Common Criteria for Information Technology Security Evaluation

Part 3: Security assurance components

July 2005

Version 3.0
Revision 2

CCMB-2005-07-003

Foreword

This version of the Common Criteria for Information Technology Security Evaluation (CC v3.0) is the first major revision since being published as CC v2.1 in 1999 and CC v2.2 in 2004.

CC v3.0 is released for public comment and aims to eliminate redundant evaluation activities; reduce/eliminate those activities that contributed little to the final assurance of a product; clarify CC terminology to reduce misunderstandings; restructure and refocus the evaluation activities to those areas where security assurance would truly be gained; and add new CC requirements if needed.

This revision 2 of the CC v3.0 includes all editorial updates as of the release date.

CC version 3.0 consists of the following parts:

– Part 1: Introduction and general model

– Part 2: Security functional components

– Part 3: Security assurance components

Trademarks:

– Microsoft is a registered trademark of Microsoft Corporation

– POSIX is a registered trademark of the IEEE

– UNIX is a registered trademark of The Open Group in the United States and other countries

– Windows is a registered trademark of Microsoft Corporation in the United States and other countries

Legal Notice:

The governmental organisations listed below contributed to the development of this version of the Common Criteria for Information Technology Security Evaluation. As the joint holders of the copyright in the Common Criteria for Information Technology Security Evaluation, version 3.0 *Parts 1 through 3 (called "CC* 3.0*"), they hereby grant non-exclusive license to ISO/IEC to use CC* 3.0 *in the continued development/maintenance of the ISO/IEC 15408 international standard. However, these governmental organisations retain the right to use, copy, distribute, translate or modify CC* 3.0 *as they see fit.*

Australia/New Zealand:	*The Defence Signals Directorate and the Government Communications Security Bureau respectively;*
Canada:	*Communications Security Establishment;*
France:	*Direction Centrale de la Sécurité des Systèmes d'Information;*
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-Electronics Security Group;*
United States:	*The National Security Agency and the National Institute of Standards and Technology.*

Table of Contents

Table of contents

Table of contents

List of figures

List of tables

1 Introduction

1 Security assurance components, as defined in this CC Part 3, are the basis for the security assurance requirements expressed in a Protection Profile (PP) or a Security Target (ST).

2 These requirements establish a standard way of expressing the assurance requirements for TOEs. This CC Part 3 catalogues the set of assurance components, families and classes. This CC Part 3 also defines evaluation criteria for PPs and STs and presents evaluation assurance levels that define the predefined CC scale for rating assurance for TOEs, which is called the Evaluation Assurance Levels (EALs).

3 The audience for this CC Part 3 includes consumers, developers, and evaluators of secure IT systems and products. CC Part 1 Chapter 7 provides additional information on the target audience of the CC, and on the use of the CC by the groups that comprise the target audience. These groups may use this part of the CC as follows:

a) Consumers, who use this CC Part 3 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 CC Part 3 when interpreting statements of assurance requirements and determining assurance approaches of TOEs.

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

2 Scope

4 This CC Part 3 defines the assurance requirements of the CC. It includes the evaluation assurance levels (EALs) that define a scale for measuring assurance for component TOEs, the composition assurance packages (CAPs) that define a scale for measuring assurance for composed TOEs, the individual assurance components from which the assurance levels and packages are composed, and the criteria for evaluation of PPs and STs.

3 Normative references

5 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.

CC-1 Common Criteria for Information Technology Security Evaluation, Version 3.0, revision 2, June 2005. Part 1: Introduction and general model.

CC-2 Common Criteria for Information Technology Security Evaluation, Version 3.0, revision 2, June 2005. Part 2: Functional security components.

4 Terms and definitions

6 For the purposes of this document, the following terms and definitions apply.

7 **coherent —** an entity is logically ordered and has a discernible meaning. For documentation, this addresses both the actual text and the structure of the document, in terms of whether it is understandable by its target audience.

8 **complete —** all necessary parts of an entity have been provided. In terms of documentation, this means that all relevant information is covered in the documentation, at such a level of detail that no further explanation is required at that level of abstraction.

9 **confirm —** this term is used to indicate that something needs to be reviewed in detail, and that an independent determination of sufficiency needs to be made. The level of rigour required depends on the nature of the subject matter. This term is only applied to evaluator actions.

