate and the Government Communications Security Bureau respectively;
Canada:	Communications Security Establishment;
France:	Agence nationale de la sécurité des systèmes d'information (ANSSI);
Germany:	Bundesamt für Sicherheit in der Informationstechnik;
Japan:	Information Technology Promotion Agency;
Netherlands:	Netherlands National Communications Security Agency;
Spain:	Ministerio de Administraciones Públicas and Centro Criptológico Nacional;
United Kingdom:	Communications-Electronic Security Group;
United States:	The National Security Agency and the National Institute of Standards and Technology.

Introduction

Security assurance components, as defined in this document, are the basis for the security assurance requirements expressed in a Protection Profile (PP) or a Security Target (ST).

These requirements establish a standard way of expressing the assurance requirements for TOEs. This document catalogues the set of assurance components, families and classes. It also defines evaluation criteria for PPs, PP-Modules, Packages and STs..

The audience for this document includes consumers, developers, and evaluators of secure IT products. ISO/IEC 15408-1:XXXX, Clause 5 provides additional information on the target audience of ISO/IEC 15408, and on the use of ISO/IEC 15408 by the groups that comprise the target audience. These groups may use this document as follows:

a) Consumers, who use this document when selecting components to express assurance requirements to satisfy the security objectives expressed in a PP or ST, determining required levels of security assurance of the TOE.

b) Developers, who respond to actual or perceived consumer security requirements in constructing a TOE, reference this document when interpreting statements of assurance requirements and determining assurance approaches of TOEs.

c) Evaluators, who use the assurance requirements defined in this document as a mandatory statement of evaluation criteria when determining the assurance of TOEs and when evaluating PPs and STs.

Information technology Security techniques — Evaluation criteria for IT security —

Part 3: Security assurance components

1 Scope

This document defines the assurance requirements of ISO/IEC 15408. It includes the individual assurance components from which the evaluation assurance levels and other packages contained in part 5 are composed, and the criteria for evaluation of Protection Profiles (PPs) and Security Targets (STs).

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 15408-1, IT security techniques -- Evaluation criteria for IT security — *Part 1: Introduction and general model*

ISO/IEC 15408-2, IT security techniques -- Evaluation criteria for IT security — *Part 2: Security functional components*

ISO/IEC 15408-5, IT security techniques -- Evaluation criteria for IT security — *Part 5: Pre-defined packages of security requirements*

3 Terms and definitions, symbols and abbreviated terms

For the purposes of this document, the terms, definitions, symbols and abbreviated terms given in ISO/IEC 15408-1 apply.

4 Overview

4.1 Organisation of this document

Clause 5 describes the paradigm used in the security assurance requirements of this document.

Clause 6 describes the presentation structure of the assurance classes, families, components, evaluation assurance levels along with their relationships, and the structure of the composed assurance packages. It also characterises the assurance classes and families found in Clauses 7 through 15.

Clauses 7 through 15 provide the detailed definitions of this document assurance classes.

Annex A provides further explanations and examples of the concepts behind the Development class.

Annex B provides an explanation of the concepts behind composed TOE evaluations and the Composition class.

Annex C provides a summary of the dependencies between the assurance components.

Annex A provides a cross reference between PPs and the families and components of the APE class.

5 Assurance paradigm

5.1 Introduction

The purpose of this clause is to document the philosophy that underpins ISO/IEC 15408 approach to assurance. An understanding of this clause will permit the reader to understand the rationale behind this document assurance requirements.

5.2 ISO/IEC 15408 philosophy

ISO/IEC 15408 philosophy is that the threats to security and organisational security policy commitments should be clearly articulated and the proposed security controls be demonstrably sufficient for their intended purpose.

Furthermore, measures should be adopted that reduce the likelihood of vulnerabilities, the ability to exercise (i.e. intentionally exploit or unintentionally trigger) a vulnerability, and the extent of the damage that could occur from a vulnerability being exercised. Additionally, measures should be adopted that facilitate the subsequent identification of vulnerabilities and the elimination, mitigation, and/or notification that a vulnerability has been exploited or triggered.

5.3 Assurance approach

5.3.1 Introduction

ISO/IEC 15408 philosophy is to provide assurance based upon an evaluation (active investigation) of the IT product that is to be trusted. Evaluation has been the traditional means of providing assurance and is the basis for prior evaluation criteria documents. In aligning the existing approaches, ISO/IEC 15408 adopts the same philosophy. ISO/IEC 15408 proposes measuring the validity of the documentation and of the resulting IT product by expert evaluators with increasing emphasis on scope, depth, and rigour.

ISO/IEC 15408 does not exclude, nor does it comment upon, the relative merits of other means of gaining assurance. Research continues with respect to alternative ways of gaining assurance. As mature alternative approaches emerge from these research activities, they will be considered for inclusion in ISO/IEC 15408, which is so structured as to allow their future introduction.

5.3.2 Significance of vulnerabilities

It is assumed that there are threat agents that will actively seek to exploit opportunities to violate security policies both for illicit gains and for well-intentioned, but nonetheless insecure actions. Threat agents may also accidentally trigger security vulnerabilities, causing harm to the organisation. Due to the need to process sensitive information and the lack of availability of sufficiently trusted products, there is significant risk due to failures of IT. It is, therefore, likely that IT security breaches could lead to significant loss.

IT security breaches arise through the intentional exploitation or the unintentional triggering of vulnerabilities in the application of IT within business concerns.

Steps should be taken to prevent vulnerabilities arising in IT products. To the extent feasible, vulnerabilities should be:

a) eliminated - that is, active steps should be taken to expose, and remove or neutralise, all exercisable vulnerabilities;

b) minimised - that is, active steps should be taken to reduce, to an acceptable residual level, the potential impact of any exercise of a vulnerability;

c) monitored - that is, active steps should be taken to ensure that any attempt to exercise a residual vulnerability will be detected so that steps can be taken to limit the damage.

5.3.3 Cause of vulnerabilities

Vulnerabilities can arise through failures in:

a) requirements -- that is, an IT product may possess all the functions and features required of it and still contain vulnerabilities that render it unsuitable or ineffective with respect to security;

b) design – that is, an IT product has been poorly designed. Building a secure product, system, or application requires not only the implementation of functional requirements but also an architecture that allows for the effective enforcement of specific security properties the product, system, or application is supposed to enforce. The ability to withstand attacks the product, system, or application may be face in its intended operational environment is highly dependent on an architecture that prohibits those attacks or – if they cannot be prohibited – allows for detection of such attacks and/or limitation of the damage such an attack can cause;

c) development -- that is, an IT product does not meet its specifications and/or vulnerabilities have been introduced as a result of poor development standards or incorrect design choices;

d) delivery, installation and configuration – that is, an IT product has vulnerabilities introduced during the delivery, installation and configuration of the product;

e) operation -- that is, an IT product has been constructed correctly to a correct specification but vulnerabilities have been introduced as a result of inadequate controls upon the operation.

f) maintenance – that is, an IT product is maintained in such a way that new vulnerabilities are introduced.

5.3.4 ISO/IEC 15408 assurance

Assurance can be derived from reference to sources such as unsubstantiated assertions, prior relevant experience, or specific experience. However, ISO/IEC 15408 provides assurance through active investigation. Active investigation is an evaluation of the IT product in order to determine its security properties.

5.3.5 Assurance through evaluation

Evaluation has been the traditional means of gaining assurance, and is the basis of ISO/IEC 15408 approach. Evaluation techniques can include, but are not limited to:

a) analysis and checking of process(es) and procedure(s);

b) checking that process(es) and procedure(s) are being applied;

c) analysis of the correspondence between TOE design representations;

d) analysis of the TOE design representation against the requirements;

e) verification of proofs;

f) analysis of guidance documents;

g) analysis of functional tests developed and the results provided;

h) independent functional testing;

i) analysis for vulnerabilities (including flaw hypothesis);

j) penetration testing;

k) analysis of the delivery process.

5.4 ISO/IEC 15408 evaluation assurance scale

ISO/IEC 15408 philosophy asserts that greater assurance results from the application of greater evaluation effort, and that the goal is to apply the minimum effort required to provide the necessary assurance. The increasing level of effort is based upon:

a) scope -- that is, the effort is greater because a larger portion of the IT product is included;

b) depth -- that is, the effort is greater because it is deployed to a finer level of design and implementation detail;

c) rigour -- that is, the effort is greater because it is applied in a more structured, formal manner.

6 Security assurance components

6.1 Security assurance classes, families and components structure

The following subclauses describe the constructs used in representing the assurance classes, families, and components.

Figure 1 illustrates the security assurance requirements (SARs) defined in this document. Note that the most abstract collection of SARs is referred to as a class. Each class contains assurance families, which then contain assurance components, which in turn contain assurance elements. Classes and families are used to provide a taxonomy for classifying SARs, while components are used to specify SARs in a PP/ST.

6.1.1 Assurance class structure

Figure 1 illustrates the assurance class structure.

6.1.1.1 Class name

Each assurance class is assigned a unique name. The name indicates the topics covered by the assurance class.

A unique short form of the assurance class name is also provided. This is the primary means for referencing the assurance class. The convention adopted is an "A" followed by two letters related to the class name.

6.1.1.2 Class introduction

Each assurance class has an introductory subclause that describes the composition of the class and contains supportive text covering the intent of the class.

6.1.1.3 Assurance families

Each assurance class contains at least one assurance family. The structure of the assurance families is described in the following subclause.

Figure 1 illustrates the assurance family structure.

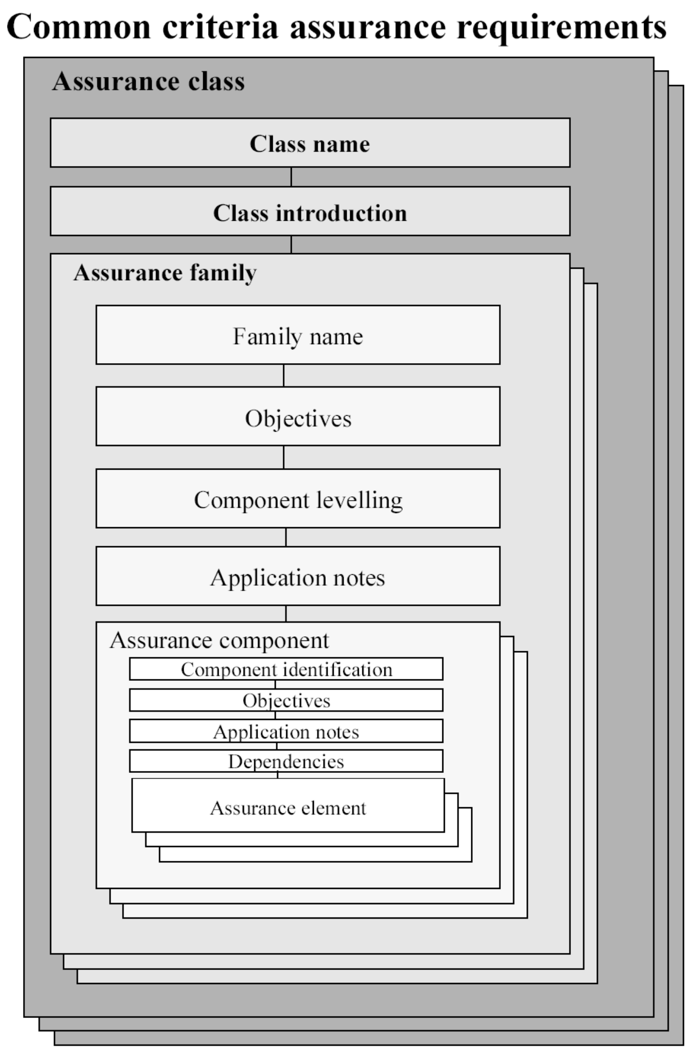

Figure 1 — Assurance class/family/component/element hierarchy

6.1.2 Assurance family structure

6.1.2.1 Family name

Every assurance family is assigned a unique name. The name provides descriptive information about the topics covered by the assurance family. Each assurance family is placed within the assurance class that contains other families with the same intent.

A unique short form of the assurance family name is also provided. This is the primary means used to reference the assurance family. The convention adopted is that the short form of the class name is used, followed by an underscore, and then three letters related to the family name.

6.1.2.2 Objectives

The objectives subclause of the assurance family presents the intent of the assurance family.

This subclause describes the objectives, particularly those related to ISO/IEC 15408 assurance paradigm, that the family is intended to address. The description for the assurance family is kept at a general level. Any specific details required for objectives are incorporated in the particular assurance component.

6.1.2.3 Component levelling

Each assurance family contains one or more assurance components. This subclause of the assurance family describes the components available and explains the distinctions between them. Its main purpose is to differentiate between the assurance components once it has been determined that the assurance family is a necessary or useful part of the SARs for a PP/ST.

Assurance families containing more than one component are levelled and rationale is provided as to how the components are levelled. This rationale is in terms of scope, depth, and/or rigour.

6.1.2.4 Application notes

The application notes subclause of the assurance family, if present, contains additional information for the assurance family. This information should be of particular interest to users of the assurance family (e.g. PP and ST authors, designers of TOEs, evaluators). The presentation is informal and covers, for example, warnings about limitations of use and areas where specific attention may be required.

6.1.2.5 Assurance components

Each assurance family has at least one assurance component. The structure of the assurance components is provided in the following subclause.

6.1.3 Assurance component structure

Figure 2 illustrates the assurance component structure.

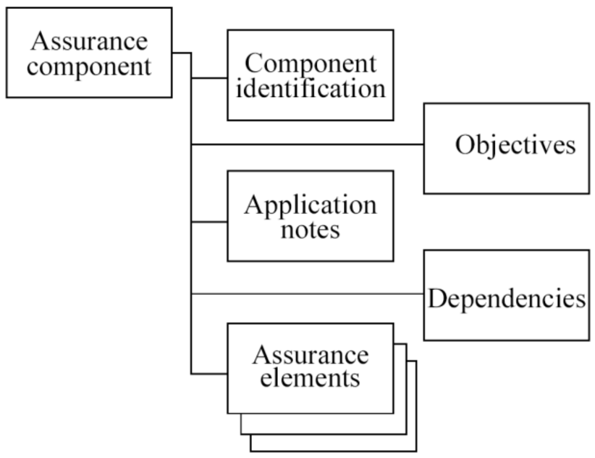

Figure 2 — Assurance component structure

The relationship between components within a family is highlighted using a bolding convention. Those parts of the requirements that are new, enhanced or modified beyond the requirements of the previous component within a hierarchy are bolded.

6.1.3.1 Component identification

The component identification subclause provides descriptive information necessary to identify, categorise, register, and reference a component.

Every assurance component is assigned a unique name. The name provides descriptive information about the topics covered by the assurance component. Each assurance component is placed within the assurance family that shares its security objective.

A unique short form of the assurance component name is also provided. This is the primary means used to reference the assurance component. The convention used is that the short form of the family name is used, followed by a period, and then a numeric character. The numeric characters for the components within each family are assigned sequentially, starting from 1.

6.1.3.2 Objectives

The objectives subclause of the assurance component, if present, contains specific objectives for the particular assurance component. For those assurance components that have this subclause, it presents the specific intent of the component and a more detailed explanation of the objectives.

6.1.3.3 Application notes

The application notes subclause of an assurance component, if present, contains additional information to facilitate the use of the component.

6.1.3.4 Dependencies

Dependencies among assurance components arise when a component is not self-sufficient, and relies upon the presence of another component.

Each assurance component provides a complete list of dependencies to other assurance components. Some components may list “No dependencies”, to indicate that no dependencies have been identified. The components depended upon may have dependencies on other components.

The dependency list identifies the minimum set of assurance components which are relied upon. Components which are hierarchical to a component in the dependency list may also be used to satisfy the dependency.

In specific situations the indicated dependencies might not be applicable. The PP/ST author, by providing rationale for why a given dependency is not applicable, may elect not to satisfy that dependency.

6.1.3.5 Assurance elements

A set of assurance elements is provided for each assurance component. An assurance element is a security requirement which, if further divided, would not yield a meaningful evaluation result. It is the smallest security requirement recognised in ISO/IEC 15408.

Each assurance element is identified as belonging to one of the three sets of assurance elements:

a) Developer action elements: the activities that shall be performed by the developer. This set of actions is further qualified by evidential material referenced in the following set of elements. Requirements for developer actions are identified by appending the letter “D” to the element number.

b) Content and presentation of evidence elements: the evidence required, what the evidence shall demonstrate, and what information the evidence shall convey. Requirements for content and presentation of evidence are identified by appending the letter “C” to the element number.

c) Evaluator action elements: the activities that shall be performed by the evaluator. This set of actions explicitly includes confirmation that the requirements prescribed in the content and presentation of evidence elements have been met. It also includes explicit actions and analysis that shall be performed in addition to that already performed by the developer. Implicit evaluator actions are also to be performed as a result of developer action elements which are not covered by content and presentation of evidence requirements. Requirements for evaluator actions are identified by appending the letter “E” to the element number.

The developer actions and content and presentation of evidence define the assurance requirements that are used to represent a developer's responsibilities in demonstrating assurance in the TOE meeting the SFRs of a PP or ST.

The evaluator actions define the evaluator's responsibilities in the two aspects of evaluation. The first aspect is validation of the PP/ST, in accordance with the classes APE and ASE in Clauses APE: Protection Profile evaluation and ASE: Security Target evaluation. The second aspect is verification of the TOE's conformance with its SFRs and SARs. By demonstrating that the PP/ST is valid and that the requirements are met by the TOE, the evaluator can provide a basis for confidence that the TOE in its operational environment solves the defined security problem.

The developer action elements, content and presentation of evidence elements, and explicit evaluator action elements, identify the evaluator effort that shall be expended in verifying the security claims made in the ST of the TOE.

6.1.4 Assurance elements

Each element represents a requirement to be met. These statements of requirements are intended to be clear, concise, and unambiguous. Therefore, there are no compound sentences: each separable requirement is stated as an individual element.

6.1.5 Component taxonomy

This document contains classes of families and components that are grouped on the basis of related assurance. At the start of each class is a diagram that indicates the families in the class and the components in each family.

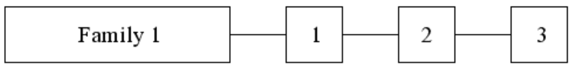

Figure 3 — Sample class decomposition diagram

In Figure 3, above, the class as shown contains a single family. The family contains three components that are linearly hierarchical (i.e. component 2 requires more than component 1, in terms of specific actions, specific evidence, or rigour of the actions or evidence). The assurance families in this document are all linearly hierarchical, although linearity is not a mandatory criterion for assurance families that may be added in the future.

7 Class APE: Protection Profile evaluation

7.1 Introduction

Evaluating a PP is required to demonstrate that the PP is sound and internally consistent, and, if the PP is based on one or more other PPs or on packages, that the PP is a correct instantiation of these PPs and packages. These properties are necessary for the PP to be suitable for use as the basis for writing an ST or another PP.

This clause should be used in conjunction with Annexes A, B and C in ISO/IEC 15408-1:2009, as these annexes clarify the concepts here and provide many examples.

Figure 4 shows the families within this class, and the hierarchy of components within the families.

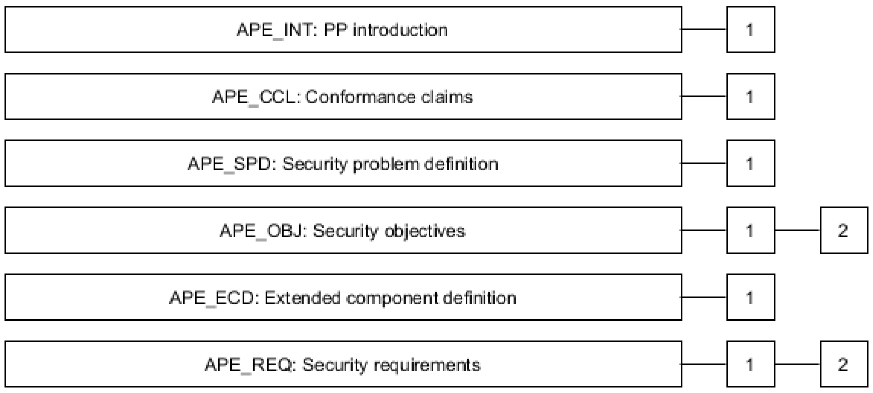
APE_INT: PP introduction
1
APE_CCL: Conformance claims
1
APE_SPD: Security problem definition
1
APE_OBJ: Security objectives
1
2
APE_ECD: Extended component definition
1
APE_REQ: Security requirements
1
2

Figure 4 — APE: Protection Profile evaluation class decomposition

7.2 PP introduction (APE_INT)

7.2.1 Objectives

The objective of this family is to describe the TOE in a narrative way.

Evaluation of the PP introduction is required to demonstrate that the PP is correctly identified, and that the PP reference and TOE overview are consistent with each other.

7.2.2 APE_INT.1 PP introduction

Dependencies: No dependencies.

7.2.2.1 Developer action elements

7.2.2.1.1 APE_INT.1.1D

The developer shall provide a PP introduction.

7.2.2.2 Content and presentation elements

7.2.2.2.1 APE_INT.1.1C

The PP introduction shall contain a PP reference and a TOE overview.

7.2.2.2.2 APE_INT.1.2C

The PP reference shall uniquely identify the PP.

7.2.2.2.3 APE_INT.1.3C

The TOE overview shall summarise the usage and major security features of the TOE.

7.2.2.2.4 APE_INT.1.4C

The TOE overview shall identify the TOE type.

7.2.2.2.5 APE_INT.1.5C

The TOE overview shall identify any non-TOE hardware/software/firmware available to the TOE.

7.2.2.3 Evaluator action elements

7.2.2.3.1 APE_INT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.3 Conformance claims (APE_CCL)

7.3.1 Objectives

The objective of this family is to determine the validity of the conformance claim. In addition, this family specifies how STs and other PPs are to claim conformance with the PP.

7.3.2 APE_CCL.1 Conformance claims

Dependencies: APE_INT.1 PP introduction

APE_ECD.1 Extended components definition

APE_REQ.1 Stated security requirements

7.3.2.1 Developer action elements

7.3.2.1.1 APE_CCL.1.1D

The developer shall provide a conformance claim.

7.3.2.1.2 APE_CCL.1.2D

The developer shall provide a conformance claim rationale.

7.3.2.1.3 APE_CCL.1.3D

The developer shall provide a conformance statement.

7.3.2.2 Content and presentation elements

7.3.2.2.1 APE_CCL.1.1C

The conformance claim shall contain an ISO/IEC 15408 conformance claim that identifies the ISO/IEC 15408-1 edition to which the PP claims conformance.

7.3.2.2.2 APE_CCL.1.2C

ISO/IEC 15408 conformance claim shall describe the conformance of the PP to ISO/IEC 15408-2 as either ISO/IEC 15408-2 conformant or ISO/IEC 15408-2 extended.

7.3.2.2.3 APE_CCL.1.3C

The ISO/IEC 15408 conformance claim shall describe the conformance of the PP to this document as either "ISO/IEC 15408-3 conformant" or ISO/IEC 15408-3 extended."

7.3.2.2.4 APE_CCL.1.4C

ISO/IEC 15408 conformance claim shall be consistent with the extended components definition.

7.3.2.2.5 APE_CCL.1.5C

The conformance claim shall identify all PPs and security requirement packages to which the PP claims conformance.

7.3.2.2.6 APE_CCL.1.6C

The conformance claim shall describe any conformance of the PP to a package as either package-conformant or package-augmented.

7.3.2.2.7 APE_CCL.1.7C

The conformance claim rationale shall demonstrate that the TOE type is consistent with the TOE type in the PPs for which conformance is being claimed.

7.3.2.2.8 APE_CCL.1.8C

The conformance claim rationale shall demonstrate that the statement of the security problem definition is consistent with the statement of the security problem definition in the PPs for which conformance is being claimed.

7.3.2.2.9 APE_CCL.1.9C

The conformance claim rationale shall demonstrate that the statement of security objectives is consistent with the statement of security objectives in the PPs for which conformance is being claimed.

7.3.2.2.10 APE_CCL.1.10C

The conformance claim rationale shall demonstrate that the statement of security requirements is consistent with the statement of security requirements in the PPs for which conformance is being claimed.

7.3.2.2.11 APE_CCL.1.11C

The conformance statement shall describe the conformance required of any PPs/STs to the PP as exact-PP, strict-PP,or demonstrable-PP conformance.

7.3.2.2.12 APE_CCL.1.12C

The conformance statement shall identify the set of PPs (if any) to which, in combination with the PP under evaluation, exact conformance is allowed to be claimed.

7.3.2.2.13 APE_CCL.1.13C

The conformance statement shall identify the set of PP-modules (if any) that are allowed to be used with the PP under evaluation in a PP-Configuration.

7.3.2.2.14 APE_CCL.1.14C

The conformance statement shall identify the set of derived Evaluation Methods and Evaluation Activities (if any) that shall be used with the PP under evaluation. This list shall contain:

- **any Evaluation Methods and Evaluation Activities that are specified for the PP under evaluation**
- **any Evaluation Methods and Evaluation Activities specified in conformance statements of PPs to which conformance is being claimed by the PP under evaluation**
- **any Evaluation Methods and Evaluation Activities specified in Application Notes of packages to which conformance is being claimed by the PP under evaluation.**

7.3.2.3 Evaluator action elements

7.3.2.3.1 APE_CCL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.4 Security problem definition (APE_SPD)

7.4.1 Objectives

This part of the PP defines the security problem to be addressed by the TOE and the operational environment of the TOE.

Evaluation of the security problem definition is required to demonstrate that the security problem intended to be addressed by the TOE and its operational environment, is clearly defined.

7.4.2 APE_SPD.1 Security problem definition

Dependencies: No dependencies.

7.4.2.1 Developer action elements

7.4.2.1.1 APE_SPD.1.1D

The developer shall provide a security problem definition.

7.4.2.2 Content and presentation elements

7.4.2.2.1 APE_SPD.1.1C

The security problem definition shall describe the threats.

7.4.2.2.2 APE_SPD.1.2C

All threats shall be described in terms of a threat agent, an asset, and an adverse action.

7.4.2.2.3 APE_SPD.1.3C

The security problem definition shall describe the OSPs.

7.4.2.2.4 APE_SPD.1.4C

The security problem definition shall describe the assumptions about the operational environment of the TOE.

7.4.2.3 Evaluator action elements

7.4.2.3.1 APE_SPD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.5 Security objectives (APE_OBJ)

7.5.1 Objectives

The security objectives are a concise statement of the intended response to the security problem defined through the Security problem definition (APE_SPD) family.

Evaluation of the security objectives is required to demonstrate that the security objectives adequately and completely address the security problem definition and that the division of this problem between the TOE and its operational environment is clearly defined.

7.5.2 Component levelling

The components in this family are levelled on whether they prescribe only security objectives for the operational environment, or also security objectives for the TOE.

7.5.3 APE_OBJ.1 Security objectives for the operational environment

Dependencies: No dependencies.

7.5.3.1 Developer action elements

7.5.3.1.1 APE_OBJ.1.1D

The developer shall provide a statement of security objectives.

7.5.3.2 Content and presentation elements

7.5.3.2.1 APE_OBJ.1.1C

The statement of security objectives shall describe the security objectives for the operational environment.

7.5.3.3 Evaluator action elements

7.5.3.3.1 APE_OBJ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.5.4 APE_OBJ.2 Security objectives

Dependencies: APE_SPD.1 Security problem definition

7.5.4.1 Developer action elements

7.5.4.1.1 APE_OBJ.2.1D

The developer shall provide a statement of security objectives.

7.5.4.1.2 APE_OBJ.2.2D

The developer shall provide a security objectives rationale.

7.5.4.2 Content and presentation elements

7.5.4.2.1 APE_OBJ.2.1C

The statement of security objectives shall describe the security objectives for the **TOE and the security objectives for the** operational environment.

7.5.4.2.2 APE_OBJ.2.2C

The security objectives rationale shall trace each security objective for the TOE back to threats countered by that security objective and OSPs enforced by that security objective.

7.5.4.2.3 APE_OBJ.2.3C

The security objectives rationale shall trace each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

7.5.4.2.4 APE_OBJ.2.4C

The security objectives rationale shall demonstrate that the security objectives counter all threats.

7.5.4.2.5 APE_OBJ.2.5C

The security objectives rationale shall demonstrate that the security objectives enforce all OSPs.

7.5.4.2.6 APE_OBJ.2.6C

The security objectives rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

7.5.4.3 Evaluator action elements

7.5.4.3.1 APE_OBJ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.6 Extended components definition (APE_ECD)

7.6.1 Objectives

Extended security requirements are requirements that are not based on components from ISO/IEC 15408-2 or this document, but are based on extended components: components defined by the PP author.

Evaluation of the definition of extended components is necessary to determine that they are clear and unambiguous, and that they are necessary, i.e. they may not be clearly expressed using existing ISO/IEC 15408-2 or this document components.

7.6.2 APE_ECD.1 Extended components definition

Dependencies: No dependencies.

7.6.2.1 Developer action elements

7.6.2.1.1 APE_ECD.1.1D

The developer shall provide a statement of security requirements.

7.6.2.1.2 APE_ECD.1.2D

The developer shall provide an extended components definition.

7.6.2.2 Content and presentation elements

7.6.2.2.1 APE_ECD.1.1C

The statement of security requirements shall identify all extended security requirements.

7.6.2.2.2 APE_ECD.1.2C

The extended components definition shall define an extended component for each extended security requirement.

7.6.2.2.3 APE_ECD.1.3C

The extended components definition shall describe how each extended component is related to the existing ISO/IEC 15408 components, families, and classes.

7.6.2.2.4 APE_ECD.1.4C

The extended components definition shall use the existing ISO/IEC 15408 components, families, classes, and methodology as a model for presentation.

7.6.2.2.5 APE_ECD.1.5C

The extended components shall consist of measurable and objective elements such that conformance or nonconformance to these elements can be demonstrated.

7.6.2.3 Evaluator action elements

7.6.2.3.1 APE_ECD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.6.2.3.2 APE_ECD.1.2E

The evaluator shall confirm that no extended component may be clearly expressed using existing components.

7.7 Security requirements (APE_REQ)

7.7.1 Objectives

The SFRs form a clear, unambiguous and well-defined description of the expected security behaviour of the TOE. The SARs form a clear, unambiguous and well-defined description of the expected activities that will be undertaken to gain assurance in the TOE.

Evaluation of the security requirements is required to ensure that they are clear, unambiguous and well-defined.

7.7.2 Component levelling

The components in this family are levelled on whether they are stated as is, or whether the SFRs are derived from security objectives for the TOE.

7.7.3 APE_REQ.1 Stated security requirements

Dependencies: APE_ECD.1 Extended components definition

7.7.3.1 Developer action elements

7.7.3.1.1 APE_REQ.1.1D

The developer shall provide a statement of security requirements.

7.7.3.1.2 APE_REQ.1.2D

The developer shall provide a security requirements rationale.

7.7.3.2 Content and presentation elements

7.7.3.2.1 APE_REQ.1.1C

The statement of security requirements shall describe the SFRs and the SARs.

7.7.3.2.2 APE_REQ.1.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

7.7.3.2.3 APE_REQ.1.3C

The statement of security requirements shall include a natural language description, part of which describes how the SFRs combine together to provide security functionality in terms of the architecture that is observable to Administrators and other users, or in terms of internal features or properties.

7.7.3.2.4 APE_REQ.1.4C

The statement of security requirements shall identify all operations on the security requirements.

7.7.3.2.5 APE_REQ.1.5C

All operations shall be performed correctly.

7.7.3.2.6 APE_REQ.1.6C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

7.7.3.2.7 APE_REQ.1.7C

The security requirements rationale shall trace each SFR back to the threats countered by that SFR and the OSPs enforced by that SFR.

7.7.3.2.8 APE_REQ.1.8C

The security requirements rationale shall trace each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

7.7.3.2.9 APE_REQ.1.9C

The security requirements rationale shall demonstrate that the SFRs (in conjunction with the security objectives for the environment) counter all threats for the TOE.

7.7.3.2.10 APE_REQ.1.10C

The security requirements rationale shall demonstrate that the SFRs (in conjunction with the security objectives for the environment) enforce all OSPs for the TOE.

7.7.3.2.11 APE_REQ.1.11C

The security requirements rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

7.7.3.2.12 APE_REQ.1.12C

The statement of security requirements shall be internally consistent.

7.7.3.3 Evaluator action elements

7.7.3.3.1 APE_REQ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

7.7.4 APE_REQ.2 Derived security requirements

Dependencies: APE_OBJ.2 Security objectives

APE_ECD.1 Extended components definition

7.7.4.1 Developer action elements

7.7.4.1.1 APE_REQ.2.1D

The developer shall provide a statement of security requirements.

7.7.4.1.2 APE_REQ.2.2D

The developer shall provide a security requirements rationale.

7.7.4.2 Content and presentation elements

7.7.4.2.1 APE_REQ.2.1C

The statement of security requirements shall describe the SFRs and the SARs.

7.7.4.2.2 APE_REQ.2.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

7.7.4.2.3 APE_REQ.2.3C

Editor's Note:

To be discussed whether the "natural language" should remain or not.

The statement of security requirements shall include a natural language description, part of which describes how the SFRs combine together to provide security functionality in terms of the architecture that is observable to Administrators and other users, or in terms of internal features or properties.

7.7.4.2.4 APE_REQ.2.4C

The statement of security requirements shall identify all operations on the security requirements.

7.7.4.2.5 APE_REQ.2.5C

All operations shall be performed correctly.

7.7.4.2.6 APE_REQ.2.6C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

7.7.4.2.7 APE_REQ.2.7C

The security requirements rationale shall trace each SFR back to the security objectives for the TOE and OSPs enforced by that SFR.

7.7.4.2.8 APE_REQ.2.8C

The security requirements rationale shall demonstrate that the SFRs meet all security objectives for the TOE.

7.7.4.2.9 APE_REQ.2.9C

The security requirements rationale shall demonstrate that the SFRs enforce all OSPs.

7.7.4.2.10 APE_REQ.2.10C

The security requirements rationale shall explain why the SARs were chosen.

7.7.4.2.11 APE_REQ.2.11C

The statement of security requirements shall be internally consistent.

7.7.4.3 Evaluator action elements

7.7.4.3.1 APE_REQ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8 Class ACE: Protection Profile Configuration evaluation

8.1 Introduction

Evaluating a PP-Configuration is required to demonstrate that the PP-Configuration is sound and consistent. These properties are necessary for the PP-Configuration to be suitable for use as the basis for writing an ST.

The class ACE is defined for the evaluation of a PP-Configuration composed of PPs and PP-Modules[1]. The evaluation of PPs is addressed in Class APE. The present class ACE defines the requirements for

- Evaluating the PP-Modules under the assumption that their base PPs/PP-Modules is internally consistent.
- Evaluating the consistency of the combination of all the PPs and PP-Modules that belong to the PP-Configuration.

[1] Two PP-Modules may define each other in their basis, which means that a PP-Configuration that contains one of them also contains the other.

The evaluator shall decide the order in which the unevaluated components of a PP-Configuration (PPs and PP-Modules) are evaluated.

This Clause should be used in conjunction with Annexes B and D in ISO/IEC 1540-1, as these Annexes clarify the concepts and provide examples.

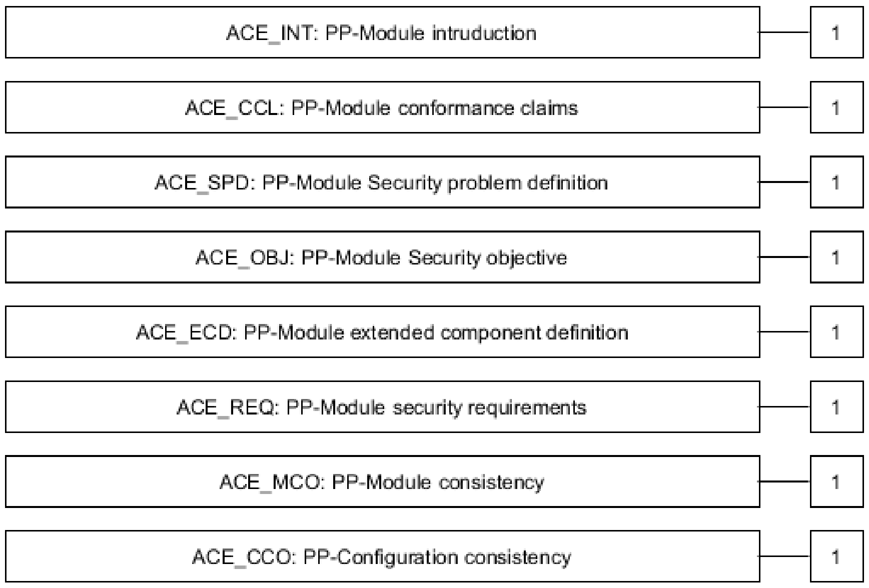
ACE_INT: PP-Module intruduction
1
ACE_CCL: PP-Module conformance claims
1
ACE_SPD: PP-Module Security problem definition
1
ACE_OBJ: PP-Module Security objective
1
ACE_ECD: PP-Module extended component definition
1
ACE_REQ: PP-Module security requirements
1
ACE_MCO: PP-Module consistency
1
ACE_CCO: PP-Configuration consistency
1

Figure 5: ACE: Protection Profile Configuration evaluation class decomposition

8.2 PP-Module introduction (ACE_INT)

8.2.1 Objectives

The objective of this family is to describe the TOE in a narrative way.

The evaluation of the PP-Module introduction is required to demonstrate that the PP-Module is correctly identified, and that the PP-Module reference and TOE overview are consistent with each other.

8.2.2 ACE_INT.1 PP-Module introduction

Dependencies: No dependencies.

8.2.2.1 Developer action elements

8.2.2.1.1 ACE_INT.1.1D

The developer shall provide a PP-Module introduction.

8.2.2.2 Content and presentation elements

8.2.2.2.1 ACE_INT.1.1C

The PP-Module introduction shall contain a PP-Module reference and a TOE overview.

8.2.2.2.2 ACE_INT.1.2C

The PP-Module reference shall uniquely identify the PP-Module.

8.2.2.2.3 ACE_INT.1.3C

The TOE overview shall summarise the usage and major security features of the TOE.

8.2.2.2.4 ACE_INT.1.4C

8.2.2.2.5 The TOE overview shall identify the TOE type.

8.2.2.2.6 ACE_INT.1.5C

The TOE overview shall identify any non-TOE hardware/software/firmware available to the TOE.

8.2.2.2.7 ACE_INT.1.6C

The PP-Module introduction shall uniquely identify the base PPs and PP-Modules it depends on.

8.2.2.2.8 ACE_INT.1.7C

The PP-Module introduction shall describe the dependency structure of the base PPs and PP-Modules.

8.2.2.2.9 ACE_INT.1.8C

The TOE overview shall describe the differences of the TOE with regard to the TOEs defined in the base PPs and PP-Modules.

8.2.2.3 Evaluator action elements

8.2.2.3.1 ACE_INT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.3 PP-Module conformance claims (ACE_CCL)

8.3.1 Objectives

The objective of this family is to determine the validity of the conformance claim and conformance statement. Unlike standard Protection Profiles, a PP-Module cannot claim conformance to another PP or PP-Module.

8.3.2 ACE_CCL.1 PP-Module conformance claims

Dependencies: ACE_INT.1 PP-Module introduction

ACE_ECD.1 PP-Module extended components definition

ACE_REQ.1 PP-Module stated security requirements or ACE_REQ.2 PP-Module security requirements

8.3.2.1.1 ACE_CCL.1.1D

The developer shall provide a conformance claim.

8.3.2.1.2 ACE_CCL.1.2D

The developer shall provide a conformance statement.

8.3.2.2 Content and presentation elements

8.3.2.2.1 ACE_CCL.1.1C

The conformance claim shall contain an ISO/IEC 15408 conformance claim that identifies the ISO/IEC 15408-1 edition to which the PP-Module claims conformance.

8.3.2.2.2 ACE_CCL.1.2C

ISO/IEC 15408 conformance claim shall describe the conformance of the PP-Module to ISO/IEC 15408-2 as either ISO/IEC 15408-2 conformant or ISO/IEC 15408-2 extended.

8.3.2.2.3 ACE_CCL.1.3C

The ISO/IEC 15408 conformance claim shall describe the conformance of the PP-Module to this document as either “ISO/IEC 15408-3 conformant” or ISO/IEC 15408-3 extended.”

8.3.2.2.4 ACE_CCL.1.4C

ISO/IEC 15408 conformance claim shall be consistent with the extended components definition.

8.3.2.2.5 ACE_CCL.1.5C

The conformance claim shall identify all security requirement packages to which the PP claims conformance.

8.3.2.2.6 ACE_CCL.1.6C

The conformance claim shall describe any conformance of the PP-Module to a package as either package-conformant or package-augmented.

8.3.2.2.7 ACE_CCL.1.7C

The conformance statement shall describe the conformance required of any PP-Configuration/ST to the PP-Module as one of exact, strict, or demonstrable.

8.3.2.2.8 ACE_CCL.1.8C

The conformance statement shall identify the set of PPs and PP-Modules to which, in combination with the PP-Module under evaluation, exact conformance is allowed to be claimed.

8.3.2.2.9 ACE_CCL.1.10C

The conformance statement shall identify the set of derived Evaluation Methods and Evaluation Activities (if any) that shall be used with the PP-Module under evaluation. This list shall contain any Evaluation Methods and Evaluation Activities that are specified in the PP-Module but also

any Evaluation Activities and Evaluation Methods specified in the base PPs and/or PP-modules and/or in the packages (if any) for which conformance is being claimed by the PP-Module under evaluation.

8.3.2.3 Evaluator action elements

8.3.2.3.1 ACE_CCL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.4 PP-Module Security problem definition (ACE_SPD)

8.4.1 Objectives

This part of the PP-Module defines the security problem to be addressed by the TOE and the operational environment of the TOE.

Evaluation of the security problem definition is required to demonstrate that the security problem intended to be addressed by the TOE and its operational environment, is clearly defined.

8.4.2 ACE_SPD.1 PP-Module Security problem definition

Dependencies: No dependencies.

8.4.2.1 Developer action elements

8.4.2.1.1 ACE_SPD.1.1D

The developer shall provide a security problem definition.

8.4.2.2 Content and presentation elements

8.4.2.2.1 ACE_SPD.1.1C

The security problem definition shall describe the threats.

8.4.2.2.2 ACE_SPD.1.2C

All threats shall be described in terms of a threat agent, an asset, and an adverse action.

8.4.2.2.3 ACE_SPD.1.3C

The security problem definition shall describe the OSPs.

8.4.2.2.4 ACE_SPD.1.4C

The security problem definition shall describe the assumptions about the operational environment of the TOE.

8.4.2.3 Evaluator action elements

8.4.2.3.1 ACE_SPD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.5 PP-Module Security objectives (ACE_OBJ)

8.5.1 Objectives

The security objectives are a concise statement of the intended response to the security problem defined through the Security problem definition (APE_SPD) family.

Evaluation of the security objectives is required to demonstrate that the security objectives adequately and completely address the security problem definition and that the division of this problem between the TOE and its operational environment is clearly defined.

8.5.2 Component levelling

The components in this family are levelled on whether they prescribe only security objectives for the operational environment (see ACE_OBJ.1), or also security objectives for the TOE (see ACE_OBJ.2).

8.5.3 ACE_OBJ.1 Direct Rationale PP-Module Security objectives

Dependencies: No dependencies.

8.5.4 Application notes

If the PP-Module uses the Direct Rationale approach then all the elements defined in ACE_OBJ.1 hold.

8.5.4.1 Developer action elements

8.5.4.1.1 ACE_OBJ.1.1D

The developer shall provide a statement of security objectives for the PP-Module.

8.5.4.2 Content and presentation elements

8.5.4.2.1 ACE_OBJ.1.1C

The statement of security objectives shall describe the security objectives for the operational environment.

8.5.4.3 Evaluator action elements

8.5.4.3.1 ACE_OBJ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.5.5 ACE_OBJ.2 PP-Module Security objectives

Dependencies: ACE_SPD.1 PP-Module security problem definition.

8.5.6 Application notes

If the PP-Module does not use the Direct Rationale approach then all elements of ACE_OBJ.2 hold.

8.5.6.1 Developer action elements

8.5.6.1.1 ACE_OBJ.2.1D

The developer shall provide a statement of security objectives for the PP-Module.

8.5.6.1.2 ACE_OBJ.2.2D

The developer shall provide a security objectives rationale for the PP-Module.

8.5.6.2 Content and presentation elements

8.5.6.2.1 ACE_OBJ.2.1C

The statement of security objectives shall describe the security objectives for the TOE and the security objectives for the operational environment.

8.5.6.2.2 ACE_OBJ.2.2C

The security objectives rationale shall trace each security objective for the TOE back to threats countered by that security objective and OSPs enforced by that security objective.

8.5.6.2.3 ACE_OBJ.2.3C

The security objectives rationale shall trace each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

8.5.6.2.4 ACE_OBJ.2.4C

The security objectives rationale shall demonstrate that the security objectives counter all threats.

8.5.6.2.5 ACE_OBJ.2.5C

The security objectives rationale shall demonstrate that the security objectives enforce all OSPs.

8.5.6.2.6 ACE_OBJ.2.6C

The security objectives rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

8.5.6.3 Evaluator action elements

8.5.6.3.1 ACE_OBJ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.6 PP-Module extended components definition (ACE_ECD)

8.6.1 Objectives

Extended security functional requirements are requirements that are not based on components from **ISO/IEC 15408-2**, but are based on extended components: components defined by the PP-Module author.

Evaluation of the definition of extended functional components is necessary to determine that they are clear and unambiguous, and that they are necessary, i.e. they may not be clearly expressed using existing **ISO/IEC 15408-2** components.

8.6.2 ACE_ECD.1 PP-Module extended components definition

Dependencies: No dependencies.

8.6.2.1 Developer action elements

8.6.2.1.1 ACE_ECD.1.1D

The developer shall provide a statement of security requirements for the PP-Module.

8.6.2.1.2 ACE_ECD.1.2D

The developer shall provide an extended components definition for the PP-Module.

8.6.2.2 Content and presentation elements

8.6.2.2.1 ACE_ECD.1.1C

The statement of security requirements shall identify all the extended security requirements.

8.6.2.2.2 ACE_ECD.1.2C

The extended components definition shall define an extended component for each extended security requirement.

8.6.2.2.3 ACE_ECD.1.3C

The extended components definition shall describe how each extended component is related to the existing ISO/IEC 15408 components, families, and classes.

8.6.2.2.4 ACE_ECD.1.4C

The extended components definition shall use the existing ISO/IEC 15408 components, families, classes, and methodology as a model for presentation.

8.6.2.2.5 ACE_ECD.1.5C

8.6.2.2.6 The extended components shall consist of measurable and objective elements such that conformance or nonconformance to these elements can be demonstrated.

8.6.2.3 Evaluator action elements

8.6.2.3.1 ACE_ECD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.6.2.3.2 ACE_ECD.1.2E

The evaluator shall confirm that no extended component may be clearly expressed using existing components.

8.7 PP-Module security requirements (ACE_REQ)

8.7.1 Objectives

The SFRs form a clear, unambiguous and well-defined description of the expected security behaviour of the TOE. The SARs form a clear, unambiguous and well-defined description of the expected activities that will be undertaken to gain assurance in the TOE.

Evaluation of the security requirements is required to ensure that they are clear, unambiguous and well-defined.

8.7.2 Component levelling

The components in this family are levelled on whether they are stated as is (see ACE_REQ.1), or whether the SFRs are derived from security objectives for the TOE (see ACE_REQ.2.).

Editor's note:

The title of ACE_REQ.1 is confusing. We propose to rename it as "Direct rationale PP-Module security requirements".

Unless experts pronounce themselves against this proposal, this change will be made in the next draft.

The same applies to the title of APE_REQ.1

8.7.3 ACE_REQ.1 PP-Module stated security requirements

Dependencies: APE_ECD.1 Extended components definition

ACE_SPD.1 PP-Module security problem definition

8.7.3.1 Developer action elements

8.7.3.1.1 ACE_REQ.1.1D

The developer shall provide a statement of security requirements for the PP-Module.

8.7.3.1.2 ACE_REQ.1.2D

The developer shall provide a security requirements rationale for the PP-Module.

8.7.3.2 Content and presentation elements

8.7.3.2.1 ACE_REQ.1.1C

The statement of security requirements shall describe the SFRs and the SARs.

8.7.3.2.2 ACE_REQ.1.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

8.7.3.2.3 ACE_REQ.1.3C

The statement of security requirements shall include a natural language description, part of which describes how the SFRs combine together to provide security functionality in terms of the architecture that is observable to Administrators and other users, or in terms of internal features or properties.

8.7.3.2.4 ACE_REQ.1.4C

The statement of security requirements shall identify all operations on the security requirements.

8.7.3.2.5 ACE_REQ.1.5C

All operations shall be performed correctly.

8.7.3.2.6 ACE_REQ.1.6C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

8.7.3.2.7 ACE_REQ.1.7C

The security requirements rationale shall trace each SFR back to the threats countered by that SFR and the OSPs enforced by that SFR.

8.7.3.2.8 ACE_REQ.1.8C

The security requirements rationale shall trace each security objective for the operational environment back to the threats countered by that security objective, the OSPs enforced by that security objective, and the assumptions upheld by that security objective.

8.7.3.2.9 ACE_REQ.1.9C

The security requirements rationale shall demonstrate that the SFRs (in conjunction with the security objectives for the environment) counter all the threats for the TOE.

8.7.3.2.10 ACE_REQ.1.10C

The security requirements rationale shall demonstrate that the SFRs (in conjunction with the security objectives for the environment) enforce all the OSPs for the TOE.

8.7.3.2.11 ACE_REQ.1.11C

The security requirements rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

8.7.3.2.12 ACE_REQ.1.12C

The statement of security requirements shall be internally consistent.

8.7.3.3 Evaluator action elements

8.7.3.3.1 ACE_REQ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.7.4 ACE_REQ.2 PP-Module derived security requirements

Dependencies: ACE_ECD.1 PP-Module extended components definition

ACE_OBJ.1 PP-Module Security objectives

8.7.4.1 Developer action elements

8.7.4.1.1 ACE_REQ.2.1D

The developer shall provide a statement of security requirements for the PP-Module.

8.7.4.1.2 ACE_REQ.2.2D

The developer shall provide a security requirement rationale for the PP-Module.

8.7.4.2 Content and presentation elements

8.7.4.2.1 ACE_REQ.2.1C

The statement of security requirements shall describe the SFRs and the SARs.

8.7.4.2.2 ACE_REQ.2.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

8.7.4.2.3 ACE_REQ.2.3C

The statement of security requirements shall include a natural language description, part of which describes how the SFRs combine together to provide security functionality in terms of the architecture that is observable to Administrators and other users, or in terms of internal features or properties.

8.7.4.2.4 ACE_REQ.2.4C

The statement of security requirements shall identify all operations on the security requirements.

8.7.4.2.5 ACE_REQ.2.5C

All operations shall be performed correctly.

8.7.4.2.6 ACE_REQ.2.6C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

8.7.4.2.7 ACE_REQ.2.7C

The security requirements rationale shall trace each SFR back to the security objectives for the TOE and OSPs enforced by that SFR.

8.7.4.2.8 ACE_REQ.2.8C

The security requirements rationale shall demonstrate that the SFRs meet all security objectives for the TOE.

8.7.4.2.9 ACE_REQ.2.9C

The security requirements rationale shall demonstrate that the SFRs enforce all OSPs.

8.7.4.2.10 ACE_REQ.2.10C

The security requirements rationale shall explain why the SARs were chosen.

8.7.4.2.11 ACE_REQ.2.11C

The statement of security requirements shall be internally consistent.

8.7.4.3 Evaluator action elements

8.7.4.3.1 ACE_REQ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.8 PP-Module consistency (ACE_MCO)

8.8.1 Objectives

The objective of this family is to determine the consistency of the PP-Module.

8.8.2 ACE_MCO.1 PP-Module consistency

Dependencies: ACE_INT.1 PP-Module introduction

ACE_SPD.1 PP-Module Security problem definition

ACE_OBJ.1 Direct Rationale PP-Module Security objectives for the environment or ACE_OBJ.2 PP-Module Security objectives

ACE_REQ.1 PP-Module stated security requirements or ACE_REQ.2 PP-Module security requirements

8.8.2.1 Developer action elements

8.8.2.1.1 ACE_MCO.1.1D

The developer shall provide a consistency rationale of the PP-Module for each of the alternative sets of Base-PPs and PP-Modules identified in the PP-Module introduction.

8.8.2.2 Content and presentation elements

8.8.2.2.1 ACE_MCO.1.1C

The consistency rationale shall demonstrate that the TOE type of the PP-Module and the TOE types of its base PPs and PP-Modules are consistent.

8.8.2.2.2 ACE_MCO.1.2C

Editor's Note: this is also meaningful for APE and ASE when the ST claims conformance to more than one PP or when the ST adds elements to the PPs it conforms to: The change has not been proposed yet in ASE/APE, but if experts agree, we suggest cascading this change in the next CD.

The consistency rationale shall identify the assets of the PP-Module that also belong to some of its base PP(s) and/or PP-Module(s) and amongst them those for which the PP-Module and the base PP(s) and PP-Module(s) define different security problems.

8.8.2.2.3 ACE_MCO.1.3C

The consistency rationale shall demonstrate that the security problem definition of the PP-Module and the security problem definition of its base PPs and PP-Modules are consistent.

8.8.2.2.4 ACE_MCO.1.4C

The consistency rationale shall demonstrate that the security objectives of the PP-Module and the security objectives of its base PPs and PP-Modules are consistent.

8.8.2.2.5 ACE_MCO.1.5C

The consistency rationale shall demonstrate that the security functional requirements of the PP-Module and the security functional requirements of its base PPs and PP-Modules are consistent.

8.8.2.2.6 ACE_MCO.1.6C

The consistency rationale shall demonstrate that the security assurance requirements of the PP-Module and the security assurance requirements of its base PPs and PP-Modules are consistent.

8.8.2.3 Evaluator action elements

8.8.2.3.1 ACE_MCO.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence. If the PP-Module specifies alternative sets of Base-PPs and PP-Modules, the evaluator shall perform this action for each consistency rationale.

8.9 PP-Configuration consistency (ACE_CCO)

8.9.1 Objectives

The objective of this family is to determine the well-formedness and the consistency of the PP-Configuration.

8.9.2 ACE_CCO.1 PP-Configuration consistency

Dependencies: ACE_INT.1 PP-Module introduction

ACE_CCL.1 PP-Module conformance claims

ACE_SPD.1 PP-Module security problem definition

ACE_OBJ.1 Direct Rationale PP-Module security objectives for the environment or ACE_OBJ.2 PP-Module Security objectives

ACE_ECD.1 PP-Module extended component definition

ACE_REQ.1 PP-Module stated security requirements or ACE_REQ.2 PP-Module security requirements

ACE_MCO.1 PP-Module consistency

APE_*

8.9.2.1 Developer action elements

8.9.2.1.1 ACE_CCO.1.1D

The developer shall provide the reference of the PP-Configuration.

8.9.2.1.2 ACE_CCO.1.2D

The developer shall provide a components list.

8.9.2.1.3 ACE_CCO.1.3D

The developer shall provide a TOE overview.

8.9.2.1.4 ACE_CCO.1.4D

The developer shall provide a conformance claim.

8.9.2.1.5 ACE_CCO.1.5D

The developer shall provide a conformance statement.

8.9.2.1.6 ACE_CCO.1.7D

The developer shall provide a consistency rationale.

8.9.2.2 Content and presentation elements

8.9.2.2.1 ACE_CCO.1.1C

The PP-Configuration reference shall uniquely identify the PP-Configuration.

8.9.2.2.2 ACE_CCO.1.2C

The components list shall uniquely identify the PPs and PP-Modules that compose the PP-Configuration.

8.9.2.2.3 ACE_CCO.1.3C

For each PP-Module identified in the components list of the PP-Configuration, the list contains at least one of its sets of base PPs and PP-Modules.

8.9.2.3 ACE_CCO.1.4C

The TOE overview shall identify the TOE type.

8.9.2.4 ACE_CCO.1.5C

The TOE overview shall describe the organisation of the TOE in terms of the sub-TSFs (TSF parts) defined in the PPs and PP-Modules that belong to the components list.

8.9.2.5 ACE_CCO.1.6C

The conformance claim shall contain an ISO/IEC 15408 conformance claim that identifies the ISO/IEC 15408-1 edition(s) to which the PPs and PP-Modules that compose the PP-Configuration claim conformance.

8.9.2.6 ACE_CCO.1.7C

ISO/IEC 15408 conformance claim shall describe the conformance of the PP-Configuration to ISO/IEC 15408-2 as either ISO/IEC 15408-2 conformant or ISO/IEC 15408-2 extended.

8.9.2.7 ACE_CCO.1.8C

The ISO/IEC 15408 conformance claim shall describe the conformance of the PP-Configuration to this document as either "ISO/IEC 15408-3 conformant" or ISO/IEC 15408-3 extended."

8.9.2.8 ACE_CCO.1.9C

ISO/IEC 15408 conformance claim shall be consistent with the extended components definition of the composing PPs and PP-Modules.

8.9.2.9 ACE_CCO.1.10C

The conformance statement shall specify the required conformance to the PP-Configuration as one of exact, strict, demonstrable, or list of strict and demonstrable types inherited from its composing PPs and PP-Modules.

8.9.2.10 ACE_CCO.1.11C

The conformance statement of a PP-Configuration of strict, demonstrable, or strict and demonstrable conformance shall define the applicable SARs/assurance packages:

- **The global set of SARs/assurance package that applies to the entire TOE.**
- **For each sub-TSF (TSF part) defined in the composing PPs and PP-Modules, the applicable set of SARs/assurance package.**

8.9.2.11 ACE_CCO.1.12C

The conformance statement of a PP-Configuration of exact conformance type shall identify the set of derived Evaluation Methods and Evaluation Activities (if any) that shall be used with the PP under evaluation. This list shall contain any Evaluation Methods and Evaluation Activities that are specified in the PP it but also any Evaluation Activities and Evaluation Methods specified in PPs and/or PP-modules and/or packages for which conformance is being claimed by the PP under evaluation.

8.9.2.1 ACE_CCO.1.13C

The consistency rationale shall demonstrate that the TOE type defined in the PP-Configuration is consistent with the TOE types defined in the PPs and PP-Modules that belong to the PP-Configuration components list.

8.9.2.2 ACE_CCO.1.14C

The consistency rationale shall demonstrate that the union of all the SPDs, security objectives and security functional requirements defined in the PPs and PP-Modules of the PP-Configuration components list is consistent.

8.9.2.3 ACE_CCO.1.15C

The consistency rationale of a PP-Configuration of strict, demonstrable, or strict and demonstrable conformance type shall demonstrate

- **the consistency of the global set of SARs/assurance package with regard to the threat models as defined in the SPDs of the component PPs and PP-Modules, and**
- **the consistency of the global set of SARs/assurance package and all the sets of SARs/assurance packages for the sub-TSFs (TSF parts) with each other.**

8.9.2.4 Evaluator action elements

8.9.2.4.1 ACE_CCO.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

8.9.2.4.2 ACE_CCO.1.2E

The evaluator shall check that the PP-Configuration consisting of all the PPs and PP-Modules identified in the components list is consistent.

9 Class ASE: Security Target evaluation

9.1 Introduction

Evaluating an ST is required to demonstrate that the ST is sound and internally consistent, and, if the ST is based on a PP-Configuration, or one or more PPs or packages, that the ST is a correct instantiation of the PP-Configuration, PPs, and packages. These properties are necessary for the ST to be suitable for use as the basis for a TOE evaluation.

This clause should be used in conjunction with Annexes A, B and C in ISO/IEC 15408-1:XXXX, as these annexes clarify the concepts here and provide many examples.

Figure 6 shows the families within this class, and the hierarchy of components within the families.

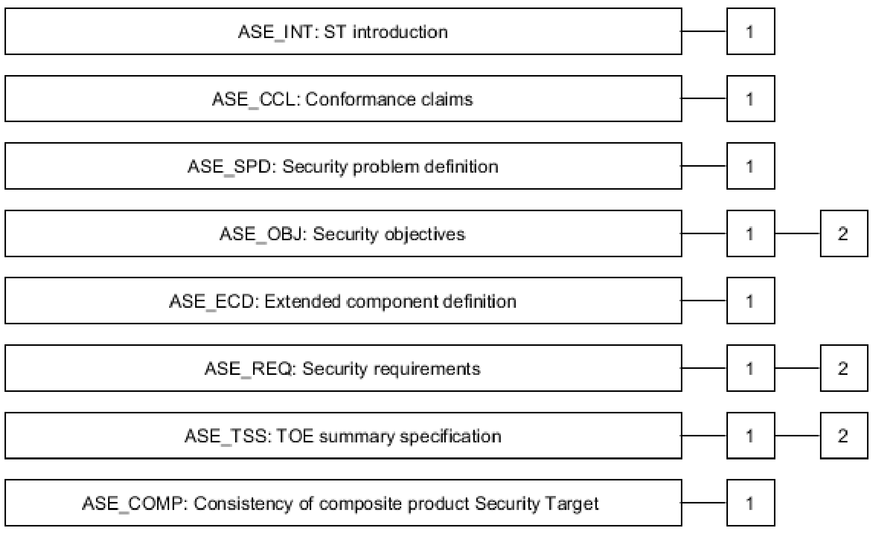
ASE_INT: ST introduction
1
ASE_CCL: Conformance claims
1
ASE_SPD: Security problem definition
1
ASE_OBJ: Security objectives
1
2
ASE_ECD: Extended component definition
1
ASE_REQ: Security requirements
1
2
ASE_TSS: TOE summary specification
1
2
ASE_COMP: Consistency of composite product Security Target
1

Figure 6 — ASE: Security Target evaluation class decomposition

9.2 ST introduction (ASE_INT)

9.2.1 Objectives

The objective of this family is to describe the TOE in a narrative way on three levels of abstraction: TOE reference, TOE overview and TOE description.

Evaluation of the ST introduction is required to demonstrate that the ST and the TOE are correctly identified, that the TOE is correctly described at three levels of abstraction and that these three descriptions are consistent with each other.

9.2.2 ASE_INT.1 ST introduction

Dependencies: No dependencies.

9.2.2.1 Developer action elements

9.2.2.1.1 ASE_INT.1.1D

The developer shall provide an ST introduction.

9.2.2.2 Content and presentation elements

9.2.2.2.1 ASE_INT.1.1C

The ST introduction shall contain an ST reference, a TOE reference, a TOE overview and a TOE description.

9.2.2.2.2 ASE_INT.1.2C

The ST reference shall uniquely identify the ST.

9.2.2.2.3 ASE_INT.1.3C

The TOE reference shall uniquely identify the TOE.

9.2.2.2.4 ASE_INT.1.4C

The TOE overview shall summarise the usage and major security features of the TOE.

9.2.2.2.5 ASE_INT.1.5C

The TOE overview shall identify the TOE type.

9.2.2.2.6 ASE_INT.1.6C

The TOE overview shall identify any non-TOE hardware/software/firmware required by the TOE.

9.2.2.2.7 ASE_INT.1.7C

The TOE description shall describe the physical scope of the TOE.

9.2.2.2.8 ASE_INT.1.8C

The TOE description shall describe the logical scope of the TOE.

9.2.2.3 Evaluator action elements

9.2.2.3.1 ASE_INT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.2.2.3.2 ASE_INT.1.2E

The evaluator shall confirm that the TOE reference, the TOE overview, and the TOE description are consistent with each other.

9.3 Conformance claims (ASE_CCL)

9.3.1 Objectives

The objective of this family is to determine the validity of the conformance claim. In addition, this family specifies how STs are to claim conformance with the PP or PP-Configuration.

9.3.2 ASE_CCL.1 Conformance claims

Dependencies: ASE_INT.1 ST introduction

ASE_ECD.1 Extended components definition

ASE_REQ.1 Stated security requirements

9.3.2.1 Developer action elements

9.3.2.1.1 ASE_CCL.1.1D

The developer shall provide a conformance claim.

9.3.2.1.2 ASE_CCL.1.2D

The developer shall provide a conformance claim rationale.

9.3.2.2 Content and presentation elements

9.3.2.2.1 ASE_CCL.1.1C

The conformance claim shall contain an ISO/IEC 15408 conformance claim that identifies the edition of ISO/IEC 15408 to which the ST and the TOE claim conformance.

9.3.2.2.2 ASE_CCL.1.2C

ISO/IEC 15408 conformance claim shall describe the conformance of the ST to ISO/IEC 15408-2 as either ISO/IEC 15408-2 conformant or ISO/IEC 15408-2 extended.

9.3.2.2.3 ASE_CCL.1.3C

ISO/IEC 15408 conformance claim shall describe the conformance of the ST to this document as either “ISO/IEC 15408-3 conformant” or ISO/IEC 15408-3 extended.”

9.3.2.2.4 ASE_CCL.1.4C

ISO/IEC 15408 conformance claim shall be consistent with the extended components definition.

9.3.2.2.5 ASE_CCL.1.5C

The conformance claim shall identify a PP-Configuration, or all PPs and security requirement packages to which the ST claims conformance.

9.3.2.2.6 ASE_CCL.1.6C

The conformance claim shall describe any conformance of the ST to a package as either package-conformant or package-augmented.

9.3.2.2.7 ASE_CCL.1.7C

The conformance claim rationale shall demonstrate that the TOE type is consistent with the TOE type in the PP-Configuration or PPs for which conformance is being claimed.

9.3.2.2.8 ASE_CCL.1.8C

The conformance claim rationale shall demonstrate that the statement of the security problem definition is consistent with the statement of the security problem definition in the PP-Configuration or PPs for which conformance is being claimed.

9.3.2.2.9 ASE_CCL.1.9C

The conformance claim rationale shall demonstrate that the statement of security objectives is consistent with the statement of security objectives in the PP-Configuration or PPs for which conformance is being claimed.

9.3.2.2.10 ASE_CCL.1.10C

The conformance claim rationale shall demonstrate that the statement of security requirements is consistent with the statement of security requirements in the PP-Configuration or PPs for which conformance is being claimed.

9.3.2.2.11 ASE_CCL.1.11C

The conformance claim for PP(s) and PP-Configuration(s) shall be exact, strict, or demonstrable.

9.3.2.3 Evaluator action elements

9.3.2.3.1 ASE_CCL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.4 Security problem definition (ASE_SPD)

9.4.1 Objectives

This part of the ST defines the security problem to be addressed by the TOE and the operational environment of the TOE.

Evaluation of the security problem definition is required to demonstrate that the security problem intended to be addressed by the TOE and its operational environment, is clearly defined.

9.4.2 ASE_SPD.1 Security problem definition

Dependencies: No dependencies.

9.4.2.1 Developer action elements

9.4.2.1.1 ASE_SPD.1.1D

The developer shall provide a security problem definition.

9.4.2.2 Content and presentation elements

9.4.2.2.1 ASE_SPD.1.1C

The security problem definition shall describe the threats.

9.4.2.2.2 ASE_SPD.1.2C

All threats shall be described in terms of a threat agent, an asset, and an adverse action.

9.4.2.2.3 ASE_SPD.1.3C

The security problem definition shall describe the OSPs.

9.4.2.2.4 ASE_SPD.1.4C

The security problem definition shall describe the assumptions about the operational environment of the TOE.

9.4.2.3 Evaluator action elements

9.4.2.3.1 ASE_SPD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.5 Security objectives (ASE_OBJ)

9.5.1 Objectives

The security objectives are a concise statement of the intended response to the security problem defined through the Security problem definition (ASE_SPD) family.

Evaluation of the security objectives is required to demonstrate that the security objectives adequately and completely address the security problem definition, that the division of this problem between the TOE and its operational environment is clearly defined.

9.5.2 Component levelling

The components in this family are levelled on whether they prescribe only security objectives for the operational environment, or also security objectives for the TOE.

9.5.3 ASE_OBJ.1 Security objectives for the operational environment

Dependencies: No dependencies

9.5.3.1 Developer action elements

9.5.3.1.1 ASE_OBJ.1.1D

The developer shall provide a statement of security objectives.

9.5.3.2 Content and presentation elements

9.5.3.2.1 ASE_OBJ.1.1C

The statement of security objectives shall describe the security objectives for the operational environment.

9.5.3.3 Evaluator action elements

9.5.3.3.1 ASE_OBJ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.5.4 ASE_OBJ.2 Security objectives

Dependencies: ASE_SPD.1 Security problem definition

9.5.4.1 Developer action elements

9.5.4.1.1 ASE_OBJ.2.1D

The developer shall provide a statement of security objectives.

9.5.4.1.2 ASE_OBJ.2.2D

The developer shall provide a security objectives rationale.

9.5.4.2 Content and presentation elements

9.5.4.2.1 ASE_OBJ.2.1C

The statement of security objectives shall describe the security objectives for the TOE and the security objectives for the operational environment.

9.5.4.2.2 ASE_OBJ.2.2C

The security objectives rationale shall trace each security objective for the TOE back to threats countered by that security objective and OSPs enforced by that security objective.

9.5.4.2.3 ASE_OBJ.2.3C

The security objectives rationale shall trace each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

9.5.4.2.4 ASE_OBJ.2.4C

The security objectives **rationale** shall **demonstrate that** the security objectives **counter all threats.**

9.5.4.2.5 ASE_OBJ.2.5C

The security objectives rationale shall demonstrate that the security objectives enforce all OSPs.

9.5.4.2.6 ASE_OBJ.2.6C

The security objectives rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

9.5.4.3 Evaluator action elements

9.5.4.3.1 ASE_OBJ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.6 Extended components definition (ASE_ECD)

9.6.1 Objectives

Extended security requirements are requirements that are not based on components from ISO/IEC 15408-2 or this document, but are based on extended components: components defined by the ST author.

Evaluation of the definition of extended components is necessary to determine that they are clear and unambiguous, and that they are necessary, i.e. they may not be clearly expressed using existing ISO/IEC 15408-2 or this document components.

9.6.2 ASE_ECD.1 Extended components definition

Dependencies: No dependencies.

9.6.2.1 Developer action elements

9.6.2.1.1 ASE_ECD.1.1D

The developer shall provide a statement of security requirements.

9.6.2.1.2 ASE_ECD.1.2D

The developer shall provide an extended components definition.

9.6.2.2 Content and presentation elements

9.6.2.2.1 ASE_ECD.1.1C

The statement of security requirements shall identify all extended security requirements.

9.6.2.2.2 ASE_ECD.1.2C

The extended components definition shall define an extended component for each extended security requirement.

9.6.2.2.3 ASE_ECD.1.3C

The extended components definition shall describe how each extended component is related to the existing ISO/IEC 15408 components, families, and classes.

9.6.2.2.4 ASE_ECD.1.4C

The extended components definition shall use the existing ISO/IEC 15408 components, families, classes, and methodology as a model for presentation.

9.6.2.2.5 ASE_ECD.1.5C

The extended components shall consist of measurable and objective elements such that conformance or nonconformance to these elements can be demonstrated.

9.6.2.3 Evaluator action elements

9.6.2.3.1 ASE_ECD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.6.2.3.2 ASE_ECD.1.2E

The evaluator shall confirm that no extended component can be clearly expressed using existing components.

9.7 Security requirements (ASE_REQ)

9.7.1 Objectives

The SFRs form a clear, unambiguous and well-defined description of the expected security behaviour of the TOE. The SARs form a clear, unambiguous and canonical description of the expected activities that will be undertaken to gain assurance in the TOE.

Evaluation of the security requirements is required to ensure that they are clear, unambiguous and well-defined.

9.7.2 Component levelling

The components in this family are levelled on whether they are stated as is.

9.7.3 ASE_REQ.1 Stated security requirements

Dependencies: ASE_ECD.1 Extended components definition

9.7.3.1 Developer action elements

9.7.3.1.1 ASE_REQ.1.1D

The developer shall provide a statement of security requirements.

9.7.3.1.2 ASE_REQ.1.2D

The developer shall provide a security requirements rationale.

9.7.3.2 Content and presentation elements

9.7.3.2.1 ASE_REQ.1.1C

The statement of security requirements shall describe the SFRs and the SARs.

9.7.3.2.2 ASE_REQ.1.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

9.7.3.2.3 ASE_REQ.1.3C

The statement of security requirements shall include a natural language description, part of which describes how the SFRs combine together to provide security functionality in terms of the architecture that is observable to Administrators and other users, or in terms of internal features or properties.

9.7.3.2.4 ASE_REQ.1.4C

The statement of security requirements shall identify all operations on the security requirements.

9.7.3.2.5 ASE_REQ.1.5C

All operations shall be performed correctly.

9.7.3.2.6 ASE_REQ.1.6C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

9.7.3.2.7 ASE_REQ.1.7C

The security requirements rationale shall trace each SFR back to the security objectives threats countered by that SFR and OSPs enforced by that SFR.

9.7.3.2.8 ASE_REQ.1.8C

The security requirements rationale shall trace each security objective for the operational environment back to threats countered by that security objective, OSPs enforced by that security objective, and assumptions upheld by that security objective.

9.7.3.2.9 ASE_REQ.1.9C

The security requirements rationale shall demonstrate that the SFRs counter all threats for the TOE.

9.7.3.2.10 ASE_REQ.1.10C

The security requirements rationale shall demonstrate that the SFRs enforce all OSPs.

9.7.3.2.11 ASE_REQ.1.11C

The security requirements rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

9.7.3.2.12 ASE_REQ.1.12C

The statement of security requirements shall be internally consistent.

9.7.3.3 Evaluator action elements

9.7.3.3.1 ASE_REQ.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.7.4 ASE_REQ.2 Derived security requirements

Dependencies: ASE_OBJ.2 Security objectives

ASE_ECD.1 Extended components definition

9.7.4.1 Developer action elements

9.7.4.1.1 ASE_REQ.2.1D

The developer shall provide a statement of security requirements.

9.7.4.1.2 ASE_REQ.2.2D

The developer shall provide a security requirements rationale.

9.7.4.2 Content and presentation elements

9.7.4.2.1 ASE_REQ.2.1C

The statement of security requirements shall describe the SFRs and the SARs.

9.7.4.2.2 ASE_REQ.2.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

9.7.4.2.3 ASE_REQ.2.3C

The statement of security requirements shall identify all operations on the security requirements.

9.7.4.2.4 ASE_REQ.2.4C

All operations shall be performed correctly.

9.7.4.2.5 ASE_REQ.2.5C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

9.7.4.2.6 ASE_REQ.2.6C

The security requirements rationale shall trace each SFR back to the SPD elements **for the TOE.**

9.7.4.2.7 ASE_REQ.2.8C

The security requirements rationale shall demonstrate that the SFRs meet all security objectives for the TOE.

9.7.4.2.8 ASE_REQ.2.9C

The security requirements rationale shall explain why the SARs were chosen.

9.7.4.2.9 ASE_REQ.2.10C

The statement of security requirements shall be internally consistent.

9.7.4.2.10 ASE_REQ.2.11C

9.7.4.3 The security requirements rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.

9.7.4.4 Evaluator action elements

9.7.4.4.1 ASE_REQ.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.8 TOE summary specification (ASE_TSS)

9.8.1 Objectives

The TOE summary specification enables evaluators and potential consumers to gain a general understanding of how the TOE is implemented.

Evaluation of the TOE summary specification is necessary to determine whether it is adequately described how the TOE:

- meets its SFRs;
- protects itself against interference, logical tampering and bypass;

and whether the TOE summary specification is consistent with other narrative descriptions of the TOE.

9.8.2 Component levelling

The components in this family are levelled on whether the TOE summary specification only needs to describe how the TOE meets the SFRs, or whether the TOE summary specification also needs to describe how the TOE protects itself against logical tampering and bypass. This additional description may be used in special circumstances where there might be a specific concern regarding the TOE security architecture.

9.8.3 ASE_TSS.1 TOE summary specification

Dependencies:	ASE_INT.1 ST introduction
	ASE_REQ.1 Stated security requirements
	ADV_FSP.1 Basic functional specification

9.8.3.1 Developer action elements

9.8.3.1.1 ASE_TSS.1.1D

The developer shall provide a TOE summary specification.

9.8.3.2 Content and presentation elements

9.8.3.2.1 ASE_TSS.1.1C

The TOE summary specification shall describe how the TOE meets each SFR.

9.8.3.3 Evaluator action elements

9.8.3.3.1 ASE_TSS.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.8.3.3.2 ASE_TSS.1.2E

The evaluator shall confirm that the TOE summary specification is consistent with the TOE overview and the TOE description.

9.8.4 ASE_TSS.2 TOE summary specification with architectural design summary

Dependencies:	ASE_INT.1 ST introduction
	ASE_REQ.1 Stated security requirements
	ADV_ARC.1 Security architecture description

9.8.4.1 Developer action elements

9.8.4.1.1 ASE_TSS.2.1D

The developer shall provide a TOE summary specification.

9.8.4.2 Content and presentation elements

9.8.4.2.1 ASE_TSS.2.1C

The TOE summary specification shall describe how the TOE meets each SFR.

9.8.4.2.2 ASE_TSS.2.2C

The TOE summary specification shall describe how the TOE protects itself against interference and logical tampering.

9.8.4.2.3 ASE_TSS.2.3C

The TOE summary specification shall describe how the TOE protects itself against bypass.

9.8.4.3 Evaluator action elements

9.8.4.3.1 ASE_TSS.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

9.8.4.3.2 ASE_TSS.2.2E

The evaluator shall confirm that the TOE summary specification is consistent with the TOE overview and the TOE description.

9.9 Consistency of composite product Security Target (ASE_COMP)

9.9.1 Objectives

The aim of this activity is to determine whether the Security Target of the composite product[2] does not contradict the Security Target of the underlying platform[3].

9.9.2 ASE_COMP.1 Consistency of Security Target

Dependencies: No dependencies

9.9.2.1 Application notes

These application notes aid the developer to create as well as the evaluator to analyse a composite Security Target and describe a general methodology for it. For detailed information / guidance please refer to the single work units below.

In order to create a composite Security Target, the developer should perform the following steps:

Step 1: The developer formulates a preliminary Security Target for the composite product (the Composite-ST) using the standard code of practice. The Composite-ST can be formulated independently of the Security Target of the underlying platform (Platform-ST) – at least as long as there are no formal PP conformance claims.

Step 2: The developer determines the overlap between Platform-ST and Composite-ST through analysing and comparing their TOE Security Functionality (TSF) [4][5]:

[2] denoted by Composite-ST in the following

[3] denoted by Platform-ST in the following. Generally, a Security Target expresses a security policy for the TOE defined.

[4] because the TSF enforce the Security Target (together with organisational measures enforcing security objectives for the operational environment of the TOE).

[5] The comparison shall be performed on the abstraction level of SFRs. If the developer defined security functionality groups (TSF-groups) in the TSS part of his Security Target, the evaluator should also consider them in order to get a better understanding for the context of the security services offered by the TOE.

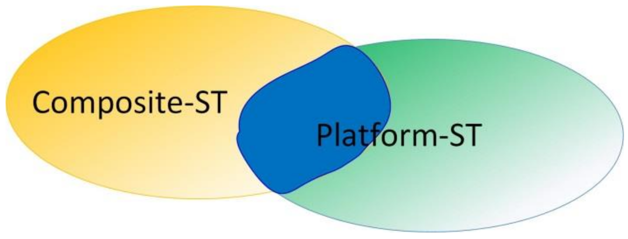

Figure 7 - Overlap between Platform-ST and Composite-ST

Step 3: The developer determines under which conditions he can trust in and rely on the Platform-TSF being used by the Composite-ST without a new examination.

Having undertaken these steps the developer completes the preliminary Security Target for the composite product.

It is not mandatory that the platform and the composite TOE are being certified according to same edition of 15408. It is due to the fact that the application can rely on some security services of the platform, if (i) the assurance level of the platform covers the intended assurance level of the composite TOE and (ii) the platform's security certificate is valid and up-to-date. Equivalence of single assurance components (and, hence, of assurance levels) belonging to different **ISO/IEC 15408 editions** shall be established / acknowledged by the Composite Product Certification Body, cf. chapter Annex A.

If a PP conformance is claimed (e.g. composite ST claim conformance to a PP that claims conformance to a hardware PP), the consistency check can be reduced to the elements of the Security Target having not already been covered by these Protection Profiles.

The fact of compliance to a PP is not sufficient to avoid inconsistencies. Assume the following situation, where → stands for "complies with"

Composite-ST → SW PP → HW PP ← platform-ST

The SW PP may require any kind of conformance[6], but this does not change the 'additional elements' that the platform-ST may introduce to the HW PP. In conclusion, these additions are not necessarily consistent with the composite-ST/SW PP additions: There is no scenario that ensures the consistency 'by construction'.

Note that consistency may not be direct matching: e.g. objectives for the platform environment may become objectives for the composite TOE.

9.9.2.2 Developer action elements

9.9.2.2.1 ASE_COMP.1.1D

The developer shall provide a statement of compatibility between the Composite Security Target and the Platform Security Target. This statement can be provided within the Composite Product Security Target.

[6] e.g. "strict", "exact" or "demonstrable" according to ISO/IEC 15408.

9.9.2.3 Content and presentation elements

9.9.2.3.1 ASE_COMP.1.1C

The statement of compatibility shall describe the separation of the Platform-TSF into relevant Platform-TSF being used by the Composite-ST and others.

9.9.2.3.2 ASE_COMP.1.2C

The statement of compatibility between the Composite Security Target and the Platform Security Target shall show (e.g. in form of a mapping) that the Security Targets of the composite product and of the underlying platform match, i.e. that there is no conflict between security environments, security objectives, and security requirements of the Composite Security Target and the Platform Security Target. It can be provided by indicating of the concerned elements directly in the Security Target for the composite product followed by explanatory text, if necessary.

9.9.2.4 Evaluator action elements

9.9.2.4.1 ASE_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10 Class ADV: Development

10.1 Introduction

The requirements of the Development class provide information about the TOE. The knowledge obtained by this information is used as the basis for conducting vulnerability analysis and testing upon the TOE, as described in the AVA and ATE classes.

The Development class encompasses seven families of requirements for structuring and representing the TSF at various levels and varying forms of abstraction. These families include:

- requirements for the description (at the various levels of abstraction) of the design and implementation of the SFRs (ADV_FSP, ADV_TDS, ADV_IMP and ADV_COMP)
- requirements for the description of the architecture-oriented features of domain separation, TSF self-protection and non-bypassability of the security functionality (ADV_ARC)
- requirements for a security policy model and for correspondence mappings between security policy model and the functional specification (ADV_SPM)
- requirements on the internal structure of the TSF, which covers aspects such as modularity, layering, and minimisation of complexity (ADV_INT)

When documenting the security functionality of a TOE, there are two properties that need to be demonstrated. The first property is that the security functionality works correctly; that is, it performs as specified. The second property, and one that is arguably harder to demonstrate, is that the TOE cannot be used in a way such that the security functionality can be corrupted or bypassed. These two properties require somewhat different approaches in analysis, and so the families in ADV are structured to support these different approaches. The families Functional specification (ADV_FSP), TOE design (ADV_TDS), Implementation representation (ADV_IMP), and Security policy modelling

(ADV_SPM) deal with the first property: the specification of the security functionality. The families Security Architecture (ADV_ARC) and TSF internals (ADV_INT) deal with the second property: the specification of the design of the TOE demonstrating the security functionality cannot be corrupted or bypassed. It should be noted that both properties need to be realised: the more confidence one has that the properties are satisfied, the more trustworthy the TOE is. The TSF of a composite product are represented at various levels of abstraction in the families of the development class ADV. The family Composite design compliance (ADV_COMP) determines whether the requirements on the application, imposed by the underlying platform, are fulfilled in a composite product. Due to the distribution of the TSF of a compistie product to various levels in the families of the class ADV, this family is not represented in Figure 8. The components in the families are designed so that more assurance can be gained as the components hierarchically increase.

The paradigm for the families targeted at the first property is one of design decomposition. At the highest level, there is a functional specification of the TSF in terms of its interfaces (describing *what* the TSF does in terms of requests to the TSF for services and resulting responses), decomposing the TSF into smaller units (dependent on the assurance desired and the complexity of the TOE) and describing *how* the TSF accomplishes its functions (to a level of detail commensurate with the assurance level), and showing the implementation of the TSF. A formal model of the security behaviour also may be given. All levels of decomposition are used in determining the completeness and accuracy of all other levels, ensuring that the levels are mutually supportive. The requirements for the various TSF representations are separated into different families, to allow the PP/ST author to specify which TSF representations are required. The level chosen will dictate the assurance desired/gained.

Figure 8 indicates the relationships among the various TSF representations of the ADV class, as well as their relationships with other classes. As the figure indicates, the APE and ASE classes define the requirements for the correspondence between the SFRs and the security objectives for the TOE. Class ASE also defines requirements for the correspondence between both the security objectives and SFRs, and for the TOE summary specification which explains how the TOE meets its SFRs. The activities of ALC_CMC.5.2E include the verification that the TSF that is tested under the ATE and AVA classes is in fact the one described by all of the ADV decomposition levels.

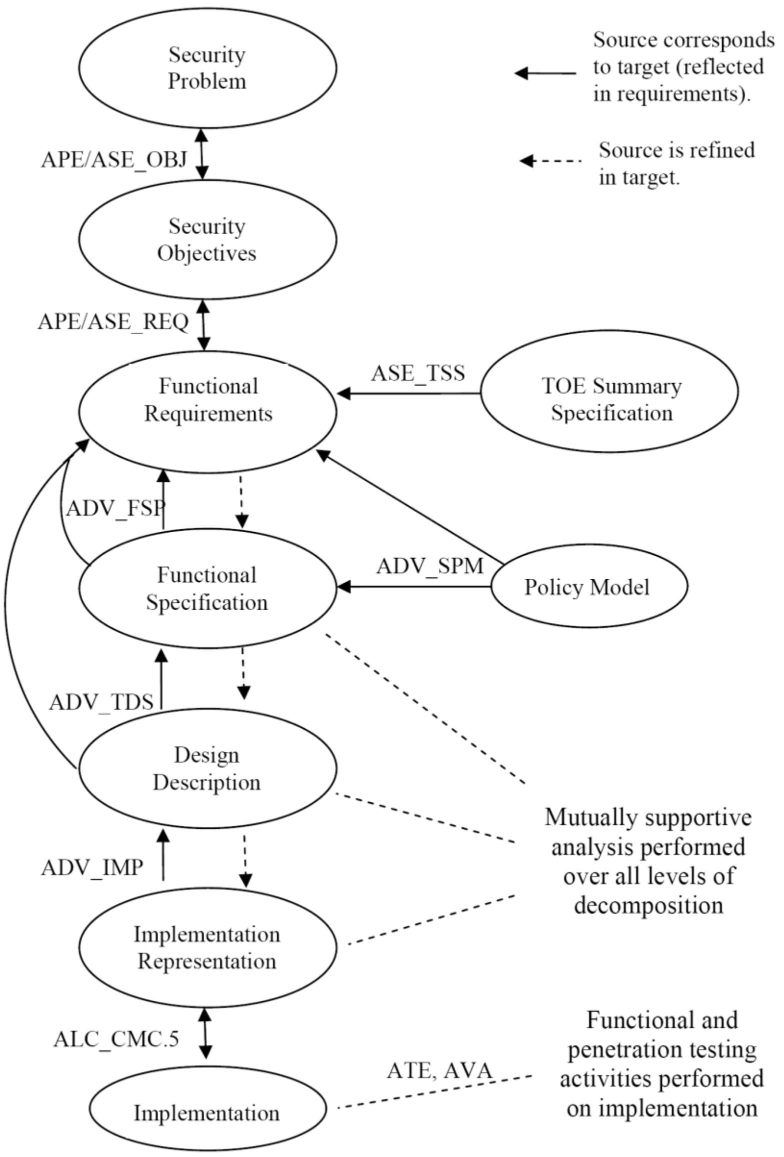
Security Problem
Source corresponds to target (reflected in requirements).
Source is refined in target.
APE/ASE_OBJ
Security Objectives
APE/ASE_REQ
Functional Requirements
ASE_TSS
TOE Summary Specification
ADV_FSP
Functional Specification
ADV_SPM
Policy Model
ADV_TDS
Design Description
Mutually supportive analysis performed over all levels of decomposition
ADV_IMP
Implementation Representation
ALC_CMC.5
Functional and penetration testing activities performed on implementation
ATE, AVA
Implementation

Figure 8 — Relationships of ADV constructs to one another and to other families

The requirements for all other correspondence shown in Figure 8 are defined in the ADV class for the TOE. The Security policy modelling (ADV_SPM) family defines the requirements for formally modelling selected SFRs, and providing correspondence between the functional specification and the formal model. Each assurance family specific to a TSF representation (i.e., Functional specification (ADV_FSP), TOE design (ADV_TDS) and Implementation representation (ADV_IMP)) defines requirements relating that TSF representation to the SFRs. All decompositions must accurately reflect all other decompositions (i.e., be mutually supportive); the developer supplies the tracings in the last .C elements of the components. Assurance relating to this factor is obtained during the analysis for each of the levels of decomposition by referring to other levels of decomposition (in a recursive fashion) while the analysis of a particular level of decomposition is being performed; the evaluator verifies the correspondence as part of the second E element. The understanding gained from these levels of decomposition form the basis of the functional and penetration testing efforts.

The ADV_INT family is not represented in this figure, as it is related to the internal structure of the TSF, and is only indirectly related to the process of refinement of the TSF representations. Similarly, the

ADV_ARC family is not represented in the figure because it relates to the architectural soundness, rather than representation, of the TSF. Both ADV_INT and ADV_ARC relate to the analysis of the property that the TOE cannot be made to circumvent or corrupt its security functionality.

The TOE security functionality (TSF) consists of all parts of the TOE that have to be relied upon for enforcement of the SFRs. The TSF includes both functionality that directly enforces the SFRs, as well as functionality that, while not directly enforcing the SFRs, contributes to their enforcement in a more indirect manner, including functionality with the capability to cause the SFRs to be violated. This includes portions of the TOE that are invoked on start-up that are responsible for putting the TSF into its initial secure state.

Several important concepts were used in the development of the components of the ADV families. These concepts, while introduced briefly here, are explained more fully in the application notes for the families.

One over-riding notion is that, as more information becomes available, greater assurance can be obtained that the security functionality 1) is correctly implemented; 2) cannot be corrupted; and 3) cannot be bypassed. This is done through the verification that the documentation is correct and consistent with other documentation, and by providing information that can be used to ensure that the testing activities (both functional and penetration testing) are comprehensive. This is reflected in the levelling of the components of the families. In general, components are levelled based on the amount of information that is to be provided (and subsequently analysed).

While not true for all TOEs, it is generally the case that the TSF is sufficiently complex that there are portions of the TSF that deserve more intense examination than other portions of the TSF. Determining those portions is unfortunately somewhat subjective, thus terminology and components have been defined such that as the level of assurance increases, the responsibility for determining what portions of the TSF need to be examined in detail shifts from the developer to the evaluator. To aid in expressing this concept, the following terminology is introduced. It should be noted that in the families of the class, this terminology is used when expressing SFR-related portions of the TOE (that is, elements and work units embodied in the Functional specification (ADV_FSP), TOE design (ADV_TDS), and Implementation representation (ADV_IMP) families). While the general concept (that some portions of the TOE are more *interesting* than others) applies to other families, the criteria are expressed differently in order to obtain the assurance required.