10 **consistent —** this term describes a relationship between two or more entities, indicating that there are no apparent contradictions between these entities.

11 **counter (verb) —** this term is typically used when the impact of a particular threat is mitigated but not necessarily eradicated.

12 **demonstrate —** this term refers to an analysis leading to a conclusion, which is less rigorous than a “proof”.

13 **describe —** this term requires that specific details of an entity be provided.

14 **determine —** this term requires an independent analysis to be made, with the objective of reaching a particular conclusion. The usage of this term differs from “confirm” or “verify”, since these other terms imply that an analysis has already been performed which needs to be reviewed, whereas the usage of “determine” implies a truly independent analysis, usually in the absence of any previous analysis having been performed.

15 **ensure —** this term, used by itself, implies a strong causal relationship between an action and its consequences. When this term is preceded by the word “helps” it indicates that the consequence is not fully certain, on the basis of that action alone.

16 **exhaustive —** this term is used in the CC with respect to conducting an analysis or other activity. It is related to “systematic” but is considerably stronger, in that it indicates not only that a methodical approach has been taken to perform the analysis or activity according to an unambiguous plan, but that the plan that was followed is sufficient to ensure that all possible avenues have been exercised.

17 **explain —** this term differs from both "describe" and "demonstrate". It is intended to answer the question "Why?" without actually attempting to argue that the course of action that was taken was necessarily optimal.

18 **internally consistent —** this term means that there are no apparent contradictions between any aspects of an entity. In terms of documentation, this means that there can be no statements within the documentation that can be taken to contradict each other.

19 **justification —** this term refers to an analysis leading to a conclusion, but is more rigorous than a demonstration. This term requires significant rigour in terms of very carefully and thoroughly explaining every step of a logical argument.

20 **prove —** this term refers to a formal analysis in its mathematical sense. It is completely rigorous in all ways. Typically, "prove" is used when there is a desire to show correspondence between two TSF representations at a high level of rigour.

21 **specify —** this term is used in the same context as "describe", but is intended to be more rigorous and precise. It is very similar to "define".

22 **trace (verb) —** this term is used to indicate that an informal correspondence is required between two entities with only a minimal level of rigour.

23 **verify —** this term is similar in context to "confirm", but has more rigorous connotations. This term when used in the context of evaluator actions indicates that an independent effort is required of the evaluator.

4.1 Terms and definitions related to the ADV class

24 The following terms are used in the requirements for software internal structuring. Some of these are derived from the *Institute of Electrical and Electronics Engineers Glossary of software engineering terminology, IEEE Std 610.12-1990*.

25 **administrator —** an entity that has complete trust with respect to all policies implemented by the TSF.

26 **interface —** a rigorous means of interaction with a component or module.

27 **interaction —** a general communication-based relationship between entities.

28 **modular decomposition —** the process of breaking a system into components to facilitate design and development.

29 **cohesion (also called module strength) —** 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.

30 **functional cohesion —** a module with this characteristic 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.

31 **sequential cohesion —** a module with this characteristic 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.

32 **communicational cohesion —** a module with this characteristic 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.

33 **temporal cohesion —** a module with this characteristic contains functions that need to be executed at about the same time. Examples of temporally cohesive modules include *initialisation*, *recovery*, and *shutdown* modules.

34 **logical (or procedural) cohesion —** a module with this characteristic performs similar activities on different data structures. A module exhibits logical cohesion if its functions perform related, but different, operations on different inputs.

35 **coincidental cohesion —** a module with this characteristic performs unrelated, or loosely related, activities.

36 **coupling —** 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.

37 **call tree —** a diagram that identifies the modules in a system and shows which modules call one another. All the modules named in a call tree that originates with (i.e., is rooted by) a specific module are the modules that directly or indirectly implement the functions of the originating module.

38 **software engineering —** the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. As with engineering practises in general, some amount of judgement must be used in applying engineering principles. Many factors affect choices, not just the

application of measures of modular decomposition, layering, and minimisation. For example, a developer may design a system with future applications in mind that will not be implemented initially. The developer may choose to include some logic to handle these future applications without fully implementing them; further, the developer may include some calls to as-yet unimplemented modules, leaving call stubs. The developer's justification for such deviations from well-structured programs will have to be assessed using judgement, as well as the application of good software engineering discipline.