All portions of the TSF are *security relevant*, meaning that they must preserve the security of the TOE as expressed by the SFRs and requirements for domain separation and non-bypassability. One aspect of security relevance is the degree to which a portion of the TSF enforces a security requirement. Since different portions of the TOE play different roles (or no apparent role at all) in enforcing security requirements, this creates a continuum of SFR relevance: at one end of this continuum are portions of the TOE that are termed *SFR-enforcing*. Such portions play a direct role in implementing any SFR on the TOE. Such SFRs refer to any functionality provided by one of the SFRs contained in the ST. It should be noted that the definition of *plays a role in* for SFR-enforcing functionality is impossible to express quantitatively. For example, in the implementation of a Discretionary Access Control (DAC) mechanism, a very narrow view of *SFR-enforcing* might be the several lines of code that actually perform the check of a subject's attributes against the object's attributes. A broader view would include the software entity (e.g., C function) that contained the several lines of code. A broader view still would include callers of the C function, since they would be responsible for enforcing the decision returned by the attribute check. A still broader view would include any code in the call tree (or programming equivalent for the implementation language used) for that C function (e.g., a sort function that sorted access control list entries in a first-match algorithm implementation). At some point, the component is not so much *enforcing* the security policy but rather plays a *supporting* role; such components are termed *SFR supporting*. One of the characteristics of SFR-supporting functionality is that it is trusted to preserve the correctness of the SFR implementation by operating without error. Such functionality may be depended on by SFR-enforcing functionality, but the dependence is generally at a functional level; for example, memory management, buffer management, etc. Further down on the security relevance continuum is functionality termed *SFR non-interfering*. Such functionality has no role in implementing the SFRs, and is likely part of the TSF because of its environment; for example, any code running in a privileged hardware mode on an operating system. It needs to be considered part of the TSF because, if compromised (or replaced by malicious code), it could compromise the correct operation of an SFR by virtue of its operating in the privileged hardware mode. An example of SFR non-interfering functionality might be a set of mathematical floating point operations implemented in kernel mode for speed considerations.

The architecture family (Security Architecture (ADV_ARC)) provides for requirements and analysis of the TOE based on properties of domain separation, self-protection, and non-bypassability. These properties relate to the SFRs in that, if these properties are not present, it will likely lead to the failure of mechanisms implementing SFRs. Functionality and design relating to these properties *is not* considered a part of the continuum described above, but instead is treated separately due to its fundamentally different nature and analysis requirements.

The difference in analysis of the implementation of SFRs (SFR-enforcing and SFR-supporting functionality) and the implementation of somewhat fundamental security properties of the TOE, which include the initialisation, self-protection, and non-bypassability concerns, is that the SFR-related functionality is more or less directly visible and relatively easy to test, while the above-mentioned properties require varying degrees of analysis on a much broader set of functionality. Further, the depth of analysis for such properties will vary depending on the design of the TOE. The ADV families are constructed to address this by a separate family (Security Architecture (ADV_ARC)) devoted to analysis of the initialisation, self-protection, and non-bypassability requirements, while the other families are concerned with analysis of the functionality supporting SFRs.

Even in cases where different descriptions are necessary for the multiple levels of abstraction, it is not absolutely necessary for each and every TSF representation to be in a separate document. Indeed, it may be the case that a single document meets the documentation requirements for more than one TSF representation, since it is the information about each of these TSF representations that is required, rather than the resulting document structure. In cases where multiple TSF representations are combined within a single document, the developer should indicate which portions of the documents meet which requirements.

Three types of specification style are mandated by this class: informal, semiformal and formal. The functional specification and TOE design documentation are always written in either informal or semiformal style. A semiformal style reduces the ambiguity in these documents over an informal presentation. A formal specification may also be required *in addition to* the semi-formal presentation; the value is that a description of the TSF in more than one way will add increased assurance that the TSF has been completely and accurately specified.

An informal specification is written as prose in natural language. Natural language is used here as meaning communication in any commonly spoken tongue (e.g. Spanish, German, French, English, Dutch). An informal specification is not subject to any notational or special restrictions other than those required as ordinary conventions for that language (e.g. grammar and syntax). While no notational restrictions apply, the informal specification is also required to provide defined meanings for terms that are used in a context other than that accepted by normal usage.

The difference between semiformal and informal documents is only a matter of formatting or presentation: a semiformal notation includes such things as an explicit glossary of terms, a standardised presentation format, etc. A semiformal specification is written to a standard presentation template. The presentation should use terms consistently if written in a natural language. The presentation may also use more structured languages/diagrams (e.g. data-flow diagrams, state transition diagrams, entity-relationship diagrams, data structure diagrams, and process or program structure diagrams). Whether based on diagrams or natural language, a set of conventions must be used in the presentation. The glossary explicitly identifies the words that are being used in a precise and constant manner; similarly, the standardised format implies that extreme care has been taken in methodically preparing the document in a manner that maximises clarity. It should be noted that fundamentally different portions of the TSF may have different semiformal notation conventions and presentation styles (as long as the number of different "semiformal notations" is small); this still conforms to the concept of a *semiformal presentation*.

A formal specification is written in a notation based upon well-established mathematical concepts, and is typically accompanied by supporting explanatory (informal) prose. These mathematical concepts are used to define the syntax and semantics of the notation and the proof rules that support logical reasoning. The syntactic and semantic rules supporting a formal notation should define how to recognise constructs unambiguously and determine their meaning. There needs to be evidence that it is impossible to derive contradictions, and all rules supporting the notation need to be defined or referenced.

Figure 9 shows the families within this class, and the hierarchy of components within the families.

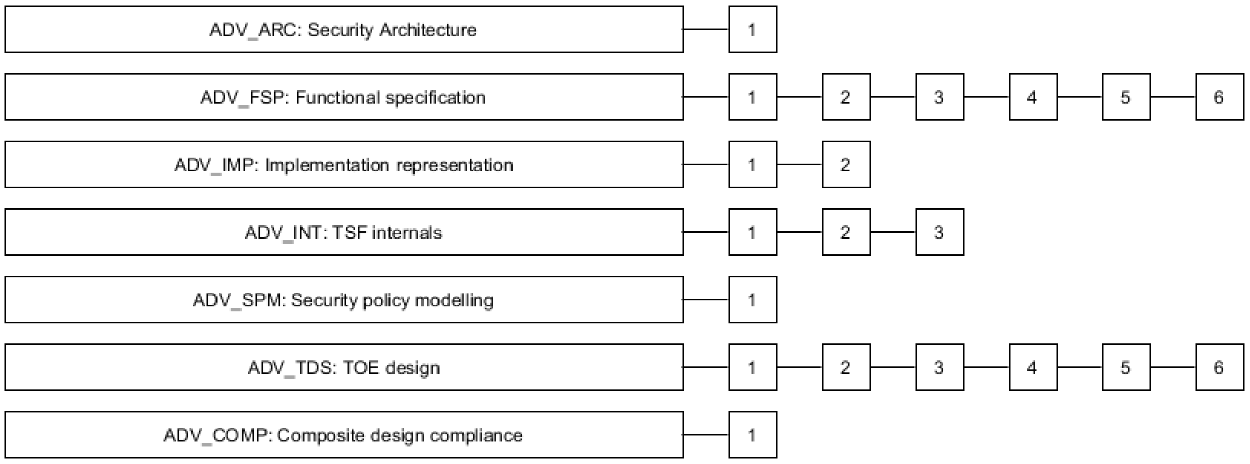
ADV_ARC: Security Architecture
1
ADV_FSP: Functional specification
1
2
3
4
5
6
ADV_IMP: Implementation representation
1
2
ADV_INT: TSF internals
1
2
3
ADV_SPM: Security policy modelling
1
ADV_TDS: TOE design
1
2
3
4
5
6
ADV_COMP: Composite design compliance
1

Figure 9 — ADV: Development class decomposition

In case of a **multi-assurance evaluation** the requirements for the description (at the various levels of abstraction) of the design and implementation of the SFRs (ADV_FSP, ADV_TDS, ADV_IMP and ADV_COMP) will be presented for the **sub-TSF of** the TOE**.** The architecture family (Security Architecture (ADV_ARC)) provides for requirements and analysis of the TOE based on properties of domain separation, self-protection, and non-bypassability which also may hold for boundaries between the **sub-TSF**.

10.2 Security Architecture (ADV_ARC)

10.2.1 Objectives

The objective of this family is for the developer to provide a description of the security architecture of the TSF. This will allow analysis of the information that, when coupled with the other evidence presented for the TSF, will confirm the TSF achieves the desired properties. The security architecture descriptions supports the implicit claim that security analysis of the TOE can be achieved by examining the TSF; without a sound architecture, the entire TOE functionality would have to be examined.

10.2.2 Component levelling

This family contains only one component.

10.2.3 Application notes

The properties of self-protection, domain separation, and non-bypassability are distinct from security functionality expressed by **ISO/IEC 15408-2** SFRs because self-protection and non-bypassability largely have no directly observable interface at the TSF. Rather, they are properties of the TSF that are achieved through the design of the TOE and TSF, and enforced by the correct implementation of that design.

The approach used in this family is for the developer to design and provide a TSF that exhibits the above-mentioned properties, and to provide evidence (in the form of documentation) that explains these properties of the TSF. This explanation is provided at the same level of detail as the description of the SFR-enforcing elements of the TOE in the TOE design document. The evaluator has the responsibility for looking at the evidence and, coupled with other evidence delivered for the TOE and TSF, determining that the properties are achieved.

Specification of security functionality implementing the SFRs (in the Functional specification (ADV_FSP) and TOE design (ADV_TDS)) will not necessarily describe mechanisms employed in implementing self-protection and non-bypassability (e.g. memory management mechanisms). Therefore, the material needed to provide the assurance that these requirements are being achieved is better suited to a presentation separate from the design decomposition of the TSF as embodied in ADV_FSP and ADV_TDS. This is not to imply that the security architecture description called for by this component cannot reference or make use of the design decomposition material; but it is likely that much of the detail present in the decomposition documentation will not be relevant to the argument being provided for the security architecture description document.

The description of architectural soundness can be thought of as a developer's vulnerability analysis, in that it provides the justification for why the TSF is sound and enforces all of its SFRs. Where the soundness is achieved through specific security mechanisms, these will be tested as part of the Depth (ATE_DPT) requirements; where the soundness is achieved solely through the architecture, the behaviour will be tested as part of the AVA: Vulnerability assessment requirements.

This family consists of requirements for a security architecture description that describes the self-protection, domain separation, non-bypassability principles, including a description of how these principles are supported by the parts of the TOE that are used for TSF initialisation.

In case of a **multi-assurance evaluation** the properties of self-protection, domain separation, and non-bypassability may also be described for boundaries between the **sub-TSF.**

Additional information on the security architecture properties of self-protection, domain separation, and non-bypassability can be found in Annex A.1, ADV_ARC: Supplementary material on security architectures.

10.2.4 ADV_ARC.1 Security architecture description

Dependencies: ADV_FSP.1 Basic functional specification

ADV_TDS.1 Basic design

10.2.4.1 Developer action elements

10.2.4.1.1 ADV_ARC.1.1D

The developer shall design and implement the TOE so that the security features of the TSF cannot be bypassed.

10.2.4.1.2 ADV_ARC.1.2D

The developer shall design and implement the TSF so that it is able to protect itself from tampering by untrusted active entities.

10.2.4.1.3 ADV_ARC.1.3D

The developer shall provide a security architecture description of the TSF.

10.2.4.2 Content and presentation elements

10.2.4.2.1 ADV_ARC.1.1C

The security architecture description shall be at a level of detail commensurate with the description of the SFR-enforcing abstractions described in the TOE design document.

10.2.4.2.2 ADV_ARC.1.2C

The security architecture description shall describe the security domains maintained by the TSF consistently with the SFRs.

10.2.4.2.3 ADV_ARC.1.3C

The security architecture description shall describe how the TSF initialisation process is secure.

10.2.4.2.4 ADV_ARC.1.4C

The security architecture description shall demonstrate that the TSF protects itself from tampering.

10.2.4.2.5 ADV_ARC.1.5C

The security architecture description shall demonstrate that the TSF prevents bypass of the SFR-enforcing functionality.

10.2.4.3 Evaluator action elements

10.2.4.3.1 ADV_ARC.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3 Functional specification (ADV_FSP)

10.3.1 Objectives

This family levies requirements upon the functional specification, which describes the TSF interfaces (TSFIs). The TSFI consists of all means by which external entities (or subjects in the TOE but outside of the TSF) supply data to the TSF, receive data from the TSF and invoke services from the TSF. It does *not* describe how the TSF processes those service requests, nor does it describe the communication when the TSF invokes services from its operational environment; this information is addressed by the TOE design (ADV_TDS) and Reliance of dependent component (ACO_REL) families, respectively.

This family provides assurance directly by allowing the evaluator to understand how the TSF meets the claimed SFRs. It also provides assurance indirectly, as input to other assurance families and classes:

- ADV_ARC, where the description of the TSFIs may be used to gain better understanding of how the TSF is protected against corruption (i.e. subversion of self-protection or domain separation) and/or bypass;
- ATE, where the description of the TSFIs is an important input for both developer and evaluator testing;
- AVA, where the description of the TSFIs is used to search for vulnerabilities.

10.3.2 Component levelling

The components in this family are levelled on the degree of detail required of the description of the TSFIs, and the degree of formalism required of the description of the TSFIs.

10.3.3 Application notes

Once the TSFIs are determined (see XXX for guidance and examples of determining TSFI), they are described. At lower-level components, developers focus their documentation (and evaluators focus their analysis) on the more security-relevant aspects of the TOE. Three categories of TSFIs are defined, based upon the relevance the services available through them have to the SFRs being claimed:

- If a service available through an interface can be traced to one of the SFRs levied on the TSF, then that interface is termed *SFR-enforcing*. Note that it is possible that an interface may have various services and results, some of which may be SFR-enforcing and some of which may not.
- Interfaces to (or services available through an interface relating to) services that SFR-enforcing functionality depends upon, but need only to function correctly in order for the security policies of the TOE to be preserved, are termed *SFR-supporting*.
- Interfaces to services on which SFR-enforcing functionality has no dependence are termed *SFR non-interfering*.

It should be noted that in order for an interface to be SFR-supporting or SFR non-interfering it must have *no* SFR-enforcing services or results. In contrast, an SFR-enforcing interface may have SFR-supporting services (for example, the ability to set the system clock may be an SFR-enforcing service of an interface, but if that same interface is used to display the system date that service may be only SFR-supporting). An example of a purely SFR-supporting interface is a system call interface that is used both by users and by a portion of the TSF that is running on behalf of users.

As more information about the TSFIs becomes available, the greater the assurance that can be gained that the interfaces are correctly categorised/analysed. The requirements are structured such that, at the lowest level, the information required for SFR non-interfering interfaces is the minimum necessary in order for the evaluator to make this determination in an effective manner. At higher levels, more information becomes available so that the evaluator has greater confidence in the designation.

The purpose in defining these labels (SFR-enforcing, SFR-supporting, and SFR-non-interfering) and for levying different requirements upon each (at the lower assurance components) is to provide a first approximation of where to focus the analysis and the evidence upon which that analysis is performed. If the developer's documentation of the TSF interfaces describes all of the interfaces to the degree specified in the requirements for the SFR-enforcing interfaces (that is, if the documentation exceeds the requirements), there is no need for the developer to create new evidence to match the requirements. Similarly, because the labels are merely a means of differentiating the interface types within the requirements, there is no need for the developer to update the evidence solely to label the interfaces as SFR-enforcing, SFR-supporting, and SFR-non-interfering. The primary purpose of this labelling is to allow developers with less mature development methodologies (and associated artefacts, such as detailed interface and design documentation) to provide only the necessary evidence without undue cost.

The last C element of each component within this family provides a direct correspondence between the SFRs and the functional specification; that is, an indication of which interfaces are used to invoke each of the claimed SFRs. In the cases where the ST contains such functional requirements as 15408-2, whose functionality may not manifest itself at the TSFIs, the functional specification and/or the tracing is expected to identify these SFRs; including them in the functional specification helps to ensure that they are not lost at lower levels of decomposition, where they will be relevant.

10.3.3.1 Detail about the Interfaces

The requirements define collections of details about TSFI to be provided. For the purposes of the requirements, interfaces are specified (in varying degrees of detail) in terms of their purpose, method of use, parameters, parameter descriptions, and error messages.

The *purpose* of an interface is a high-level description of the general goal of the interface (e.g. process GUI commands, receive network packets, provide printer output, etc.).

The interface's *method of use* describes how the interface is supposed to be used. This description should be built around the various interactions available at that interface. For instance, if the interface were a Unix command shell, *ls*, *mv* and *cp* would be interactions for that interface. For each interaction the method of use describes what the interaction does, both for behaviour seen at the interface (e.g. the programmer calling the API, the Windows users changing a setting in the registry, etc.) as well as behaviour at other interfaces (e.g. generating an audit record).

Parameters are explicit inputs to and outputs from an interface that control the behaviour of that interface. For example, parameters are the arguments supplied to an API; the various fields in a packet for a given network protocol; the individual key values in the Windows Registry; the signals across a set of pins on a chip; the flags that can be set for the *ls*, etc. The parameters are “identified” with a simple list of what they are.

A *parameter description* tells what the parameter is in some meaningful way. For instance, an acceptable parameter description for interface *foo(i)* would be “parameter i is an integer that indicates the number of users currently logged in to the system”. A description such as “parameter i is an integer” is not an acceptable.

The description of an interface's *actions* describes what the interface does. This is more detailed than the purpose in that, while the “purpose” reveals why one might want to use it, the “actions” reveals everything that it does. These actions might be related to the SFRs or not. In cases where the interface's action is not related to SFRs, its description is said to be *summarised*, meaning the description merely makes clear that it is indeed not SFR-related.

The *error message description* identifies the condition that generated it, what the message is, and the meaning of any error codes. An error message is generated by the TSF to signify that a problem or irregularity of some degree has been encountered. The requirements in this family refer to different kinds of error messages:

- a “direct” error message is a security-relevant response through a specific TSFI invocation.
- an “indirect” error cannot be tied to a specific TSFI invocation because it results from system-wide conditions (e.g. resource exhaustion, connectivity interruptions, etc.). Error messages that are not security-relevant are also considered “indirect”.
- “remaining” errors are any other errors, such as those that might be referenced within the code. For example, the use of condition-checking code that checks for conditions that would not logically occur (e.g. a final “else” after a list of “case” statements), would provide for generating a catch-all error message; in an operational TOE, these error messages should never be seen.

An example functional specification is provided in A.2.4.

10.3.3.2 Components of this Family

Increasing assurance through increased completeness and accuracy in the interface specification is reflected in the documentation required from the developer as detailed in the various hierarchical components of this family.

At ADV_FSP.1 Basic functional specification, the only documentation required is a characterisation of all TSFIs and a high level description of SFR-enforcing and SFR-supporting TSFIs. To provide some assurance that the “important” aspects of the TSF have been correctly characterised at the TSFIs, the developer is required to provide the purpose and method of use, parameters for the SFR-enforcing and SFR-supporting TSFIs.

At ADV_FSP.2 Security-enforcing functional specification, the developer is required to provide the purpose, method of use, parameters, and parameter descriptions for all TSFIs. Additionally, for the SFR-enforcing TSFIs the developer has to describe the SFR-enforcing actions and direct error messages.

At ADV_FSP.3 Functional specification with complete summary, the developer must now, in addition to the information required at ADV_FSP.2, provide enough information about the SFR-supporting and SFR-non-interfering actions to show that they are not SFR-enforcing. Further, the developer must now document all of the direct error messages resulting from the invocation of SFR-enforcing TSFIs.

At ADV_FSP.4 Complete functional specification, all TSFIs - whether SFR-enforcing, SFR-supporting or SFR-non-interfering - must be described to the same degree, including all of the direct error messages.

At ADV_FSP.5 Complete semi-formal functional specification with additional error information, the TSFIs descriptions also include error messages that do not result from an invocation of a TSFI.

At ADV_FSP.6 Complete semi-formal functional specification with additional formal specification, in addition to the information required by ADV_FSP.5, all remaining error messages are included. The developer must also provide a formal description of the TSFI. This provides an alternative view of the TSFI that may expose inconsistencies or incomplete specification.

10.3.4 ADV_FSP.1 Basic functional specification

Dependencies: No dependencies.

10.3.4.1 Developer action elements

10.3.4.1.1 ADV_FSP.1.1D

The developer shall provide a functional specification.

10.3.4.1.2 ADV_FSP.1.2D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.4.2 Content and presentation elements

10.3.4.2.1 ADV_FSP.1.1C

The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supportingTSFI.

10.3.4.2.2 ADV_FSP.1.2C

The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.

10.3.4.2.3 ADV_FSP.1.3C

The functional specification shall provide rationale for the implicit categorisation of interfaces as SFR-non-interfering.

10.3.4.2.4 ADV_FSP.1.4C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.4.3 Evaluator action elements

10.3.4.3.1 ADV_FSP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.4.3.2 ADV_FSP.1.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.3.5 ADV_FSP.2 Security-enforcing functional specification

Dependencies: ADV_TDS.1 Basic design

10.3.5.1 Developer action elements

10.3.5.1.1 ADV_FSP.2.1D

The developer shall provide a functional specification.

10.3.5.1.2 ADV_FSP.2.2D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.5.2 Content and presentation elements

10.3.5.2.1 ADV_FSP.2.1C

The functional specification shall completely represent the TSF.

10.3.5.2.2 ADV_FSP.2.2C

The functional specification shall describe the purpose and method of use for **all** TSFI.

10.3.5.2.3 ADV_FSP.2.3C

The functional specification shall identify **and describe** all parameters associated with each TSFI.

10.3.5.2.4 ADV_FSP.2.4C

For each SFR-enforcing TSFI, the functional specification shall **describe the SFR-enforcing actions** associated with **the** TSFI.

10.3.5.2.5 ADV_FSP.2.5C

For each SFR-enforcing TSFI, the functional specification shall describe direct error messages resulting from processing associated with the SFR-enforcing actions.

10.3.5.2.6 ADV_FSP.2.6C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.5.3 Evaluator action elements

10.3.5.3.1 ADV_FSP.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.5.3.2 ADV_FSP.2.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.3.6 ADV_FSP.3 Functional specification with complete summary

Dependencies: ADV_TDS.1 Basic design

10.3.6.1 Developer action elements

10.3.6.1.1 ADV_FSP.3.1D

The developer shall provide a functional specification.

10.3.6.1.2 ADV_FSP.3.2D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.6.2 Content and presentation elements

10.3.6.2.1 ADV_FSP.3.1C

The functional specification shall completely represent the TSF.

10.3.6.2.2 ADV_FSP.3.2C

The functional specification shall describe the purpose and method of use for all TSFI.

10.3.6.2.3 ADV_FSP.3.3C

The functional specification shall identify and describe all parameters associated with each TSFI.

10.3.6.2.4 ADV_FSP.3.4C

For each SFR-enforcing TSFI, the functional specification shall describe the SFR-enforcing actions associated with the TSFI.

10.3.6.2.5 ADV_FSP.3.5C

For each SFR-enforcing TSFI, the functional specification shall describe direct error messages resulting from **SFR-enforcing actions and exceptions** associated with **invocation of** the **TSFI.**

10.3.6.2.6 ADV_FSP.3.6C

The functional specification shall summarise the SFR-supporting ans SFR-non-interfering actions associated with each TSFI.

10.3.6.2.7 ADV_FSP.3.7C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.6.3 Evaluator action elements

10.3.6.3.1 ADV_FSP.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.6.3.2 ADV_FSP.3.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.3.7 ADV_FSP.4 Complete functional specification

Dependencies: ADV_TDS.1 Basic design

10.3.7.1 Developer action elements

10.3.7.1.1 ADV_FSP.4.1D

The developer shall provide a functional specification.

10.3.7.1.2 ADV_FSP.4.2D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.7.2 Content and presentation elements

10.3.7.2.1 ADV_FSP.4.1C

The functional specification shall completely represent the TSF.

10.3.7.2.2 ADV_FSP.4.2C

The functional specification shall describe the purpose and method of use for all TSFI.

10.3.7.2.3 ADV_FSP.4.3C

The functional specification shall identify and describe all parameters associated with each TSFI.

10.3.7.2.4 ADV_FSP.4.4C

The functional specification shall **describe all** actions associated with each TSFI.

10.3.7.2.5 ADV_FSP.4.5C

The functional specification shall describe all direct error messages that may result from an invocation of each TSFI.

10.3.7.2.6 ADV_FSP.4.6C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.7.3 Evaluator action elements

10.3.7.3.1 ADV_FSP.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.7.3.2 ADV_FSP.4.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.3.8 ADV_FSP.5 Complete semi-formal functional specification with additional error information

Dependencies: ADV_TDS.1 Basic design

ADV_IMP.1 Implementation representation of the TSF

10.3.8.1 Developer action elements

10.3.8.1.1 ADV_FSP.5.1D

The developer shall provide a functional specification.

10.3.8.1.2 ADV_FSP.5.2D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.8.2 Content and presentation elements

10.3.8.2.1 ADV_FSP.5.1C

The functional specification shall completely represent the TSF.

10.3.8.2.2 ADV_FSP.5.2C

The functional specification shall describe the TSFI using a semi-formal style.

10.3.8.2.3 ADV_FSP.5.3C

The functional specification shall describe the purpose and method of use for all TSFI.

10.3.8.2.4 ADV_FSP.5.4C

The functional specification shall identify and describe all parameters associated with each TSFI.

10.3.8.2.5 ADV_FSP.5.5C

The functional specification shall describe all actions associated with each TSFI.

10.3.8.2.6 ADV_FSP.5.6C

The functional specification shall describe all direct error messages that may result from an invocation of each TSFI.

10.3.8.2.7 ADV_FSP.5.7C

The functional specification shall describe all error messages that do not result from an invocation of a TSFI.

10.3.8.2.8 ADV_FSP.5.8C

The functional specification shall provide a rationale for each error message contained in the TSF implementation yet does not result from an invocation of a TSFI.

10.3.8.2.9 ADV_FSP.5.9C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.8.3 Evaluator action elements

10.3.8.3.1 ADV_FSP.5.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.8.3.2 ADV_FSP.5.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.3.9 ADV_FSP.6 Complete semi-formal functional specification with additional formal specification

Dependencies:	ADV_TDS.1 Basic design
	ADV_IMP.1 Implementation representation of the TSF

10.3.9.1 Developer action elements

10.3.9.1.1 ADV_FSP.6.1D

The developer shall provide a functional specification.

10.3.9.1.2 ADV_FSP.6.2D

The developer shall provide a formal presentation of the functional specification of the TSF.

10.3.9.1.3 ADV_FSP.6.3D

The developer shall provide a tracing from the functional specification to the SFRs.

10.3.9.2 Content and presentation elements

10.3.9.2.1 ADV_FSP.6.1C

The functional specification shall completely represent the TSF.

10.3.9.2.2 ADV_FSP.6.2C

The functional specification shall describe the TSFI using a **formal** style.

10.3.9.2.3 ADV_FSP.6.3C

The functional specification shall describe the purpose and method of use for all TSFI.

10.3.9.2.4 ADV_FSP.6.4C

The functional specification shall identify and describe all parameters associated with each TSFI.

10.3.9.2.5 ADV_FSP.6.5C

The functional specification shall describe all actions associated with each TSFI.

10.3.9.2.6 ADV_FSP.6.6C

The functional specification shall describe all direct error messages that may result from an invocation of each TSFI.

10.3.9.2.7 ADV_FSP.6.7C

The functional specification shall describe all error messages **contained in the TSF implementation representation.**

10.3.9.2.8 ADV_FSP.6.8C

The functional specification shall provide a rationale for each error message contained in the TSF implementation **that is** not **otherwise described in the functional specification justifying why it is not associated with** a TSFI.

10.3.9.2.9 ADV_FSP.6.9C

The formal presentation of the functional specification of the TSF shall describe the TSFI using a formal style, supported by informal, explanatory text where appropriate.

10.3.9.2.10 ADV_FSP.6.10C

The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

10.3.9.3 Evaluator action elements

10.3.9.3.1 ADV_FSP.6.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.3.9.3.2 ADV_FSP.6.2E

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

10.4 Implementation representation (ADV_IMP)

10.4.1 Objectives

The function of the Implementation representation (ADV_IMP) family is for the developer to make available the implementation representation (and, at higher levels, the implementation itself) of the TOE in a form that can be analysed by the evaluator. The implementation representation is used in analysis activities for other families (analysing the TOE design, for instance) to demonstrate that the TOE conforms its design and to provide a basis for analysis in other areas of the evaluation (e.g., the search for vulnerabilities). The implementation representation is expected to be in a form that captures the detailed internal workings of the TSF. This may be software source code, firmware source code, hardware diagrams and/or IC hardware design language code or layout data.

10.4.2 Component levelling

The components in this family are levelled on the amount of implementation that is mapped to the TOE design description.

10.4.3 Application notes

Source code or hardware diagrams and/or IC hardware design language code or layout data that are used to build the actual hardware are examples of parts of an implementation representation. It is important to note that while the implementation representation must be made available to the evaluator, this does not imply that the evaluator needs to possess that representation. For instance, the developer may require that the evaluator review the implementation representation at a site of the developer's choosing.

The entire implementation representation is made available to ensure that analysis activities are not curtailed due to lack of information. This does not, however, imply that all of the representation is examined when the analysis activities are being performed. This is likely impractical in almost all cases, in addition to the fact that it most likely will not result in a higher-assurance TOE vs. targeted sampling of the implementation representation. The implementation representation is made available to allow analysis of other TOE design decompositions (e.g., functional specification, TOE design), and to gain confidence that the security functionality described at a higher level in the design actually appear to be implemented in the TOE. Conventions in some forms of the implementation representation may make it difficult or impossible to determine from just the implementation representation itself what the actual result of the compilation or run-time interpretation will be. For example, compiler directives for C language compilers will cause the compiler to exclude or include entire portions of the code. For this reason, it is important that such “extra” information or related tools (scripts, compilers, etc.) be provided so that the implementation representation can be accurately determined.

The purpose of the mapping between the implementation representation and the TOE design description is to aid the evaluator's analysis. The internal workings of the TOE may be better understood when the TOE design is analysed with corresponding portions of the implementation representation. The mapping serves as an index into the implementation representation. At the lower component, only a subset of the implementation representation is mapped to the TOE design description. Because of the uncertainty of which portions of the implementation representation will need such a mapping, the developer may choose either to map the entire implementation representation beforehand, or to wait to see which portions of the implementation representation the evaluator requires to be mapped.

The implementation representation is manipulated by the developer in a form that is suitable for transformation to the actual implementation. For instance, the developer may work with files containing source code, which is eventually compiled to become part of the TSF. The developer makes available the implementation representation in the form used by the developer, so that the evaluator may use automated techniques in the analysis. This also increases the confidence that the implementation representation examined is actually the one used in the production of the TSF (as opposed to the case where it is supplied in an alternate presentation format, such as a word processor document). It should be noted that other forms of the implementation representation may also be used by the developer; these forms are supplied as well. The overall goal is to supply the evaluator with the information that will maximise the effectiveness of the evaluator's analysis efforts.

Some forms of the implementation representation may require additional information because they introduce significant barriers to understanding and analysis. Examples include “shrouded” source code or source code that has been obfuscated in other ways such that it prevents understanding and/or analysis. These forms of implementation representation typically result from the TOE developer taking a version of the implementation representation and running a shrouding or obfuscation program on it. While the shrouded representation is what is compiled and may be closer to the implementation (in terms of structure) than the original, un-shrouded representation, supplying such obfuscated code may cause significantly more time to be spent in analysis tasks involving the representation. When such forms of representation are created, the components require details on the shrouding tools/algorithms used so that the un-shrouded representation can be supplied, and the additional information can be used to gain confidence that the shrouding process does not compromise any security functionality.

10.4.4 ADV_IMP.1 Implementation representation of the TSF

Dependencies: ADV_TDS.3 Basic modular design

ALC_TAT.1 Well-defined development tools

10.4.4.1 Developer action elements

10.4.4.1.1 ADV_IMP.1.1D

The developer shall make available the implementation representation for the entire TSF.

10.4.4.1.2 ADV_IMP.1.2D

The developer shall provide a mapping between the TOE design description and the sample of the implementation representation.

10.4.4.2 Content and presentation elements

10.4.4.2.1 ADV_IMP.1.1C

The implementation representation shall define the TSF to a level of detail such that the TSF can be generated without further design decisions.

10.4.4.2.2 ADV_IMP.1.2C

The implementation representation shall be in the form used by the development personnel.

10.4.4.2.3 ADV_IMP.1.3C

The mapping between the TOE design description and the sample of the implementation representation shall demonstrate their correspondence.

10.4.4.3 Evaluator action elements

10.4.4.3.1 ADV_IMP.1.1E

The evaluator shall confirm that, for the selected sample of the implementation representation, the information provided meets all requirements for content and presentation of evidence.

10.4.5 ADV_IMP.2 Complete mapping of the implementation representation of the TSF

Dependencies: ADV_TDS.3 Basic modular design

ALC_TAT.1 Well-defined development tools

ALC_CMC.5 Advanced support

10.4.5.1 Developer action elements

10.4.5.1.1 ADV_IMP.2.1D

The developer shall make available the implementation representation for the entire TSF.

10.4.5.1.2 ADV_IMP.2.2D

The developer shall provide a mapping between the TOE design description and the **entire** implementation representation.

10.4.5.2 Content and presentation elements

10.4.5.2.1 ADV_IMP.2.1C

The implementation representation shall define the TSF to a level of detail such that the TSF can be generated without further design decisions.

10.4.5.2.2 ADV_IMP.2.2C

The implementation representation shall be in the form used by the development personnel.

10.4.5.2.3 ADV_IMP.2.3C

The mapping between the TOE design description and the **entire** implementation representation shall demonstrate their correspondence.

10.4.5.3 Evaluator action elements

10.4.5.3.1 ADV_IMP.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.5 TSF internals (ADV_INT)

10.5.1 Objectives

This family addresses the assessment of the internal structure of the TSF. A TSF whose internals are well-structured is easier to implement and less likely to contain flaws that could lead to vulnerabilities; it is also easier to maintain without the introduction of flaws.

10.5.2 Component levelling

The components in this family are levelled on the basis of the amount of structure and minimisation of complexity required. ADV_INT.1 Well-structured subset of TSF internals places requirements for well-structured internals on only selected parts of the TSF. This component is not included in an EAL because this component is viewed for use in special circumstances (e.g., the sponsor has a specific concern regarding a cryptographic module, which is isolated from the rest of the TSF) and would not be widely applicable.

At the next level, the requirements for well-structured internals are placed on the entire TSF. Finally, minimisation of complexity is introduced in the highest component.

10.5.3 Application notes

These requirements, when applied to the internal structure of the TSF, typically result in improvements that aid both the developer and the evaluator in understanding the TSF, and also provide the basis for designing and evaluating test suites. Further, improving understandability of the TSF should assist the developer in simplifying its maintainability.

The requirements in this family are presented at a fairly abstract level. The wide variety of TOEs makes it impossible to codify anything more specific than “well-structured” or “minimum complexity”. Judgements on structure and complexity are expected to be derived from the specific technologies used in the TOE. For example, software is likely to be considered well-structured if it exhibits the characteristics cited in the software engineering disciplines. The components within this family call for identifying the standards for measuring the characteristic of being well-structured and not overly-complex.

10.5.4 ADV_INT.1 Well-structured subset of TSF internals

Dependencies: ADV_IMP.1 Implementation representation of the TSF

ADV_TDS.3 Basic modular design

ALC_TAT.1 Well-defined development tools

10.5.4.1 Objectives

The objective of this component is to provide a means for requiring specific portions of the TSF to be well-structured. The intent is that the entire TSF has been designed and implemented using sound engineering principles, but the analysis is performed upon only a specific subset.

10.5.4.2 Application notes

This component requires the PP or ST author to fill in an assignment with the subset of the TSF. This subset may be identified in terms of the internals of the TSF at any layer of abstraction. For example:

a) the structural elements of the TSF as identified in the TOE design (e.g. “The developer shall design and implement *the audit subsystem* such that it has well-structured internals.”)

b) the implementation (e.g. “The developer shall design and implement *the encrypt.c and decrypt.c files* such that it has well-structured internals.” or “The developer shall design and implement *the 6227 IC chip* such that it has well-structured internals.”)

It is likely this would not be readily accomplished by referencing the claimed SFRs (e.g. “The developer shall design and implement *the portion of the TSF that provide anonymity as defined in FPR_ANO.2* such that it has well-structured internals.”) because this does not indicate where to focus the analysis.

This component has limited value and would be suitable in cases where potentially-malicious users/subjects have limited or strictly controlled access to the TSFIs or where there is another means of protection (e.g., domain separation) that ensures the chosen subset of the TSF cannot be adversely affected by the rest of the TSF (e.g., the cryptographic functionality, which is isolated from the rest of the TSF, is well-structured).

10.5.4.3 Developer action elements

10.5.4.3.1 ADV_INT.1.1D

The developer shall design and implement [assignment: *subset of the TSF*] such that it has well-structured internals.

10.5.4.3.2 ADV_INT.1.2D

The developer shall provide an internals description and justification.

10.5.4.4 Content and presentation elements

10.5.4.4.1 ADV_INT.1.1C

The justification shall explain the characteristics used to judge the meaning of "well-structured".

10.5.4.4.2 ADV_INT.1.2C

The TSF internals description shall demonstrate that the assigned subset of the TSF is well-structured.

10.5.4.5 Evaluator action elements

10.5.4.5.1 ADV_INT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.5.4.5.2 ADV_INT.1.2E

The evaluator shall perform an internals analysis on the assigned subset of the TSF.

10.5.5 ADV_INT.2 Well-structured internals

Dependencies: ADV_IMP.1 Implementation representation of the TSF

ADV_TDS.3 Basic modular design

ALC_TAT.1 Well-defined development tools

10.5.5.1 Objectives

The objective of this component is to provide a means for requiring the TSF to be well-structured. The intent is that the entire TSF has been designed and implemented using sound engineering principles.

10.5.5.2 Application notes

Judgements on the adequacy of the structure are expected to be derived from the specific technologies used in the TOE. This component calls for identifying the standards for measuring the characteristic of being well-structured.

10.5.5.3 Developer action elements

10.5.5.3.1 ADV_INT.2.1D

The developer shall design and implement the **entire TSF** such that it has well-structured internals.

10.5.5.3.2 ADV_INT.2.2D

The developer shall provide an internals description and justification.

10.5.5.4 Content and presentation elements

10.5.5.4.1 ADV_INT.2.1C

The justification shall **describe** the characteristics used to judge the meaning of “well-structured”.

10.5.5.4.2 ADV_INT.2.2C

The TSF internals description shall demonstrate that the **entire** TSF is well-structured.

10.5.5.5 Evaluator action elements

10.5.5.5.1 ADV_INT.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.5.5.5.2 ADV_INT.2.2E

The evaluator shall perform an internals analysis on the TSF.

10.5.6 ADV_INT.3 Minimally complex internals

Dependencies:	ADV_IMP.1 Implementation representation of the TSF
	ADV_TDS.3 Basic modular design
	ALC_TAT.1 Well-defined development tools

10.5.6.1 Objectives

The objective of this component is to provide a means for requiring the TSF to be well-structured and of minimal complexity. The intent is that the entire TSF has been designed and implemented using sound engineering principles.

10.5.6.2 Application notes

Judgements on the adequacy of the structure and complexity are expected to be derived from the specific technologies used in the TOE. This component calls for identifying the standards for measuring the structure and complexity.

10.5.6.3 Developer action elements

10.5.6.3.1 ADV_INT.3.1D

The developer shall design and implement the entire TSF such that it has well-structured internals.

10.5.6.3.2 ADV_INT.3.2D

The developer shall provide an internals description and justification.

10.5.6.4 Content and presentation elements

10.5.6.4.1 ADV_INT.3.1C

The justification shall describe the characteristics used to judge the meaning of “well-structured” **and “complex”.**

10.5.6.4.2 ADV_INT.3.2C

The TSF internals description shall demonstrate that the entire TSF is well-structured **and is not overly complex.**

10.5.6.5 Evaluator action elements

10.5.6.5.1 ADV_INT.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.5.6.5.2 ADV_INT.3.2E

The evaluator shall perform an internals analysis on the **entire** TSF.

10.6 Security policy modelling (ADV_SPM)

10.6.1 Objectives

It is the objective of this family to provide additional assurance from the development of a formal *security policy model* of the TSFI behaviour of the TSF, and establishing a correspondence between the functional specification and this security policy model. Preserving internal consistency the security policy model is expected to formally establish the security principles from its characteristics by means of a mathematical proof.

10.6.2 Component levelling

This family contains only one component.

10.6.3 Application notes

Inadequacies in a TOE can result either from a failure in understanding the security requirements or from a flawed implementation of those security requirements. Defining the security requirements adequately to ensure their understanding may be problematic because the definition must be sufficiently precise to prevent undesired results or subtle flaws during implementation of the TOE. Throughout the design, implementation, and review processes, the modelled security requirements may and should be used as precise design and implementation guidance, thereby providing increased assurance that the modelled security requirements modelled via the TSFI behaviour are satisfied by the TOE. The precision of the model and resulting guidance is significantly improved by casting the model in a formal language and verifying the security requirements by automated formal proof techniques.

The creation of a formal security policy model helps to identify and eliminate ambiguous, inconsistent, incomplete, contradictory, or unenforceable security policy elements. Once the TOE has been built, the formal model serves the evaluation effort by contributing to the evaluator's judgement of how well the developer has understood the security functionality being implemented and whether there are inconsistencies between the security requirements and the TOE design. The confidence in the model is accompanied by a correspondence analysis for model elements and the functional specification, and a proof that the model contains no inconsistencies.

A formal security policy model is a precise formal presentation of the important aspects of security and their relationship to the behaviour of the TOE; it identifies the set of rules and practises that regulates how the TSF manages, protects, and otherwise controls the system resources. The model includes the set of restrictions and properties that specify how information and computing resources are prevented from being used to violate the SFRs, accompanied by a persuasive set of engineering arguments showing that these restrictions and properties play a key role in the enforcement of the SFRs. It consists both of the formalisms that express the security functionality, as well as ancillary text to explain the model and to provide it with context. The security behaviour of the TSF is modelled both in terms of external behaviour (i.e. how the TSF interacts with the rest of the TOE and with its operational environment), as well as its internal behaviour.

The security policy model of the TOE is informally abstracted from its realisation by considering the TSFI behaviour defined in the functional specification, which is strongly connected to the SFRs and security policies expressed in the ST. The purpose of formal methods lies within the enhancement of the rigour of enforcement. Informal arguments are always prone to fallacies; especially if relationships among subjects, objects and operations get more and more involved. In order to minimise the risk of insecure state reachability the rules and characteristics of the security policy model are mapped to respective properties and features within some formal system, whose rigour and strength can afterwards be used to obtain the security properties by means of theorems and formal proof.

While the term “formal security policy model” is used in academic circles, ISO/IEC 15408's approach has no fixed definition of “security”; it would equate to whatever SFRs are being claimed. Therefore, the formal security policy model is merely a formal representation of the set of SFRs being claimed by the TOE.

The term *security policy* has traditionally been associated with only access control policies, whether label-based (mandatory access control) or user-based (discretionary access control). However, a security policy is not limited to access control; there are also audit policies, identification policies, authentication policies, encryption policies, management policies, and any other security policies that are enforced by the TOE, as described in the PP/ST.

10.6.4 ADV_SPM.1 Formal TOE security policy model

Dependencies: ADV_FSP.5 Complete semi-formal functional specification with additional error information

ADV_FSP.6 Complete semi-formal functional specification with additional formal specification

10.6.4.1 Developer action elements

10.6.4.1.1 ADV_SPM.1.1D

The developer shall provide a formal security policy model for the TSFI behaviour of the TOE.

10.6.4.1.2 ADV_SPM.1.2D

The developer shall determine all TSFIs and analyze for each TSFI whether its behaviour can be modelled by the formal security policy model. If a TSFI cannot be modelled, for example caused by technical limitations, the developer shall analyze the impact of not modelling the TSFI behaviour on the security of the TOE. If parts of the TSFI behaviour cannot be modelled due to technical limitations, the remaining parts shall nevertheless be covered by the formal model.

10.6.4.1.3 ADV_SPM.1.3D

The formal security policy model shall identify the modelled TSFIs. For each TSFI covered by the formal security policy model, the model shall identify the related SFRs and security policies in the ST. For each SFR covered by the formal security policy model, the model shall identify the relevant portions of the statement of SFRs.

10.6.4.1.4 ADV_SPM.1.4D

For all TSFIs that are not modelled by the formal security policy model, the developer shall identify the affected SFRs and security policies in the ST.

10.6.4.1.5 ADV_SPM.1.5D

The developer shall provide a formal proof of correspondence between the model and any formal functional specification. The proof of correspondence shall relate model elements and TSFIs. With ADV_SPM.1.3D, the given proof of correspondence thereby implicitly provides a correspondence between model elements and SFRs, as well as model elements and security policies. The developer defines a structured process for identifying and presenting corresponding items formally.

10.6.4.1.6 ADV_SPM.1.6D

The developer shall provide a demonstration of correspondence between the model and the functional specification. This item shall demonstrate the correspondence between model elements and TSFIs.

10.6.4.2 Content and presentation elements

10.6.4.2.1 ADV_SPM.1.1C

The model shall define security for the TOE and provide a formal proof that the TOE cannot reach a state that is not secure.

10.6.4.2.2 ADV_SPM.1.2C

The developer shall provide an analysis why the chosen modelling formalism is appropriate.

10.6.4.2.3 ADV_SPM.1.3C

If tool support is used, the developer shall identify the tool chain used to verify the formal security policy model, including environments and version numbers. The developer shall provide arguments why the tool chain is suited and trustworthy.

10.6.4.2.4 ADV_SPM.1.4C

The developer shall define how the formal analysis of the formal security policy model can be reproduced (for example, applying an interactive theorem prover to prove correctness of the formal security policy model).

10.6.4.2.5 ADV_SPM.1.5C

The model shall be in a formal style, supported by explanatory text as required, and identify the TSFIs that are modelled. Additionally, the SFRs and security policies of the TSF that are modelled via the TSFI behaviour shall be presented. The model shall identify all TSFIs that are not modelled (compare ADV_SPM.1.2D) and present the affected SFRs and security policies. The model shall explain the reason for not modelling TSFIs and provide an impact analysis which shows that correctness of the formal model is not affected.

10.6.4.2.6 ADV_SPM.1.6C

The correspondence between the model and the functional specification shall be at the correct level of formality. The developer shall describe the correspondence analysis process and define the applied understanding of correspondence. If a semi-formal functional specification is provided, the correspondence must be shown semi-formally. If a formal functional specification is provided, the correspondence must be shown formally.

10.6.4.2.7 ADV_SPM.1.7C

The correspondence shall show that the model is consistent and complete with respect to the functional specification.