39 **complexity —** this is a measure of how difficult software is to understand, and thus to analyse, test, and maintain. Reducing complexity is the ultimate goal for using modular decomposition, layering and minimisation. Controlling coupling and cohesion contributes significantly to this goal.

40 A good deal of effort in the software engineering field has been expended in attempting to develop metrics to measure the complexity of source code. Most of these metrics use easily computed properties of the source code, such as the number of operators and operands, the complexity of the control flow graph (cyclomatic complexity), the number of lines of source code, the ratio of comments to executable code, and similar measures. Coding standards have been found to be a useful tool in generating code that is more readily understood.

41 This TSF internals (ADV_INT) family calls for a *complexity analysis* in all components. It is expected that the developer will provide support for the claims that there has been a sufficient reduction in complexity. This support could include the developer's programming standards, and an indication that all modules meet the standard (or that there are some exceptions that are justified by software engineering arguments). It could include the results of tools used to measure some of the properties of the source code. Or it could include other support that the developer finds appropriate.

42 **layering —** the design software such that separate groups of modules (the *layers*) are hierarchically organised to have separate responsibilities such that one layer depends only on layers below it in the hierarchy for services, and provides its services only to the layers above it. Strict layering adds the constraint that each layer receives services only from the layer immediately beneath it, and provides services only to the layer immediately above it.

4.2 Terms and definitions related to the AGD class

43 **installation —** the procedures which the user has to perform normally only once after receipt and acceptance of the TOE to progress it to the secure configuration as described in the ST including the embedding of the TOE in its operational environment. - If similar processes have to be performed by the developer they are denoted as “generation” throughout ALC: Life-cycle support. - If the TOE requires an initial start-up which has not to be repeated regularly, this process would be classified as installation here.

44 **operation —** the end-usage phase of the TOE. - This includes “normal usage”, administration and maintenance of the TOE.

45 **preparation —** the product life-cycle phase comprising the customer's acceptance of the delivered TOE and its installation, progressing the TOE to a state ready for operation.

4.3 Terms and definitions related to the ALC class

46 **acceptance criteria —** the criteria to be applied when performing the acceptance procedures (e.g. successful document review, or successful testing in the case of software, firmware or hardware).

47 **acceptance procedures —** the procedures followed in order to accept newly created or modified configuration items as part of the TOE, or to move them to the next step of the life-cycle. These procedures identify the roles or individuals responsible for the acceptance and the criteria to be applied to decide on the acceptance.

48 There are several types of acceptance situations some of which may overlap:

a) accepting an item into the CM system for the first time, in particular inclusion of software, firmware and hardware components from other manufacturers into the TOE (“integration”);

b) moving configuration items to the next life-cycle phase at each stage of the construction of the TOE (e.g. module, subsystem, system);

c) subsequent to transports between different development sites;

d) subsequent to the delivery of the TOE to the consumer.

49 **CM documentation (documentation of the CM system) —** overall term for the CM usage documentation and the CM output documentation.

50 **CM evidence —** everything that may be used to establish confidence in the correct operation of the CM system, e.g. CM output, rationales provided by the developer, observations, experiments or interviews made by the evaluator during a site visit.

51 **CM item (configuration item) —** object managed by the CM system during the TOE development. - These may be either parts of the TOE or objects related to the development of the TOE like evaluation documents or development tools. CM items may be stored in the CM system directly (e.g. for files) or by reference (e.g. for hardware parts) together with their version.

52 **CM list (configuration list) —** a CM output document listing all configuration items for a specific product together with the exact version of each CM item relevant for a specific version of the complete product. - This list allows distinguishing the items belonging to the evaluated version of the product from other versions of these items belonging to other versions of the

product. The final CM list is a specific document for a specific version of a specific product. (Of course the list can be an electronic document inside of a CM tool. In that case it can be seen as a specific view into the system or a part of the system rather than an output of the system. However, for the practical use in an evaluation the configuration list will probably be delivered as a part of the evaluation documentation.) The configuration list defines the items that are under the CM requirements of ALC_CMC.

53 **CM output —** CM related results produced or enforced by the CM system. - These CM related results could occur as documents (e.g. filled paper forms, CM system records, logging data, hard copies and electronic output data) as well as actions (e.g. manual measures to fulfil CM instructions). Examples of such CM outputs are configuration lists, CM plans and/or behaviours during the product life-cycle.