10.6.4.3 Evaluator action elements

10.6.4.3.1 ADV_SPM.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7 TOE design (ADV_TDS)

10.7.1 Objectives

The design description of a TOE provides both context for a description of the TSF, and a thorough description of the TSF. As assurance needs increase, the level of detail provided in the description also increases. As the size and complexity of the TSF increase, multiple levels of decomposition are appropriate. The design requirements are intended to provide information (commensurate with the given assurance level) so that a determination can be made that the security functional requirements are realised.

10.7.2 Component levelling

The components in this family are levelled on the basis of the amount of information that is required to be presented with respect to the TSF, and on the degree of formalism required of the design description.

10.7.3 Application notes

The goal of design documentation is to provide sufficient information to determine the TSF boundary, and to describe *how* the TSF implements the Security Functional Requirements. The amount and structure of the design documentation will depend on the complexity of the TOE and the number of SFRs; in general, a very complex TOE with a large number of SFRs will require more design documentation than a very simple TOE implementing only a few SFRs. Very complex TOEs will benefit (in terms of the assurance provided) from the production of differing levels of decomposition in describing the design, while very simple TOEs do not require both high-level and low-level descriptions of its implementation.

This family uses two levels of decomposition: the *subsystem* and the *module*. A module is the most specific description of functionality: it is a description of the implementation. A developer should be able to implement the part of the TOE described by the module with no further design decisions. A subsystem is a description of the design of the TOE; it helps to provide a high-level description of what a portion of the TOE is doing and how. As such, a subsystem may be further divided into lower-level subsystems, or into modules. Very complex TOEs might require several levels of subsystems in order to adequately convey a useful description of how the TOE works. Very simple TOEs, in contrast, might not require a subsystem level of description; the module might clearly describe how the TOE works.

The general approach adopted for design documentation is that, as the level of assurance increases, the emphasis of description shifts from the general (subsystem level) to more (module level) detail. In cases where a module-level of abstraction is appropriate because the TOE is simple enough to be described at the module level, yet the level of assurance calls for a subsystem level of description, the module-level description alone will suffice. For complex TOEs, however, this is not the case: an enormous amount of (module-level) detail would be incomprehensible without an accompanying subsystem level of description.

This approach follows the general paradigm that providing additional detail about the implementation of the TSF will result in greater assurance that the SFRs are implemented correctly, and provide information that can be used to demonstrate this in testing (ATE: Tests).

In the requirements for this family, the term *interface* is used as the means of communication (between two subsystems or modules). It describes how the communication is invoked; this is similar to the details of TSFI (see Functional specification (ADV_FSP)). The term *interaction* is used to identify the purpose for communication; it identifies why two subsystems or modules are communicating.

10.7.3.1 Detail about the Subsystems and Modules

The requirements define collections of details about subsystems and modules to be provided:

a) The subsystems and modules are *identified* with a simple list of what they are.

b) Subsystems and modules may be *categorised* (either implicitly or explicitly) as “SFR-enforcing”, “SFR-supporting”, or “SFR-non-interfering”; these terms are used the same as they are used in Functional specification (ADV_FSP).

c) A subsystem's *behaviour* is what it does. The behaviour may also be categorised as SFR-enforcing, SFR-supporting, or SFR-non-interfering. The behaviour of the subsystem is never categorised as more SFR-relevant than the category of the subsystem itself. For example, an SFR-enforcing subsystem can have SFR-enforcing behaviour as well as SFR-supporting or SFR-non-interfering behaviour.

d) A *behaviour summary* of a subsystem is an overview of the actions it performs (e.g. "The TCP subsystem assembles IP datagrams into reliable byte streams").

e) A *behaviour description* of a subsystem is an explanation of everything it does. This description should be at a level of detail that one can readily determine whether the behaviour has any relevance to the enforcement of the SFRs.

f) A *description of interactions* among or between subsystems or modules identifies the reason that subsystems or modules communicate, and characterises the information that is passed. It need not define the information to the same level of detail as an interface specification. For example, it would be sufficient to say "subsystem X requests a block of memory from the memory manager, which responds with the location of the allocated memory.

g) A *description of interfaces* provides the details of how the interactions among modules are achieved. Rather than describing the reason the modules are communicating or the purpose of their communication (that is, the description of interactions), the description of interfaces describes the details of how that communication is accomplished, in terms of the structure and contents of the messages, semaphores, internal process communications, etc.

h) The *purpose* describes how a module provides their functionality. It provides sufficient detail that no further design decisions are needed. The correspondence between the implementation representation that implements the module, and the purpose of the module should be readily apparent.

i) A module is otherwise *described* in terms of whatever is identified in the element.

Subsystems and modules, and "SFR-enforcing", etc. are all further explained in greater detail in A.4, ADV_TDS: Subsystems and Modules.

10.7.4 ADV_TDS.1 Basic design

Dependencies: ADV_FSP.2 Security-enforcing functional specification

10.7.4.1 Developer action elements

10.7.4.1.1 ADV_TDS.1.1D

The developer shall provide the design of the TOE.

10.7.4.1.2 ADV_TDS.1.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.4.2 Content and presentation elements

10.7.4.2.1 ADV_TDS.1.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.4.2.2 ADV_TDS.1.2C

The design shall identify all subsystems of the TSF.

10.7.4.2.3 ADV_TDS.1.3C

The design shall provide the behaviour summary of each SFR-supporting or SFR-non-interfering TSF subsystem.

10.7.4.2.4 ADV_TDS.1.4C

The design shall summarise the SFR-enforcing behaviour of the SFR-enforcing subsystems.

10.7.4.2.5 ADV_TDS.1.5C

The design shall provide a description of the interactions among SFR-enforcing subsystems of the TSF, and between the SFR-enforcing subsystems of the TSF and other subsystems of the TSF.

10.7.4.2.6 ADV_TDS.1.6C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.4.3 Evaluator action elements

10.7.4.3.1 ADV_TDS.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.4.3.2 ADV_TDS.1.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.7.5 ADV_TDS.2 Architectural design

Dependencies: ADV_FSP.3 Functional specification with complete summary

10.7.5.1 Developer action elements

10.7.5.1.1 ADV_TDS.2.1D

The developer shall provide the design of the TOE.

10.7.5.1.2 ADV_TDS.2.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.5.2 Content and presentation elements

10.7.5.2.1 ADV_TDS.2.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.5.2.2 ADV_TDS.2.2C

The design shall identify all subsystems of the TSF.

10.7.5.2.3 ADV_TDS.2.3C

The design shall provide the behaviour summery of each SFR non-interfering subsystem of the TSF.

10.7.5.2.4 ADV_TDS.2.4C

The design shall **describe** the SFR-enforcing behaviour of the SFR-enforcing subsystems.

10.7.5.2.5 ADV_TDS.2.5C

The design shall summarise the **SFR-supporting and SFR-non-interfering** behaviour of the SFR-enforcing subsystems.

10.7.5.2.6 ADV_TDS.2.6C

The design shall summarise the behaviour of the **SFR-supporting** subsystems.

10.7.5.2.7 ADV_TDS.2.7C

The design shall provide a description of the interactions among all subsystems of the TSF.

10.7.5.2.8 ADV_TDS.2.8C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.5.3 Evaluator action elements

10.7.5.3.1 ADV_TDS.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.5.3.2 ADV_TDS.2.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.7.6 ADV_TDS.3 Basic modular design

Dependencies: ADV_FSP.4 Complete functional specification

10.7.6.1 Developer action elements

10.7.6.1.1 ADV_TDS.3.1D

The developer shall provide the design of the TOE.

10.7.6.1.2 ADV_TDS.3.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.6.2 Content and presentation elements

10.7.6.2.1 ADV_TDS.3.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.6.2.2 ADV_TDS.3.2C

The design shall describe the TSF in terms of modules.

10.7.6.2.3 ADV_TDS.3.3C

The design shall identify all subsystems of the TSF.

10.7.6.2.4 ADV_TDS.3.4C

The design shall **provide a description** of **each subsystem of** the **TSF.**

10.7.6.2.5 ADV_TDS.3.5C

The design shall provide a description of the interactions among all subsystems of the TSF.

10.7.6.2.6 ADV_TDS.3.6C

The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

10.7.6.2.7 ADV_TDS.3.7C

The design shall describe each **SFR-enforcing module in terms** of **its purpose and relationship with other modules.**

10.7.6.2.8 ADV_TDS.3.8C

The design shall describe each SFR-enforcing module in terms of its SFR-related interfaces, return values from those interfaces, interaction with other modules and called SFR-related interfaces to other SFR-enforcing modules.

10.7.6.2.9 ADV_TDS.3.9C

The design shall describe each **SFR-supporting and SFR-non-interfering module in terms** of **its purpose and interaction with other modules.**

10.7.6.2.10 ADV_TDS.3.10C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.6.3 Evaluator action elements

10.7.6.3.1 ADV_TDS.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.6.3.2 ADV_TDS.3.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.7.7 ADV_TDS.4 Semiformal modular design

Dependencies: ADV_FSP.5 Complete semi-formal functional specification with additional error information

10.7.7.1 Developer action elements

10.7.7.1.1 ADV_TDS.4.1D

The developer shall provide the design of the TOE.

10.7.7.1.2 ADV_TDS.4.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.7.2 Content and presentation elements

10.7.7.2.1 ADV_TDS.4.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.7.2.2 ADV_TDS.4.2C

The design shall describe the TSF in terms of modules, **designating each module as SFR-enforcing, SFR-supporting, or SFR-non-interfering.**

10.7.7.2.3 ADV_TDS.4.3C

The design shall identify all subsystems of the TSF.

10.7.7.2.4 ADV_TDS.4.4C

The design shall provide a **semiformal** description of each subsystem of the TSF, **supported by informal, explanatory text where appropriate.**

10.7.7.2.5 ADV_TDS.4.5C

The design shall provide a description of the interactions among all subsystems of the TSF.

10.7.7.2.6 ADV_TDS.4.6C

The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

10.7.7.2.7 ADV_TDS.4.7C

The design shall describe each SFR-enforcing **and SFR-supporting** module in terms of its purpose and relationship with other modules.

10.7.7.2.8 ADV_TDS.4.8C

The design shall describe each SFR-enforcing **and SFR-supporting** module in terms of its SFR-related interfaces, return values from those interfaces, interaction with other modules and called SFR-related interfaces to other SFR-enforcing **or SFR-supporting** modules.

10.7.7.2.9 ADV_TDS.4.9C

The design shall describe each SFR-non-interfering module in terms of its purpose and interaction with other modules.

10.7.7.2.10 ADV_TDS.4.10C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.7.3 Evaluator action elements

10.7.7.3.1 ADV_TDS.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.7.3.2 ADV_TDS.4.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.7.8 ADV_TDS.5 Complete semiformal modular design

Dependencies: ADV_FSP.5 Complete semi-formal functional specification with additional error information

10.7.8.1 Developer action elements

10.7.8.1.1 ADV_TDS.5.1D

The developer shall provide the design of the TOE.

10.7.8.1.2 ADV_TDS.5.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.8.2 Content and presentation elements

10.7.8.2.1 ADV_TDS.5.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.8.2.2 ADV_TDS.5.2C

The design shall describe the TSF in terms of modules, designating each module as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

10.7.8.2.3 ADV_TDS.5.3C

The design shall identify all subsystems of the TSF.

10.7.8.2.4 ADV_TDS.5.4C

The design shall provide a semiformal description of each subsystem of the TSF, supported by informal, explanatory text where appropriate.

10.7.8.2.5 ADV_TDS.5.5C

The design shall provide a description of the interactions among all subsystems of the TSF.

10.7.8.2.6 ADV_TDS.5.6C

The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

10.7.8.2.7 ADV_TDS.5.7C

The design shall **provide a semiformal description of** each module in terms of its **purpose, interaction,** interfaces, return values from those interfaces, and called interfaces to other modules, **supported by informal, explanatory text where appropriate.**

10.7.8.2.8 ADV_TDS.5.8C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.8.3 Evaluator action elements

10.7.8.3.1 ADV_TDS.5.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.8.3.2 ADV_TDS.5.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.7.9 ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation

Dependencies: ADV_FSP.6 Complete semi-formal functional specification with additional formal specification

10.7.9.1 Developer action elements

10.7.9.1.1 ADV_TDS.6.1D

The developer shall provide the design of the TOE.

10.7.9.1.2 ADV_TDS.6.2D

The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.

10.7.9.1.3 ADV_TDS.6.3D

The developer shall provide a formal specification of the TSF subsystems.

10.7.9.1.4 ADV_TDS.6.4D

The developer shall provide a proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification.

10.7.9.2 Content and presentation elements

10.7.9.2.1 ADV_TDS.6.1C

The design shall describe the structure of the TOE in terms of subsystems.

10.7.9.2.2 ADV_TDS.6.2C

The design shall describe the TSF in terms of modules, designating each module as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

10.7.9.2.3 ADV_TDS.6.3C

The design shall identify all subsystems of the TSF.

10.7.9.2.4 ADV_TDS.6.4C

The design shall provide a semiformal description of each subsystem of the TSF, supported by informal, explanatory text where appropriate.

10.7.9.2.5 ADV_TDS.6.5C

The design shall provide a description of the interactions among all subsystems of the TSF.

10.7.9.2.6 ADV_TDS.6.6C

The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

10.7.9.2.7 ADV_TDS.6.7C

The design shall **describe** each module in **semiformal style in** terms of its purpose, interaction, interfaces, return values from those interfaces, and called interfaces to other modules, supported by informal, explanatory text where appropriate.

10.7.9.2.8 ADV_TDS.6.8C

The formal specification of the TSF subsystems shall describe the TSF using a formal style, supported by informal, explanatory text where appropriate.

10.7.9.2.9 ADV_TDS.6.9C

The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

10.7.9.2.10 ADV_TDS.6.10C

The proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification shall demonstrate that all behaviour described in the TOE design is a correct and complete refinement of the TSFI that invoked it.

10.7.9.3 Evaluator action elements

10.7.9.3.1 ADV_TDS.6.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

10.7.9.3.2 ADV_TDS.6.2E

The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

10.8 Composite design compliance (ADV_COMP)

10.8.1 Objectives

The aim of this activity is to determine whether the requirements on the application, imposed by the underlying platform, are fulfilled in the composite product.

10.8.2 Component levelling

This family contains only one component.

10.8.3 Application notes

The requirements on the application, imposed by the underlying platform, can be formulated in the relevant certification report (e.g. in form of constraints and recommendations), user guidance and ETR_COMP (in form of observations and recommendations) for the platform. The developer of the composite product shall regard each of these sources, if available (cf. Table D2, chapter Annex A), and implement the composite product in such a way that the applicable requirements are fulfilled.

The TSF of the composite product is represented at various levels of abstraction in the families of the development class ADV. Experiential, the appropriate levels of design representation for examining, whether the requirements of the platform are fulfilled by the composite product, are the TOE design (ADV_TDS), security architecture (ADV_ARC) and the implementation (ADV_IMP). In case, these design representation levels are not available (e.g. due to the assurance package chosen is EAL1), the current activity is not applicable (see the next paragraph for the reason).

Due to the definition of the composite TOE (cf. Clause 13.3.2.2 in ISO/IEC 15408-1) the interface between the underlying platform and the application is the internal one, hence, a functional specification (ADV_FSP) as representation level is not appropriate for analysing the design compliance.

Security architecture ADV_ARC as assurance family is dedicated to ensure that integrative security services like domain separation, self-protection and non-bypassability properly work. It is impossible and not the sense of the composite evaluation to have an insight into the architectural internals of the underlying platform (it is a matter of the platform evaluation). What the Composite Evaluator has to do in the context of ADV_ARC is

i. to determine whether the application uses services of the underlying platform within its own Composite-ST to provide domain separation, self-protection, non-bypassability and protected start-up; if no, there is no further composite activities for ADV_ARC; if yes, then

ii. the evaluator has to determine, whether the application uses these platform-services in an appropriate/secure way (please refer to the platform user guidance, cf. item #3 in Table D1, chapter Annex A).

Since consistency of the composite product security policy has already been considered in the context of the Security Target in the assurance family ASE_COMP, there is no necessity to consider non-contradictoriness of the security policy model (ADV_SPM) of the composite TOE and the security policy model of the underlying platform.

10.8.4 ADV_COMP.1 Design compliance with the platform certification report, guidance and ETR_COMP

Dependencies: No dependencies

10.8.4.1

10.8.4.2 Developer action elements

10.8.4.2.1 ADV_COMP.1.1D

The developer shall provide a design compliance justification; cf. item #6 as well as items #3, #4, #5 in Table D1, chapter Annex A.

10.8.4.3 Content and presentation elements

10.8.4.3.1 ADV_COMP.1.1C

The design compliance justification shall provide a rationale for design compliance – on an appropriate representation level – of how the requirements on the application, imposed by the underlying platform, are fulfilled in the composite product.

10.8.4.4 Evaluator action elements

10.8.4.4.1 ADV_COMP.1.1E

The evaluator shall confirm that the rationale for design compliance is complete, coherent, and internally consistent.

11 Class AGD: Guidance documents

11.1 Introduction

The guidance documents class provides the requirements for guidance documentation for all user roles. For the secure preparation and operation of the TOE it is necessary to describe all relevant aspects for the secure handling of the TOE. The class also addresses the possibility of unintended incorrect configuration or handling of the TOE.

In many cases it may be appropriate that guidance is provided in separate documents for preparation and operation of the TOE, or even separate for different user roles as end-users, administrators, application programmers using software or hardware interfaces, etc.

The guidance documents class is subdivided into two families which are concerned with the preparative user guidance (what has to be done to transform the delivered TOE into its evaluated configuration in the operational environment as described in the ST) and with the operational user guidance (what has to be done during the operation of the TOE in its evaluated configuration).

Figure 10 shows the families within this class, and the hierarchy of components within the families.

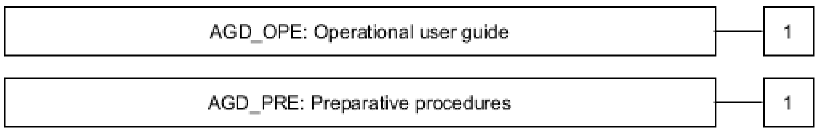
AGD_OPE: Operational user guide
1
AGD_PRE: Preparative procedures
1

Figure 10 — AGD: Guidance documents class decomposition

11.2 Operational user guidance (AGD_OPE)

11.2.1 Objectives

Operational user guidance refers to written material that is intended to be used by all types of users of the TOE in its evaluated configuration: end-users, persons responsible for maintaining and administering the TOE in a correct manner for maximum security, and by others (e.g. programmers) using the TOE's external interfaces. Operational user guidance describes the security functionality provided by the TSF, provides instructions and guidelines (including warnings), helps to understand the TSF and includes the security-critical information, and the security-critical actions required, for its secure use. Misleading and unreasonable guidance should be absent from the guidance documentation, and secure procedures for all modes of operation should be addressed. Insecure states should be easy to detect.

The operational user guidance provides a measure of confidence that non-malicious users, administrators, application providers and others exercising the external interfaces of the TOE will understand the secure operation of the TOE and will use it as intended. The evaluation of the user guidance includes investigating whether the TOE can be used in a manner that is insecure but that the user of the TOE would reasonably believe to be secure. The objective is to minimise the risk of human or other errors in operation that may deactivate, disable, or fail to activate security functionality, resulting in an undetected insecure state.

11.2.2 Component levelling

This family contains only one component.

11.2.3 Application notes

There may be different user roles or groups that are recognised by the TOE and that can interact with the TSF. These user roles and groups should be taken into consideration by the operational user guidance. They may be roughly grouped into administrators and non-administrative users, or more specifically grouped into persons responsible for receiving, accepting, installing and maintaining the TOE, application programmers, revisors, auditors, daily-management, end-users. Each role can encompass an extensive set of capabilities, or can be a single one.

The requirement AGD_OPE.1.1C encompasses the aspect that any warnings to the users during operation of a TOE with regard to the security problem definition and the security objectives for the operational environment described in the PP/ST are appropriately covered in the user guidance.

The concept of secure values, as employed in AGD_OPE.1.3C, has relevance where a user has control over security parameters. Guidance needs to be provided on secure and insecure settings for such parameters.

AGD_OPE.1.4C requires that the user guidance describes the appropriate reactions to all security-relevant events. Although many security-relevant events are the result of performing functions, this need not always be the case (e.g. the audit log fills up, an intrusion is detected). Furthermore, a security-relevant event may happen as a result of a specific chain of functions or, conversely, several security-relevant events may be triggered by one function.

AGD_OPE.1.7C requires that the user guidance is clear and reasonable. Misleading or unreasonable guidance may result in a user of the TOE believing that the TOE is secure when it is not.

An example of misleading guidance would be the description of a single guidance instruction that could be parsed in more than one way, one of which may result in an insecure state.

An example of unreasonable guidance would be a recommendation to follow a procedure that is so complicated that it cannot reasonably be expected that users will follow this guidance.

11.2.4 AGD_OPE.1 Operational user guidance

Dependencies: ADV_FSP.1 Basic functional specification

11.2.4.1 Developer action elements

11.2.4.1.1 AGD_OPE.1.1D

The developer shall provide operational user guidance.

11.2.4.2 Content and presentation elements

11.2.4.2.1 AGD_OPE.1.1C

The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.

11.2.4.2.2 AGD_OPE.1.2C

The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the TOE in a secure manner.

11.2.4.2.3 AGD_OPE.1.3C

The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate.

11.2.4.2.4 AGD_OPE.1.4C

The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the TSF.

11.2.4.2.5 AGD_OPE.1.5C

The operational user guidance shall identify all possible modes of operation of the TOE (including operation following failure or operational error), their consequences and implications for maintaining secure operation.

11.2.4.2.6 AGD_OPE.1.6C

The operational user guidance shall, for each user role, describe the security controls to be followed in order to fulfil the security objectives for the operational environment as described in the ST.

11.2.4.2.7 AGD_OPE.1.7C

The operational user guidance shall be clear and reasonable.

11.2.4.3 Evaluator action elements

11.2.4.3.1 AGD_OPE.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

11.3 Preparative procedures (AGD_PRE)

11.3.1 Objectives

Preparative procedures are useful for ensuring that the TOE has been received and installed in a secure manner as intended by the developer. The requirements for preparation call for a secure transition from the delivered TOE to its initial operational environment. This includes investigating whether the TOE can be configured or installed in a manner that is insecure but that the user of the TOE would reasonably believe to be secure.

11.3.2 Component levelling

This family contains only one component.

11.3.3 Application notes

It is recognised that the application of these requirements will vary depending on aspects such as whether the TOE is delivered in an operational state, or whether it has to be installed at the TOE owner's site, etc.

The first process covered by the preparative procedures is the consumer's secure acceptance of the received TOE in accordance with the developer's delivery procedures. If the developer has not defined delivery procedures, security of the acceptance has to be ensured otherwise.

Installation of the TOE includes transforming its operational environment into a state that conforms to the security objectives for the operational environment provided in the ST.

It might also be the case that no installation is necessary, for example a smart card. In this case it may be inappropriate to require and analyse installation procedures.

The requirements in this assurance family are presented separately from those in the Operational user guidance (AGD_OPE) family, due to the infrequent, possibly one-time use of the preparative procedures.

11.3.4 AGD_PRE.1 Preparative procedures

Dependencies: No dependencies.

11.3.4.1 Developer action elements

11.3.4.1.1 AGD_PRE.1.1D

The developer shall provide the TOE including its preparative procedures.

11.3.4.2 Content and presentation elements

11.3.4.2.1 AGD_PRE.1.1C

The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered TOE in accordance with the developer's delivery procedures.

11.3.4.2.2 AGD_PRE.1.2C

The preparative procedures shall describe all the steps necessary for secure installation of the TOE and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the ST.

11.3.4.3 Evaluator action elements

11.3.4.3.1 AGD_PRE.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

11.3.4.3.2 AGD_PRE.1.2E

The evaluator shall apply the preparative procedures to confirm that the TOE can be prepared securely for operation.

12 Class ALC: Life-cycle support

12.1 Introduction

Life-cycle support is an aspect of establishing appropriate security controls in the development, production, delivery and maintenance of the TOE. Confidence in the correspondence between the TOE security requirements and the TOE is greater if security analysis and the production of the evidence are done on a regular basis as an integral part of the development, production, delivery and maintenance activities.

During the life-cycle of the TOE it is distinguished whether the TOE is under the responsibility of the TOE developer or the user rather than whether it is located in the development or the user environment. The point of transition is when the TOE is accepted by the user. User in this context relates to the end-user as well as product- and system integrators.

The ALC class consists of nine families:

- Development Life-cycle definition (ALC_LCD) provides requirements for the developer’s description of the life-cycle model used in the development, production, delivery and maintenance life-cycle of the TOE;

- CM capabilities (ALC_CMC) provides requirements for the management of the configuration items;

- CM scope (ALC_CMS) requires a minimum set of configuration items to be managed in the defined way;
- Developer environment security (ALC_DVS) is concerned with the developer's physical, logical, procedural, personnel, and other security controls;
- Tools and techniques (ALC_TAT) provides requirements for the development tools and implementation standards used by the developer;
- Flaw remediation (ALC_FLR) provides requirements for the handling of security flaws.
- Delivery (ALC_DEL) provides requirements for the procedures used for the delivery of the TOE to the downstream user. Delivery processes occurring during the development of the TOE are denoted rather as transfers, and are handled in the context of integration and acceptance procedures in other families of this class.
- ALC_TDA
- ALC_COMP

Throughout this class, development and related terms (developer, develop) are meant in the more general sense to comprise development and production, whereas production specifically means the process of transforming the implementation representation into the final TOE.

Figure 11 shows the families within this class, and the hierarchy of components within the families.

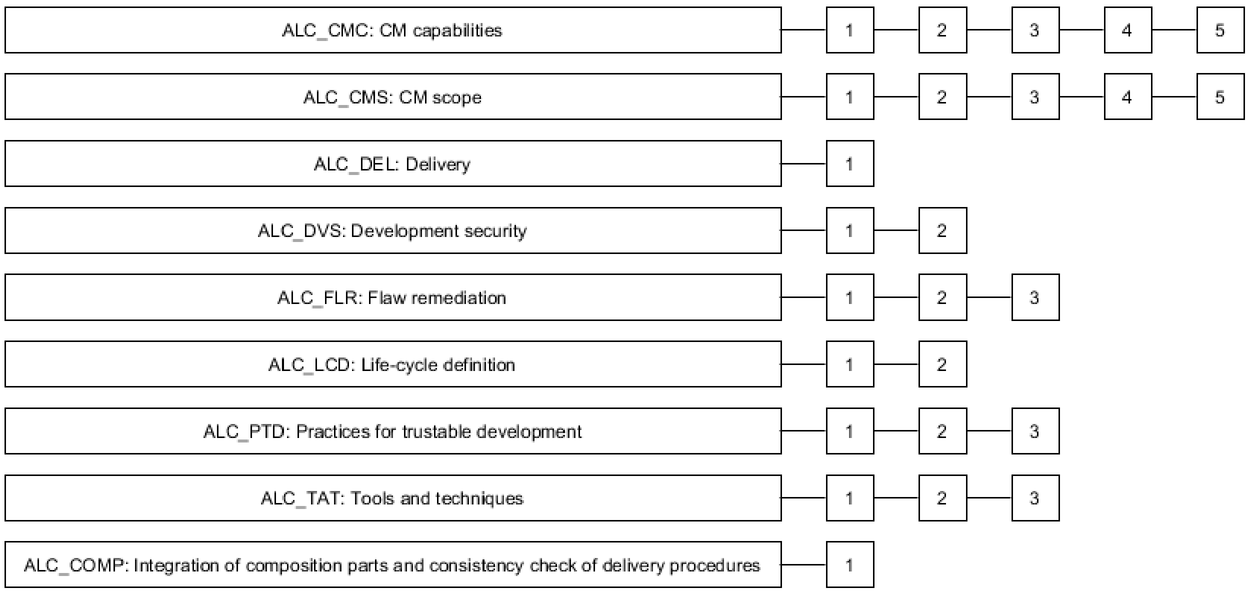
ALC_CMC: CM capabilities
1
2
3
4
5
ALC_CMS: CM scope
1
2
3
4
5
ALC_DEL: Delivery
1
ALC_DVS: Development security
1
2
ALC_FLR: Flaw remediation
1
2
3
ALC_LCD: Life-cycle definition
1
2
ALC_PTD: Practices for trustable development
1
2
3
ALC_TAT: Tools and techniques
1
2
3
ALC_COMP: Integration of composition parts and consistency check of delivery procedures
1

Figure 11 — ALC: Life-cycle support class decomposition

12.2 CM capabilities (ALC_CMC)

12.2.1 Objectives

Configuration management (CM) techniques, properly defined as part of the development life-cycle model, contribute to the assurance argument that the TOE meets the SFRs. A Configuration

Management (CM) system that is managed and operated correctly will help ensure the integrity of the portions of the TOE that are controlled, by providing a method of tracking any changes to the TOE, and to help ensure that all changes to the TOE are authorised.

The objective of this family is to require the TOE developer's CM system to have certain capabilities. These capabilities are intented to reduce the likelihood that accidental or unauthorised modifications of the configuration items will occur. The CM system should support maintaining the integrity of the TOE throughout the part of the TOE’s life-cycle that is under the control of the developer.

The objective of introducing automated CM tools is to increase the effectiveness of the CM system. While both automated and manual CM systems can be bypassed, ignored, or proven insufficient to prevent unauthorised modification, automated systems are less susceptible to human error or negligence.

The objectives of this family include the following:

a) ensuring that the TOE is identifiable and complete before it is sent to the downstream user;

b) ensuring that no configuration items are missed during evaluation;

c) preventing unauthorised modification, addition, or deletion of TOE configuration items.

12.2.2 Component levelling

The components in this family are levelled on the basis of the CM system capabilities, the scope of the CM documentation and the evidence provided by the developer.

12.2.3 Application notes

In the case where the TOE is a subset of a product, the requirements of this family apply only to the TOE configuration items, not to the product as a whole.

For developer organizations that specify more than one CM application, or include different instances of a CM application within the scope of the TOEs design, development, production and maintenance, it is required to document all of them. For evaluation purposes, the set of CM applications should be regarded as parts of an overall CM system, applicable to the TOE, which is addressed in the criteria.

The overall CM system should address any aspects of integration between component CM applications.

Several elements of this family refer to configuration items. These elements identify CM requirements to be imposed on all items identified in the configuration list, but leave the contents of the list to the discretion of the developer. CM scope (ALC_CMS) can be used to narrow this discretion by identifying specific items that must be included in the configuration list, and hence within the scope of the overall CM system.

ALC_CMC.2.3C introduces a requirement that the CM system uniquely identify all configuration items. This also requires that modifications to configuration items result in a new, unique identifier being assigned to the configuration item.

ALC_CMC.3.8C introduces the requirement that the evidence shall demonstrate that the CM system operates in accordance with the CM plan. Examples of such evidence might be documentation such as screen snapshots or audit trail output from the CM system, or a detailed demonstration of the CM system by the developer. The evaluator is responsible for determining that this evidence is sufficient to show that the CM system operates in accordance with the CM plan.

ALC_CMC.4.5C introduces a requirement that the CM system provide an automated means to support the production of the TOE. This requires that the CM system provide an automated means to assist in determining that the correct configuration items are used in generating the TOE.

ALC_CMC.5.10C introduces a requirement that the CM system provide an automated means to ascertain the changes between the TOE and its preceding version. If no previous version of the TOE exists, the developer still needs to provide an automated means to ascertain the changes between the TOE and a future version of the TOE.

12.2.4 ALC_CMC.1 Labelling of the TOE

Dependencies: ALC_CMS.1 TOE CM coverage

12.2.4.1 Objectives

A unique reference is required to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated. Labelling the TOE with its reference ensures that users of the TOE can be aware of which instance of the TOE they are using.

12.2.4.2 Developer action elements

12.2.4.2.1 ALC_CMC.1.1D

The developer shall provide the TOE and a unique reference for the TOE.

12.2.4.3 Content and presentation elements

12.2.4.3.1 ALC_CMC.1.1C

The TOE shall be labelled with its unique reference.

12.2.4.4 Evaluator action elements

12.2.4.4.1 ALC_CMC.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.2.5 ALC_CMC.2 Use of the CM system

Dependencies: ALC_CMS.1 TOE CM coverage

12.2.5.1 Objectives

A unique reference is required to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated. Labelling the TOE with its reference ensures that users of the TOE can be aware of which instance of the TOE they are using.

Unique identification of the configuration items leads to a clearer understanding of the composition of the TOE, which in turn helps to determine those items which are subject to the evaluation requirements for the TOE.

The use of a CM system increases assurance that the configuration items are maintained in a controlled manner.

12.2.5.2 Developer action elements

12.2.5.2.1 ALC_CMC.2.1D

The developer shall provide the TOE and a unique reference for the TOE.

12.2.5.2.2 ALC_CMC.2.2D

The developer shall provide the CM documentation.

12.2.5.2.3 ALC_CMC.2.3D

The developer shall use a CM system.

12.2.5.3 Content and presentation elements

12.2.5.3.1 ALC_CMC.2.1C

The TOE shall be labelled with its unique reference.

12.2.5.3.2 ALC_CMC.2.2C

The CM documentation shall describe the method used to uniquely identify the configuration items.

12.2.5.3.3 ALC_CMC.2.3C

The CM system shall uniquely identify all configuration items.

12.2.5.4 Evaluator action elements

12.2.5.4.1 ALC_CMC.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.2.6 ALC_CMC.3 Authorisation controls

Dependencies: ALC_CMS.1 TOE CM coverage

ALC_DVS.1 Identification of security measures

A life-cycle model encompasses the procedures, tools and techniques used to develop and maintain the TOE. Aspects of the process that may be covered by such a model include design methods, review procedures, project management controls, change control procedures, test methods and acceptance procedures. An effective life-cycle model will address these aspects of the development and maintenance process within an overall management structure that assigns responsibilities and monitors progress.

There are different types of acceptance situations that are dealt with at different locations in the criteria:

- acceptance of parts delivered by subcontractors ("integration") should be treated in this family
- Development Life-cycle definition (ALC_LCD),
- acceptance subsequent to internal transportations in Developer environment security (ALC_DVS),
- acceptance of parts into the CM system in CM capabilities (ALC_CMC), and
- acceptance of the delivered TOE by the consumer in Delivery (ALC_DEL).

The first three types may overlap.

Although life-cycle definition deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, its evaluation adds assurance through an analysis of the life-cycle information for the TOE provided at the time of the evaluation.

A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the model enables sufficient minimisation of the danger that the TOE will not meet its security requirement.

A measurable life-cycle model is a model using some quantitative valuation (arithmetic parameters and/or metrics) of the managed product in order to measure development properties of the product. Typical metrics are source code complexity metrics, defect density (errors per size of code) or mean time to failure. For the security evaluation all those metrics are of relevance, which are used to increase quality by decreasing the probability of faults and thereby in turn increasing assurance in the security of the TOE.

One should take into account that there exist standardised life-cycle models on the one hand (like the waterfall model) and standardised metrics on the other hand (like error density), which may be combined. ISO/IEC 15408 does not require the life-cycle to follow exactly one standard defining both aspects.

ALC_LCD.1 Developer defined life-cycle processes

12.2.6.1 Objectives

A unique reference is required to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated. Labelling the TOE with its reference ensures that users of the TOE can be aware of which instance of the TOE they are using.

Unique identification of the configuration items leads to a clearer understanding of the composition of the TOE, which in turn helps to determine those items which are subject to the evaluation requirements for the TOE.

The use of a CM system increases assurance that the configuration items are maintained in a controlled manner.

Providing controls to ensure that unauthorised modifications are not made to the TOE ("CM access control"), and ensuring proper functionality and use of the CM system, helps to maintain the integrity of the TOE.

12.2.6.2 Developer action elements

12.2.6.2.1 ALC_CMC.3.1D

The developer shall provide the TOE and a unique reference for the TOE.

12.2.6.2.2 ALC_CMC.3.2D

The developer shall provide the CM documentation.

12.2.6.2.3 ALC_CMC.3.3D

The developer shall use a CM system.

12.2.6.3 Content and presentation elements

12.2.6.3.1 ALC_CMC.3.1C

The TOE shall be labelled with its unique reference.

12.2.6.3.2 ALC_CMC.3.2C

The CM documentation shall describe the method used to uniquely identify the configuration items.

12.2.6.3.3 ALC_CMC.3.3C

The CM system shall uniquely identify all configuration items.

12.2.6.3.4 ALC_CMC.3.4C

The CM system shall provide controls such that only authorised changes are made to the configuration items.

12.2.6.3.5 ALC_CMC.3.5C

The CM documentation shall include a CM plan.

12.2.6.3.6 ALC_CMC.3.6C

The CM plan shall describe how the CM system is used for the development of the TOE.

12.2.6.3.7 ALC_CMC.3.7C

The evidence shall demonstrate that all configuration items are being maintained under the CM system.

12.2.6.3.8 ALC_CMC.3.8C

The evidence shall demonstrate that the CM system is being operated in accordance with the CM plan.

12.2.6.4 Evaluator action elements

12.2.6.4.1 ALC_CMC.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.2.7 ALC_CMC.4 Production support, acceptance procedures and automation

Dependencies: ALC_CMS.1 TOE CM coverage

ALC_DVS.1 Identification of security measures

A life-cycle model encompasses the procedures, tools and techniques used to develop and maintain the TOE. Aspects of the process that may be covered by such a model include design methods, review procedures, project management controls, change control procedures, test methods and acceptance procedures. An effective life-cycle model will address these aspects of the development and maintenance process within an overall management structure that assigns responsibilities and monitors progress.

There are different types of acceptance situations that are dealt with at different locations in the criteria:

- acceptance of parts delivered by subcontractors (“integration”) should be treated in this family
- Development Life-cycle definition (ALC_LCD),
- acceptance subsequent to internal transportations in Developer environment security (ALC_DVS),
- acceptance of parts into the CM system in CM capabilities (ALC_CMC), and
- acceptance of the delivered TOE by the consumer in Delivery (ALC_DEL).

The first three types may overlap.

Although life-cycle definition deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, its evaluation adds assurance through an analysis of the life-cycle information for the TOE provided at the time of the evaluation.

A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the model enables sufficient minimisation of the danger that the TOE will not meet its security requirement.

A measurable life-cycle model is a model using some quantitative valuation (arithmetic parameters and/or metrics) of the managed product in order to measure development properties of the product. Typical metrics are source code complexity metrics, defect density (errors per size of code) or mean time to failure. For the security evaluation all those metrics are of relevance, which are used to increase quality by decreasing the probability of faults and thereby in turn increasing assurance in the security of the TOE.

One should take into account that there exist standardised life-cycle models on the one hand (like the waterfall model) and standardised metrics on the other hand (like error density), which may be combined. ISO/IEC 15408 does not require the life-cycle to follow exactly one standard defining both aspects.

ALC_LCD.1 Developer defined life-cycle processes

12.2.7.1 Objectives

A unique reference is required to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated. Labelling the TOE with its reference ensures that users of the TOE can be aware of which instance of the TOE they are using.

Unique identification of the configuration items leads to a clearer understanding of the composition of the TOE, which in turn helps to determine those items which are subject to the evaluation requirements for the TOE.

The use of a CM system increases assurance that the configuration items are maintained in a controlled manner.

Providing access controls to help ensure that unauthorised modifications are not made to the TOE ("CM access control"), and ensuring proper functionality and use of the CM system, helps to maintain the integrity of the TOE.

The purpose of the acceptance procedures is to ensure that the parts of the TOE are of adequate quality and to confirm that any creation or modification of configuration items is authorised. Acceptance procedures are an essential element in integration processes and in the life-cycle management of the TOE.

In a CM system where the quantity and organization of configuration items is complex, it is difficult to control changes without the support of automated tools. In particular, these automated tools need to be able to support the numerous changes that occur during development and ensure that those changes are authorised. It is an objective of this component to ensure that the configuration items are controlled through automated means. In the case where the overall CM system includes more than one CM application then automated tools can also support integration between the CM applications and of the TOE.

Production support procedures help to ensure that the generation of the TOE from a managed set of configuration items is correctly performed in an authorised manner, particularly in the case when different developers are involved and integration processes have to be carried out.

12.2.7.2 Developer action elements

12.2.7.2.1 ALC_CMC.4.1D

The developer shall provide the TOE and a unique reference for the TOE.

12.2.7.2.2 ALC_CMC.4.2D

The developer shall provide the CM documentation.

12.2.7.2.3 ALC_CMC.4.3D

The developer shall use a CM system.

12.2.7.3 Content and presentation elements

12.2.7.3.1 ALC_CMC.4.1C

The TOE shall be labelled with its unique reference.

12.2.7.3.2 ALC_CMC.4.2C

The CM documentation shall describe the method or methods used to uniquely identify the configuration items.

12.2.7.3.3 ALC_CMC.4.3C

The CM system shall uniquely identify all configuration items.

12.2.7.3.4 ALC_CMC.4.4C

The CM system shall provide **automated** controls such that only authorised changes are made to the configuration items.

12.2.7.3.5 ALC_CMC.4.5C

The CM system shall support the production of the TOE by automated means.

12.2.7.3.6 ALC_CMC.4.6C

The CM documentation shall include a CM plan.

12.2.7.3.7 ALC_CMC.4.7C

The CM plan shall describe how the CM system is used for the development of the TOE.

12.2.7.3.8 ALC_CMC.4.8C

The CM plan shall describe the procedures used to accept modified or newly created configuration items as part of the TOE.

12.2.7.3.9 ALC_CMC.4.9C

The evidence shall demonstrate that all configuration items are being maintained under the CM system.

12.2.7.3.10 ALC_CMC.4.10C

The evidence shall demonstrate that the CM system is being operated in accordance with the CM plan.

12.2.7.4 Evaluator action elements

12.2.7.4.1 ALC_CMC.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.2.8 ALC_CMC.5 Advanced support

Dependencies: ALC_CMS.1 TOE CM coverage

ALC_DVS.2 Sufficiency of security measures

A life-cycle model encompasses the procedures, tools and techniques used to develop and maintain the TOE. Aspects of the process that may be covered by such a model include design methods, review procedures, project management controls, change control procedures, test methods and acceptance procedures. An effective life-cycle model will address these aspects of the development and maintenance process within an overall management structure that assigns responsibilities and monitors progress.

There are different types of acceptance situations that are dealt with at different locations in the criteria:

- acceptance of parts delivered by subcontractors ("integration") should be treated in this family
- Development Life-cycle definition (ALC_LCD),
- acceptance subsequent to internal transportations in Developer environment security (ALC_DVS),
- acceptance of parts into the CM system in CM capabilities (ALC_CMC), and
- acceptance of the delivered TOE by the consumer in Delivery (ALC_DEL).

The first three types may overlap.

Although life-cycle definition deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, its evaluation adds assurance through an analysis of the life-cycle information for the TOE provided at the time of the evaluation.

A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the model enables sufficient minimisation of the danger that the TOE will not meet its security requirement.

A measurable life-cycle model is a model using some quantitative valuation (arithmetic parameters and/or metrics) of the managed product in order to measure development properties of the product. Typical metrics are source code complexity metrics, defect density (errors per size of code) or mean time to failure. For the security evaluation all those metrics are of relevance, which are used to increase quality by decreasing the probability of faults and thereby in turn increasing assurance in the security of the TOE.

One should take into account that there exist standardised life-cycle models on the one hand (like the waterfall model) and standardised metrics on the other hand (like error density), which may be combined. ISO/IEC 15408 does not require the life-cycle to follow exactly one standard defining both aspects.

ALC_LCD.1 Developer defined life-cycle processes

12.2.8.1 Objectives

A unique reference is required to ensure that there is no ambiguity in terms of which instance of the TOE is being evaluated. Labelling the TOE with its reference ensures that users of the TOE can be aware of which instance of the TOE they are using.

Unique identification of the configuration items leads to a clearer understanding of the composition of the TOE, which in turn helps to determine those items which are subject to the evaluation requirements for the TOE.

The use of a CM system increases assurance that the configuration items are maintained in a controlled manner.

Providing controls to ensure that unauthorised modifications are not made to the TOE ("CM access control"), and ensuring proper functionality and use of the CM system, helps to maintain the integrity of the TOE.

The purpose of the acceptance procedures is to ensure that the parts of the TOE meet defined criteria in regard to the integrity of the TOE. Criteria for acceptance procedures may include code review, checking for vulnerabilities, authenticity checking, and functional testing to confirm that any creation or modification of configuration items is authorised. Acceptance procedures are an essential element in integration processes and in the life-cycle management of the TOE.

In development environments where the configuration items are complex, it is difficult to control changes without the support of automated tools. In particular, these automated tools need to be able to support the numerous changes that occur during development and ensure that those changes are authorised. It is an objective of this component to ensure that the configuration items are controlled through automated means. If the TOE is developed by multiple developers, i.e. integration has to take place, the use of automatic tools is adequate.

Production support procedures help to ensure that the generation of the TOE from a managed set of configuration items is correctly performed in an authorised manner, particularly in the case when different developers are involved and integration processes have to be carried out.

Requiring that the CM system be able to identify the version of the implementation representation from which the TOE is generated helps to ensure that the integrity of this material is preserved by the appropriate technical, physical and procedural safeguards.

Providing an automated means of ascertaining changes between versions of the TOE and identifying which configuration items are affected by modifications to other configuration items assists in determining the impact of the changes between successive versions of the TOE. This in turn can provide valuable information in determining whether changes to the TOE result in all configuration items being consistent with one another.

12.2.8.2 Developer action elements

12.2.8.2.1 ALC_CMC.5.1D

The developer shall provide the TOE and a unique reference for the TOE.

12.2.8.2.2 ALC_CMC.5.2D

The developer shall provide the CM documentation.

12.2.8.2.3 ALC_CMC.5.3D

The developer shall use a CM system.

12.2.8.3 Content and presentation elements

12.2.8.3.1 ALC_CMC.5.1C

The TOE shall be labelled with its unique reference.

12.2.8.3.2 ALC_CMC.5.2C

The CM documentation shall describe the method used to uniquely identify the configuration items.

12.2.8.3.3 ALC_CMC.5.3C

The CM documentation shall justify that the acceptance procedures provide for an adequate and appropriate review of changes to all configuration items.

12.2.8.3.4 ALC_CMC.5.4C

The CM system shall uniquely identify all configuration items.

12.2.8.3.5 ALC_CMC.5.5C

The CM system shall provide automated controls such that only authorised changes are made to the configuration items.

12.2.8.3.6 ALC_CMC.5.6C

The CM system shall support the production of the TOE by automated means.

12.2.8.3.7 ALC_CMC.5.7C

The CM system shall ensure that the person responsible for accepting a configuration item into CM is not the person who developed it.

12.2.8.3.8 ALC_CMC.5.8C

The CM system shall identify the configuration items that comprise the TSF.

12.2.8.3.9 ALC_CMC.5.9C

The CM system shall support the audit of all changes to the TOE by automated means, including the originator, date, and time in the audit trail.

12.2.8.3.10 ALC_CMC.5.10C

The CM system shall provide an automated means to identify all other configuration items that are affected by the change of a given configuration item.

12.2.8.3.11 ALC_CMC.5.11C

The CM system shall be able to identify the version of the implementation representation from which the TOE is generated.

12.2.8.3.12 ALC_CMC.5.12C

The CM documentation shall include a CM plan.

12.2.8.3.13 ALC_CMC.5.13C

The CM plan shall describe how the CM system is used for the development of the TOE.

12.2.8.3.14 ALC_CMC.5.14C

The CM plan shall describe the procedures used to accept modified or newly created configuration items as part of the TOE.

12.2.8.3.15 ALC_CMC.5.15C

The evidence shall demonstrate that all configuration items are being maintained under the CM system.

12.2.8.3.16 ALC_CMC.5.16C

The evidence shall demonstrate that the CM system is being operated in accordance with the CM plan.

12.2.8.4 Evaluator action elements

12.2.8.4.1 ALC_CMC.5.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.2.8.4.2 ALC_CMC.5.2E

The evaluator shall determine that the application of the production support procedures results in a TOE as provided by the developer for testing activities.

12.3 CM scope (ALC_CMS)

12.3.1 Objectives

The objective of this family is to identify items to be included as configuration items and hence placed under the CM requirements of CM capabilities (ALC_CMC). Applying configuration management to these additional items provides additional assurance that the integrity of TOE is maintained.

12.3.2 Component levelling

The components in this family are levelled on the basis of which of the following are required to be included as configuration items: the TOE and the evaluation evidence required by the SARs; the parts of the TOE; the implementation representation; security flaws; and development tools and related information.

12.3.3 Application notes

While CM scope (ALC_CMS) mandates a list of configuration items and that each item on this list be under CM, CM capabilities (ALC_CMC) leaves the contents of the configuration list to the discretion of the developer. CM scope (ALC_CMS) narrows this discretion by identifying items that must be included in the configuration list, and hence come under the CM requirements of CM capabilities (ALC_CMC).

12.3.4 ALC_CMS.1 TOE CM coverage

Dependencies: No dependencies.

12.3.4.1 Objectives

A CM system can control changes only to those items that have been placed under CM (i.e., the configuration items identified in the configuration list). Placing the TOE itself and the evaluation evidence required by the other SARs in the ST under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

12.3.4.2 Application notes

ALC_CMS.1.1C introduces the requirement that the TOE itself and the evaluation evidence required by the other SARs in the ST be included in the configuration list and hence be subject to the CM requirements of CM capabilities (ALC_CMC).

12.3.4.3 Developer action elements

12.3.4.3.1 ALC_CMS.1.1D

The developer shall provide a configuration list for the TOE.

12.3.4.4 Content and presentation elements

12.3.4.4.1 ALC_CMS.1.1C

The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.

12.3.4.4.2 ALC_CMS.1.2C

The configuration list shall uniquely identify the configuration items.

12.3.4.5 Evaluator action elements

12.3.4.5.1 ALC_CMS.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.3.5 ALC_CMS.2 Parts of the TOE CM coverage

Dependencies: No dependencies.

12.3.5.1 Objectives

A CM system can control changes only to those items that have been placed under CM (i.e., the configuration items identified in the configuration list). Placing the TOE itself, the parts that comprise the TOE, and the evaluation evidence required by the other SARs under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

12.3.5.2 Application notes

ALC_CMS.2.1C introduces the requirement that the parts that comprise the TOE (all parts that are delivered to the consumer, for example hardware parts or executable files) be included in the configuration list and hence be subject to the CM requirements of CM capabilities (ALC_CMC).

ALC_CMS.2.3C introduces the requirement that the configuration list indicate the developer of each TSF relevant configuration item.

12.3.5.3 Developer action elements

12.3.5.3.1 ALC_CMS.2.1D

The developer shall provide a configuration list for the TOE.

12.3.5.4 Content and presentation elements

12.3.5.4.1 ALC_CMS.2.1C

The configuration list shall include the following: the TOE itself; the evaluation evidence required by the SARs; **and the parts that comprise the TOE.**

12.3.5.4.2 ALC_CMS.2.2C

The configuration list shall uniquely identify the configuration items.

12.3.5.4.3 ALC_CMS.2.3C

For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

12.3.5.5 Evaluator action elements

12.3.5.5.1 ALC_CMS.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.3.6 ALC_CMS.3 Implementation representation CM coverage

Dependencies: No dependencies.

12.3.6.1 Objectives

A CM system can control changes only to those items that have been placed under CM (i.e., the configuration items identified in the configuration list). Placing the TOE itself, the parts that comprise the TOE, the TOE implementation representation and the evaluation evidence required by the other SARs under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

12.3.6.2 Application notes

ALC_CMS.3.1C introduces the requirement that the TOE implementation representation be included in the list of configuration items and hence be subject to the CM requirements of CM capabilities (ALC_CMC).

12.3.6.3 Developer action elements

12.3.6.3.1 ALC_CMS.3.1D

The developer shall provide a configuration list for the TOE.

12.3.6.4 Content and presentation elements

12.3.6.4.1 ALC_CMS.3.1C

The configuration list shall include the following: the TOE itself; the evaluation evidence required by the SARs; the parts that comprise the TOE; **and the implementation representation.**

12.3.6.4.2 ALC_CMS.3.2C

The configuration list shall uniquely identify the configuration items.

12.3.6.4.3 ALC_CMS.3.3C

For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

12.3.6.5 Evaluator action elements

12.3.6.5.1 ALC_CMS.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.3.7 ALC_CMS.4 Problem tracking CM coverage

Dependencies: No dependencies.

12.3.7.1 Objectives

A CM system can control changes only to those items that have been placed under CM (i.e., the configuration items identified in the configuration list). Placing the TOE itself, the parts that comprise the TOE, the TOE implementation representation and the evaluation evidence required by the other SARs under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

Placing security flaw reports under CM ensures that the integrity of the reports is maintained and that access to them is managed, further, it may support developers in tracking security flaws to their resolution.

12.3.7.2 Application notes

ALC_CMS.4.1C introduces the requirement that reports of identified security flaws be included in the configuration list and hence be subject to the CM requirements of CM capabilities (ALC_CMC). This requires that information regarding previously identified security flaw reports and their resolution be maintained.

12.3.7.3 Developer action elements

12.3.7.3.1 ALC_CMS.4.1D

The developer shall provide a configuration list for the TOE.

12.3.7.4 Content and presentation elements

12.3.7.4.1 ALC_CMS.4.1C

The configuration list shall include the following: the TOE itself; the evaluation evidence required by the SARs; the parts that comprise the TOE; the implementation representation; **and security flaw reports and resolution status.**

12.3.7.4.2 ALC_CMS.4.2C

The configuration list shall uniquely identify the configuration items.

12.3.7.4.3 ALC_CMS.4.3C

For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

12.3.7.5 Evaluator action elements

12.3.7.5.1 ALC_CMS.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.3.8 ALC_CMS.5 Development tools CM coverage

Dependencies: No dependencies.

12.3.8.1 Objectives

A CM system can control changes only to those items that have been placed under CM (i.e., the configuration items identified in the configuration list). Placing the TOE itself, the parts that comprise the TOE, the TOE implementation representation and the evaluation evidence required by the other SARs under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

Placing security flaw reports under CM ensures that the integrity of the reports is maintained and that access to them is managed, further, it may support developers in tracking security flaws to their resolution.

Development tools play an important role in ensuring the production of a quality version of the TOE. Therefore, it is important to control modifications to these tools.

12.3.8.2 Application notes

ALC_CMS.5.1C introduces the requirement that development tools and other related information be included in the list of configuration items and hence be subject to the CM requirements of CM capabilities (ALC_CMC). Examples of development tools are programming languages and compilers. Information pertaining to TOE generation items (such as compiler options, generation options, and build options) is an example of information relating to development tools.

12.3.8.3 Developer action elements

12.3.8.3.1 ALC_CMS.5.1D

The developer shall provide a configuration list for the TOE.

12.3.8.4 Content and presentation elements

12.3.8.4.1 ALC_CMS.5.1C

The configuration list shall include the following: the TOE itself; the evaluation evidence required by the SARs; the parts that comprise the TOE; the implementation representation; security flaw reports and resolution status; **and development tools and related information.**

12.3.8.4.2 ALC_CMS.5.2C

The configuration list shall uniquely identify the configuration items.

12.3.8.4.3 ALC_CMS.5.3C

For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

12.3.8.5 Evaluator action elements

12.3.8.5.1 ALC_CMS.5.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.4 Delivery (ALC_DEL)

12.4.1 Objectives

The concern of this family is the secure transfer of the finished TOE from the development environment into the responsibility of the user.

The requirements for delivery call for system control and distribution facilities and procedures that detail the controls necessary to provide assurance that the security of the TOE is maintained during distribution of the TOE to the user. For a valid distribution of the TOE, the procedures used for the distribution of the TOE address the implied or identified objectives identified in the PP/ST relating to the security of the TOE during delivery.

12.4.2 Component levelling

This family contains only one component. An increasing level of protection for the TOE is established by requiring that the delivery procedures are commensurate with the assumed attack potential in the family Vulnerability analysis (AVA_VAN) specified in the ST.

12.4.3 Application notes

Transfers from subcontractors to the developer or between different development sites are not considered here, but in the family Developer environment security (ALC_DVS).

The end of the delivery phase is marked by the acceptance of the transfer of the TOE into the responsibility of the downstream user.

NOTE: This does not necessarily coincide with the arrival of the TOE at the downstream user's location.

The delivery procedures should consider, if applicable, issues such as:

a) ensuring that the TOE received by the consumer corresponds precisely to the evaluated version of the TOE;

b) avoiding or detecting any tampering with the actual version of the TOE;

c) preventing submission of a counterfeit version of the TOE;

d) avoiding unwanted knowledge of distribution of the TOE to the consumer: there might be cases where potential attackers should not know when and how it is delivered;

e) avoiding or detecting the TOE being intercepted during delivery; and

f) avoiding the TOE being delayed or stopped during distribution.

The delivery procedures should include the recipient's actions implied by these issues. The consistent description of these implied actions is examined in the Preparative procedures (AGD_PRE) family, if present.

12.4.4 ALC_DEL.1 Delivery procedures

Dependencies: No dependencies.

12.4.4.1 Developer action elements

12.4.4.1.1 ALC_DEL.1.1D

The developer shall document and provide procedures for delivery of the TOE or parts of it to the consumer.

12.4.4.1.2 ALC_DEL.1.2D

The developer shall use the delivery procedures.

12.4.4.2 Content and presentation elements

12.4.4.2.1 ALC_DEL.1.1C

The delivery documentation shall describe all procedures that are necessary to maintain security when distributing versions of the TOE to the consumer.

12.4.4.3 Evaluator action elements

12.4.4.3.1 ALC_DEL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.5 Developer environment security (ALC_DVS)

12.5.1 Objectives

Development security is concerned with the determination and specification of security controls relating to the developer provided environment.

NOTE: Such controls include coverage of security relevant aspects of asset management, human resources security, physical and environmental security, communications and operations management, access control, information systems acquisition, development and maintenance, information security incident management, and business continuity management.

12.5.2 Component levelling

The components in this family are levelled on the basis of whether justification of the sufficiency of the security controls is required.

12.5.3 Application notes

This family deals with controls to remove or reduce threads and security risks existing at the developer's site.

The evaluator should visit the site(s) in order to assess evidence for development security. This may include sites of subcontractors involved in the TOE development and production. Any decision not to visit shall be agreed with the evaluation authority.

Although development security deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, the Developer environment security (ALC_DVS) requirements specify only that the development security controls be in place at the time of evaluation. Furthermore, Developer environment security (ALC_DVS) does not contain any requirements related to the sponsor's intention to apply the development security controls in the future, after completion of the evaluation.

It is recognised that confidentiality may not always be an issue for the protection of the TOE in its development environment. The use of the word “necessary” allows for the selection of appropriate safeguards.

12.5.4 ALC_DVS.1 Identification of security controls

Dependencies: No dependencies.

12.5.4.1 Developer action elements

12.5.4.1.1 ALC_DVS.1.1D

The developer shall produce and provide development security documentation.

12.5.4.2 Content and presentation elements

12.5.4.2.1 ALC_DVS.1.1C

The development security documentation shall describe all the physical, logical, procedural, personnel, and other security controls that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment.

12.5.4.3 Evaluator action elements

12.5.4.3.1 ALC_DVS.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.5.4.3.2 ALC_DVS.1.2E

The evaluator shall confirm that the security controls are being applied.

12.5.5 ALC_DVS.2 Sufficiency of security controls

Dependencies: No dependencies.

12.5.5.1 Developer action elements

12.5.5.1.1 ALC_DVS.2.1D

The developer shall produce and provide development security documentation.

12.5.5.2 Content and presentation elements

12.5.5.2.1 ALC_DVS.2.1C

The development security documentation shall describe all the physical, procedural, personnel, and other security controls that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment.

12.5.5.2.2 ALC_DVS.2.2C

The development security documentation shall justify that the security controls provide the necessary level of protection to maintain the confidentiality and integrity of the TOE.

12.5.5.3 Evaluator action elements

12.5.5.3.1 ALC_DVS.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.5.5.3.2 ALC_DVS.2.2E

The evaluator shall confirm that the security controls are being applied.

12.6 Flaw remediation (ALC_FLR)

12.6.1 Objectives

Flaw remediation requires that discovered security flaws be tracked and corrected by the developer. Although future compliance with flaw remediation procedures cannot be determined at the time of the TOE evaluation, it is possible to evaluate the policies and procedures that a developer has in place to track and correct flaws, and to distribute the flaw information and corrections.

12.6.2 Component levelling

The components in this family are levelled on the basis of the increasing extent in scope of the flaw remediation procedures and the rigour of the flaw remediation policies.

12.6.3 Application notes

This family provides assurance that the TOE will be maintained and supported in the future, requiring the TOE developer to track and correct flaws in the TOE. Additionally, requirements are included for the distribution of flaw corrections. However, this family does not impose evaluation requirements beyond the current evaluation.

The TOE user is considered to be the focal point in the user organisation that is responsible for receiving and implementing fixes to security flaws. This is not necessarily an individual user, but may be an organisational representative who is responsible for the handling of security flaws. The use of the term TOE user recognises that different organisations have different procedures for handling flaw reporting, which may be done either by an individual user, or by a central administrative body.

The flaw remediation procedures should describe the methods for dealing with all types of flaws encountered. These flaws may be reported by the developer, by users of the TOE, or by other parties with familiarity with the TOE. Some flaws may not be reparable immediately. There may be some occasions where a flaw cannot be fixed and other (e.g. procedural) controls must be taken. The documentation provided should cover the procedures for providing the operational sites with fixes, and providing information on flaws where fixes are delayed (and what to do in the interim) or when fixes are not possible.

Changes applied to a TOE after its release render it unevaluated; although some information from the original evaluation may still apply. The phrase "release of the TOE" used in this family therefore refers to a version of a product that is a release of a certified TOE, to which changes have been applied.

12.6.4 ALC_FLR.1 Basic flaw remediation

Dependencies: No dependencies.

12.6.4.1 Developer action elements

12.6.4.1.1 ALC_FLR.1.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

12.6.4.2 Content and presentation elements

12.6.4.2.1 ALC_FLR.1.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

12.6.4.2.2 ALC_FLR.1.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

12.6.4.2.3 ALC_FLR.1.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

12.6.4.2.4 ALC_FLR.1.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

12.6.4.3 Evaluator action elements

12.6.4.3.1 ALC_FLR.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.6.5 ALC_FLR.2 Flaw reporting procedures

Dependencies: No dependencies.

12.6.5.1 Objectives

In order for the developer to be able to act appropriately upon security flaw reports from TOE users, and to know to whom to send corrective fixes, TOE users need to understand how to submit security flaw reports to the developer. Flaw remediation guidance from the developer to the TOE user ensures that TOE users are aware of this important information.

12.6.5.2 Developer action elements

12.6.5.2.1 ALC_FLR.2.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

12.6.5.2.2 ALC_FLR.2.2D

The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws.

12.6.5.2.3 ALC_FLR.2.3D

The developer shall provide flaw remediation guidance addressed to TOE users.

12.6.5.3 Content and presentation elements

12.6.5.3.1 ALC_FLR.2.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

12.6.5.3.2 ALC_FLR.2.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

12.6.5.3.3 ALC_FLR.2.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

12.6.5.3.4 ALC_FLR.2.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

12.6.5.3.5 ALC_FLR.2.5C

The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE.

12.6.5.3.6 ALC_FLR.2.6C

The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users.

12.6.5.3.7 ALC_FLR.2.7C

The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

12.6.5.3.8 ALC_FLR.2.8C

The flaw remediation guidance shall describe a means by which TOE users report to the developer any suspected security flaws in the TOE.

12.6.5.4 Evaluator action elements

12.6.5.4.1 ALC_FLR.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.6.6 ALC_FLR.3 Systematic flaw remediation

Dependencies: No dependencies.

12.6.6.1 Objectives

In order for the developer to be able to act appropriately upon security flaw reports from TOE users, and to know to whom to send corrective fixes, TOE users need to understand how to submit security flaw reports to the developer, and how to register themselves with the developer so that they may receive these corrective fixes. Flaw remediation guidance from the developer to the TOE user ensures that TOE users are aware of this important information.

12.6.6.2 Developer action elements

12.6.6.2.1 ALC_FLR.3.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

12.6.6.2.2 ALC_FLR.3.2D

The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws.

12.6.6.2.3 ALC_FLR.3.3D

The developer shall provide flaw remediation guidance addressed to TOE users.

12.6.6.3 Content and presentation elements

12.6.6.3.1 ALC_FLR.3.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

12.6.6.3.2 ALC_FLR.3.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

12.6.6.3.3 ALC_FLR.3.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

12.6.6.3.4 ALC_FLR.3.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

12.6.6.3.5 ALC_FLR.3.5C

The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE.

12.6.6.3.6 ALC_FLR.3.6C

The flaw remediation procedures shall include a procedure requiring timely response and the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.

12.6.6.3.7 ALC_FLR.3.7C

The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users.

12.6.6.3.8 ALC_FLR.3.8C

The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

12.6.6.3.9 ALC_FLR.3.9C

The flaw remediation guidance shall describe a means by which TOE users report to the developer any suspected security flaws in the TOE.

12.6.6.3.10 ALC_FLR.3.10C

The flaw remediation guidance shall describe a means by which TOE users may register with the developer, to be eligible to receive security flaw reports and corrections.

12.6.6.3.11 ALC_FLR.3.11C

The flaw remediation guidance shall identify the specific points of contact for all reports and enquiries about security issues involving the TOE.

12.6.6.4 Evaluator action elements

12.6.6.4.1 ALC_FLR.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.7 Development Life-cycle definition (ALC_LCD)

12.7.1 Objectives

Poorly defined or uncontrolled processes applied during the development, production and maintenance of the TOE can result in a TOE that does not meet all of its security objectives. Therefore, it is important that well defined and controlled processes be established as early as possible in the TOE's life-cycle.

Defining and implementing such processes does not guarantee that the TOE meets all of its SFRs. It is possible that the processes will be insufficient or inadequate.

Adopting a life-cycle model, or models that meets the needs of the developer's organization will improve the likelihood that the development, production and maintenance processes applied to TOE support the correct design and implementation of a TOE that meets the specified SFRs.

The determination of appropriate process controls in order to support process improvement is a long established best practice.

12.7.2 Component levelling

The components in this family are levelled on the basis of increasing requirements for measurability of the life-cycle model, and for compliance with that model.

12.7.3 Application notes

A life-cycle model encompasses the procedures, tools and techniques used to develop and maintain the TOE. Aspects of the process that may be covered by such a model include design methods, review procedures, project management controls, change control procedures, test methods and acceptance procedures. An effective life-cycle model will address these aspects of the development and maintenance process within an overall management structure that assigns responsibilities and monitors progress.

There are different types of acceptance situations that are dealt with at different locations in the criteria:

- acceptance of parts delivered by subcontractors ("integration") should be treated in this family
- Development Life-cycle definition (ALC_LCD),
- acceptance subsequent to internal transportations in Developer environment security (ALC_DVS),

- acceptance of parts into the CM system in CM capabilities (ALC_CMC), and
- acceptance of the delivered TOE by the consumer in Delivery (ALC_DEL).

The first three types may overlap.

Although life-cycle definition deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, its evaluation adds assurance through an analysis of the life-cycle information for the TOE provided at the time of the evaluation.

A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the model enables sufficient minimisation of the danger that the TOE will not meet its security requirement.

A measurable life-cycle model is a model using some quantitative valuation (arithmetic parameters and/or metrics) of the managed product in order to measure development properties of the product. Typical metrics are source code complexity metrics, defect density (errors per size of code) or mean time to failure. For the security evaluation all those metrics are of relevance, which are used to increase quality by decreasing the probability of faults and thereby in turn increasing assurance in the security of the TOE.

One should take into account that there exist standardised life-cycle models on the one hand (like the waterfall model) and standardised metrics on the other hand (like error density), which may be combined. ISO/IEC 15408 does not require the life-cycle to follow exactly one standard defining both aspects.

12.7.4 ALC_LCD.1 Developer defined life-cycle processes

Dependencies: No dependencies.

12.7.4.1 Developer action elements

12.7.4.1.1 ALC_LCD.1.1D

The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE.

12.7.4.1.2 ALC_LCD.1.2D

The developer shall provide life-cycle definition documentation.

12.7.4.2 Content and presentation elements

12.7.4.2.1 ALC_LCD.1.1C

The life-cycle definition documentation shall describe the processes used to develop and maintain the TOE.

12.7.4.2.2 ALC_LCD.1.2C

The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.

12.7.4.3 Evaluator action elements

12.7.4.3.1 ALC_LCD.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.7.5 ALC_LCD.2 Measurable life-cycle model

Dependencies: No dependencies.

12.7.5.1 Developer action elements

12.7.5.1.1 ALC_LCD.2.1D

The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE **that is based on a measurable life-cycle model**.

12.7.5.1.2 ALC_LCD.2.2D

The developer shall provide life-cycle definition documentation.

12.7.5.1.3 ALC_LCD.2.3D

The developer shall measure the TOE development using the measurable life-cycle model..

12.7.5.1.4 ALC_LCD.2.4D

The developer shall provide life-cycle output documentation.

12.7.5.2 Content and presentation elements

12.7.5.2.1 ALC_LCD.2.1C

The life-cycle definition documentation shall describe the model used to develop and maintain the TOE **including the details of its arithmetic parameters and/or metrics used to measure the quality of the TOE and/or its development.**

12.7.5.2.2 ALC_LCD.2.2C

The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.

12.7.5.2.3 ALC_LCD.2.3C

The life-cycle output documentation shall provide the results of the measurements of the TOE development using the measurable life-cycle model.

12.7.5.3 Evaluator action elements

12.7.5.3.1 ALC_LCD.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.7.5.3.2 ALC_LCD.2.2E

The evaluator shall confirm that the measurements of the TOE development processes and security relevant properties of the TOE support improvements in the development processes and/or the TOE itself.

12.8 TOE Development Artifacts (ALC_TDA)

12.8.1 Objectives

This family aims to add trust to the development process or a development (for brevity). It focuses on the generation of certain artifacts in the development process. These artifacts are used at a later point in time to assess the degree to which the development process or the development is trustable. This trust is realized through the validation of the generated artifacts for confirming them as sufficient evidence for trustable development.

This family introduces developer practices within the development process to generate the required artifacts for realizing trustable development. If a requirement in this family does not explicitly specify the use of automation to generate the required artifacts, the developer is free to undertake the corresponding practice manually, or using some integrated automation in the development process, or using a hybrid method of both. It is expected that the degree of trust in the development process is proportional to the degree of automation adoption to implement the corresponding practice in the development process.

This family also has a closer relationship with ALC_TAT than most of the other families. As ALC_TAT focuses on the tools and techniques aspect for developing, analysing, and implementing the TOE, it provides the necessary context when describing the practices of this family being introduced into the development process.

12.8.2 Component levelling

The components in this family are levelled on the basis of increasing cross-checking for consistency with relevant evidence from components of other families of other security assurance classes.

12.8.3 Application notes

The requirements in ALC_TDA.1 provide a degree of trust in the developer’s ability to identify the set of implementation representation which actually has been used during the TOE generation time. This degree of trust helps to positively answer the question “is that really the source code for this software” or "is that really the register-transfer level (RTL) design or description for this integrated circuit hardware"” or “is that really the set of implementation representation for this TOE”, which is potentially relevant in an evaluation. Such degree of trust is built on

a) the timing of when the set of implementation representation identifiers is recorded or logged,

b) the integrity and authenticity of the record of implementation representation identifiers, and

c) the traceability of implementation representation identifiers from the TOE.

In the case where some implementation representation elements are also covered in the configuration list due to ALC_CMS.3, the requirements in ALC_TDA.2 make sure that these implementation representation elements actually are identifiable through the use of the implementation representation identifiers of ALC_TDA.1.

With the accurate recording or logging of the actual implementation representation being used by the development tools under the scope of ALC_TAT, it provides an additional evidence to convince a third party that a regeneration of the TOE is functionally equivalent to the original TOE.

The requirements in ALC_TDA.3 provide the developer an opportunity to testify the absence of functional differences between the two possibly visibly different TOEs which have been independently generated from the identical set of implementation representation.

12.8.4 ALC_TDA.1 Uniquely identifying implementation representation

Dependencies: No dependencies.

12.8.4.1 Developer action elements

12.8.4.1.1 ALC_TDA.1.1D

The developer shall identify individual elements of the TOE implementation representation to record the list of unique TOE implementation representation identifiers, as the development tool generates the TOE.

12.8.4.1.2 ALC_TDA.1.2D

The developer shall use the current date and time to timestamp the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.4.1.3 ALC_TDA.1.3D

The developer shall maintain the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.4.1.4 ALC_TDA.1.4D

The developer shall ensure the authenticity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time, with the maintenance of the (author) origination information.

12.8.4.1.5 ALC_TDA.1.5D

The developer shall be able to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.4.1.6 ALC_TDA.1.6D

The developer shall produce and provide documentation describing

a) **the developer's creation of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;**

b) **the developer's timestamp being applied to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;**

c) **the maintenance of the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;**

d) the maintenance of the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information;

e) the developer's mechanism to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.4.2 Content and presentation elements

12.8.4.2.1 ALC_TDA.1.1C

The list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall demonstrate the correspondence between the TOE implementation representation element identifiers and the TOE implementation representation element names.

12.8.4.2.2 ALC_TDA.1.2C

The TOE implementation representation element names shall be in the same form as used or referenced by the development tool to generate the TOE.

12.8.4.2.3 ALC_TDA.1.3C

The timestamp of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the creation time of the TOE.

12.8.4.2.4 ALC_TDA.1.4C

The (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the (author) origination information of the TOE.

12.8.4.3 Evaluator action elements

12.8.4.3.1 ALC_TDA.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.8.4.3.2 ALC_TDA.1.2E

The evaluator shall confirm that the development tool for generating the TOE is capable to use or reference the implementation representation element names.

12.8.4.3.3 ALC_TDA.1.3E

The evaluator shall confirm that the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the creation time of the TOE.

12.8.4.3.4 ALC_TDA.1.4E

The evaluator shall confirm that the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the (author) origination information of the TOE.

12.8.4.3.5 ALC_TDA.1.5E

The evaluator shall check the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information.

12.8.4.3.6 ALC_TDA.1.6E

The evaluator shall confirm the developer's ability to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5 ALC_TDA.2 Matching CMS scope of implementation representation

Dependencies: ALC_CMS.3 Implementation representation CM coverage

12.8.5.1 Developer action elements

12.8.5.1.1 ALC_TDA.2.1D

The developer shall identify individual elements of the TOE implementation representation to record the list of unique TOE implementation representation identifiers, as the development tool generates the TOE.

12.8.5.1.2 ALC_TDA.2.2D

The developer shall use the current date and time to timestamp the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.1.3 ALC_TDA.2.3D

The developer shall maintain the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.1.4 ALC_TDA.2.4D

The developer shall ensure the authenticity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time, with the maintenance of the (author) origination information.

12.8.5.1.5 ALC_TDA.2.5D

The developer shall be able to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.1.6 ALC_TDA.2.6D

The developer shall produce and provide documentation describing

a) the developer's creation of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;
b) the developer's timestamp being applied to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;
c) the maintenance of the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;

d) the maintenance of the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information;
e) the developer's mechanism to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.1.7 ALC_TDA.2.7D

The developer shall provide evidence that the elements of implementation representation under the configuration scope of ALC_CMS.3 are identified by the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.2 Content and presentation elements

12.8.5.2.1 ALC_TDA.2.1C

The list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall demonstrate the correspondence between the TOE implementation representation element identifiers and the TOE implementation representation element names.

12.8.5.2.2 ALC_TDA.2.2C

The TOE implementation representation element names shall be in the same form as used or referenced by the development tool to generate the TOE.

12.8.5.2.3 ALC_TDA.2.3C

The timestamp of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the creation time of the TOE.

12.8.5.2.4 ALC_TDA.2.4C

The (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the (author) origination information of the TOE.

12.8.5.2.5 ALC_TDA.2.5C

The list of identifiers of the elements of implementation representation under the configuration scope of ALC_CMS.3 shall match with the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.3 Evaluator action elements

12.8.5.3.1 ALC_TDA.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.8.5.3.2 ALC_TDA.2.2E

The evaluator shall confirm that the development tool for generating the TOE is capable to use or reference the implementation representation element names.

12.8.5.3.3 ALC_TDA.2.3E

The evaluator shall confirm that the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the creation time of the TOE.

12.8.5.3.4 ALC_TDA.2.4E

The evaluator shall confirm that the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the (author) origination information of the TOE.

12.8.5.3.5 ALC_TDA.2.5E

The evaluator shall check the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information.

12.8.5.3.6 ALC_TDA.2.6E

The evaluator shall confirm the developer's ability to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.5.3.7 ALC_TDA.2.7E

The evaluator shall confirm that the list of identifiers of the elements of implementation representation under the configuration scope of ALC_CMS.3 matches with the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6 ALC_TDA.3 Regenerate TOE with well-defined development tools

Dependencies: ALC_CMS.3 Implementation representation CM coverage

ALC_TAT.1 Well-defined development tools and

ADV_IMP.1 Implementation representation of the TSF

12.8.6.1 Developer action elements

12.8.6.1.1 ALC_TDA.3.1D

The developer shall identify individual elements of the TOE implementation representation to record the list of unique TOE implementation representation identifiers, as the development tool generates the TOE.

12.8.6.1.2 ALC_TDA.3.2D

The developer shall use the current date and time to timestamp the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.1.3 ALC_TDA.3.3D

The developer shall maintain the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.1.4 ALC_TDA.3.4D

The developer shall ensure the authenticity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time, with the maintenance of the (author) origination information.

12.8.6.1.5 ALC_TDA.3.5D

The developer shall be able to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.1.6 ALC_TDA.3.6D

The developer shall produce and provide documentation describing

a) the developer's creation of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;
b) the developer's timestamp being applied to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;
c) the maintenance of the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time;
d) the maintenance of the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information;
e) the developer's mechanism to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.1.7 ALC_TDA.3.7D

The developer shall provide evidence that the elements of implementation representation under the configuration scope of ALC_CMS.3 are identified by the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.1.8 ALC_TDA.3.8D

After applying the development tools to another copy of the TOE implementation representation according to the list of unique TOE implementation representation identifiers to regenerate a TOE copy, the developer shall explain the functional differences, if any, between the TOE copy and the original TOE.

12.8.6.1.9 ALC_TDA.3.9D

The developer shall produce and provide documentation explaining the functional differences, if any, between the regenerated TOE copy and the original TOE.

12.8.6.2 Content and presentation elements

12.8.6.2.1 ALC_TDA.3.1C

The list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall demonstrate the correspondence between the TOE implementation representation element identifiers and the TOE implementation representation element names.

12.8.6.2.2 ALC_TDA.3.2C

The TOE implementation representation element names shall be in the same form as used or referenced by the development tool to generate the TOE.

12.8.6.2.3 ALC_TDA.3.3C

The timestamp of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the creation time of the TOE.

12.8.6.2.4 ALC_TDA.3.4C

The (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time shall be consistent with the (author) origination information of the TOE.

12.8.6.2.5 ALC_TDA.3.5C

The list of identifiers of the elements of implementation representation under the configuration scope of ALC_CMS.3 shall match with the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.2.6 ALC_TDA.3.6C

The developer's explanation of the functional differences, if any, between the regenerated TOE copy and the original TOE shall take into account all visible differences, if any, between the regenerated TOE copy and the original TOE.

12.8.6.3 Evaluator action elements

12.8.6.3.1 ALC_TDA.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.8.6.3.2 ALC_TDA.3.2E

The evaluator shall confirm that the development tool for generating the TOE is capable to use or reference the implementation representation element names.

12.8.6.3.3 ALC_TDA.3.3E

The evaluator shall confirm that the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the creation time of the TOE.

12.8.6.3.4 ALC_TDA.3.4E

The evaluator shall confirm that the (author) origination information of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time is consistent with the (author) origination information of the TOE.

12.8.6.3.5 ALC_TDA.3.5E

The evaluator shall check the integrity of the list of unique TOE implementation representation identifiers as recorded during the TOE generation time and its associated timestamp and (author) origination information.

12.8.6.3.6 ALC_TDA.3.6E

The evaluator shall confirm the developer’s ability to trace from the TOE to the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.3.7 ALC_TDA.3.7E

The evaluator shall confirm that the list of identifiers of the elements of implementation representation under the configuration scope of ALC_CMS.3 matches with the list of unique TOE implementation representation identifiers as recorded during the TOE generation time.

12.8.6.3.8 ALC_TDA.3.8E

The evaluator shall check that the developer’s explanation of the functional differences, if any, between the regenerated TOE copy and the original TOE takes into account all visible differences, if any, between the regenerated TOE copy and the original TOE.

12.9 Tools and techniques (ALC_TAT)

12.9.1 Objectives

Tools and techniques is an aspect of selecting tools that are used to develop, analyse and implement the TOE. It includes requirements to prevent ill-defined, inconsistent or incorrect development tools from being used to develop the TOE. This includes, but is not limited to, programming languages, documentation, implementation standards, and other parts of the TOE such as supporting runtime libraries.

12.9.2 Component levelling

The components in this family are levelled on the basis of increasing requirements on the description and scope of the implementation standards and the documentation of implementation-dependent options.

12.9.3 Application notes

There is a requirement for well-defined development tools. These are tools that are clearly and completely described. For example, programming languages and computer aided design (CAD) systems that are based on a standard published by standards bodies are considered to be well-defined. Self-made tools would need further investigation to clarify whether they are well-defined.

The requirement in ALC_TAT.1.2C is especially applicable to programming languages so as to ensure that all statements in the source code have an unambiguous meaning.

In ALC_TAT.2 and ALC_TAT.3, implementation guidelines may be accepted as an implementation standard if they have been approved by some group of experts (e.g. academic experts, standards bodies). Implementation standards are normally public, well accepted and common practise in a specific industry, but developer-specific implementation guidelines may also be accepted as a standard; the emphasis is on the expertise.

Tools and techniques distinguishes between the implementation standards applied by the developer (ALC_TAT.2.3D) and the implementation standards for "all parts of the TOE" (ALC_TAT.3.3D) which include third party software, hardware, or firmware. The configuration list introduced in CM scope (ALC_CMS) requires that for each TSF relevant configuration item to indicate if it has been generated by the TOE developer or by third party developers

12.9.4 ALC_TAT.1 Well-defined development tools

Dependencies: ADV_IMP.1 Implementation representation of the TSF

12.9.4.1 Developer action elements

12.9.4.1.1 ALC_TAT.1.1D

The developer shall provide the documentation identifying each development tool being used for the TOE.

12.9.4.1.2 ALC_TAT.1.2D

The developer shall document and provide the selected implementation-dependent options of each development tool.

12.9.4.2 Content and presentation elements

12.9.4.2.1 ALC_TAT.1.1C

Each development tool used for implementation shall be well-defined.

12.9.4.2.2 ALC_TAT.1.2C

The documentation of each development tool shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.

12.9.4.2.3 ALC_TAT.1.3C

The documentation of each development tool shall unambiguously define the meaning of all implementation-dependent options.

12.9.4.3 Evaluator action elements

12.9.4.3.1 ALC_TAT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.9.5 ALC_TAT.2 Compliance with implementation standards

Dependencies: ADV_IMP.1 Implementation representation of the TSF

12.9.5.1 Developer action elements

12.9.5.1.1 ALC_TAT.2.1D

The developer shall provide the documentation identifying each development tool being used for the TOE.

12.9.5.1.2 ALC_TAT.2.2D

The developer shall document and provide the selected implementation-dependent options of each development tool.

12.9.5.1.3 ALC_TAT.2.3D

The developer shall describe and provide the implementation standards that are being applied by the developer.

12.9.5.2 Content and presentation elements

12.9.5.2.1 ALC_TAT.2.1C

Each development tool used for implementation shall be well-defined.

12.9.5.2.2 ALC_TAT.2.2C

The documentation of each development tool shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.

12.9.5.2.3 ALC_TAT.2.3C

The documentation of each development tool shall unambiguously define the meaning of all implementation-dependent options.

12.9.5.3 Evaluator action elements

12.9.5.3.1 ALC_TAT.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.9.5.3.2 ALC_TAT.2.2E

The evaluator shall confirm that the implementation standards have been applied.

12.9.6 ALC_TAT.3 Compliance with implementation standards - all parts

Dependencies: ADV_IMP.1 Implementation representation of the TSF

12.9.6.1 Developer action elements

12.9.6.1.1 ALC_TAT.3.1D

The developer shall provide the documentation identifying each development tool being used for the TOE.

12.9.6.1.2 ALC_TAT.3.2D

The developer shall document and provide the selected implementation-dependent options of each development tool.

12.9.6.1.3 ALC_TAT.3.3D

The developer shall describe and provide the implementation standards that are being applied by the developer **and by any third-party providers for all parts of the TOE.**

12.9.6.2 Content and presentation elements

12.9.6.2.1 ALC_TAT.3.1C

Each development tool used for implementation shall be well-defined.

12.9.6.2.2 ALC_TAT.3.2C

The documentation of each development tool shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.

12.9.6.2.3 ALC_TAT.3.3C

The documentation of each development tool shall unambiguously define the meaning of all implementation-dependent options.

12.9.6.3 Evaluator action elements

12.9.6.3.1 ALC_TAT.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.9.6.3.2 ALC_TAT.3.2E

The evaluator shall confirm that the implementation standards have been applied.

12.10 Integration of composition parts and consistency check of delivery procedures (ALC_COMP)

12.10.1 Objectives

The aims of this activity are to determine whether

- the correct version of the application is installed onto/into the correct version of the underlying platform, and
- the preparative guidance procedures of Platform and Application Developers are compatible with the acceptance procedure of the Composite Product Integrator.

12.10.2 Component levelling

This family contains only one component.

12.10.3 ALC_COMP.1 Integration of the application into the underlying platform and Consistency check for delivery and acceptance procedures

Dependencies: No dependencies

12.10.3.1Developer action elements

12.10.3.1.1 ALC_COMP.1.1D

The developer shall provide components configuration evidence; cf. item #7, item #8 and item #3 in Table D1, section Annex A.

12.10.3.2 Content and presentation elements

12.10.3.2.1 ALC_COMP.1.1C

The components configuration evidence shall show that the evaluated version of the application has been installed onto / embedded into the certified version of the underlying platform.

12.10.3.2.2 ALC_COMP.1.2C

The components configuration evidence shall show that:

i. The evidence for delivery and acceptance compatibility shall show that the delivery procedures of the Platform and Application Developers are compatible with the acceptance procedure of the Composite Product Integrator.

ii. the evidence shall show that preparative guidance procedures prescribed by the Platform and Application Developers are either actually being used by the Composite Product Integrator or compatible with the Composite Product Integrator guidance and do not contradict each other.

12.10.3.3 Evaluator action elements

12.10.3.3.1 ALC_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

12.10.3.3.2 ALC_COMP.1.2E

The evaluator shall confirm that the evidence for delivery compatibility is complete, coherent, and internally consistent.

13 Class ATE: Tests

13.1 Introduction

The class “Tests” encompasses five families: Coverage (ATE_COV), Depth (ATE_DPT), Independent testing (ATE_IND) (i.e. functional testing performed by evaluators), Functional tests (ATE_FUN) and Composite functional testing (ATE_COMP). Testing provides assurance that the TSF behaves as described (in the functional specification, TOE design, implementation representation, and allows straightforward traceability of SFR in test scenario).

The emphasis in this class is on confirmation that the TSF operates according to its design descriptions. This class does not address penetration testing, which is based upon an analysis of the TSF that specifically seeks to identify vulnerabilities in the design and implementation of the TSF. Penetration testing is addressed separately as an aspect of vulnerability assessment in the AVA: Vulnerability assessment class.

The ATE: Tests class separates testing into developer testing and evaluator testing. The Coverage (ATE_COV), and Depth (ATE_DPT) families address the completeness of developer testing. Coverage (ATE_COV) addresses the rigour with which the functional specification is tested; Depth (ATE_DPT) addresses whether testing against other design descriptions (security architecture, TOE design, and implementation representation) is required.

Functional tests (ATE_FUN) addresses the performing of the tests by the developer and how this testing should be documented. Finally, Independent testing (ATE_IND) then addresses evaluator testing: whether the evaluator should repeat part or all of the developer testing and how much independent testing the evaluator should do.

Composite functional testing (ATE_COMP) determines whether composite product as a whole exhibits the properties necessary to satisfy the functional requirements of its Security Target.

Figure 12 shows the families within this class, and the hierarchy of components within the families.

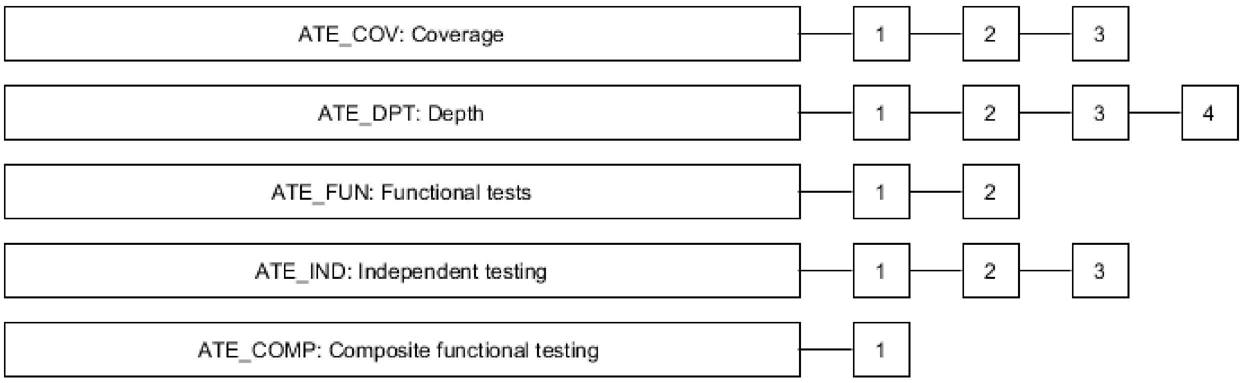
ATE_COV: Coverage
1
2
3
ATE_DPT: Depth
1
2
3
4
ATE_FUN: Functional tests
1
2
ATE_IND: Independent testing
1
2
3
ATE_COMP: Composite functional testing
1

Figure 12 — ATE: Tests class decomposition

13.2 Coverage (ATE_COV)

13.2.1 Objectives

This family establishes that the TSF has been tested against its functional specification. This is achieved through an examination of developer evidence of correspondence.

13.2.2 Component levelling

The components in this family are levelled on the basis of specification.

13.2.3 Application notes

13.2.4 ATE_COV.1 Evidence of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification

ATE_FUN.1 Functional testing

13.2.4.1 Objectives

The objective of this component is to establish that some of the TSFIs have been tested.

13.2.4.2 Application notes

In this component the developer shows how tests in the test documentation correspond to TSFIs in the functional specification. This can be achieved by a statement of correspondence, perhaps using a table.

13.2.4.3 Developer action elements

13.2.4.3.1 ATE_COV.1.1D

The developer shall provide evidence of the test coverage.

13.2.4.4 Content and presentation elements

13.2.4.4.1 ATE_COV.1.1C

The evidence of the test coverage shall show the correspondence between the tests in the test documentation and the TSFIs in the functional specification.

13.2.4.5 Evaluator action elements

13.2.4.5.1 ATE_COV.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.2.5 ATE_COV.2 Analysis of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification

ATE_FUN.1 Functional testing

13.2.5.1 Objectives

The objective of this component is to confirm that all of the TSFIs have been tested.

13.2.5.2 Application notes

In this component the developer confirms that tests in the test documentation correspond to all of the TSFIs in the functional specification. This can be achieved by a statement of correspondence, perhaps using a table, but the developer also provides an analysis of the test coverage.

13.2.5.3 Developer action elements

13.2.5.3.1 ATE_COV.2.1D

The developer shall provide **an analysis** of the test coverage.

13.2.5.4 Content and presentation elements

13.2.5.4.1 ATE_COV.2.1C

The **analysis** of the test coverage shall **demonstrate** the correspondence between the tests in the test documentation and the TSFIs in the functional specification.