54 **CM plan —** part of the CM documentation describing, how the CM system is used for the TOE. The objective of issuing a CM plan is that each staff can see clearly what he has to do. - From the point of view of the overall CM system this can be seen as an output document (because it may be produced as part of the application of the CM system). From the point of view of the concrete project it is a usage document because members of the project team use it in order to understand, which steps they have to do during the project. The CM plan defines the usage of the system for the specific product; the same system may be used to a different extent for other products. That means the CM plan defines and describes the output of the CM system of a company which is used during the TOE development.

55 **CM system —** overall term for the set of procedures and tools (including their documentation) used by a developer to develop and maintain configurations of his products during their life-cycles. - CM systems may have varying degrees of rigour and function. At higher levels, CM systems may be automated, with flaw remediation, change controls, and other tracking mechanisms.

56 **CM system records —** those CM output documents which are produced during the operation of the CM system documenting important activities. Examples of CM system records are CM item change control forms or CM item access approval forms.

57 **CM tools —** tools realising or supporting a CM system, e.g. tools for the version management of the parts of the TOE. They may require manual operation or may be automated.

58 **CM usage documentation —** part of the CM system, which describes how the CM system is defined and applied by using e.g. handbooks, regulations and/or documentation of tools and procedures.

59 **delivery —** the product life-cycle phase which is concerned with the transmission of the finished TOE from the production environment into the hands of the customer. - This may include packaging and storage at the

development site, but does not include transports of the unfinished TOE or parts of the TOE between different developers or different development sites.

60 **develop —** overall term for the activities of the developer to perform the development of the TOE.

61 **developer —** the company in charge of the development of the TOE.

62 **development —** the product life-cycle phase which is concerned with generating the implementation representation of the TOE. - Throughout the ALC requirements, development and related terms (developer, develop) are meant in the more general sense to comprise development *and production*.

63 **development tools —** tools (including test software, if applicable) supporting the development and production of the TOE. - E.g. for a software TOE, development tools are usually programming languages, compilers, linkers and generating tools.

64 **implementation representation —** the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement. - Source code that is then compiled or a hardware drawing that is used to build the actual hardware are examples of parts of an implementation representation.

65 **life-cycle —** the sequence of stages of existence of an object (e.g. a product or a system) in time.

66 **life-cycle definition —** the definition of the life-cycle model.

67 **life-cycle model —** description of the stages and their relations to each other that are used in the management of the life-cycle of a certain object, how the sequence of stages looks like and which high level characteristics the stages have.

68 **measurable life-cycle model —** a life-cycle 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.

69 **production —** the production life-cycle phase follows the development phase and consists of transforming the implementation representation into the implementation of the TOE, i.e. into a state acceptable for delivery to the customer. - This phase may comprise manufacturing, integration, generation, internal transports, storage, and labelling of the TOE.

70 **standardised life-cycle model —** a standardised life-cycle model is a model that has been approved by some group of experts (e.g. academic experts, standards bodies). - Standardised life-cycle models are normally public, well accepted common practise in a specific industry, but developer specific models are not excluded; the emphasis is on the expertise. Examples of

standardised life-cycle models are the Waterfall Model, Rapid Prototyping or the V-Model and the Spiral Model.