13.2.5.4.2 ATE_COV.2.2C

The analysis of the test coverage shall demonstrate that all TSFIs in the functional specification have been tested.

13.2.5.5 Evaluator action elements

13.2.5.5.1 ATE_COV.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.2.6 ATE_COV.3 Rigorous analysis of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification

ATE_FUN.1 Functional testing

13.2.6.1 Objectives

In this component, the objective is to confirm that the developer performed exhaustive tests of all interfaces in the functional specification.

The objective of this component is to confirm that all parameters of all of the TSFIs have been tested.

13.2.6.2 Application notes

In this component the developer is required to show how tests in the test documentation correspond to all of the TSFIs in the functional specification. This can be achieved by a statement of correspondence, perhaps using a table, but in addition the developer is required to demonstrate that the tests exercise all of the parameters of all TSFIs. This additional requirement includes bounds testing (i.e. verifying that errors are generated when stated limits are exceeded) and negative testing (e.g. when access is given to User A, verifying not only that User A now has access, but also that User B did not suddenly gain access). This kind of testing is not, strictly speaking, *exhaustive* because not every possible value of the parameters is expected to be checked.

13.2.6.3 Developer action elements

13.2.6.3.1 ATE_COV.3.1D

The developer shall provide an analysis of the test coverage.

13.2.6.4 Content and presentation elements

13.2.6.4.1 ATE_COV.3.1C

The analysis of the test coverage shall demonstrate the correspondence between the tests in the test documentation and the TSFIs in the functional specification.

13.2.6.4.2 ATE_COV.3.2C

The analysis of the test coverage shall demonstrate that all TSFIs in the functional specification have been **completely** tested.

13.2.6.5 Evaluator action elements

13.2.6.5.1 ATE_COV.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.3 Depth (ATE_DPT)

13.3.1 Objectives

The components in this family deal with the level of detail to which the TSF is tested by the developer. Testing of the TSF is based upon increasing depth of information derived from additional design representations and descriptions (TOE design, implementation representation, and security architecture description).

The objective is to counter the risk of missing an error in the development of the TOE. Testing that exercises specific internal interfaces can provide assurance not only that the TSF exhibits the desired external security behaviour, but also that this behaviour stems from correctly operating internal functionality.

13.3.2 Component levelling

The components in this family are levelled on the basis of increasing detail provided in the TSF representations, from the TOE design to the implementation representation. This levelling reflects the TSF representations presented in the ADV class.

13.3.3 Application notes

The TOE design describes the internal components (e.g. subsystems) and, perhaps, modules of the TSF, together with a description of the interfaces among these components and modules. Evidence of testing of this TOE design must show that the internal interfaces have been exercised and seen to behave as described. This may be achieved through testing via the external interfaces of the TSF, or by testing of the TOE subsystem or module interfaces in isolation, perhaps employing a test harness. In cases where some aspects of an internal interface cannot be tested via the external interfaces, there should either be justification that these aspects need not be tested, or the internal interface needs to be tested directly. In the latter case the TOE design needs to be sufficiently detailed in order to facilitate direct testing.

In cases where the description of the TSF's architectural soundness (in Security Architecture (ADV_ARC)) cites specific mechanisms, the tests performed by the developer must show that the mechanisms have been exercised and seen to behave as described.

At the highest component of this family, the testing is performed not only against the TOE design, but also against the implementation representation.

13.3.4 ATE_DPT.1 Testing: basic design

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_TDS.2 Architectural design
	ATE_FUN.1 Functional testing

13.3.4.1 Objectives

The subsystem descriptions of the TSF provide a high-level description of the internal workings of the TSF. Testing at the level of the TOE subsystems provides assurance that the TSF subsystems behave and interact as described in the TOE design and the security architecture description.

13.3.4.2 Developer action elements

13.3.4.2.1 ATE_DPT.1.1D

The developer shall provide the analysis of the depth of testing.

13.3.4.3 Content and presentation elements

13.3.4.3.1 ATE_DPT.1.1C

The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems in the TOE design.

13.3.4.3.2 ATE_DPT.1.2C

The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested.

13.3.4.4 Evaluator action elements

13.3.4.4.1 ATE_DPT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.3.5 ATE_DPT.2 Testing: security enforcing modules

Dependencies: ADV_ARC.1 Security architecture description

ADV_TDS.3 Basic modular design

ATE_FUN.1 Functional testing

13.3.5.1 Objectives

The subsystem and module descriptions of the TSF provide a high-level description of the internal workings, and a description of the interfaces of the SFR-enforcing modules, of the TSF. Testing at this level of TOE description provides assurance that the TSF subsystems and SFR-enforcing modules behave and interact as described in the TOE design and the security architecture description.

13.3.5.2 Developer action elements

13.3.5.2.1 ATE_DPT.2.1D

The developer shall provide the analysis of the depth of testing.

13.3.5.3 Content and presentation elements

13.3.5.3.1 ATE_DPT.2.1C

The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems **and SFR-enforcing modules** in the TOE design.

13.3.5.3.2 ATE_DPT.2.2C

The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested.

13.3.5.3.3 ATE_DPT.2.3C

The analysis of the depth of testing shall demonstrate that the SFR-enforcing modules in the TOE design have been tested.

13.3.5.4 Evaluator action elements

13.3.5.4.1 ATE_DPT.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.3.6 ATE_DPT.3 Testing: modular design

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_TDS.4 Semiformal modular design
	ATE_FUN.1 Functional testing

13.3.6.1 Objectives

The subsystem and module descriptions of the TSF provide a high-level description of the internal workings, and a description of the interfaces of the modules, of the TSF. Testing at this level of TOE description provides assurance that the TSF subsystems and modules behave and interact as described in the TOE design and the security architecture description.

13.3.6.2 Developer action elements

13.3.6.2.1 ATE_DPT.3.1D

The developer shall provide the analysis of the depth of testing.

13.3.6.3 Content and presentation elements

13.3.6.3.1 ATE_DPT.3.1C

The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems and modules in the TOE design.

13.3.6.3.2 ATE_DPT.3.2C

The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested.

13.3.6.3.3 ATE_DPT.3.3C

The analysis of the depth of testing shall demonstrate that **all TSF** modules in the TOE design have been tested.

13.3.6.4 Evaluator action elements

13.3.6.4.1 ATE_DPT.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.3.7 ATE_DPT.4 Testing: implementation representation

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_TDS.4 Semiformal modular design
	ADV_IMP.1 Implementation representation of the TSF
	ATE_FUN.1 Functional testing

13.3.7.1 Objectives

The subsystem and module descriptions of the TSF provide a high-level description of the internal workings, and a description of the interfaces of the modules, of the TSF. Testing at this level of TOE description provides assurance that the TSF subsystems and modules behave and interact as described in the TOE design and the security architecture description, and in accordance with the implementation representation.

13.3.7.2 Developer action elements

13.3.7.2.1 ATE_DPT.4.1D

The developer shall provide the analysis of the depth of testing.

13.3.7.3 Content and presentation elements

13.3.7.3.1 ATE_DPT.4.1C

The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems and modules in the TOE design.

13.3.7.3.2 ATE_DPT.4.2C

The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested.

13.3.7.3.3 ATE_DPT.4.3C

The analysis of the depth of testing shall demonstrate that all modules in the TOE design have been tested.

13.3.7.3.4 ATE_DPT.4.4C

The analysis of the depth of testing shall demonstrate that the TSF operates in accordance with its implementation representation.

13.3.7.4 Evaluator action elements

13.3.7.4.1 ATE_DPT.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.4 Functional tests (ATE_FUN)

13.4.1 Objectives

Functional testing performed by the developer provides assurance that the tests in the test documentation are performed and documented correctly. The correspondence of these tests to the design descriptions of the TSF is achieved through the Coverage (ATE_COV) and Depth (ATE_DPT) families.

This family contributes to providing assurance that the likelihood of undiscovered flaws is relatively small.

The families Coverage (ATE_COV), Depth (ATE_DPT) and Functional tests (ATE_FUN) are used in combination to define the evidence of testing to be supplied by a developer. Independent functional testing by the evaluator is specified by Independent testing (ATE_IND).

13.4.2 Component levelling

This family contains two components, the higher requiring that ordering dependencies are analysed.

13.4.3 Application notes

Procedures for performing tests are expected to provide instructions for using test programs and test suites, including the test environment, test conditions, test data parameters and values. The test procedures should also show how the test results are derived from the test inputs.

Ordering dependencies are relevant when the successful execution of a particular test depends upon the existence of a particular state. For example, this might require that test A be executed immediately before test B, since the state resulting from the successful execution of test A is a prerequisite for the successful execution of test B. Thus, failure of test B could be related to a problem with the ordering dependencies. In the above example, test B could fail because test C (rather than test A) was executed immediately before it, or the failure of test B could be related to a failure of test A.

13.4.4 ATE_FUN.1 Functional testing

Dependencies: ATE_COV.1 Evidence of coverage

13.4.4.1 Objectives

The objective is for the developer to demonstrate that the tests in the test documentation are performed and documented correctly.

13.4.4.2 Developer action elements

13.4.4.2.1 ATE_FUN.1.1D

The developer shall test the TSF and document the results.

13.4.4.2.2 ATE_FUN.1.2D

The developer shall provide test documentation.

13.4.4.3 Content and presentation elements

13.4.4.3.1 ATE_FUN.1.1C

The test documentation shall consist of test plans, expected test results and actual test results.

13.4.4.3.2 ATE_FUN.1.2C

The test plans shall identify the tests to be performed and describe the scenarios for performing each test. These scenarios shall include any ordering dependencies on the results of other tests.

13.4.4.3.3 ATE_FUN.1.3C

The expected test results shall show the anticipated outputs from a successful execution of the tests.

13.4.4.3.4 ATE_FUN.1.4C

The actual test results shall be consistent with the expected test results.

13.4.4.4 Evaluator action elements

13.4.4.4.1 ATE_FUN.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.4.5 ATE_FUN.2 Ordered functional testing

Dependencies: ATE_COV.1 Evidence of coverage

13.4.5.1 Objectives

The objectives are for the developer to demonstrate that the tests in the test documentation are performed and documented correctly, and to ensure that testing is structured such as to avoid circular arguments about the correctness of the interfaces being tested.

13.4.5.2 Application notes

Although the test procedures may state pre-requisite initial test conditions in terms of ordering of tests, they may not provide a rationale for the ordering. An analysis of test ordering is an important factor in determining the adequacy of testing, as there is a possibility of faults being concealed by the ordering of tests.

13.4.5.3 Developer action elements

13.4.5.3.1 ATE_FUN.2.1D

The developer shall test the TSF and document the results.

13.4.5.3.2 ATE_FUN.2.2D

The developer shall provide test documentation.

13.4.5.4 Content and presentation elements

13.4.5.4.1 ATE_FUN.2.1C

The test documentation shall consist of test plans, expected test results and actual test results.

13.4.5.4.2 ATE_FUN.2.2C

The test plans shall identify the tests to be performed and describe the scenarios for performing each test. These scenarios shall include any ordering dependencies on the results of other tests.

13.4.5.4.3 ATE_FUN.2.3C

The expected test results shall show the anticipated outputs from a successful execution of the tests.

13.4.5.4.4 ATE_FUN.2.4C

The actual test results shall be consistent with the expected test results.

13.4.5.4.5 ATE_FUN.2.5C

The test documentation shall include an analysis of the test procedure ordering dependencies.

13.4.5.5 Evaluator action elements

13.4.5.5.1 ATE_FUN.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.5 Independent testing (ATE_IND)

13.5.1 Objectives

The objectives of this family are built upon the assurances achieved in the ATE_FUN, ATE_COV, and ATE_DPT families by verifying the developer testing and performing additional tests by the evaluator.

13.5.2 Component levelling

Levelling is based upon the amount of developer test documentation and test support and the amount of evaluator testing.

13.5.3 Application notes

This family deals with the degree to which there is independent functional testing of the TSF. Independent functional testing may take the form of repeating the developer's functional tests (in whole or in part) or of extending the scope or the depth of the developer's tests. These activities are complementary, and an appropriate mix must be planned for each TOE, which takes into account the availability and coverage of test results, and the functional complexity of the TSF.

Sampling of developer tests is intended to provide confirmation that the developer has carried out his planned test programme on the TSF, and has correctly recorded the results. The size of sample selected will be influenced by the detail and quality of the developer's functional test results. The evaluator will also need to consider the scope for devising additional tests, and the relative benefit that may be gained from effort in these two areas. It is recognised that repetition of all developer tests may be feasible and desirable in some cases, but may be very arduous and less productive in others. The highest component in this family should therefore be used with caution. Sampling will address the whole range of test results available, including those supplied to meet the requirements of both Coverage (ATE_COV) and Depth (ATE_DPT).

There is also a need to consider the different configurations of the TOE that are included within the evaluation. The evaluator will need to assess the applicability of the results provided, and to plan his own testing accordingly.

The suitability of the TOE for testing is based on the access to the TOE, and the supporting documentation and information required (including any test software or tools) to run tests. The need for such support is addressed by the dependencies to other assurance families.

Additionally, suitability of the TOE for testing may be based on other considerations. For example, the version of the TOE submitted by the developer may not be the final version.

The term *interfaces* refers to interfaces described in the functional specification and TOE design, and parameters passed through invocations identified in the implementation representation. The exact set of interfaces to be used is selected through Coverage (ATE_COV) and the Depth (ATE_DPT) components.

References to a subset of the interfaces are intended to allow the evaluator to design an appropriate set of tests which is consistent with the objectives of the evaluation being conducted.

13.5.4 ATE_IND.1 Independent testing - conformance

Dependencies: ADV_FSP.1 Basic functional specification

AGD_OPE.1 Operational user guidance

AGD_PRE.1 Preparative procedures

13.5.4.1 Objectives

In this component, the objective is to demonstrate that the TOE operates in accordance with its design representations and guidance documents.

13.5.4.2 Application notes

This component does not address the use of developer test results. It is applicable where such results are not available, and also in cases where the developer's testing is accepted without validation. The evaluator is required to devise and conduct tests with the objective of confirming that the TOE operates in accordance with its design representations, including but not limited to the functional specification. The approach is to gain confidence in correct operation through representative testing, rather than to conduct every possible test. The extent of testing to be planned for this purpose is a methodology issue, and needs to be considered in the context of a particular TOE and the balance of other evaluation activities.

13.5.4.3 Developer action elements

13.5.4.3.1 ATE_IND.1.1D

The developer shall provide the TOE for testing.

13.5.4.4 Content and presentation elements

13.5.4.4.1 ATE_IND.1.1C

The TOE shall be suitable for testing.

13.5.4.5 Evaluator action elements

13.5.4.5.1 ATE_IND.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.5.4.5.2 ATE_IND.1.2E

The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified.

13.5.5 ATE_IND.2 Independent testing - sample

Dependencies:	ADV_FSP.2 Security-enforcing functional specification
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
	ATE_COV.1 Evidence of coverage
	ATE_FUN.1 Functional testing

13.5.5.1 Objectives

In this component, the objective is to demonstrate that the TOE operates in accordance with its design representations and guidance documents. Evaluator testing confirms that the developer performed some tests of some interfaces in the functional specification.

13.5.5.2 Application notes

The intent is that the developer should provide the evaluator with materials necessary for the efficient reproduction of developer tests. This may include such things as machine-readable test documentation, test programs, etc.

This component contains a requirement that the evaluator has available test results from the developer to supplement the programme of testing. The evaluator will repeat a sample of the developer's tests to gain confidence in the results obtained. Having established such confidence the evaluator will build upon the developer's testing by conducting additional tests that exercise the TOE in a different manner. By using a platform of validated developer test results the evaluator is able to gain confidence that the TOE operates correctly in a wider range of conditions than would be possible purely using the developer's own efforts, given a fixed level of resource. Having gained confidence that the developer has tested the TOE, the evaluator will also have more freedom, where appropriate, to concentrate testing in areas where examination of documentation or specialist knowledge has raised particular concerns.

13.5.5.3 Developer action elements

13.5.5.3.1 ATE_IND.2.1D

The developer shall provide the TOE for testing.

13.5.5.4 Content and presentation elements

13.5.5.4.1 ATE_IND.2.1C

The TOE shall be suitable for testing.

13.5.5.4.2 ATE_IND.2.2C

The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the TSF.

13.5.5.5 Evaluator action elements

13.5.5.5.1 ATE_IND.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.5.5.5.2 ATE_IND.2.2E

The evaluator shall execute a sample of tests in the test documentation to verify the developer test results.

13.5.5.5.3 ATE_IND.2.3E

The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified.

13.5.6 ATE_IND.3 Independent testing - complete

Dependencies:	ADV_FSP.4 Complete functional specification
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
	ATE_COV.1 Evidence of coverage
	ATE_FUN.1 Functional testing

13.5.6.1 Objectives

In this component, the objective is to demonstrate that the TOE operates in accordance with its design representations and guidance documents. Evaluator testing includes repeating all of the developer tests.

13.5.6.2 Application notes

The intent is that the developer should provide the evaluator with materials necessary for the efficient reproduction of developer tests. This may include such things as machine-readable test documentation, test programs, etc.

In this component the evaluator must repeat all of the developer's tests as part of the programme of testing. As in the previous component the evaluator will also conduct tests that aim to exercise the TSF in a different manner from that achieved by the developer. In cases where developer testing has been exhaustive, there may remain little scope for this.

13.5.6.3 Developer action elements

13.5.6.3.1 ATE_IND.3.1D

The developer shall provide the TOE for testing.

13.5.6.4 Content and presentation elements

13.5.6.4.1 ATE_IND.3.1C

The TOE shall be suitable for testing.

13.5.6.4.2 ATE_IND.3.2C

The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the TSF.

13.5.6.5 Evaluator action elements

13.5.6.5.1 ATE_IND.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

13.5.6.5.2 ATE_IND.3.2E

The evaluator shall execute **all** tests in the test documentation to verify the developer test results.

13.5.6.5.3 ATE_IND.3.3E

The evaluator shall test the TSF to confirm that the **entire** TSF operates as specified.

13.6 Composite functional testing (ATE_COMP)

13.6.1 Objectives

The objective of this family is to determine whether composite product as a whole exhibits the properties necessary to satisfy the functional requirements of its Security Target.

13.6.2 Application notes

A composite product can be tested by testing the components of it separately and by testing the integrated product. Separate testing means that the platform and the application are being tested independent of each other. A lot of tests of the platform may have been performed within the scope of its accomplished evaluation. The application may be tested on a simulator or an emulator, which represent a virtual machine.

Integration testing means that the composite product is being tested as it is: the application is running on the platform.

Behaviour of implementation of some SFRs can depend on properties of the underlying platform as well as of the application (e.g. correctness of the measures of the composite product to withstand a side channel attack or correctness of the implementation of tamper resistance against physical attacks). In such a case the SFR implementation shall be tested on the final composite product, but not on a simulator or an emulator.

This activity focuses exclusively on testing of the composite product as a whole and represents merely partial efforts within the general test approach being covered by the assurance ATE. These integration tests shall be specified and performed, whereby the approach of the standard assurance families of the class ATE shall be applied.

13.6.3 ATE_COMP.1 Composite product functional testing

Dependencies: No dependencies

13.6.3.1 Developer action elements

13.6.3.1.1 ATE_COMP.1.1D

The developer shall provide a set of tests as required by the assurance package chosen.

13.6.3.1.2 ATE_COMP.1.2D

The developer shall provide the composite TOE for testing.

13.6.3.2 Content and presentation elements

13.6.3.2.1 ATE_COMP.1.1C

Content and presentation of the specification and documentation of the *integration* tests shall correspond to the standard[7] requirements of the assurance families ATE_FUN and ATE_COV.

13.6.3.2.2 ATE_COMP.1.2C

The composite TOE provided shall be suitable for testing.

[7] i.e. as defined by ISO/IEC 18045

13.6.3.3 Evaluator action elements

13.6.3.3.1 ATE_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14 Class AVA: Vulnerability assessment

14.1 Introduction

The AVA: Vulnerability assessment class addresses the possibility of exploitable vulnerabilities introduced in the development or the operation of the TOE.

Figure 13 shows the families within this class, and the hierarchy of components within the families.

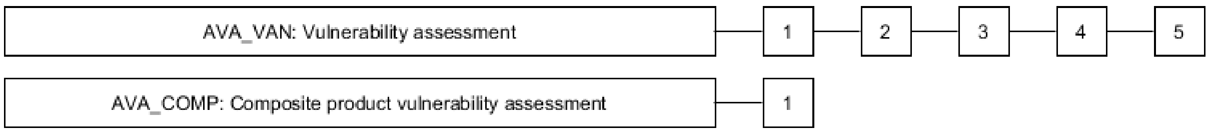
AVA_VAN: Vulnerability assessment
1
2
3
4
5
AVA_COMP: Composite product vulnerability assessment
1

Figure 13 — AVA: Vulnerability assessment class decomposition

14.2 Application notes

Generally, the vulnerability assessment activity covers various vulnerabilities in the development and operation of the TOE. Development vulnerabilities take advantage of some property of the TOE ,or the product where the TOE resides, which was introduced during its development, e.g. defeating the TSF self-protection through tampering, direct attack or monitoring of the TSF, defeating the TSF domain separation through monitoring or direct attack the TSF, or defeating non-bypassability through circumventing (bypassing) the TSF. Explicit dependencies of the TOE on IT systems in the environment must also be considered. Operational vulnerabilities take advantage of weaknesses in non-technical countermeasures to violate the TOE SFRs, e.g. misuse or incorrect configuration. Misuse investigates whether the TOE can be configured or used in a manner that is insecure, but that an administrator or user of the TOE would reasonably believe to be secure.

Assessment of development vulnerabilities is covered by the assurance family AVA_VAN. Basically, all development vulnerabilities can be considered in the context of AVA_VAN due to the fact, that this family allows application of a wide range of assessment methodologies being unspecific to the kind of an attack scenario. These unspecific assessment methodologies comprise, among other, also the specific methodologies for those TSF where covert channels are to be considered (a channel capacity estimation can be done using informal engineering measurements, as well as actual test measurements) or can be overcome by the use of sufficient resources in the form of a direct attack (underlying technical concept of those TSF is based on probabilistic or permutational mechanisms; a qualification of their security behaviour and the effort required to overcome them can be made using a quantitative or statistical analysis).

If there are security objectives specified in the ST to either to prevent one user of the TOE from observing activity associated with another user of the TOE, or to ensure that information flows cannot be used to achieve enforced illicit data signals, covert channel analysis should be considered during the conduct of the vulnerability analysis. This is often reflected by the inclusion of Unobservability

(FPR_UNO) and multilevel access control policies specified through Access control policy (FDP_ACC) and/or Information flow control policy (FDP_IFC) requirements in the ST.

14.3 Vulnerability analysis (AVA_VAN)

14.3.1 Objectives

Vulnerability analysis is an assessment to determine whether potential vulnerabilities identified, during the evaluation of the development and anticipated operation of the TOE or by other methods (e.g. by flaw hypotheses or quantitative or statistical analysis of the security behaviour of the underlying security mechanisms), could allow attackers to violate the SFRs.

Vulnerability analysis deals with the threats that an attacker will be able to discover flaws that will allow unauthorised access to data and functionality, allow the ability to interfere with or alter the TSF, or interfere with the authorised capabilities of other users.

In case of a **multi-assurance evaluation** the vulnerability analysis will assess the defined **sub-TSF** as well as the TOE as a whole.

14.3.2 Component levelling

Levelling is based on an increasing rigour of vulnerability analysis by the evaluator and increased levels of attack potential required by an attacker to identify and exploit the potential vulnerabilities.

14.3.3 AVA_VAN.1 Vulnerability survey

Dependencies:	ADV_FSP.1 Basic functional specification
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures

14.3.3.1 Objectives

A vulnerability survey of information available in the public domain is performed by the evaluator to ascertain potential vulnerabilities that may be easily found by an attacker.

The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the evaluator assuming an attack potential of Basic.

14.3.3.2 Developer action elements

14.3.3.2.1 AVA_VAN.1.1D

The developer shall provide the TOE for testing.

14.3.3.3 Content and presentation elements

14.3.3.3.1 AVA_VAN.1.1C

The TOE shall be suitable for testing.

14.3.3.4 Evaluator action elements

14.3.3.4.1 AVA_VAN.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14.3.3.4.2 AVA_VAN.1.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE.

14.3.3.4.3 AVA_VAN.1.3E

The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing Basic attack potential.

14.3.4 AVA_VAN.2 Vulnerability analysis

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_FSP.2 Security-enforcing functional specification
	ADV_TDS.1 Basic design
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures

14.3.4.1 Objectives

A vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the evaluator assuming an attack potential of Basic.

14.3.4.2 Developer action elements

14.3.4.2.1 AVA_VAN.2.1D

The developer shall provide the TOE for testing.

14.3.4.2.2 AVA_VAN.2.2D

The developer shall provide a list of third party components included in the TOE and the TOE delivery.

14.3.4.3 Content and presentation elements

14.3.4.3.1 AVA_VAN.2.1C

The TOE shall be suitable for testing.

14.3.4.3.2 AVA_VAN.2.2C

The list of third party components shall include components provided by third parties, and that are part of the TOE or otherwise part of the TOE delivery.

14.3.4.4 Evaluator action elements

14.3.4.4.1 AVA_VAN.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14.3.4.4.2 AVA_VAN.2.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE **the components in the list of third party components, and specific IT products in the environment that the TOE depends on.**

14.3.4.4.3 AVA_VAN.2.3E

The evaluator shall perform an independent vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design and security architecture description to identify potential vulnerabilities in the TOE.

14.3.4.4.4 AVA_VAN.2.4E

The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing Basic attack potential.

14.3.5 AVA_VAN.3 Focused vulnerability analysis

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_FSP.4 Complete functional specification
	ADV_TDS.3 Basic modular design
	ADV_IMP.1 Implementation representation of the TSF
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
	ATE_DPT.1 Testing: basic design

14.3.5.1 Objectives

A vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the evaluator assuming an attack potential of Enhanced-Basic.

14.3.5.2 Developer action elements

14.3.5.2.1 AVA_VAN.3.1D

The developer shall provide the TOE for testing.

14.3.5.2.2 AVA_VAN.3.2D

The developer shall provide a list of third party components included in the TOE and the TOE delivery.

14.3.5.3 Content and presentation elements

14.3.5.3.1 AVA_VAN.3.1C

The TOE shall be suitable for testing.

14.3.5.3.2 AVA_VAN.3.2C

The list of third party components shall include components provided by third parties, and that are part of the TOE or otherwise part of the TOE delivery.

14.3.5.4 Evaluator action elements

14.3.5.4.1 AVA_VAN.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14.3.5.4.2 AVA_VAN.3.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE the components in the list of third party components, and specific IT products in the environment that the TOE depends on.

14.3.5.4.3 AVA_VAN.3.3E

The evaluator shall perform an independent, **focused** vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design, security architecture description **and implementation representation** to identify potential vulnerabilities in the TOE.

14.3.5.4.4 AVA_VAN.3.4E

The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing **Enhanced-Basic** attack potential.

14.3.6 AVA_VAN.4 Methodical vulnerability analysis

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_FSP.4 Complete functional specification
	ADV_TDS.3 Basic modular design
	ADV_IMP.1 Implementation representation of the TSF

AGD_OPE.1 Operational user guidance

AGD_PRE.1 Preparative procedures

ATE_DPT.1 Testing: basic design

14.3.6.1 Objectives

A methodical vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the evaluator assuming an attack potential of Moderate.

14.3.6.2 Developer action elements

14.3.6.2.1 AVA_VAN.4.1D

The developer shall provide the TOE for testing.

14.3.6.2.2 AVA_VAN.4.2D

The developer shall provide a list of third party components included in the TOE and the TOE delivery.

14.3.6.3 Content and presentation elements

14.3.6.3.1 AVA_VAN.4.1C

The TOE shall be suitable for testing.

14.3.6.3.2 AVA_VAN.4.2C

The list of third party components shall include components provided by third parties, and that are part of the TOE or otherwise part of the TOE delivery.

14.3.6.4 Evaluator action elements

14.3.6.4.1 AVA_VAN.4.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14.3.6.4.2 AVA_VAN.4.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE the components in the list of third party components, and specific IT products in the environment that the TOE depends on.

14.3.6.4.3 AVA_VAN.4.3E

The evaluator shall perform an independent, **methodical** vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design, security architecture description and implementation representation to identify potential vulnerabilities in the TOE.

14.3.6.4.4 AVA_VAN.4.4E

The evaluator shall conduct penetration testing based on the identified potential vulnerabilities to determine that the TOE is resistant to attacks performed by an attacker possessing **Moderate** attack potential.

14.3.7 AVA_VAN.5 Advanced methodical vulnerability analysis

Dependencies:	ADV_ARC.1 Security architecture description
	ADV_FSP.4 Complete functional specification
	ADV_TDS.3 Basic modular design
	ADV_IMP.1 Implementation representation of the TSF
	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
	ATE_DPT.1 Testing: basic design

14.3.7.1 Objectives

A methodical vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the evaluator assuming an attack potential of High.

14.3.7.2 Developer action elements

14.3.7.2.1 AVA_VAN.5.1D

The developer shall provide the TOE for testing.

14.3.7.2.2 AVA_VAN.5.2D

The developer shall provide a list of third party components included in the TOE and the TOE delivery.

14.3.7.3 Content and presentation elements

14.3.7.3.1 AVA_VAN.5.1C

The TOE shall be suitable for testing.

14.3.7.3.2 AVA_VAN.2.2C

The list of third party components shall include components provided by third parties, and that are part of the TOE or otherwise part of the TOE delivery.

14.3.7.4 Evaluator action elements

14.3.7.4.1 AVA_VAN.5.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

14.3.7.4.2 AVA_VAN.5.2E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE the components in the list of third party components, and specific IT products in the environment that the TOE depends on.

14.3.7.4.3 AVA_VAN.5.3E

The evaluator shall perform an independent, methodical vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design, security architecture description and implementation representation to identify potential vulnerabilities in the TOE.

14.3.7.4.4 AVA_VAN.5.4E

The evaluator shall conduct penetration testing based on the identified potential vulnerabilities to determine that the TOE is resistant to attacks performed by an attacker possessing **High** attack potential.

14.4 Composite vulnerability assessment (AVA_COMP)

14.4.1 Objectives

The aim of this activity is to determine the exploitability of flaws or weaknesses in the composite TOE as a whole in the intended environment.

14.4.2 AVA_COMP.1 Composite product vulnerability assessment

Dependencies: No dependencies

14.4.2.1 Application notes

This activity focuses exclusively on vulnerability assessment of the composite product *as a whole* and represents merely *partial efforts* within the general approach being covered by the standard[8] assurance family of the class AVA: AVA_VAN.

The results of the vulnerability assessment for the underlying platform represented in the ETR_COMP can be reused under the following conditions: they are up to date and all composite activities for correctness – ASE_COMP.1, ALC_COMP.1, ADV_COMP.1 and ATE_COMP.1 – are finalised with the verdict PASS.

The yellow marked references need to be aligned with CEM. This will be done later on if the necessary content has been included in CEM.

[8] i.e. as defined by ISO/IEC 18045

Due to composing of the platform and the application a new quality arises, which can cause additional vulnerabilities of the platform which might be not mentioned in the ETR_COMP. In these circumstances [R44] in chapter Annex A applies.

14.4.2.2 Developer action elements

14.4.2.2.1 AVA_COMP.1.1D

The developer shall provide the composite TOE for penetrating testing.

14.4.2.3 Content and presentation elements

14.4.2.3.1 AVA_COMP.1.1C

The composite TOE provided shall be suitable for testing as a whole.

14.4.2.4 Evaluator action elements

14.4.2.4.1 AVA_COMP.1.1E

The evaluator shall conduct penetration testing of the composite product *as a whole* building on evaluator’s own vulnerability analysis, to ensure that the vulnerabilities being relevant for the Composite-ST are not exploitable.

15 Class ACO: Composition

15.1 Introduction

The class ACO: Composition encompasses five families. These families specify assurance requirements that are designed to provide confidence that a composed TOE will operate securely when relying upon security functionality provided by previously evaluated software, firmware or hardware components.

Composition involves taking two or more IT entities successfully evaluated against ISO/IEC 15408 security assurance requirements packages (base components and dependent components, see Annex B) and combining them for use, with no further development of either IT entity. The development of additional IT entities is not included (entities that have not previously been the subject of a component evaluation). The composed TOE forms a new product that can be installed and integrated into any specific environment instance that meets the objectives for the environment.

This approach does not provide an alternative approach for the evaluation of components. Composition under ACO provides a composed TOE integrator a method, which can be used as an alternative to other assurance levels specified in ISO/IEC 15408, to gain confidence in a TOE that is the combination of two or more successfully evaluated components without having to re-evaluate the composite TSF. (The composed TOE integrator is referred to as “developer” throughout the ACO class, with any references to the developer of the base or dependent components clarified as such.)

Composed Assurance Packages, as defined in part 5 provide an assurance scale for composed TOEs. This assurance scale is required in addition to other assurance packages, for example the EALs, because to combine components evaluated against another assurance package and gain equivalent assurance in the resulting composed TOE, all SARs have to be applied to the composed TOE. Although reuse can be made of the component TOE evaluation results, there are often additional aspects of the components that have to be considered in the composed TOE, as described in Annex B.3. Due to the different parties involved in a composed TOE evaluation activity it is generally not possible to gain all necessary evidence about these additional aspects of the components to apply the appropriate EAL. Hence, CAPs have been defined to address the issue of combining evaluated components and gaining a meaningful result. This is discussed further in Annex B.

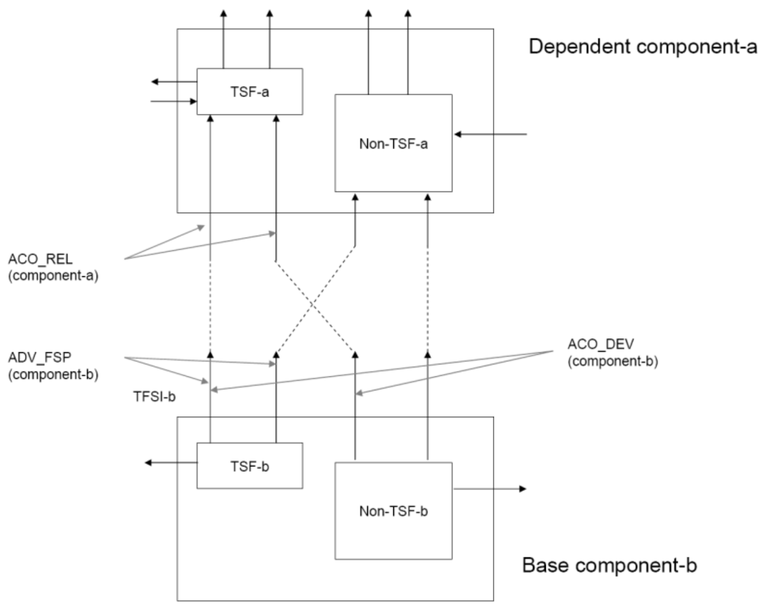

Figure 14 — Relationship between ACO families and interactions between components

In a composed TOE it is generally the case that one component relies on the services provided by another component. The component requiring services is termed the dependent component and the component providing the services is termed the base component. This interaction and distinct is discussed further in Annex B. It is assumed to be the case that the developer of the dependent component is supporting the composed TOE evaluation in some manner (as developer, sponsor, or just cooperating and providing the necessary evaluation evidence from the dependent component evaluation) The ACO components included in the CAP assurance packages should not be used as augmentations for component TOE evaluations, as this would provide no meaningful assurance for the component.

The families within the ACO class interact in a similar manner to the ADV, ATE and AVA classes in a component TOE evaluation and hence leverage from the specification of requirements from those classes where applicable. There are however a few items specific to composed TOE evaluations. To determine how the components interact and identify any deviations from the evaluations of the components, the dependencies that the dependent component has upon the underlying base component are identified (ACO_REL). This reliance on the base component is specified in terms of the interfaces through which the dependent component makes calls for services in support of the dependent component SFRs. The interfaces, and at higher levels the supporting behaviour, provided by the base component in response to those service requests are analysed in ACO_DEV. The ACO_DEV family is based on the ADV_TDS family, as at the simplest level the TSF of each component can be viewed as a subsystem of the composed TOE, with additional portions of each component seen as additional subsystems. Therefore, the interfaces between the components are seen as interactions between subsystems in a component TOE evaluation.

It is possible that the interfaces and supporting behaviour descriptions provided for ACO_DEV are incomplete. This is determined during the conduct of ACO_COR. The ACO_COR family takes the outputs of ACO_REL and ACO_DEV and determines whether the components are being used in their evaluated configuration and identifies where any specifications are incomplete, which are then identified as inputs into testing (ACO_CTT) and vulnerability analysis (ACO_VUL) activities of the composed TOE.

Testing of the composed TOE is performed to determine that the composed TOE exhibits the expected behaviour as determined by the composed TOE SFRs, and at higher levels demonstrates the compatibility of the interfaces between the components of the composed TOE.

The vulnerability analysis of the composed TOE leverages from the outputs of the vulnerability analysis of the component evaluations. The composed TOE vulnerability analysis considers any residual vulnerabilities from the component evaluations to determine that the residual vulnerabilities are not applicable to the composed TOE. A search of publicly available information relating to the components is also performed to identify any issues reported in the components since the completion of the respective evaluations.

The interaction between the ACO families is depicted in Figure 15 below. This shows by solid arrowed lines where the evidence and understanding gained in one family feeds into the next activity and the dashed arrows identify where an activity explicitly traces back to the composed TOE SFRs, as described above.

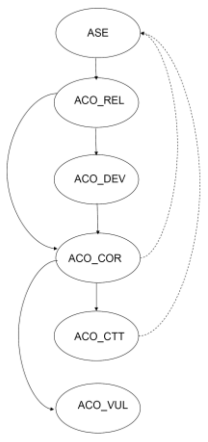

Figure 15 — Relationship between ACO families

Further discussion of the definition and interactions within composed TOEs is provided in Annex B.

Figure 16 shows the families within this class, and the hierarchy of components within the families.

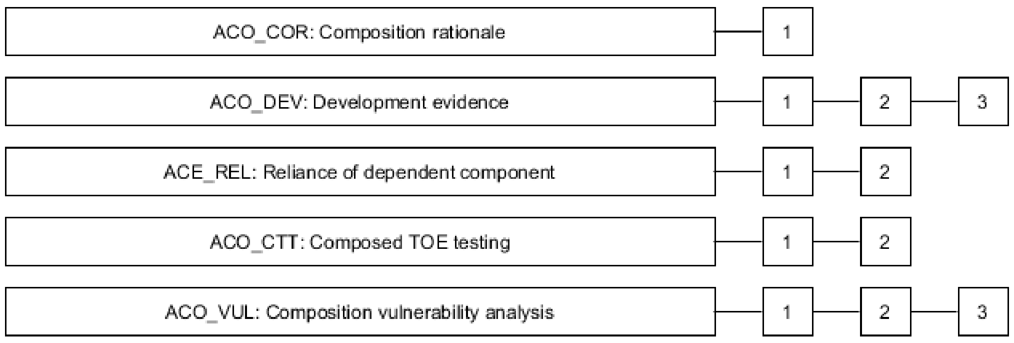
ACO_COR: Composition rationale
1
ACO_DEV: Development evidence
1
2
3
ACE_REL: Reliance of dependent component
1
2
ACO_CTT: Composed TOE testing
1
2
ACO_VUL: Composition vulnerability analysis
1
2
3

Figure 16 — ACO: Composition class decomposition

15.2 Composition rationale (ACO_COR)

15.2.1 Objectives

This family addresses the requirement to demonstrate that the base component can provide an appropriate level of assurance for use in composition.

15.2.2 Component levelling

There is only a single component in this family.

15.2.3 ACO_COR.1 Composition rationale

Dependencies: ACO_DEV.1 Functional Description

ALC_CMC.1 Labelling of the TOE

ACO_REL.1 Basic reliance information

15.2.3.1 Developer action elements

15.2.3.1.1 ACO_COR.1.1D

The developer shall provide composition rationale for the base component.

15.2.3.2 Content and presentation elements

15.2.3.2.1 ACO_COR.1.1C

The composition rationale shall demonstrate that a level of assurance at least as high as that of the dependent component has been obtained for the support functionality of the base component, when the base component is configured as required to support the TSF of the dependent component.

15.2.3.3 Evaluator action elements

15.2.3.3.1 ACO_COR.1.1E

The evaluator shall confirm that the information meets all requirements for content and presentation of evidence.

15.3 Development evidence (ACO_DEV)

15.3.1 Objectives

This family sets out requirements for a specification of the base component in increasing levels of detail. Such information is required to gain confidence that the appropriate security functionality is provided to support the requirements of the dependent component (as identified in the reliance information).

15.3.2 Component levelling

The components are levelled on the basis of increasing amounts of detail about the interfaces provided, and how they are implemented.

15.3.3 Application notes

The TSF of the base component is often defined without knowledge of the dependencies of the possible applications with which it may by composed. The TSF of this base component is defined to include all parts of the base component that have to be relied upon for enforcement of the base component SFRs. This will include all parts of the base component required to implement the base component SFRs.

The functional specification of the base component will describe the TSFI in terms of the interfaces the base component provides to allow an external entity to invoke operations of the TSF. This includes interfaces to the human user to permit interaction with the operation of the TSF invoking SFRs and also interfaces allowing an external IT entity to make calls into the TSF.

The functional specification only provides a description of what the TSF provides at its interface and the means by which that TSF functionality are invoked. Therefore, the functional specification does not necessarily provide a complete interface specification of all possible interfaces available between an external entity and the base component. It does not include what the TSF expects/requires from the operational environment. The description of what a dependent component TSF relies upon of a base component is considered in Reliance of dependent component (ACO_REL) and the development information evidence provides a response to the interfaces specified.

The development information evidence includes a specification of the base component. This may be the evidence used during evaluation of the base component to satisfy the ADV requirements, or may be another form of evidence produced by either the base component developer or the composed TOE developer. This specification of the base component is used during Development evidence (ACO_DEV) to gain confidence that the appropriate security functionality is provided to support the requirements of the dependent component. The level of detail required of this evidence increases to reflect the level of required assurance in the composed TOE. This is expected to broadly reflect the increasing confidence gained from the application of the assurance packages to the components. The evaluator determines that this description of the base component is consistent with the reliance information provided for the dependent component.

15.3.4 ACO_DEV.1 Functional Description

Dependencies: ACO_REL.1 Basic reliance information

15.3.4.1 Objectives

A description of the interfaces in the base component, on which the dependent component relies, is required. This is examined to determine whether or not it is consistent with the description of interfaces on which the dependent component relies, as provided in the reliance information.

15.3.4.2 Developer action elements

15.3.4.2.1 ACO_DEV.1.1D

The developer shall provide development information for the base component.

15.3.4.3 Content and presentation elements

15.3.4.3.1 ACO_DEV.1.1C

The development information shall describe the purpose of each interface of the base component used in the composed TOE.

15.3.4.3.2 ACO_DEV.1.2C

The development information shall show correspondence between the interfaces, used in the composed TOE, of the base component and the dependent component to support the TSF of the dependent component.

15.3.4.4 Evaluator action elements

15.3.4.4.1 ACO_DEV.1.1E

The evaluator shall confirm that the information meets all requirements for content and presentation of evidence.

15.3.4.4.2 ACO_DEV.1.2E

The evaluator shall determine that the interface description provided is consistent with the reliance information provided for the dependent component.

15.3.5 ACO_DEV.2 Basic evidence of design

Dependencies: ACO_REL.1 Basic reliance information

15.3.5.1 Objectives

A description of the interfaces in the base component, on which the dependent component relies, is required. This is examined to determine whether or not it is consistent with the description of interfaces on which the dependent component relies, as provided in the reliance information.

In addition, the security behaviour of the base component that supports the dependent component TSF is described.

15.3.5.2 Developer action elements

15.3.5.2.1 ACO_DEV.2.1D

The developer shall provide development information for the base component.

15.3.5.3 Content and presentation elements

15.3.5.3.1 ACO_DEV.2.1C

The development information shall describe the purpose **and method** of **use of** each interface of the base component used in the composed TOE.

15.3.5.3.2 ACO_DEV.2.2C

The development information shall provide a high-level description of the behaviour of the base component, which supports the enforcement of the dependent component SFRs.

15.3.5.3.3 ACO_DEV.2.3C

The development information shall show correspondence between the interfaces, used in the composed TOE, of the base component and the dependent component to support the TSF of the dependent component.

15.3.5.4 Evaluator action elements

15.3.5.4.1 ACO_DEV.2.1E

The evaluator shall confirm that the information meets all requirements for content and presentation of evidence.

15.3.5.4.2 ACO_DEV.2.2E

The evaluator shall determine that the interface description provided is consistent with the reliance information provided for the dependent component.

15.3.6 ACO_DEV.3 Detailed evidence of design

Dependencies: ACO_REL.2 Reliance information

15.3.6.1 Objectives

A description of the interfaces in the base component, on which the dependent component relies, is required. This is examined to determine whether or not it is consistent with the description of interfaces on which the dependent component relies, as provided in the reliance information.

The interface description of the architecture of the base component is provided to enable the evaluator to determine whether or not that interface formed part of the TSF of the base component.

15.3.6.2 Developer action elements

15.3.6.2.1 ACO_DEV.3.1D

The developer shall provide development information for the base component.

15.3.6.3 Content and presentation elements

15.3.6.3.1 ACO_DEV.3.1C

The development information shall describe the purpose and method of use of each interface of the base component used in the composed TOE.

15.3.6.3.2 ACO_DEV.3.2C

The development information shall identify the subsystems of the base component that provide interfaces of the base component used in the composed TOE.

15.3.6.3.3 ACO_DEV.3.3C

The development information shall provide a high-level description of the behaviour of the base component **subsystems,** which **support** the enforcement of the dependent component SFRs.

15.3.6.3.4 ACO_DEV.3.4C

The development information shall provide a mapping from the interfaces to the subsystems of the base component.

15.3.6.3.5 ACO_DEV.3.5C

The development information shall show correspondence between the interfaces, used in the composed TOE, of the base component and the dependent component to support the TSF of the dependent component.