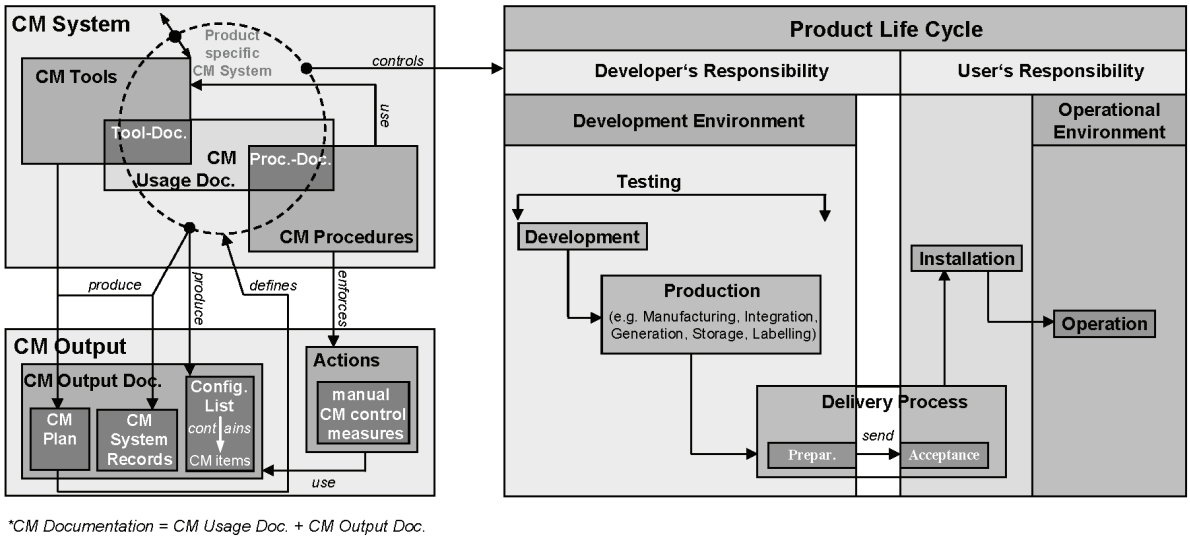

Figure 1 - Terminology in CM and in the product life-cycle

4.4 Terms and definitions related to the AVA class

71 **vulnerability —** a weakness in the TOE that can be used to violate a security policy in some environment.

72 **potential vulnerability —** a weakness the existence of which is suspected (by virtue of a postulated attack path), but not confirmed, to violate the TSP.

73 **exploitable vulnerability —** a weakness in the TOE that can be used to violate the TSP in the operational environment for the TOE.

74 **residual vulnerability —** a weakness that cannot be exploited in the operational environment for the TOE, but that could be used to violate the TSP by an attacker with greater attack potential than is anticipated in the operational environment for the TOE.

75 **encountered potential vulnerabilities —** potential weakness in the TOE identified by the evaluator while performing evaluation activities that could be used to violate the TSP

4.5 Terms and definitions related to the ACO class

76 **base component —** the entity in a composed TOE, which has itself been the subject of an evaluation, providing services and resources to a dependent entity.

77 **dependent component —** the entity in a composed TOE, which is itself the subject of an evaluation, relying on the provision on services by a trusted IT entity.

5 Symbols and abbreviated terms

78 For the purposes of this document, the symbols and abbreviated terms given in CC Part 1 apply.

6 Overview

6.1 Organisation of CC Part 3

79 Chapter 7 describes the paradigm used in the security assurance requirements of CC Part 3.

80 Chapter 8 describes the presentation structure of the assurance classes, families, components, evaluation assurance levels along with their relationships, and the structure of the composition assurance packages. It also characterises the assurance classes and families found in chapter 10 and in chapters 14 through 20.

81 Chapter 9 summarises the assurance classes and families.

82 Chapter 10 provides a brief introduction to the evaluation criteria for PPs, followed by detailed explanations of the families and components that are used for those evaluations.

83 Chapter 11 provides detailed definitions of the EALs.

84 Chapter 12 provides detailed definitions of the CAPs.

85 Chapter 13 provides a brief introduction to the assurance classes and is followed by chapters 14 through 20 that provide detailed definitions of those classes.

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

87 Annex B provides an explanation of the concepts behind composition evaluations and the Composition criteria.

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

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

90 Annex E provides a cross reference between the EALs and the assurance components.

91 Annex F provides a cross reference between the CAPs and the assurance components.

7 Assurance paradigm

92 The purpose of this chapter is to document the philosophy that underpins the CC approach to assurance. An understanding of this chapter will permit the reader to understand the rationale behind the CC Part 3 assurance requirements.

7.1 CC philosophy

93 The CC philosophy is that the threats to security and organisational security policy commitments should be clearly articulated and the proposed security measures be demonstrably sufficient for their intended purpose.

94 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.

7.2 Assurance approach

95 The CC 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, the CC adopts the same philosophy. The CC proposes measuring the validity of the documentation and of the resulting IT product by expert evaluators with increasing emphasis on scope, depth, and rigour.

96 The CC 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 the CC, which is so structured as to allow their future introduction.