15.3.6.4 Evaluator action elements

15.3.6.4.1 ACO_DEV.3.1E

The evaluator shall confirm that the information meets all requirements for content and presentation of evidence.

15.3.6.4.2 ACO_DEV.3.2E

The evaluator shall determine that the interface description provided is consistent with the reliance information provided for the dependent component.

15.4 Reliance of dependent component (ACO_REL)

15.4.1 Objectives

The purpose of this family is to provide evidence that describes the reliance that a dependent component has upon the base component. This information is useful to persons responsible for integrating the component with other evaluated IT components to form the composed TOE, and for providing insight into the security properties of the resulting composition.

This provides a description of the interface between the dependent and base components of the composed TOE that may not have been analysed during evaluation of the individual components, as the interfaces were not TSFIs of the individual component TOEs.

15.4.2 Component levelling

The components in this family are levelled according to the amount of detail provided in the description of the reliance by the dependent component upon the base component.

15.4.3 Application notes

The Reliance of dependent component (ACO_REL) family considers the interactions between the components where the dependent component relies upon a service from the base component to support the operation of security functionality of the dependent component. The interfaces into these services of the base component may not have been considered during evaluation of the base component because the service in the base component was not considered security-relevant in the component evaluation, either because of the inherent purpose of the service (e.g., adjust type font) or because associated ISO/IEC 15408 SFRs are not being claimed in the base component's ST (e.g. the login interface when no FIA: Identification and authentication SFRs are claimed). These interfaces into the base component are often viewed as functional interfaces in the evaluation of the base component, and are in addition to the security interfaces (TSFI) considered in the functional specification.

In summary, the TSFIs described in the functional specification only include the calls made into a TSF by external entities and responses to those calls. Calls made by a TSF, which were not explicitly considered during evaluation of the components, are described by the reliance information provided to satisfy Reliance of dependent component (ACO_REL).

15.4.4 ACO_REL.1 Basic reliance information

Dependencies: No dependencies.

15.4.4.1 Developer action elements

15.4.4.1.1 ACO_REL.1.1D

The developer shall provide reliance information of the dependent component.

15.4.4.2 Content and presentation elements

15.4.4.2.1 ACO_REL.1.1C

The reliance information shall describe the functionality of the base component hardware, firmware and/or software that is relied upon by the dependent component TSF.

15.4.4.2.2 ACO_REL.1.2C

The reliance information shall describe all interactions through which the dependent component TSF requests services from the base component.

15.4.4.2.3 ACO_REL.1.3C

The reliance information shall describe how the dependent TSF protects itself from interference and tampering by the base component.

15.4.4.3 Evaluator action elements

15.4.4.3.1 ACO_REL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.4.5 ACO_REL.2 Reliance information

Dependencies: No dependencies.

15.4.5.1 Developer action elements

15.4.5.1.1 ACO_REL.2.1D

The developer shall provide reliance information of the dependent component.

15.4.5.2 Content and presentation elements

15.4.5.2.1 ACO_REL.2.1C

The reliance information shall describe the functionality of the base component hardware, firmware and/or software that is relied upon by the dependent component TSF.

15.4.5.2.2 ACO_REL.2.2C

The reliance information shall describe all interactions through which the dependent component TSF requests services from the base component.

15.4.5.2.3 ACO_REL.2.3C

The reliance information shall describe each interaction in terms of the interface used and the return values from those interfaces.

15.4.5.2.4 ACO_REL.2.4C

The reliance information shall describe how the dependent TSF protects itself from interference and tampering by the base component.

15.4.5.3 Evaluator action elements

15.4.5.3.1 ACO_REL.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.5 Composed TOE testing (ACO_CTT)

15.5.1 Objectives

This family requires that testing of composed TOE and testing of the base component, as used in the composed TOE, is performed.

15.5.2 Component levelling

The components in this family are levelled on the basis of increasing rigour of interface testing and increasing rigour of the analysis of the sufficiency of the tests to demonstrate that the composed TSF operates in accordance with the reliance information and the composed TOE SFRs.

15.5.3 Application notes

There are two distinct aspects of testing associated with this family:

a) testing of the interfaces between the base component and the dependent component, which the dependent component rely upon for enforcement of security functionality, to demonstrate their compatibility;

b) testing of the composed TOE to demonstrate that the TOE behaves in accordance with the SFRs for the composed TOE.

If the test configurations used during evaluation of the dependent component included use of the base component as a “platform” and the test analysis sufficiently demonstrates that the TSF behaves in accordance with the SFRs, the developer need perform no further testing of the composed TOE functionality. However, if the base component was not used in the testing of the dependent component, or the configuration of either component varied, then the developer is to perform testing of the composed TOE. This may take the form of repeating the dependent component developer testing of the dependent component, provided this adequately demonstrates the composed TOE TSF behaves in accordance with the SFRs.

The developer is to provide evidence of testing the base component interfaces used in the composition. The operation of base component TSFIs would have been tested as part of the ATE: Tests activities during evaluation of the base component. Therefore, provided the appropriate interfaces were included within the test sample of the base component evaluation and it was determined in Composition rationale (ACO_COR) that the base component is operating in accordance with the base component evaluated configuration, with all security functionality required by the dependent component included in the TSF, the evaluator action ACO_CTT.1.1E may be met through reuse of the base component ATE: Tests verdicts.

If this is not the case, the base component interfaces used relevant to the composition that are affected by any variations to the evaluated configuration and any additional security functionally will be tested to ensure they demonstrate the expected behaviour. The expected behaviour to be tested is that described in the reliance information (Reliance of dependent component (ACO_REL) evidence).

15.5.4 ACO_CTT.1 Interface testing

Dependencies: ACO_REL.1 Basic reliance information

ACO_DEV.1 Functional Description

15.5.4.1 Objectives

The objective of this component is to ensure that each interface of the base component, on which the dependent component relies, is tested.

15.5.4.2 Developer action elements

15.5.4.2.1 ACO_CTT.1.1D

The developer shall provide composed TOE test documentation.

15.5.4.2.2 ACO_CTT.1.2D

The developer shall provide base component interface test documentation.

15.5.4.2.3 ACO_CTT.1.3D

The developer shall provide the composed TOE for testing.

15.5.4.2.4 ACO_CTT.1.4D

The developer shall provide an equivalent set of resources to those that were used in the base component developer's functional testing of the base component.

15.5.4.3 Content and presentation elements

15.5.4.3.1 ACO_CTT.1.1C

The composed TOE and base component interface test documentation shall consist of test plans, expected test results and actual test results.

15.5.4.3.2 ACO_CTT.1.2C

The test documentation from the developer execution of the composed TOE tests shall demonstrate that the TSF behaves as specified.

15.5.4.3.3 ACO_CTT.1.3C

The test documentation from the developer execution of the base component interface tests shall demonstrate that the base component interface relied upon by the dependent component behaves as specified.

15.5.4.3.4 ACO_CTT.1.4C

The base component shall be suitable for testing.

15.5.4.4 Evaluator action elements

15.5.4.4.1 ACO_CTT.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.5.4.4.2 ACO_CTT.1.2E

The evaluator shall execute a sample of test in the test documentation to verify the developer test results.

15.5.4.4.3 ACO_CTT.1.3E

The evaluator shall test a subset of the TSF interfaces of the composed TOE to confirm that the composed TSF operates as specified.

15.5.5 ACO_CTT.2 Rigorous interface testing

Dependencies: ACO_REL.2 Reliance information

ACO_DEV.2 Basic evidence of design

15.5.5.1 Objectives

The objective of this component is to ensure that each interface of the base component, on which the dependent component relies, is tested.

15.5.5.2 Developer action elements

15.5.5.2.1 ACO_CTT.2.1D

The developer shall provide composed TOE test documentation.

15.5.5.2.2 ACO_CTT.2.2D

The developer shall provide base component interface test documentation.

15.5.5.2.3 ACO_CTT.2.3D

The developer shall provide the composed TOE for testing.

15.5.5.2.4 ACO_CTT.2.4D

The developer shall provide an equivalent set of resources to those that were used in the base component developer's functional testing of the base component.

15.5.5.3 Content and presentation elements

15.5.5.3.1 ACO_CTT.2.1C

The composed TOE and base component interface test documentation shall consist of test plans, expected test results and actual test results.

15.5.5.3.2 ACO_CTT.2.2C

The test documentation from the developer execution of the composed TOE tests shall demonstrate that the TSF behaves as specified **and is complete.**

15.5.5.3.3 ACO_CTT.2.3C

The test documentation from the developer execution of the base component interface tests shall demonstrate that the base component interface relied upon by the dependent component behaves as specified **and is complete.**

15.5.5.3.4 ACO_CTT.2.4C

The base component shall be suitable for testing.

15.5.5.4 Evaluator action elements

15.5.5.4.1 ACO_CTT.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.5.5.4.2 ACO_CTT.2.2E

The evaluator shall execute a sample of test in the test documentation to verify the developer test results.

15.5.5.4.3 ACO_CTT.2.3E

The evaluator shall test a subset of the TSF interfaces of the composed TOE to confirm that the composed TSF operates as specified.

15.6 Composition vulnerability analysis (ACO_VUL)

15.6.1 Objectives

This family calls for an analysis of vulnerability information available in the public domain and of vulnerabilities that may be introduced as a result of the composition.

15.6.2 Component levelling

The components in this family are levelled on the basis of increasing scrutiny of vulnerability information from the public domain and independent vulnerability analysis.

15.6.3 Application notes

The developer will provide details of any residual vulnerabilities reported during evaluation of the components. These may be gained from the component developers or evaluation reports for the components. These will be used as inputs into the evaluator's vulnerability analysis of the composed TOE in the operational environment.

The operational environment of the composed TOE is examined to ensure that the assumptions and objectives for the component operational environment (specified in each component ST) are satisfied in the composed TOE. An initial analysis of the consistency of assumptions and objectives between the components and the composed TOE STs will have been performed during the conduct of the ASE activities for the composed TOE. However, this analysis is revisited with the knowledge acquired during the ACO_REL, ACO_DEV and the ACO_COR activities to ensure that, for example, assumptions of the dependent component that were addressed by the environment in the dependent component ST are not reintroduced as a result of composition (i.e. that the base component adequately addresses the assumptions of the dependent component ST in the composed TOE).

A search by the evaluator for issues in each component will identify potential vulnerabilities reported in the public domain since completion of the evaluation of the components. Any potential vulnerabilities will then be subject to testing.

If the base component used in the composed TOE has been the subject of assurance continuity activities since certification, the evaluator will consider during the composed TOE vulnerability analysis activities the changes made in base component.

15.6.4 ACO_VUL.1 Composition vulnerability review

Dependencies: ACO_DEV.1 Functional Description

15.6.4.1 Developer action elements

15.6.4.1.1 ACO_VUL.1.1D

The developer shall provide the composed TOE for testing.

15.6.4.2 Content and presentation elements

15.6.4.2.1 ACO_VUL.1.1C

The composed TOE shall be suitable for testing.

15.6.4.3 Evaluator action elements

15.6.4.3.1 ACO_VUL.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.6.4.3.2 ACO_VUL.1.2E

The evaluator shall perform an analysis to determine that any residual vulnerabilities identified for the base and dependent components are not exploitable in the composed TOE in its operational environment.

15.6.4.3.3 ACO_VUL.1.3E

The evaluator shall perform a search of public domain sources to identify possible vulnerabilities arising from use of the base and dependent components in the composed TOE operational environment.

15.6.4.3.4 ACO_VUL.1.4E

The evaluator shall conduct penetration testing, based on the identified vulnerabilities, to demonstrate that the composed TOE is resistant to attacks by an attacker with basic attack potential.

15.6.5 ACO_VUL.2 Composition vulnerability analysis

Dependencies: ACO_DEV.2 Basic evidence of design

15.6.5.1 Developer action elements

15.6.5.1.1 ACO_VUL.2.1D

The developer shall provide the composed TOE for testing.

15.6.5.2 Content and presentation elements

15.6.5.2.1 ACO_VUL.2.1C

The composed TOE shall be suitable for testing.

15.6.5.3 Evaluator action elements

15.6.5.3.1 ACO_VUL.2.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.6.5.3.2 ACO_VUL.2.2E

The evaluator shall perform an analysis to determine that any residual vulnerabilities identified for the base and dependent components are not exploitable in the composed TOE in its operational environment.

15.6.5.3.3 ACO_VUL.2.3E

The evaluator shall perform a search of public domain sources to identify possible vulnerabilities arising from use of the base and dependent components in the composed TOE operational environment.

15.6.5.3.4 ACO_VUL.2.4E

The evaluator shall perform an independent vulnerability analysis of the composed TOE, using the guidance documentation, reliance information and composition rationale to identify potential vulnerabilities in the composed TOE.

15.6.5.3.5 ACO_VUL.2.5E

The evaluator shall conduct penetration testing, based on the identified vulnerabilities, to demonstrate that the composed TOE is resistant to attacks by an attacker with basic attack potential.

15.6.6 ACO_VUL.3 Enhanced-Basic Composition vulnerability analysis

Dependencies: ACO_DEV.3 Detailed evidence of design

15.6.6.1 Developer action elements

15.6.6.1.1 ACO_VUL.3.1D

The developer shall provide the composed TOE for testing.

15.6.6.2 Content and presentation elements

15.6.6.2.1 ACO_VUL.3.1C

The composed TOE shall be suitable for testing.

15.6.6.3 Evaluator action elements

15.6.6.3.1 ACO_VUL.3.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

15.6.6.3.2 ACO_VUL.3.2E

The evaluator shall perform an analysis to determine that any residual vulnerabilities identified for the base and dependent components are not exploitable in the composed TOE in its operational environment.

15.6.6.3.3 ACO_VUL.3.3E

The evaluator shall perform a search of public domain sources to identify possible vulnerabilities arising from use of the base and dependent components in the composed TOE operational environment.

15.6.6.3.4 ACO_VUL.3.4E

The evaluator shall perform an independent vulnerability analysis of the composed TOE, using the guidance documentation, reliance information and composition rationale to identify potential vulnerabilities in the composed TOE.

15.6.6.3.5 ACO_VUL.3.5E

The evaluator shall conduct penetration testing, based on the identified vulnerabilities, to demonstrate that the composed TOE is resistant to attacks by an attacker with **Enhanced-Basic** attack potential.

Annex A
(informative)

Development (ADV)

This annex contains ancillary material to further explain and provide additional examples for the topics brought up in families of the ADV: Development class.

A.1ADV_ARC: Supplementary material on security architectures

A security architecture is a set of properties that the TSF exhibits; these properties include self-protection, domain separation, and non-bypassability. Having these properties provides a basis of confidence that the TSF is providing its security services. This annex provides additional material on these properties, as well as discussion on contents of a security architecture description.

The remainder of this subclause first explains these properties, then discusses the kinds of information that are needed to describe how the TSF exhibits those properties.

A.1.1 Security architecture properties

Self-protection refers to the ability of the TSF to protect itself from manipulation from external entities that may result in changes to the TSF. Without these properties, the TSF might be disabled from performing its security services.

It is oftentimes the case that a TOE uses services or resources supplied by other IT entities in order to perform its functions (e.g. an application that relies upon its underlying operating system). In these cases, the TSF does not protect itself entirely on its own, because it depends on the other IT entities to protect the services it uses.

Domain separation is a property whereby the TSF creates separate *security domains* for each untrusted active entity to operate on its resources, and then keeps those domains separated from one another so that no entity can run in the domain of any other. For example, an operating system TOE supplies a domain (address space, per-process environment variables) for each process associated with untrusted entities.

For some TOEs such domains do not exist because all of the actions of the untrusted entities are brokered by the TSF. A packet-filter firewall is an example of such a TOE, where there are no untrusted entity domains; there are only data structures maintained by the TSF. The existence of domains, then, is dependant upon 1) the type of TOE and 2) the SFRs levied on the TOE. In the cases where the TOE does provide domains for untrusted entities, this family requires that those domains are isolated from one another such that untrusted entities in one domain are prevented from tampering (affecting without brokering by the TSF) from another untrusted entity's domain.

Non-bypassability is a property that the security functionality of the TSF (as specified by the SFRs) is always invoked and cannot be circumvented when appropriate for that specific mechanism. For example, if access control to files is specified as a capability of the TSF via an SFR, there must be no interfaces through which files can be accessed without invoking the TSF's access control mechanism (an interface through which a raw disk access takes place might be an example of such an interface).

As is the case with self-protection, the very nature of some TOEs might depend upon their environments to play a role in non-bypassability of the TSF. For example, a security application TOE requires that it be invoked by the underlying operating system. Similarly, a firewall depends upon the fact that there are no direct connections between the internal and external networks and that all traffic between them must go through the firewall.

A.1.2 Security architecture descriptions

The security architecture description explains how the properties described above are exhibited by the TSF. It describes how domains are defined and how the TSF keeps them separate. It describes what prevents untrusted processes from getting to the TSF and modifying it. It describes what ensures that all resources under the TSF's control are adequately protected and that all actions related to the SFRs are mediated by the TSF. It explains any role the environment plays in any of these (e.g. presuming it gets correctly invoked by its underlying environment, how are its security functions invoked?).

The security architecture description presents the TSF's properties of self-protection, domain separation, and non-bypassability in terms of the decomposition descriptions. The level of this description is commensurate with the TSF description required by the ADV_FSP, ADV_TDS and ADV_IMP requirements that are being claimed. For example, if ADV_FSP is the only TSF description available, it would be difficult to provide any meaningful security architecture description because none of the details of any internal workings of the TSF would be available.

However, if the TOE design were also available, even at the most basic level (ADV_TDS.1), there would be some information available concerning the subsystems that make up the TSF, and there would be a description of how they work to implement self-protection, domain separation, and non-bypassability. For example, perhaps all user interaction with the TOE is constrained through a process that acts on that user's behalf, adopting all of the user's security attributes; the security architecture description would describe how such a process comes into being, how the process's behaviour is constrained by the TSF (so it cannot corrupt the TSF), how all actions of that process are mediated by the TSF (thereby explaining why the TSF cannot be bypassed), etc.

If the available TOE design is more detailed (e.g. at the modular level), or the implementation representation is also available, then the security architecture description would be correspondingly more detailed, explaining how the user's process communicate with the TSF processes, how different requests are processed by the TSF, what parameters are passed, what programmatic protections (buffer overflow prevention, parameter bounds checking, time of check/time of use checking, etc.) are in place. Similarly, a TOE whose ST claimed the ADV_IMP component would go into implementation-specific detail.

The explanations provided in the security architecture description are expected to be of sufficient detail that one would be able to test their accuracy. That is, simple assertions (e.g. "The TSF keeps domains separate") provide no useful information to convince the reader that the TSF does indeed create and separate domains.

A.1.2.1 Domain Separation

In cases where the TOE exhibits domain separation entirely on its own, there would be a straightforward description of how this is attained. The security architecture description would explain the different kinds of domains that are defined by the TSF, how they are defined (i.e. what resources are allocated to each domain), how no resources are left unprotected, and how the domains are kept separated so that active entities in one domain cannot tamper with resources in another domain.

For cases where the TOE depends upon other IT entities to play a role in domain separation, that sharing of roles must be made clear. For example, a TOE that is solely application software relies upon the underlying operating system to correctly instantiate the domains that the TOE defines; if the TOE defines separate processing space, memory space, etc, for each domain, it depends upon the underlying operating system to operate correctly and benignly (e.g. allow the process to execute only in the execution space that is requested by the TOE software).

For example, mechanisms that implement domain separation (e.g., memory management, protected processing modes provided by the hardware, etc.) would be identified and described. Or, the TSF might implement software protection constructs or coding conventions that contribute to implementing separation of software domains, perhaps by delineating user address space from system address space.

The vulnerability analysis and testing (see AVA_VAN) activities will likely include attempts to defeat the described TSF domain separation through the use of monitoring or direct attack the TSF.

A.1.2.2 TSF Self-protection

In cases where the TOE exhibits self-protection entirely on its own, there would be a straightforward description of how this self-protection is attained. Mechanisms that provide domain separation to define a TSF domain that is protected from other (user) domains would be identified and described.

For cases where the TOE depends upon other IT entities to play a role in protecting itself, that sharing of roles must be made clear. For example, a TOE that is solely application software relies upon the underlying operating system to operate correctly and benignly; the application cannot protect itself against a malicious operating system that subverts it (for example, by overwriting its executable code or TSF data).

The security architecture description also covers how user input is handled by the TSF in such a way that the TSF does not subject itself to being corrupted by that user input. For example, the TSF might implement the notion of privilege and protect itself by using privileged-mode routines to handle user data. The TSF might make use of processor-based separation mechanisms (e.g. privilege levels or rings) to separate TSF code and data from user code and data. The TSF might implement software protection constructs or coding conventions that contribute to implementing separation of software, perhaps by delineating user address space from system address space.

For TOEs that start up in a low-function mode (for example, a single-user mode accessible only to installers or administrators) and then transition to the evaluated secure configuration (a mode whereby untrusted users are able to login and use the services and resources of the TOE), the security architecture description also includes an explanation of how the TSF is protected against this initialisation code that does not run in the evaluated configuration. For such TOEs, the security architecture description would explain what prevents those services that should be available only during initialisation (e.g. direct access to resources) from being accessible in the evaluated configuration. It would also explain what prevents initialisation code from running while the TOE is in the evaluated configuration.

There must also be an explanation of how the trusted initialisation code will maintain the integrity of the TSF (and of its initialisation process) such that the initialisation process is able to detect any modification that would result in the TSF being spoofed into believe it was in an initial secure state.

The vulnerability analysis and testing (see AVA_VAN) activities will likely include attempts to defeat the described TSF self protection through the use of tampering, direct attack, or monitoring of the TSF.

A.1.2.3 TSF Non-Bypassability

The property of non-bypassability is concerned with interfaces that permit the bypass of the enforcement mechanisms. In most cases this is a consequence of the implementation, where if a programmer is writing an interface that accesses or manipulates an object, it is that programmer's responsibility to use interfaces that are part of the SFR enforcement mechanism for the object and not to try to circumvent those interfaces. For the description pertaining to non-bypassability, then, there are two broad areas that have to be covered.

The first consists of those interfaces to the SFR-enforcement. The property for these interfaces is that they contain no operations or modes that allow them to be used to bypass the TSF. It is likely that the evidence for ADV_FSP and ADV_TDS can be used in large part to make this determination. Because non-bypassability is the concern, if only certain operations available through these TSFIs are documented (because they are SFR-enforcing) and others are not, the developer should consider whether additional information (to that presented in ADV_FSP and ADV_TDS) is necessary to make a determination that the SFR-supporting and SFR-non-interfering operations of the TSFI do not afford an untrusted entity the ability to bypass the policy being enforced. If such information is necessary, it is included in the security architecture description.

The second area of non-bypassability is concerned with those interfaces whose interactions are not associated with SFR-enforcement. Depending on the ADV_FSP and ADV_TDS components claimed, some information about these interfaces may or may not exist in the functional specification and TOE design documentation. The information presented for such interfaces (or groups of interfaces) should be sufficient so that a reader can make a determination (at the level of detail commensurate with the rest of the evidence supplied in the ADV: Development class) that the enforcement mechanisms cannot be bypassed.

The property that the security functionality cannot be bypassed applies to all security functionality equally. That is, the design description should cover objects that are protected under the SFRs (e.g. FDP_* components) and functionality (e.g., audit) that is provided by the TSF. The description should also identify the interfaces that are associated with security functionality; this might make use of the information in the functional specification. This description should also describe any design constructs, such as object managers, and their method of use. For instance, if routines are to use a standard macro to produce an audit record, this convention is a part of the design that contributes to the non-bypassability of the audit mechanism. It is important to note that *non-bypassability* in this context is not an attempt to answer the question “could a part of the TSF implementation, if malicious, bypass the security functionality”, but rather to document how the implementation does not bypass the security functionality.

The vulnerability analysis and testing (see AVA_VAN) activities will likely include attempts to defeat the described non-bypassability by circumventing the TSF.

A.2 ADV_FSP: Supplementary material on functional specification

The purpose in specifying the TSFIs is to provide the necessary information to conduct testing; without knowing the possible means interact with the TSF, one cannot adequately test the behaviour of the TSF.

There are two parts to specifying the TSFIs: identifying them and describing them. Because of the diversity of possible TOEs, and of different TSFs therein, there is no standard set of interfaces that constitute “TSFIs”. This annex provides guidance on the factors that determine which interfaces are TSFIs.

A.2.1 Non-TSF part of the TOE

The TSF comprises all parts of the TOE the user has to rely on in order to trust the security functionality.

To say it in other words: Those parts of the TOE that do not belong to the TSF can be modified by an attacker without any impact on the TOE security functionality. If this isn't the case, these parts of the TOE have to be included in the TSF.

If the TSF and the TSF implementation are defined then it is clear whether there exist further parts of the TOE which can be classified as non-TSF parts of the TOE. Such parts do not have to be part of the TSF but they are still part of the TOE.

The relationship between TSF and non-TSF parts of TOE is given by their definitions and the ARC properties as follows:

- non-TSF parts do not bypass the TSF and
- parts of the TSF protects themselves against tampering.

A subsystem of the TOE which is not part of the TSF has to fulfil the following condition (described as a rule of thumb[9]): The subsystem must not have any security impact of the TOE even if it were substituted by an attacker.

Therefore between the Non-TSF parts and the TSF parts it seems that some kind of "separation mechanism" is advisable[10] because such "separation mechanism" may build the basis for the assessment that there is no impact on the TSF parts from the Non-TSF parts possible.

Such "separation mechanism" could be implemented by the security architecture or by an explicitly realised part of the implementation (e.g. a firewall between TSF and Non-TSF parts of the TOE).

The analysis of the "separation mechanism" is then subject of the vulnerability assessment because it must withstand attacks by an attacker of the respective strength according to the VAN level of the evaluation.

The developer shall provide evidence for non-bypassability and self-protection in its security architecture description and the evaluator shall analyse this evidence in subactivity for ADV_ARC.1 and assess the effectiveness in the vulnerability assessment.

The goal of TOE design documentation is to provide sufficient information to determine the TSF boundary, and to describe how the TSF implements the SFR. Further attention is needed by the fact that the family ADV_TDS requires only identification of the non-TSF subsystems of the TOE. No interface description is provided for these subsystems in ADV_FSP or ADV_TDS. SFR non-interference of these subsystems is assumed but not demonstrated by the developer and not examined in details by the evaluator. However from the TOE design point of view this is not that important as long as the above mentioned separation mechanism is in place and the vulnerability assessment confirms that it is

[9] This rule is only valid to some extent because the actual requirement "The Non-TSF part must not bypass the TSF." is not that strong as the given rule of thumb.

[10] The "separation mechanism" is only an proposal here. The developer is free to provide evidence using other kind of security implementation as long as the requirement showing the non-bypassablity for the TSF part of the TOE from the non-TSF part of the TOE is fulfilled.

strong enough. Therefore this "separation mechanism" implements the TSF or enforces ARC properties as security feature. But non-bypassability may be enforced by "pure architecture properties" as well.

Parts of the TOE classified as non-TSF must not provide means to bypass the TSF (no matter whether a valid user or even an attacker makes uses of those parts) and must not contribute to the TSF. It is important that the developer provides clear evidence and demonstrate how this requirement is fulfilled.

Therefore the developer shall demonstrate and the evaluator shall examine that the TOE identification of subsystems as non-TSF (cf. ADV_TDS.x.1) is correct and consequently no detailed description of these subsystems is necessary. The evaluator examination shall include the ARC properties non-bypassability and self-protection being described in the ADV_ARC documentation provided by the developer (see the paragraphs above).

A.2.2 Determining the TSFI

In order to identify the interfaces to the TSF, the parts of the TOE that make up the TSF must first be identified. This identification is actually a part of the TOE design (ADV_TDS) analysis, but is also performed implicitly (through identification and description of the TSFI) by the developer in cases where TOE design (ADV_TDS) is not included in the assurance package. In this analysis, a portion of the TOE must be considered to be in the TSF if it contributes to the satisfaction of an SFR in the ST (in whole or in part). This includes, for example, everything in the TOE that contributes to TSF run-time initialisation, such as software that runs prior to the TSF being able to protect itself because enforcement of the SFRs has not yet begun (e.g., while booting up). Also included in the TSF are all parts of the TOE that contribute to the architectural principles of TSF self-protection, domain separation, and non-bypassability (see Security Architecture (ADV_ARC)).

Once the TSF has been defined, the TSFI are identified. The TSFI consists of all means by which external entities (or subjects in the TOE but outside of the TSF) supply data to the TSF, receive data from the TSF and invoke services from the TSF. These service invocations and responses are the means of crossing the TSF boundary. While many of these are readily apparent, others might not be as obvious. The question that should be asked when determining the TSFIs is: “How can a potential attacker interact with the TSF in an attempt to subvert the SFRs?”

Therefore from the evaluation point of view it is also important whether the interface can be misused by an attacker to get access to the security functionality in order to compromise the assets protected by TSF.

Any interface of the TSF which can be potentially used by an attacker belongs to the TSFI (regardless of the further classification as SFR-enforcing, SFR-supporting or SFR-non-interfering).

It is not important whether the TSF will be accessed from outside or whether the TSF accesses the external resources (e.g. TSF calls platform or user). The only criteria is whether there is a potential interference with the TSF from outside.

The following discussions illustrate the application of the TSFI definition in different contexts.

A.2.2.1 Electrical interfaces

In TOEs such as smart cards, where the adversary has not only logical access to the TOE, but also complete physical access to the TOE, the TSF boundary is the physical boundary. Therefore, the exposed electrical interfaces are considered TSFI because their manipulation could affect the behaviour of the TSF. As such, all these interfaces (electrical contacts) need to be described: various voltages that might be applied, etc.

A.2.2.2 Network protocol stack

The TSFIs of a TOE that performs protocol processing would be those protocol layers to which a potential attacker has direct access. This need not be the entire protocol stack, but it might be.

For example, if the TOE were some sort of a network appliance that allowed potential attackers to affect every level of the protocol stack (i.e. to send arbitrary signals, arbitrary voltages, arbitrary packets, arbitrary datagrams, etc.), then the TSF boundary exists at each layer of the stack. Therefore, the functional specification would have to address every protocol at every layer of the stack.

If, however, the TOE were a firewall that protects an internal network from the Internet, a potential attacker would have no means of directly manipulating the voltages that enter the TOE; any extreme voltages would simply not be passed though the Internet. That is, the attacker would have access only to those protocols at the Internet layer or above. The TSF boundary exists at each layer of the stack. Therefore, the functional specification would have to address only those protocols at or above the Internet layer: it would describe each of the different communication layers at which the firewall is exposed in terms of what constitutes well-formed input for what might appear on the line, and the result of both well-formed and malformed inputs. For example, the description of the Internet protocol layer would describe what constitutes a well-formed IP packet and what happens when both correctly-formed and malformed packets are received. Likewise, the description of the TCP layer would describe a successful TCP connection and what happens both when successful connections are established and when connections cannot be established or are inadvertently dropped. Presuming the firewall's purpose is to filter application-level commands (like FTP or telnet), the description of the application layer would describe the application-level commands that are recognised and filtered by the firewall, as well as the results of encountering unknown commands.

The descriptions of these layers would likely reference published communication standards (telnet, FTP, TCP, etc.) that are used, noting which user-defined options are chosen.

A.2.2.3 Wrappers

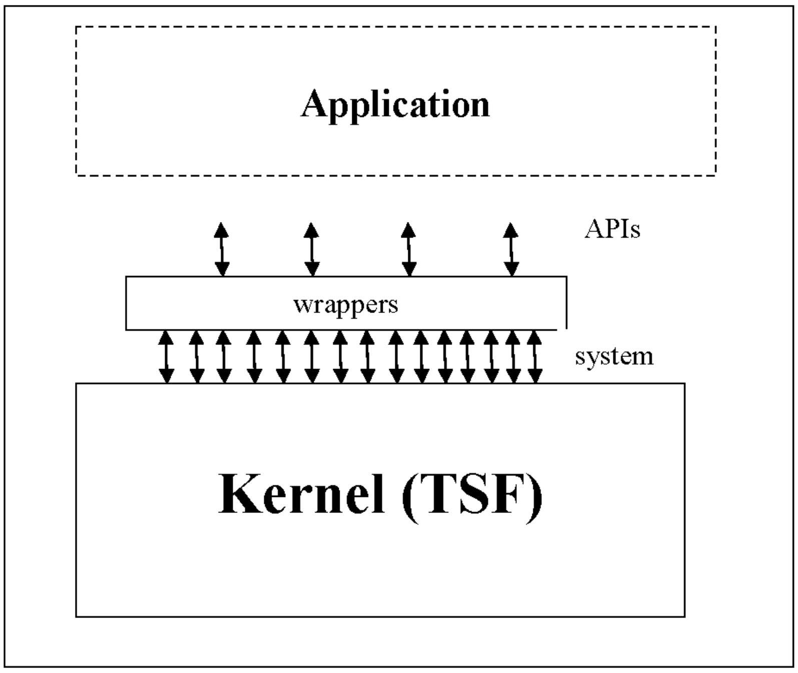

Figure A.1 — Wrappers

"Wrappers" translate complex series of interactions into simplified common services, such as when Operating Systems create APIs for use by applications (as shown in Figure A.1). Whether the TSFIs would be the system calls or the APIs depends upon what is available to the application: if the application can use the system calls directly, then the system calls are the TSFIs. If, however, there were something that prohibits their direct use and requires all communication through the APIs, then the APIs would be the TSFIs.

A Graphical User interface is similar: it translates between machine-understandable commands and user-friendly graphics. Similarly, the TSFIs would be the commands if users have access to them, or the graphics (pull-down menus, check-boxes, text fields) if the users are constrained to using them.

It is worth noting that, in both of these examples, if the user is prohibited from using the more primitive interfaces (i.e. the system calls or the commands), the description of this restriction and of its enforcement would be included in the Security Architecture Description (see A.1). Also, the wrapper would be part of the TSF.

A.2.2.4 Inaccessible interfaces

For a given TOE, not all of the interfaces may be *accessible*. That is, the security objectives for the operational environment (in the Security Target) may prevent access to these interfaces or limit access in such a way that they are practically inaccessible. Such interfaces would not be considered TSFIs. Some examples:

a) If the security objectives for the operational environment for the stand-alone firewall state that "the firewall will be operational in a server room environment to which only trusted and trained personnel will have access, and which will be equipped with an interruptible power supply (against power failure)", physical and power interfaces will not be accessible, since trusted and trained personnel will not attempt to dismantle the firewall and/or disable its power supply.

b) If the security objectives for the operational environment for the software firewall (application) state that “the OS and the hardware will provide a security domain for the application free from tampering by other programs”, the interfaces through which the firewall can be accessed by other applications on the OS (e.g. deleting or modifying the firewall executable, direct reading or writing to the memory space of the firewall) will not be accessible, since the OS/hardware part of the operational environment makes this interface inaccessible.

c) If the security objectives for the operational environment for the software firewall additionally state that the OS and hardware will faithfully execute the commands of the TOE, and will not tamper with the TOE in any manner, interfaces through which the firewall obtains primitive functionality from the OS and hardware (executing machine code instructions, OS APIs, such as creating, reading, writing or deleting files, graphical APIs etc.) will not be accessible, since the OS/hardware are the only entities that can access that interface, and they are completely trusted.

For all of these examples, these inaccessible interfaces would not be TSFIs.

A.2.3 Example: A complex DBMS

Figure A.2 illustrates a complex TOE: a database management system that relies on hardware and software that is outside the TOE boundary (referred to as the *IT environment* in the rest of this discussion). To simplify this example, the TOE is identical to the TSF. The shaded boxes represent the TSF, while the unshaded boxes represent IT entities in the environment. The TSF comprises the database engine and management GUIs (represented by the box labelled *DB*) and a kernel module that runs as part of the OS that performs some security function (represented by the box labelled *PLG*). The TSF kernel module has entry points defined by the OS specification that the OS will call to invoke some function (this could be a device driver, or an authentication module, etc.). The key is that this pluggable kernel module is providing security services specified by functional requirements in the ST.

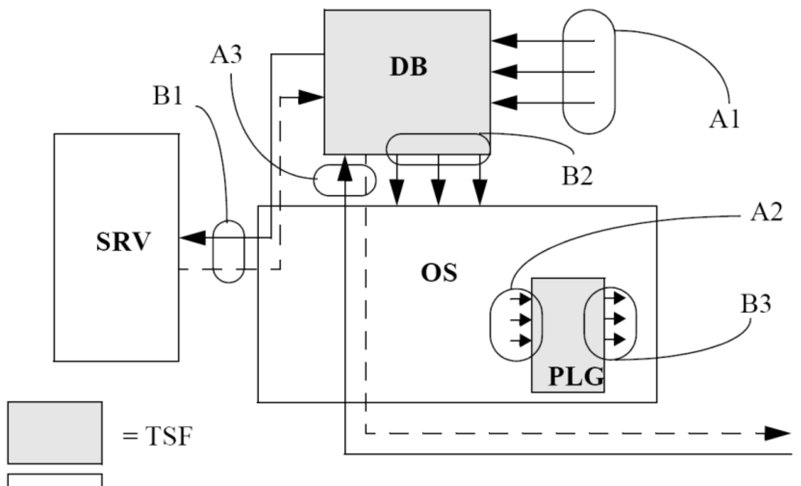

Figure A.2 — Interfaces in a DBMS system

The IT environment consists of the operating system itself (represented by the box labelled *OS*), as well as an external server (labelled *SRV*). This external server, like the OS, provides a service that the TSF depends on, and thus needs to be in the IT environment. Interfaces in the figure are labelled *Ax* for TSFI, and *Bx* for other interfaces that would be documented in ACO: Composition. Each of these groups of interfaces is now discussed.

Interface group A1 represents the most obvious set of TSFI. These are interfaces used by users to directly access the database and its security functionality and resources.

Interface group A2 represent the TSFI that the OS invokes to obtain the functionality provided by the pluggable module. These are contrasted with interface group B3, which represent calls that the pluggable module makes to obtain services from the IT environment.

Interface group A3 represent TSFI that pass through the IT environment. In this case, the DBMS communicates over the network using a proprietary application-level protocol. While the IT environment is responsible for providing various supporting protocols (e.g., Ethernet, IP, TCP), the application layer protocol that is used to obtain services from the DBMS is a TSFI and must be documented as such. The dotted line indicates return values/services from the TSF over the network connection.

The interfaces labelled *Bx* represent interfaces to functionality in the IT Environment. These interfaces are not TSFI and need only be discussed and analysed when the TOE is being used in a composite evaluation as part of the activities associated with the ACO class.

A.2.4 Example Functional Specification

The Example firewall is used between an internal network and an external network. It verifies the source address of data received (to ensure that external data is not attempting to masquerade as originating from the internal data); if it detects any such attempts, it saves the offending attempt to the audit log. The administrator connects to the firewall by establishing a telnet connection to the firewall from the internal network. Administrator actions consist of authenticating, changing passwords, reviewing the audit log, and setting or changing the addresses of the internal and external networks.

The Example firewall presents the following interfaces to the internal network:

a) IP datagrams

b) Administrator Commands

and the following interfaces to the external network:

a) IP datagrams

Interfaces Descriptions: IP Datagrams

The datagrams are in the format specified by RFC 791.

- Purpose - to transmit blocks of data (“datagrams”) from source hosts to destination hosts identified by fixed length addresses; also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through small-packet networks.
- Method of Use - they arrive from the lower-level (e.g. data link) protocol.
- Parameters - the following fields of the IP datagram header: source address, destination address, don't-fragment flag.
- Parameter description - [As defined by RFC 791, subclause 3.1 (“Internet Header Format”)]
- Actions - Transmits datagrams that are not masquerading; fragments large datagrams if necessary; reassembles fragments into datagrams.

- Error messages - (none). No reliability guaranteed (reliability to be provided by upper-level protocols) Undeliverable datagrams (e.g. must be fragmented for transmission, but don't-fragment flag is set) dropped.

Interfaces Descriptions: Administrator Commands

The administrator commands provide a means for the administrator to interact with the firewall. These c**ommands and responses ride atop a telnet (RFC 854) connection established from any host on the internal** network. Available commands are:

- **Passwd**
    - Purpose - sets administrator password
    - Method of Use - **Passwd** <*password*>
    - Parameters - password
    - Parameter description - value of new password
    - Actions - changes password to new value supplied. There are no restrictions.
    - Error messages - none.
- **Readaudit**
    - Purpose - presents the audit log to the administrator
    - Method of Use - **Readaudit**
    - Parameters - none
    - Parameter description - none
    - Actions - provides the text of the audit log
    - Error messages - none.
- **Setintaddr**
    - Purpose - sets the address of the internal address.
    - Method of Use - **Setintaddr** <*address*>
    - Parameters - address
    - Parameter description - first three fields of an IP address (as defined in RFC 791). For example: 123.123.123.
    - Actions - changes the internal value of the variable defining the internal network, the value of which is used to judge attempted masquerades.

    - Error messages - “address in use”: indicates the identified internal network is the same as the external network.

- **Setextaddr**
    - Purpose - sets the address of the external address
    - Method of Use - **Setextaddr** <*address*>
    - Parameters - address
    - Parameter description - first three fields of an IP address (as defined in RFC 791). For example: 123.123.123.
    - Actions - changes the internal value of the variable defining the external network.
    - Error messages - “address in use”: indicates the identified external network is the same as the internal network.

A.3 ADV_INT: Supplementary material on TSF internals

The wide variety of TOEs makes it impossible to codify anything more specific than “well-structured” or “minimum complexity”. Judgements on structure and complexity are expected to be derived from the specific technologies used in the TOE. For example, software is likely to be considered well-structured if it exhibits the characteristics cited in the software engineering disciplines.

This annex provides supplementary material on assessing the structure and complexity of procedure-based software portions of the TSF. This material is based on information readily available in software engineering literature. For other kinds of internals (e.g. hardware, non-procedural software such as object-oriented code, etc.), corresponding literature on good practises should be consulted.

A.3.1 Structure of procedural software

The structure of procedural software is traditionally assessed according to its *modularity*. Software written with a modular design aids in achieving understandability by clarifying what dependencies a module has on other modules (*coupling*) and by including in a module only tasks that are strongly related to each other (*cohesion*). The use of modular design reduces the interdependence between elements of the TSF and thus reduces the risk that a change or error in one module will have effects throughout the TOE. Its use enhances clarity of design and provides for increased assurance that unexpected effects do not occur. Additional desirable properties of modular decomposition are a reduction in the amount of redundant or unneeded code.

Minimising the amount of functionality in the TSF allows the evaluator as well as the developer to focus only on that functionality which is necessary for SFR enforcement, contributing further to understandability and further lowering the likelihood of design or implementation errors.

The incorporation of modular decomposition, layering and minimisation into the design and implementation process must be accompanied by sound software engineering considerations. A practical, useful software system will usually entail some undesirable coupling among modules, some modules that include loosely-related functions, and some subtlety or complexity in a module's design. These deviations from the ideals of modular decomposition are often deemed necessary to achieve some goal or constraint, be it related to performance, compatibility, future planned functionality, or some other factors, and may be acceptable, based on the developer's justification for them. In applying the requirements of this class, due consideration must be given to sound software engineering principles; however, the overall objective of achieving understandability must be achieved.

A.3.1.1 Cohesion

Cohesion is the manner and degree to which the tasks performed by a single software module are related to one another; types of cohesion include coincidental, communicational, functional, logical, sequential, and temporal. These types of cohesion are characterised below, listed in the order of decreasing desirability.

a) *functional* cohesion - a module with functional cohesion performs activities related to a single purpose. A functionally cohesive module transforms a single type of input into a single type of output, such as a stack manager or a queue manager.

b) *sequential* cohesion - a module with sequential cohesion contains functions each of whose output is input for the following function in the module. An example of a sequentially cohesive module is one that contains the functions to write audit records and to maintain a running count of the accumulated number of audit violations of a specified type.

c) *communicational* cohesion - a module with communicational cohesion contains functions that produce output for, or use output from, other functions within the module. An example of a communicationally cohesive module is an access check module that includes mandatory, discretionary, and capability checks.

d) *temporal* cohesion - a module with temporal cohesion contains functions that need to be executed at about the same time. Examples of temporally cohesive modules include initialisation, recovery, and shutdown modules.

e) *logical* (or *procedural*) cohesion - a module with logical cohesion performs similar activities on different data structures. A module exhibits logical cohesion if its functions perform related, but different, operations on different inputs.

f) *coincidental* cohesion - a module with coincidental cohesion performs unrelated, or loosely related, activities.

A.3.1.2 Coupling

Coupling is the manner and degree of interdependence between software modules; types of coupling include call, common and content coupling. These types of coupling are characterised below, listed in the order of decreasing desirability:

a) call: two modules are call coupled if they communicate strictly through the use of their documented function calls; examples of call coupling are data, stamp, and control, which are defined below.

   1) *data*: two modules are data coupled if they communicate strictly through the use of call parameters that represent single data items.

   2) *stamp*: two modules are stamp coupled if they communicate through the use of call parameters that comprise multiple fields or that have meaningful internal structures.

   3) *control*: two modules are control coupled if one passes information that is intended to influence the internal logic of the other.

b) *common*: two modules are common coupled if they share a common data area or a common system resource. Global variables indicate that modules using those global variables are common coupled. Common coupling through global variables is generally allowed, but only to a limited degree. For example, variables that are placed into a global area, but are used by only a single module, are inappropriately placed, and should be removed. Other factors that need to be considered in assessing the suitability of global variables are:

    1) The number of modules that modify a global variable: In general, only a single module should be allocated the responsibility for controlling the contents of a global variable, but there may be situations in which a second module may share that responsibility; in such a case, sufficient justification must be provided. It is unacceptable for this responsibility to be shared by more than two modules. (In making this assessment, care should be given to determining the module actually responsible for the contents of the variable; for example, if a single routine is used to modify the variable, but that routine simply performs the modification requested by its caller, it is the calling module that is responsible, and there may be more than one such module). Further, as part of the complexity determination, if two modules are responsible for the contents of a global variable, there should be clear indications of how the modifications are coordinated between them.