7.2.1 Significance of vulnerabilities

97 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.

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

99 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.

7.2.2 Cause of vulnerabilities

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

c) 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.

7.2.3 CC assurance

101 Assurance is grounds for confidence that an IT product meets its security objectives. Assurance can be derived from reference to sources such as unsubstantiated assertions, prior relevant experience, or specific experience. However, the CC provides assurance through active investigation. Active investigation is an evaluation of the IT product in order to determine its security properties.

7.2.4 Assurance through evaluation

102 Evaluation has been the traditional means of gaining assurance, and is the basis of the CC 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.

7.3 The CC evaluation assurance scale

103 The CC 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 level of 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.

8 Security assurance components

8.1 Security assurance classes, families and components structure

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

105 Figure 2 illustrates the SARs defined in this CC Part 3. 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.

8.1.1 Assurance class structure

106 Figure 2 illustrates the assurance class structure.

8.1.1.1 Class name

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

108 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.

8.1.1.2 Class introduction

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

8.1.1.3 Assurance families

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

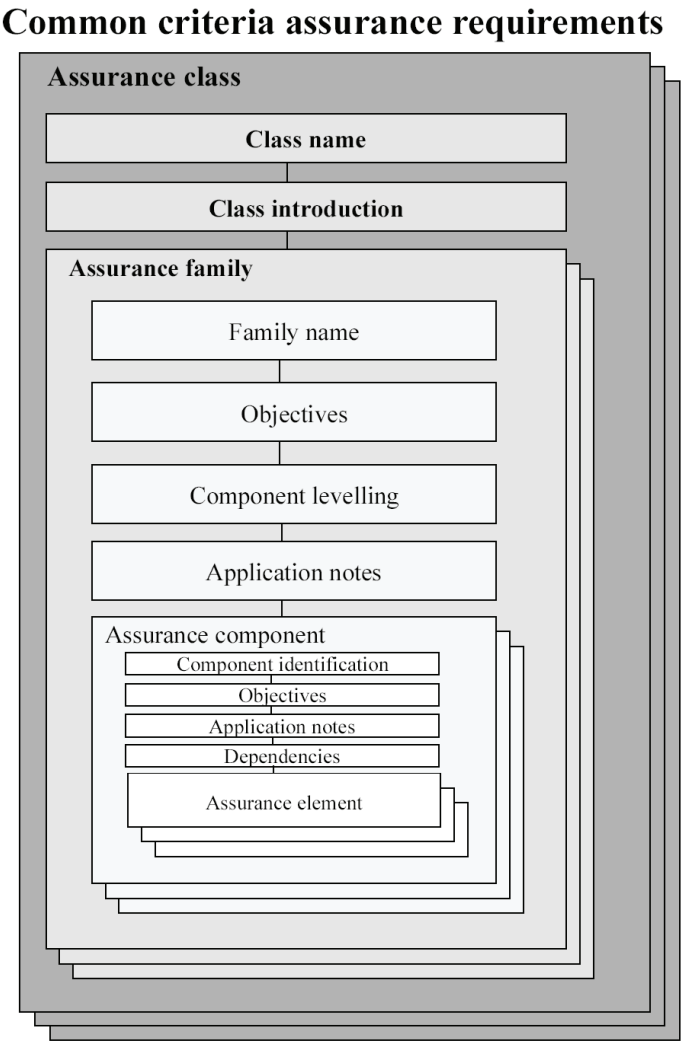

Figure 2 - Assurance class/family/component/element hierarchy

8.1.2 Assurance family structure

111 Figure 2 illustrates the assurance family structure.

8.1.2.1 Family name

112 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.

113 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.

8.1.2.2 Objectives

114 The objectives section of the assurance family presents the intent of the assurance family.

115 This section describes the objectives, particularly those related to the CC 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.

8.1.2.3 Component levelling

116 Each assurance family contains one or more assurance components. This section 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.

117 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.

8.1.2.4 Application notes

118 The application notes section 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.

8.1.2.5 Assurance components

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

8.1.3 Assurance component structure

120 Figure 3 illustrates the assurance component structure.

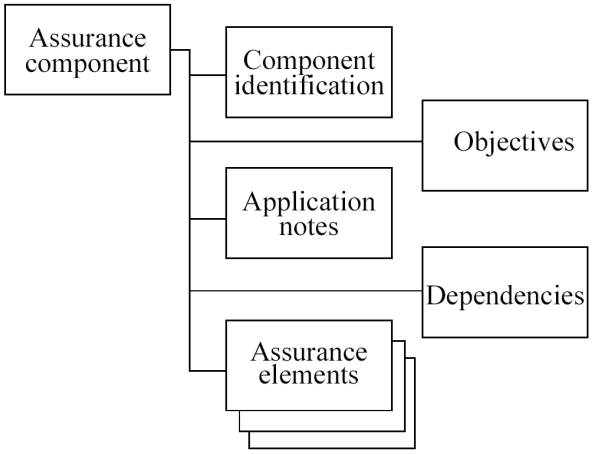

Figure 3 - Assurance component structure

121 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.

8.1.3.1 Component identification

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

123 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.

124 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.

8.1.3.2 Objectives

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

8.1.3.3 Application notes

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

8.1.3.4 Dependencies

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

128 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.

129 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.

130 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.

8.1.3.5 Assurance elements

131 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 the CC.

132 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.

133 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.

134 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 chapters 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.

135 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.

8.1.4 Assurance elements

136 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.

8.1.5 Component taxonomy

137 This CC Part 3 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.

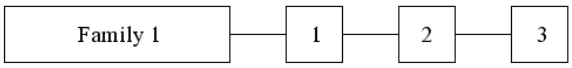

Figure 4 - Sample class decomposition diagram

138 In Figure 4, 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 CC Part 3 are all linearly hierarchical, although linearity is not a mandatory criterion for assurance families that may be added in the future.

8.2 EAL structure

139 Figure 5 illustrates the EALs and associated structure defined in this CC Part 3. Note that while the figure shows the contents of the assurance components, it is intended that this information would be included in an EAL by reference to the actual components defined in the CC.

Part 3 Assurance levels

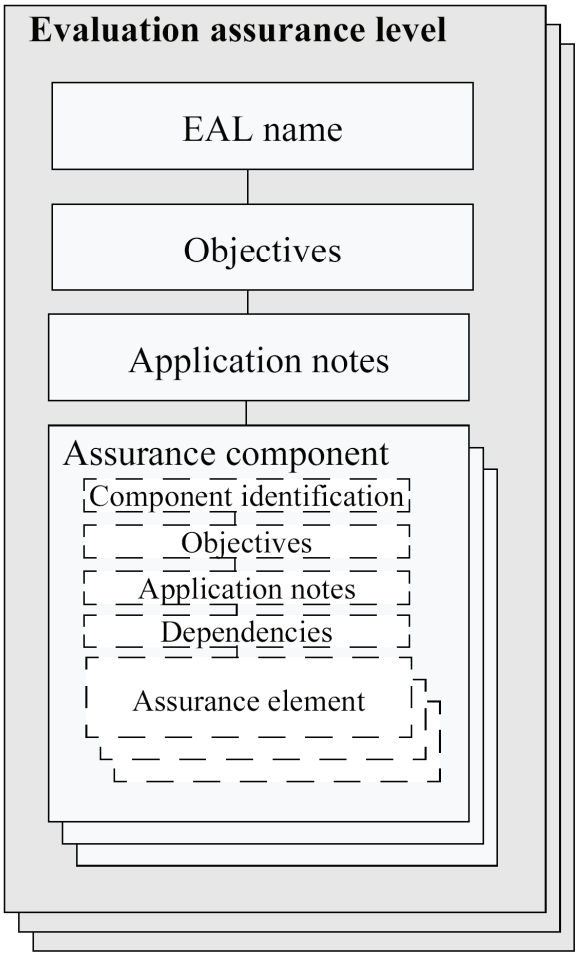

Figure 5 - EAL structure

8.2.1 EAL name

140 Each EAL is assigned a unique name. The name provides descriptive information about the intent of the EAL.

141 A unique short form of the EAL name is also provided. This is the primary means used to reference the EAL.

8.2.2 Objectives

142 The objectives section of the EAL presents the intent of the EAL.

8.2.3 Application notes

143 The application notes section of the EAL, if present, contains information of particular interest to users of the EAL (e.g. PP and ST authors, designers of TOEs targeting this EAL, evaluators). The presentation is informal and covers, for example, warnings about limitations of use and areas where specific attention may be required.