    2) The number of modules that reference a global variable: Although there is generally no limit on the number of modules that reference a global variable, cases in which many modules make such a reference should be examined for validity and necessity.

c) *content*: two modules are content coupled if one can make direct reference to the internals of the other (e.g. modifying code of, or referencing labels internal to, the other module). The result is that some or all of the content of one module are effectively included in the other. Content coupling can be thought of as using unadvertised module interfaces; this is in contrast to call coupling, which uses only advertised module interfaces.

A.3.2 Complexity of procedural software

Complexity is the measure of the decision points and logical paths of execution that code takes. Software engineering literature cites complexity as a negative characteristic of software because it impedes understanding of the logic and flow of the code. Another impediment to the understanding of code is the presence of code that is unnecessary, in that it is unused or redundant.

The use of layering to separate levels of abstraction and minimise circular dependencies further enables a better understanding of the TSF, providing more assurance that the TOE security functional requirements are accurately and completely instantiated in the implementation.

Reducing complexity also includes reducing or eliminating mutual dependencies, which pertains both to modules in a single layer and to those in separate layers. Modules that are mutually dependent may rely on one another to formulate a single result, which could result in a deadlock condition, or worse yet, a race condition (e.g., time of check vs. time of use concern), where the ultimate conclusion could be indeterminate and subject to the computing environment at the given instant in time.

Design complexity minimisation is a key characteristic of a reference validation mechanism, the purpose of which is to arrive at a TSF that is easily understood so that it can be completely analysed. (There are other important characteristics of a reference validation mechanism, such as TSF self-protection and non-bypassability; these other characteristics are covered by requirements in the ADV_ARC family.)

A.4ADV_TDS: Subsystems and Modules

This subclause provides additional guidance on the TDS family, and its use of the terms “subsystem” and “module”. This is followed by a discussion of how, as more-detailed becomes available, the requirement for the less-detailed is reduced.

A.4.1 Subsystems

Figure A.3 shows that, depending on the complexity of the TSF, the design may be described in terms of subsystems *and* modules (where subsystems are at a higher level of abstraction than modules); or it may just be described in terms of one level of abstraction (e.g., *subsystems* at lower assurance levels, *modules* at higher levels). In cases where a lower level of abstraction (modules) is presented, requirements levied on higher-level abstractions (subsystems) are essentially met by default. This concept is further elaborated in the discussion on subsystems and modules below.

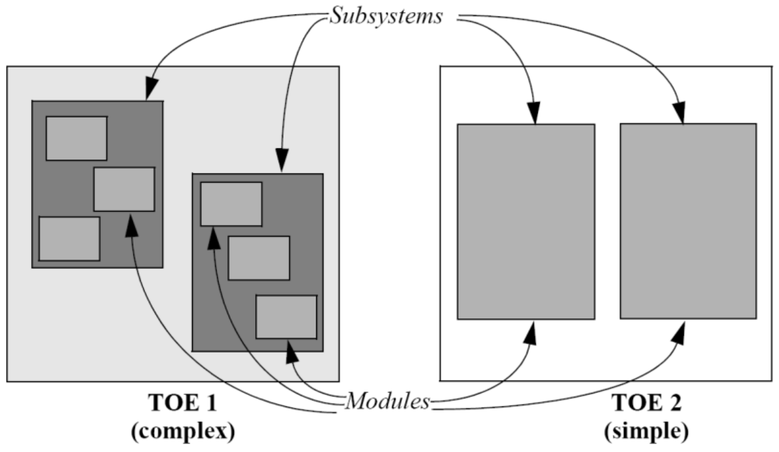

Figure A.3 — Subsystems and Modules

The developer is expected to describe the design of the TOE in terms of *subsystems*. The term “subsystem” was chosen to be specifically vague so that it could refer to units appropriate to the TOE (e.g., subsystems, modules). subsystems can even be uneven in scope, as long as the requirements for description of subsystems are met.

The first use of subsystems is to distinguish the TSF boundary; that is, the portions of the TOE that comprise the TSF. In general, a subsystem is part of the TSF if it has the capability (whether by design or implementation) to affect the correct operation of any of the SFRs. For example, for software that depends on different hardware execution modes to provide domain separation (see A.1) where SFR-enforcing code is executed in one domain, then all subsystems that execute in that domain would be considered part of the TSF. Likewise, if a server outside that domain implemented an SFR (e.g. enforced an access control policy over objects it managed), then it too would be considered part of the TSF.

The second use of subsystems is to provide a structure for describing the TSF at a level of description that, while describing how the TSF works, does not necessarily contain low-level implementation detail found in module descriptions (discussed later). subsystems are described at either a high level (lacking an abundance of implementation detail) or a detailed level (providing more insight into the implementation). The level of description provided for a subsystem is determined by the degree to which that subsystem is responsible for implementing an SFR.

An *SFR-enforcing* subsystem is a subsystem that provides mechanisms for enforcing an element of any SFR, or directly supports a subsystem that is responsible for enforcing an SFR. If a subsystem provides (implements) an SFR-enforcing TSFI, then the subsystem is SFR-enforcing.

Subsystems can also be identified as *SFR-supporting* and *SFR-non-interfering*. An SFR-supporting subsystem is one that is depended on by an SFR-enforcing subsystem in order to implement an SFR, but does not play as direct a role as an SFR-enforcing subsystem. An SFR-non-interfering subsystem is one that is not depended upon, in either a supporting or enforcing role, to implement an SFR.

A.4.2 Modules

A module is generally a relatively small architectural unit that can be characterised in terms of the properties discussed in TSF internals (ADV_INT). When both ADV_TDS.3 Basic modular design (or above) requirements and TSF internals (ADV_INT) requirements are present in a PP or ST, a “module” in terms of the TOE design (ADV_TDS) requirements refers to the same entity as a “module” for the TSF internals (ADV_INT) requirements. Unlike subsystems, modules describe the implementation in a level of detail that can serve as a guide to reviewing the implementation representation.

It is important to note that, depending on the TOE, modules and subsystems may refer to the same abstraction. For ADV_TDS.1 Basic design and ADV_TDS.2 Architectural design (which do not require description at the module level) the subsystem description provides the lowest level detail available about the TSF. For ADV_TDS.3 Basic modular design (which require module descriptions) these descriptions provide the lowest level of detail, while the subsystem descriptions (if they exist as separate entities) merely serve to put to the module descriptions in context. That is, it is not necessary to provide detailed subsystem descriptions if module descriptions exist. In TOEs that are sufficiently simple, a separate “subsystem description” is not necessary; the requirements can be met through documentation provided by modules. For complex TOEs, the purpose of the subsystem description (with respect to the TSF) is to provide the reader context so they can focus their analysis appropriately. This difference is illustrated in Figure A.3.

An SFR-enforcing module is a module that completely or partially implements a security functional requirement (SFR) in the ST. Such modules may implement an SFR-enforcing TSFI, but some functionality expressed in an SFR (for example, audit and object re-use functionality) may not be directly tied to a single TSFI. As was the case with subsystems, SFR-supporting modules are those modules that are depended upon by an SFR-enforcing module, but are not responsible for directly implementing an SFR. SFR-non-interfering modules are those modules that do not deal, directly or indirectly, with the enforcement of SFRs.

It is important to note that the determination of what “directly implements” means is somewhat subjective. In the narrowest sense of the term, it could be interpreted to mean the one or two lines of code that actually perform a comparison, zeroing operation, etc. that implements a requirement. A broader interpretation might be that it includes the module that is invoked in response to a SFR-enforcing TSFI, and all modules that may be invoked in turn by that module (and so on until the completion of the call). Neither of these interpretations is particularly satisfying, since the narrowness of the first interpretation may lead to important modules being incorrectly categorised as SFR supporting, while the second leads to modules that are actually not SFR-enforcing being classified as such.

A description of a module should be such that one could create an implementation of the module from the description, and the resulting implementation would be 1) identical to the actual TSF implementation in terms of the interfaces presented, 2) identical in the use of interfaces that are mentioned in the design, and 3) functionally equivalent to the description of the purpose of the TSF module. For instance, RFC 793 provides a high-level description of the TCP protocol. It is necessarily implementation independent. While it provides a wealth of detail, it is ***not*** a suitable design description because it is not specific to an implementation. An actual implementation can add to the protocol specified in the RFC, and implementation choices (for example, the use of global data vs. local data in various parts of the implementation) may have an impact on the analysis that is performed. The design description of the TCP module would list the interfaces presented by the implementation (rather than just those defined in RFC 793), as well as an algorithm description of the processing associated with the modules implementing TCP (assuming they were part of the TSF).

In the design, modules are described in detail in terms of the function they provide (the purpose); the interfaces they present (when required by the criteria); the return values from such interfaces; the interfaces (presented by other modules) they use (provided those interfaces are required to be also described); and a description of how they provide their functionality using a technique appropriate to the method used to implement the module.

The purpose of a module should be described indicating what function the module is providing. It should be sufficient so that the reader could get a general idea of what the module's function is in the architecture.

The interfaces presented by a module are those interfaces used by other modules to invoke the functionality provided. Interfaces include both *explicit* interfaces (e.g., a calling sequence invoked by other modules) as well as *implicit* interfaces (e.g., global data manipulated by the module). Interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. If a parameter were expected to take on a set of values (e.g., a "flag" parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are described such that each field of the data structure is identified and described. Global data should be described to the extent required to understand their purpose. The level of description required for a global data structure needs to be identical to the one for module interfaces, where the input parameter and return values correspond to the individual fields and their possible values in the data structure. Global data structures may be described separate from the modules that manipulate or read them as long as the design of the modules contain sufficient information about the global data structures updated or the information extracted from global data structures.

Note that different programming languages may have additional "interfaces" that would be non-obvious; an example would be operator/function overloading in C++. This "implicit interface" in the class description would also be described as part of the module design. Note that although a module could present only one interface, it is more common that a module presents a small set of related interfaces.

When it is required to describe the interfaces used by a module, it must be clear from either the design description of the module or the purpose of the module called, what service is expected from the module called. For example if Module A is being described, and it uses Module B's bubble sort routine, the description of the interaction between modules must allow to identify why Module B's bubble sort routine is called and what this call contributes to the implementation of the SFRs. The interface and purpose of Module B's bubble sort routine must be described as part of the interfaces of Module B (provided the level of ADV_TDS and the classification of Module B require a description its interfaces) and so Module A just needs to identify what data it needs to have sorted using this routine. An adequate description would be: "Module A invokes Module B's interface *double_bubble()* to sort the usernames in alphabetical order".

Note that if this sorting of the user names is not important for the enforcement of any SFR (e. g. it is just done to speed up things and an algorithmically identical implementation of Module A could also avoid to have the usernames sorted), the use of Module B's bubble sort routine is not SFR-enforcing and it is sufficient to explain in the description of Module A that the usernames are sorted in alphabetical order to enhance performance. Module B may be classified as "SFR-supporting" only and the level of ADV_TDS chosen indicates if the interfaces of SFR-supporting modules need to be described or if it is sufficient to just describe the purpose of Module B.

As discussed previously, the algorithmic description of the module should describe in an algorithmic fashion the implementation of the module. This can be done in pseudo-code, through flow charts, or (at ADV_TDS.3 Basic modular design) informal text. It discusses how the module inputs and called functions are used to accomplish the module's function. It notes changes to global data, system state, and return values produced by the module. It is at the level of detail that an implementation could be derived that would be very similar to the actual implementation of the TOE.

It should be noted that source code does not meet the module documentation requirements. Although the module design describes the implementation, it is *not* the implementation. The comments surrounding the source code might be sufficient documentation if they provide an explanation of the intent of the source code. In-line comments that merely state what each line of code is doing are useless because they provide no explanation of what the module is meant to accomplish.

In the elements below, the labels (SFR-enforcing, SFR-supporting, and SFR-non-interfering) discussed for subsystems and modules are used to describe the amount and type of information that needs to be made available by the developer. The elements have been structured so that there is no expectation that the developer provide *only* the information specified. That is, if the developer's documentation of the TSF provides the information in the requirements below, there is no expectation that the developer update their documentation and label subsystems and modules as SFR-enforcing, SFR-supporting or SFR-non-interfering. The primary purpose of this labelling is to allow developers with less mature development methodologies (and associated artifacts, such as detailed interface and design documentation) to provide the necessary evidence without undue cost.

A.4.3 Levelling Approach

Because there is subjectivity in determining what is SFR-enforcing vs. SFR-supporting (and in some cases, even determining what is SFR-non-interfering the following paradigm has been adopted in this family. In early components of the family, the developer makes a determination about the classification of the subsystems into SFR-enforcing, etc., supplying the appropriate information, and there is little additional evidence for the evaluator to examine to support this claim. As the level of desired assurance increases, while the developer still makes a classification determination, the evaluator obtains more and more evidence that is used to confirm the developer's classification.

In order to focus the evaluator's analysis on the SFR-related portions of the TOE, especially at lower levels of assurance, the components of the family are levelled such that initially detailed information is required only for SFR-enforcing architectural entities. As the level of assurance increases, more information is required for SFR-supporting and (eventually) SFR-non-interfering entities. It should be noted that even when complete information is required, it is not required that all of this information be analysed in the same level of detail. The focus should be in all cases on whether the *necessary* information has been provided and analysed.

Table **AA.1** summarises the information required at each of the family components for the architectural entities to be described.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.1 Basic design (informal presentation)	structure, summary of SFR-Enf. behaviour, interactions	designation support[11]	designation support			
ADV_TDS.2 Architectural design (informal presentation)	structure, detailed description of SFR-Enf. behaviour, summary of other behaviour, interactions	structure, summary of other behaviour, interactions	designation support, interactions			
ADV_TDS.3 Basic modular design (informal presentation)	description, interactions	description, interactions	description, interactions	purpose, SFR interfaces[12]	interaction, purpose	interaction, purpose
ADV_TDS.4 Semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	purpose, SFR interfaces	purpose, SFR interfaces	interaction, purpose
ADV_TDS.5 Complete semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	purpose, all interfaces[13]	purpose, all interfaces	purpose, all interfaces
ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation	description, interactions	description, interactions	description, interactions	purpose, all interfaces	purpose, all interfaces	purpose, all interfaces

[11] *designation support* means that only documentation sufficient to support the classification of the subsystem / module is needed.

[12] *SFR interfaces* means that the module description contains, for each SFR-related interface, the returned values and the called interfaces to other modules.

[13] *All interfaces* means that the module description contains, for each interface, the returned values and the called interfaces to other modules.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
(semiformal presentation; additional formal presentation)						

Table AA.1 — Description Detail Levelling

A.4.4 Security relevance

The comments to WD2 regarding this chapter are pending as the contributor was not able to answer the comments until the deadline.

ISO/IEC 15408 (all parts) concentrates the description, the evidence and the analysis on the security functionality of the TOE. This requires characterization of security relevance of functional and physical parts of the TOE. Interfaces, subsystems and modules may be categorised (either implicitly or explicitly) as "SFR-enforcing", "SFR-supporting", or "SFR-noninterfering".

The developer evidence and the evaluation analysis relates to the TOE and focus on the TSF and its SFR-enforcing and SFR-supporting implementation. The security architecture description shall demonstrate that the identified non-TSF subsystems of the TOE are not bypassing the TSF and the TSF protects themselves against corruption by non-TSF code or entities. The developer shall describe the SFR-noninterfering interfaces, subsystems and modules in the TOE design and demonstrate that they do not interfere with the TSF because of their purposes, interactions or separation of resources.

An interface, subsystem or module is

- SFR-enforcing, if it directly implements an SFR.
- SFR-supporting if it has to operate functionally correctly in order to support the proper function of the SFRs.
- SFR-non-interfering if it is not related to the implementation of the SFRs.

The focus on security enforcing and security supporting functionality requires evidence of non-interference of the other functionality. Even correct implemented security enforcing functions and security mechanisms may be **bypassed, circumvented, deactivated, corrupted,** or **directly attacked.** Non-interference implies that the TSF cannot be misused and unauthorized access to the resources of the TSF implementation is prevented or impossible. Therefore the security architecture aspects of non-bypassability and self-protection are critical if security relevance of interfaces, subsystems and modules is categorized and this categorization is used in the vulnerability analysis.

TSF self-protection is the security architecture property whereby the TSF cannot be corrupted by non-TSF code or entities. This includes non-TSF subsystems of TOE and non-TOE parts of the IT product. It is similar to the evidence for SFR-non-interfering subsystems/modules.

The security domains are environments provided by the TSF for the use by untrusted entities in such a way that these environments are isolated and protected from each other.

Therefore the analysis of non-interference during evaluation requires examination of the security architecture of the TOE (ADV_ARC) and may need more information on non-TSF subsystems than only the TOE structure in terms of subsystems as provided for ADV_TDS.x.1. The developers shall provide a rationale that TSF is correctly defined and the analysis of SFR-non-interfering module in terms of its purpose and interaction with other modules

- **purpose**: how a module provides their functionality, no further design decisions are needed.
- **interaction**: reason that subsystems or modules communicate, and characterizes the information that is passed (less details than for interfaces).

During evaluation non-interference shall be analyzed as part of the examination of functional specification and TOE design, and the vulnerability analysis. The categorization of interfaces, subsystems and modules as SFR-enforcing, SFR-supporting and SFR-noninterfering implies specific examination of the functional specification, design and testing. An interpretation of TSFI as all accessible external interfaces of the TSF would help this analysis. The functional tests of all TSF subsystems (beginning with ATE_DPT.1) and all TSF modules (ATE_DPT.3 and higher) should provide evidence for the correctness of their security categorization.

A.5Supplementary material on formal methods

Formal methods provide a mathematical representation of the TSF and its behaviour and are required by the ADV_FSP.6 Complete semi-formal functional specification with additional formal specification, ADV_SPM.1 Formal TOE security policy model, and ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation components. There are two aspects of formal methods: the *specification language* that is used for formal expression, and the *theorem prover* that mathematically proves the completeness and correctness of the formal specification.

A formal specification is expressed within a formal system based upon well-established mathematical concepts. These mathematical concepts are used to define well-defined semantics, syntax and rules of inference. A formal system is an abstract system of identities and relations that can be described by specifying a formal alphabet, a formal language over that alphabet which is based on a formal syntax, and a set of formal rules of inference for constructing derivations of sentences in the formal language.

The evaluator should examine the identified formal systems to make sure that:

- The semantics, syntax and inference rules of the formal system are defined or a definition is referenced.
- Each formal system is accompanied by explanatory text that provides defined **semantics** so that:
  1) the explanatory text provides defined meanings of terms, abbreviations and acronyms that are used in a context other than that accepted by normal usage;
  2) the use of a formal system and semiformal notation use is accompanied by supporting explanatory text in informal style appropriate for unambiguous meaning;
  3) the formal system is able to express rules and characteristics of applicable SFPs, security functionality and interfaces (providing details of effects, exceptions and error messages) of TSF, their subsystems or modules to be specified for the assurance family for which the notations are used;
  4) the notation provides rules to determine the meaning of syntactical valid constructs.

- Each formal system uses a formal syntax that provides rules to unambiguously recognise constructs.
- Each formal system provides proof rules which
    5) support logical reasoning of well-established mathematical concepts,
    6) help to prevent derivation of contradictions.

If the developer uses a formal system which is already accepted by the evaluation authority the evaluator can rely on the level of formality and strength of the system and focus on the instantiation of the formal system to the TOE specifications and correspondence proofs.

The formal style supports mathematical proofs of the security properties based on the security features, the consistency of refinements and the correspondence of the representations. Formal tool support seems adequate whenever manual derivations would otherwise become long winded and incomprehensible. Formal tools are also apt to reduce the error probability inherent in manual derivations.

Examples of formal systems:

- The **Z specification language** is highly expressive, and supports many different methods or styles of formal specification. The use of Z has been predominantly for model-oriented specification, using *schemas* to formally specify operations. See https://en.wikipedia.org/wiki/Z_notation for more information.
- **ACL2** is an open-source formal system comprising a LISP-based specification language and a theorem prover. See http://www.cs.utexas.edu/users/moore/acl2/ for further information.
- **Isabelle** is a popular generic theorem proving environment that allows mathematical formulae to be expressed in a formal language and provides tools for proving those formulae within a logical calculus (see e.g. http://www.cl.cam.ac.uk/Research/HVG/Isabelle/ for additional information).
- The **B method** is a formal system based on the propositional calculus, the first order predicate calculus with inference rules and set theory (see e.g. https://en.wikipedia.org/wiki/B-Method for further information).
- **NuSMV** (based on its predecessor SMV) is a symbolic model checker designed to be an open architecture for model checking which can be reliably used for the verification of industrial designs, as a core for custom verification tools, and as a testbed for formal verification techniques. See http://nusmv.fbk.eu/ for more information.
- **Coq** is a formal proof management system that provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. See https://coq.inria.fr/ for more information.
- **SystemVerilog** is a combined hardware description language and hardware verification language based on Verilog.

Annex B
(informative)

Composition (ACO)

The goal of this annex is to explain the concepts behind composition evaluations and the ACO criteria. This annex does not define the ASE criteria; this definition can be found in clause 9.

B.1Necessity for composed TOE evaluations

The IT market is, on the whole, made up of vendors offering a particular type of product/technology. Although there is some overlap, where a PC hardware vendor may also offer application software and/or operating systems or a chip manufacturer may also develop a dedicated operating system for their own chipset, it is often the case that an IT solution is implemented by a variety of vendors.

There is sometimes a need for assurance in the combination (composition) of components in addition to the assurance of the individual components. Although there is cooperation between these vendors, in the dissemination of certain material required for the technical integration of the components, the agreements rarely stretch to the extent of providing detailed design information and development process/procedure evidence. This lack of information from the developer of a component on which another component relies means that the dependent component developer does not have access to the type of information necessary to perform an evaluation of both the dependent and base components at EAL2 or above. Therefore, while an evaluation of the dependent component can still be performed at any assurance level, to compose components with assurance at EAL2 or above it is necessary to reuse the evaluation evidence and results of evaluations performed for the component developer.

It is intended that the ACO criteria are applicable in the situation where one IT entity is dependent on another for the provision of security services. The entity providing the services is termed the “base component”, and that receiving the services is termed the “dependent component”. This relationship may exist in a number of contexts. For example, an application (dependent component) may use services provided by an operating system (base component). Alternatively, the relationship may be peer-to-peer, in the sense of two linked applications, either running in a common operating system environment, or on separate hardware platforms. If there is a dominant peer providing the services to the minor peer, the dominant peer is considered to be the base component and the minor peer the dependent component. If the peers provide services to each other in a mutual manner, each peer will be considered to be the base component for the services offered and dependent component for the services required. This will require iterations of the ACO components applying all requirements to each type of component peer.

The criteria are also intended to be more broadly applicable, stepwise (where a composed TOE comprised of a dependent component and a base component itself becomes the base component of another composed TOE), in more complex relationships, but this may require further interpretation.

It is still required for composed TOE evaluations that the individual components are evaluated independently, as the composition evaluation builds on the results of the individual component evaluations. The evaluation of the dependent component may still be in progress when the composed TOE evaluation commences. However, the dependent component evaluation must complete before the composed TOE evaluation completes.

The composed evaluation activities may take place at the same time as the dependent component evaluation. This is due to two factors:

a) Economic/business drivers - the dependent component developer will either be sponsoring the composition evaluation activities or supporting these activities as the evaluation deliverables from the dependent component evaluation are required for composed evaluation activities.

b) Technical drivers - the components consider whether the requisite assurance is provided by the base component (e.g. considering the changes to the base component since completion of the component evaluation) with the understanding that the dependent component has recently undergone (is undergoing) component evaluation and all evaluation deliverables associated with the evaluation are available. Therefore, there are no activities during composition requesting the dependent component evaluation activities to be re-verified. Also, it is verified that the base component forms (one of) the test configurations for the testing of the dependent component during the dependent component evaluation, leaving ACO_CTT to consider the base component in this configuration.

The evaluation evidence from the evaluation of the dependent component is required input into the composed TOE evaluation activities. The only evaluation material from the evaluation of the base component that is required as input into the composed TOE evaluation activities:

a) Residual vulnerabilities in the base component, as reported during the base component evaluation. This is required for the ACO_VUL activities.

No other evaluation evidence from the base component activities should be required for the composed TOE evaluation, as the evaluation results from the component evaluation of the base component should be reused. Additional information about the base component may be required if the composed TOE TSF includes more of the base component than was considered to be TSF during component evaluation of the base component.

The component evaluation of the base and dependent components are assumed to be complete by the time final verdicts are assigned for the ACO components.

The ACO_VUL components only consider resistance against an attacker with an attack potential up to Enhanced-Basic. This is due to the level of design information that can be provided of how the base component provides the services on which the dependent component relies through application of the ACO_DEV activities. Therefore, the confidence arising from composed TOE evaluations using CAPs is limited to a level similar to that obtained from EAL4 component TOE evaluations. Although assurance in the components that comprise the composed TOE may be higher than EAL4.

B.2 Performing Security Target evaluation for a composed TOE

An ST will be submitted by the developer for the evaluation of the composed (base component + dependent component) TOE. This ST will identify the assurance package to be applied to the composed TOE, providing assurance in the composed entity by drawing upon the assurance gained in the component evaluations.

The purpose of considering the composition of components within an ST is to validate the compatibility of the components from the point of view of both the environment and the requirements, and also to assess that the composed TOE ST is consistent with the component STs and the security policies expressed within them. This includes determining that the component STs and the security policies expressed within them are compatible.

The composed TOE ST may refer out to the content of the component STs, or the ST author may chose to reiterate the material of the component STs within the composed TOE ST providing a rationale of how the component STs are represented in the composed TOE ST.

During the conduct of the ASE_CCL evaluation activities for a composed TOE ST the evaluator determines that the component STs are accurately represented in the composed TOE ST. This is achieved through determining that the composed TOE ST demonstrably conforms to the component TOE STs. Also, the evaluator will need to determine that the dependencies of the dependent component on the operational environment are adequately fulfilled in the composed TOE.

The composed TOE description will describe the composed solution. The logical and physical scope and boundary of the composed solution will be described, and the logical boundary(ies) between the components will also be identified. The description will identify the security functionality to be provided by each component.

The statement of SFRs for the composed TOE will identify which component is to satisfy an SFR. If an SFR is met by both components, then the statement will identify which component meets the different aspects of the SFR. Similarly the composed TOE Summary Specification will identify which component provides the security functionality described.

The package of ASE: Security Target evaluation requirements applied to the composed TOE ST should be consistent with the package of ASE: Security Target evaluation requirements used in the component evaluations.

Reuse of evaluation results from the evaluation of component STs can be made in the instances that the composed TOE ST directly refers to the component STs. e.g. if the composed TOE ST refers to a component ST for part of its statement of SFRs, the evaluator can understand that the requirement for the completion of all assignment and selection operations (as stated in ASE_REQ.*.3C has been satisfied in the component evaluations.

B.3 Interactions between composed IT entities

The TSF of the base component is often defined without knowledge of the dependencies of the possible applications with which it may by composed. The TSF of this base component is defined to include all parts of the base component that have to be relied upon for enforcement of the base component SFRs. This will include all parts of the base component required to implement the base component SFRs.

The TSFI of this base component represents the interfaces provided by the TSF to the external entities defined in the statement of SFRs to invoke a service of the TSF. This includes interfaces to the human user and also interfaces to external IT entities. However, the TSFI only includes those interfaces to the TSF, and therefore is not necessarily an exhaustive interface specification of all possible interfaces available between an external entity and the base component. The base component may present interfaces to services that were not considered security-relevant, either because of the inherent purpose of the service (e.g., adjust type font) or because associated ISO/IEC 15408 SFRs are not being claimed in the base component's ST (e.g. the login interface when no FIA: Identification and authentication SFRs are claimed).

The functional interfaces provided by the base component are in addition to the security interfaces (TSFIs), and are not required to be considered during the base component evaluation. These often include interfaces that are used by a dependent component to invoke a service provided by the base component.

The base component may include some indirect interfaces through which TSFIs may be called, e.g. APIs that can be used to invoke a service of the TSF, which were not considered during the evaluation of the base component.

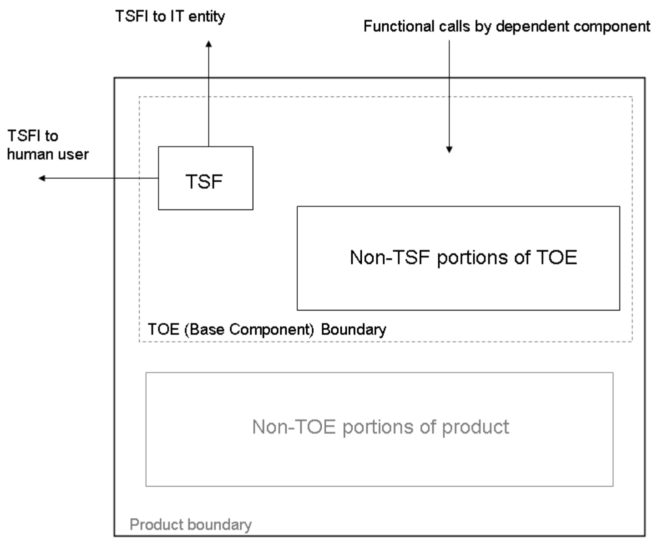
TSFI to IT entity
Functional calls by dependent component
TSFI to human user
TSF
Non-TSF portions of TOE
TOE (Base Component) Boundary
Non-TOE portions of product
Product boundary

Figure BB.1 — Base component abstraction

The dependent component, which relies on the base component, is similarly defined: interfaces to external entities defined in the SFRs of the component ST are categorised as TSFI and are examined in ADV_FSP.

Any call out from the dependent TSF to the environment in support of an SFR will indicate that the dependent TSF requires some service from the environment in order to satisfy the enforcement of the stated dependent component SFRs. Such a service is outside the dependent component boundary and the base component is unlikely to be defined in the dependent ST as an external entity. Hence, the calls for services made out by the dependent TSF to its underlying platform (the base component) will not be analysed as part of the Functional specification (ADV_FSP) activities. These dependencies on the base component are expressed in the dependent component ST as security objectives for the environment.

This abstraction of the dependent component and the interfaces is shown in Figure BB.2 below.

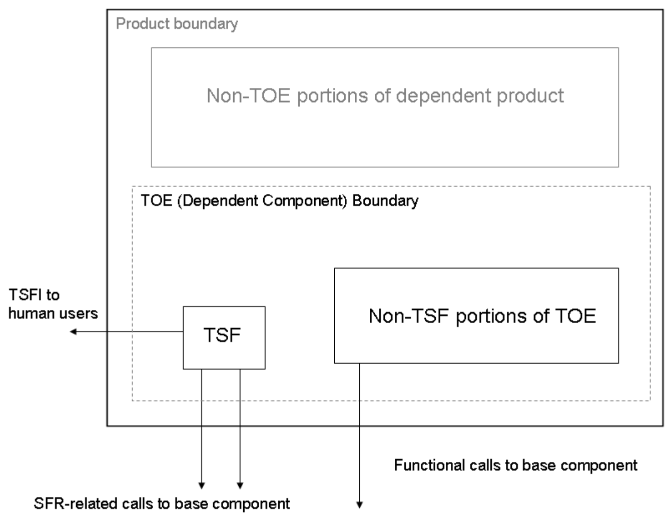
Product boundary
Non-TOE portions of dependent product
TOE (Dependent Component) Boundary
TSFI to human users
TSF
Non-TSF portions of TOE
Functional calls to base component
SFR-related calls to base component

Figure BB.2 — Dependent component abstraction

When considering the composition of the base component and the dependent component, if the dependent component's TSF requires services from the base component to support the implementation of the SFR, the interface to the service will need to be defined. If that service is provided by the base component's TSF, then that interface should be a TSFI of the base component and will therefore already be defined within the functional specification of the base component.

If, however, the service called by the dependent component's TSF is not provided by the TSF of the base component (i.e., it is implemented in the non-TSF portion of the base component or possibly even in the non-TOE portion of the base component (not illustrated in Figure B.3), there is unlikely to be a TSFI of the base component relating to the service, unless the service is mediated by the TSF of the base component. The interfaces to these services from the dependent component to the operational environment are considered in the family Reliance of dependent component (ACO_REL).

The non-TSF portion of the base component is drawn into the TSF of the composed TOE due to the dependencies the dependent component has on the base component to support the SFRs of the dependent component. Therefore, in such cases, the TSF of the composed TOE would be larger than simply the sum of the components' TSFs.

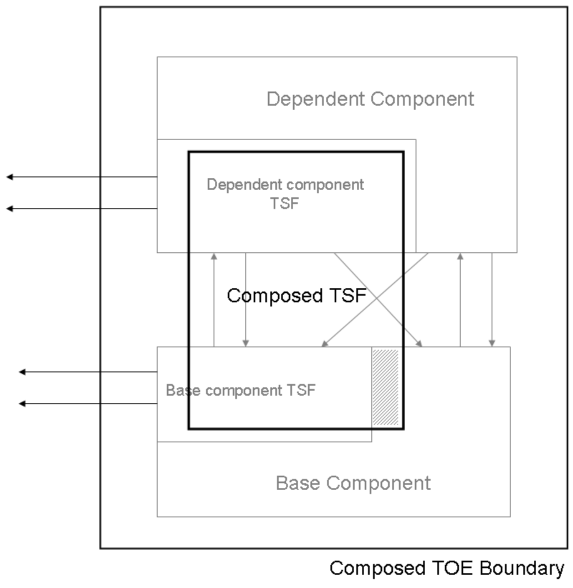

Figure B.3 — Composed TOE abstraction

It may be the case that the base component TSFI is being called in a manner that was unforeseen in the base component evaluation. Hence there would be a requirement for further testing of the base component TSFI.

The possible interfaces are further described in the following diagram (Figure B.4) and supporting text.

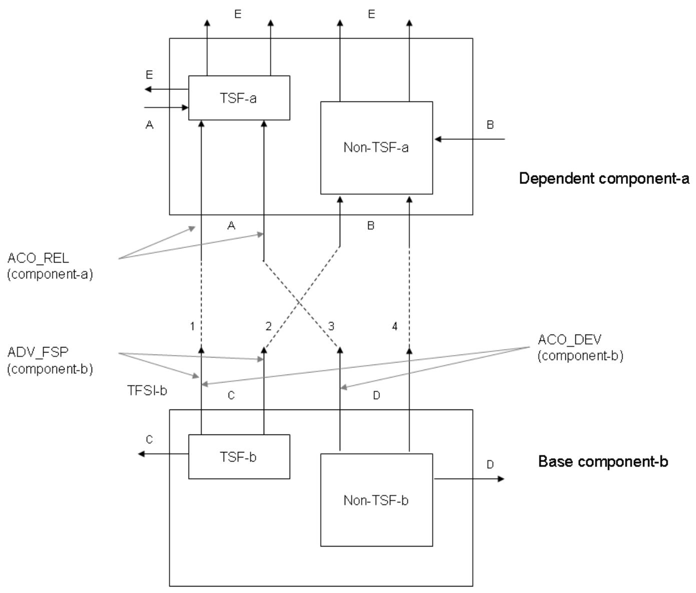

Figure B.4 — Composed component interfaces

a) Arrows going *into* 'dependent component-a' (A and B) = where the component expects the environment to respond to a service request (responding to calls out from dependent component to the environment);

b) Arrows coming *out* of 'base component-b' (C and D) = interfaces of services provided by the base component to the environment;

c) Broken lines between components = types of communication between pairs of interfaces;

d) The other (grey) arrows = interfaces that are described by the given criteria.

The following is a simplification, but explains the considerations that need to be made.

There are components a ('dependent component-a') and b ('base component-b'): the arrows coming *out* of TSF-a are services provided by TSF-a and are therefore TSFIs(a); likewise, the arrows coming *out* of TSF-b ("C") are TSFIs(b). These are each detailed in their respective functional specs. component-a is such that it requires services from its environment: those needed by the TSF(a) are labelled "A"; the other (not related to TSF-a) services are labelled "B".

When component-a and component-b are combined, there are four possible combinations of {services needed by component-a} and {services provided by component-b}, shown as broken lines (types of communication between pairs of interfaces). Any set of these might exist for a particular composition:

a) TSF-a needs those services that are provided by TSF-b ("A" is connected to "C"): this is straightforward: the details about "C" are in the FSP for component-b. In this instance the interfaces should all be defined in the functional specifications for the component-b.

b) Non-TSF-a needs those services that are provided by TSF-b ("B" is connected to "C"): this is straightforward (again, the details about "C" are in the FSP for component-b), but unimportant: security-wise.

c) Non-TSF-a needs those services that are provided by non-TSF-b ("B" is connected to "D"): we have no details about D, but there are no security implications about the use of these interfaces, so they do not need to be considered in the evaluation, although they are likely to be an integration issue for the developer.

d) TSF-a needs those services that are provided by non-TSF-b ("A" is connected to "D"): this would arise when component-a and component-b have different senses of what a "security service" is. Perhaps component-b is making no claims about I&A (has no FIA SFRs in its ST), but component-a needs authentication provided by its environment. There are no details about the "D" interfaces available (they are not TSFI (b), so they are not in component-b's FSP).

Note: if the kind of interaction described in case d above exists, then the TSF of the composed TOE would be TSF-a + TSF-b + Non-TSF-b. Otherwise, the TSF of the composed TOE would be TSF-a + TSF-b.

Interfaces types 2 and 4 of Figure B.4 are not directly relevant to the evaluation of the composed TOE. Interfaces 1 and 3 will be considered during the application of different families:

a) Functional specification (ADV_FSP) (for component-b) will describe the C interfaces.

b) Reliance of dependent component (ACO_REL) will describe the A interfaces.

c) Development evidence (ACO_DEV) will describe the C interfaces for connection type 1 and the D interfaces for connection type 3.

A typical example where composition may be applied is a database management system (DBMS) that relies upon its underlying operating system (OS). During the evaluation of the DBMS component, there will be an assessment made of the security properties of that DBMS (to whatever degree of rigour is dictated by the assurance components used in the evaluation): its TSF boundary will be identified, its functional specification will be assessed to determine whether it describes the interfaces to the security services provided by the TSF, perhaps additional information about the TSF (its design, architecture, internal structure) will be provided, the TSF will be tested, aspects of its life-cycle and its guidance documentation will be assessed, etc.

However, the DBMS evaluation will not call for any evidence concerning the dependency the DBMS has on the OS. The ST of the DBMS will most likely state assumptions about the OS in its Assumptions subclause and state security objectives for the OS in its Environment subclause. The DBMS ST may even instantiate those objectives for the environment in terms of SFRs for the OS. However, there will be no specification for the OS that mirrors the detail in the functional specification, architecture description, or other ADV evidence as for the DBMS. Reliance of dependent component (ACO_REL) will fulfil that need.

Reliance of dependent component (ACO_REL) describes the interfaces of the dependent TOE that make the calls to the base component for the provision of services. These are the interfaces to which the base component is to respond. The interface descriptions are provided from the dependent component's viewpoint.

Development evidence (ACO_DEV) describes the interfaces provided by the base component, which respond to the dependent component service requests. These interfaces are mapped to the relevant dependent component interfaces that are identified in the reliance information. (The completeness of this mapping, whether the base component interfaces described represent all dependent component interfaces, is not verified here, but in Composition rationale (ACO_COR)). At the higher levels of ACO_DEV the subsystems providing the interfaces are described.

Any interfaces required by the dependent component that have not been described for the base component are reported in the rationale for Composition rationale (ACO_COR). The rationale also reports whether the interfaces of the base component on which the dependent component relies were considered within the base component evaluation. For any interfaces that were not considered in the base component evaluation, a rationale is provided of the impact of using the interface on the base component TSF.

Annex C
(informative)

Cross reference of assurance component dependencies

The dependencies documented in the components of Clauses 7 and 9-15 are the direct dependencies between the assurance components.

The following dependency tables for assurance components show their direct, indirect and optional dependencies. Each of the components that is a dependency of some assurance component is allocated a column. Each assurance component is allocated a row. The value in the table cell indicate whether the column label component is directly required (indicated by a cross "X") or indirectly required (indicated by a dash "-"), by the row label component. If no character is presented, the component is not dependent upon another component.

	ADV_ARC.1	ADV_FSP.1	ADV_FSP.2	ADV_FSP.3	ADV_FSP.4	ADV_FSP.5	ADV_FSP.6	ADV_IMP.1	ADV_TDS.1	ADV_TDS.3	AGD_OPE.1	AGD_PRE.1	ALC_CMC.5	ALC_CMS.1	ALC_DVS.2	ALC_LCD.1	ALC_TAT.1	ALC_TDA.1	ALC_TDA.2	ALC_TDA.3
ADV_ARC.1		X	-						X											
ADV_FSP.1																				
ADV_FSP.2			-						X											
ADV_FSP.3			-						X											
ADV_FSP.4			-						X											
ADV_FSP.5			-		-			X	X	-							-			
ADV_FSP.6			-		-			X	X	-							-			
ADV_IMP.1			-		-			-	-	X							X			
ADV_IMP.2			-		-			-	-	X			X	-	-	-	X			
ADV_INT.1			-		-			X	-	X							X			
ADV_INT.2			-		-			X	-	X							X			
ADV_INT.3			-		-			X	-	X							X			
ADV_SPM.1			-		X				-											
ADV_TDS.1			X						-											
ADV_TDS.2			-	X					-											
ADV_TDS.3			-		X				-											
ADV_TDS.4			-		-	X		-	-	-							-			
ADV_TDS.5			-		-	X		-	-	-							-			
ADV_TDS.6			-		-		X	-	-	-							-			
ADV_COMP.1																				

Table C.1 — Dependency table for Class ADV: Development

	ADV_FSP.1
AGD_OPE.1	X
AGD_PRE.1	

Table C.2 — Dependency table for Class AGD: Guidance documents

	ADV_FSP.2	ADV_FSP.4	ADV_IMP.1	ADV_TDS.1	ADV_TDS.3	ALC_CMS.1	ALC_CMS.3	ALC_DVS.1	ALC_DVS.2	ALC_LCD.1	ALC_TAT.1
ALC_CMC.1						X					
ALC_CMC.2						X					
ALC_CMC.3						X		X		X	
ALC_CMC.4						X		X		X	
ALC_CMC.5						X			X	X	
ALC_CMS.1											
ALC_CMS.2											
ALC_CMS.3											
ALC_CMS.4											
ALC_CMS.5											
ALC_DEL.1											
ALC_DVS.1											
ALC_DVS.2											
ALC_FLR.1											
ALC_FLR.2											
ALC_FLR.3											
ALC_LCD.1											
ALC_LCD.2											
ALC_TDA.1											
ALC_TDA.2							X				
ALC_TDA.3					-						X
ALC_TAT.1	-	-	X	-	-						-
ALC_TAT.2	-	-	X	-	-						-
ALC_TAT.3	-	-	X	-	-						-
ALC_MOK.1											
ALC_COMP.1											

Table C.3 — Dependency table for Class ALC: Life-cycle support

	APE_ECD.1	APE_INT.1	APE_OBJ.2	APE_REQ.1	APE_SPD.1
APE_CCL.1	X	X		X	
APE_ECD.1					
APE_INT.1					
APE_OBJ.1					
APE_OBJ.2					X
APE_REQ.1	X				
APE_REQ.2	X		X		-
APE_SPD.1					

Table C.4 — Dependency table for Class APE: Protection Profile evaluation

	ACE_ECD.1	ACE_INT.1	ACE_MCO.2	ACE_OBJ.1	ACE_REQ.1	ACE_SPD.1
ACE_CCL.1	X	X		-	X	
ACE_CCO.1	-	X	X	-	X	-
ACE_ECD.1						
ACE_INT.1						
ACE_MCO.1	-	X		X	X	X
ACE_OBJ.1						
ACE_REQ.1	X			X		
ACE_SPD.1						

Table C.5 — Dependency table for Class ACE: Protection Profile Configuration evaluation

	ADV_ARC.1	ADV_FSP.1	ADV_FSP.2	ADV_TDS.1	ASE_ECD.1	ASE_INT.1	ASE_OBJ.2	ASE_REQ.1	ASE_SPD.1
ASE_CCL.1					X	X		X	
ASE_ECD.1									
ASE_INT.1									
ASE_OBJ.1									
ASE_OBJ.2									X
ASE_REQ.1					X				
ASE_REQ.2					X		X		-
ASE_SPD.1									
ASE_TSS.1		X			-	X		X	
ASE_TSS.2	X	-	-	-	-	X		X	
ASE_COMP.1									

Table C.6 — Dependency table for Class ASE: Security Target evaluation

	ADV_ARC.1	ADV_FSP.1	ADV_FSP.2	ADV_FSP.3	ADV_FSP.4	ADV_FSP.5	ADV_IMP.1	ADV_TDS.1	ADV_TDS.2	ADV_TDS.3	ADV_TDS.4	AGD_OPE.1	AGD_PRE.1	ALC_TAT.1	ATE_COV.1	ATE_FUN.1
ATE_COV.1			X					-							-	X
ATE_COV.2			X					-							-	X
ATE_COV.3			X					-							-	X
ATE_DPT.1	X	-	-	-				-	X						-	X
ATE_DPT.2	X	-	-		-			-		X					-	X
ATE_DPT.3	X	-	-		-	-	-	-		-	X			-	-	X
ATE_DPT.4	X	-	-		-	-	X	-		-	X			-	-	X
ATE_FUN.1			-					-							X	-
ATE_FUN.2			-					-							X	-
ATE_IND.1		X										X	X			
ATE_IND.2		-	X					-				X	X		X	X
ATE_IND.3		-	-		X			-				X	X		X	X
ATE_COMP.1																

Table C.7 Dependency table for Class ATE: Tests

	ADV_ARC.1	ADV_FSP.1	ADV_FSP.2	ADV_FSP.3	ADV_FSP.4	ADV_IMP.1	ADV_TDS.1	ADV_TDS.2	ADV_TDS.3	AGD_OPE.1	AGD_PRE.1	ALC_TAT.1	ATE_COV.1	ATE_DPT.1	ATE_FUN.1
AVA_VAN.1		X								X	X				
AVA_VAN.2	X	-	X				X			X	X				
AVA_VAN.3	X	-	-	-	X	X	-	-	X	X	X	-	-	X	-
AVA_VAN.4	X	-	-	-	X	X	-	-	X	X	X	-	-	X	-
AVA_VAN.5	X	-	-	-	X	X	-	-	X	X	X	-	-	X	-
AVA_COMP.1															

Table C.8 Dependency table for Class AVA: Vulnerability assessment

Annex D