8.2.3.1 Assurance components

144 A set of assurance components have been chosen for each EAL.

145 A higher level of assurance than that provided by a given EAL can be achieved by:

a) including additional assurance components from other assurance families; or

b) replacing an assurance component with a higher level assurance component from the same assurance family.

8.2.4 Relationship between assurances and assurance levels

146 Figure 6 illustrates the relationship between the SARs and the assurance levels defined in the CC. While assurance components further decompose into assurance elements, assurance elements cannot be individually referenced by assurance levels. Note that the arrow in the figure represents a reference from an EAL to an assurance component within the class where it is defined.

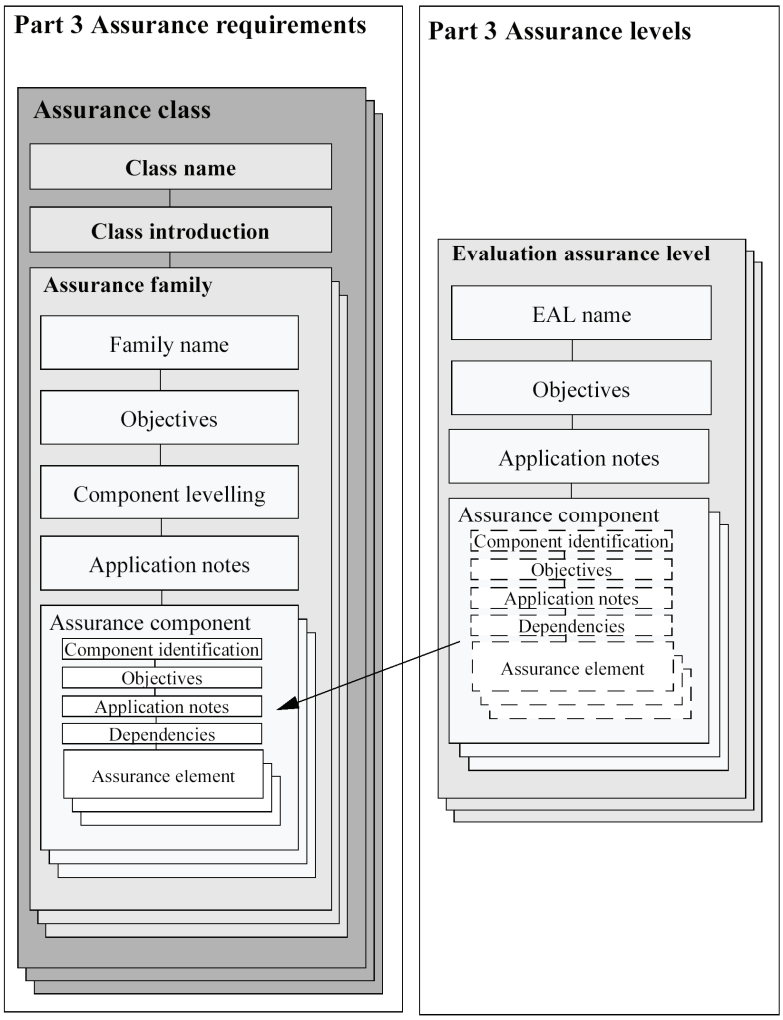

Figure 6 - Assurance and assurance level association

8.3 CAP structure

147 The structure of the CAPs is consistent with that of the EALs. The main difference between these two types of package is the type of TOE they apply to; the EALs applying to component TOEs and the CAPs applying to composed TOEs.

148 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.

9 Overview of assurance classes

149 The following summarises the assurance classes and families of chapter 10 and of chapters 14-20 .

150 The assurance classes, families, and the abbreviation for each family are shown in Table 1.

Assurance Class	Assurance Family
ACO: Composition	Composition rationale (ACO_COR)
	Development evidence (ACO_DEV)
	Reliance of dependent component (ACO_REL)
	Base TOE testing (ACO_TBT)
	Composition vulnerability analysis (ACO_VUL)
ADV: Development	Architectural design (ADV_ARC)
	Functional specification (ADV_FSP)
	Implementation representation (ADV_IMP)
	TSF internals (ADV_INT)
	Security policy modelling (ADV_SPM)
	TOE design (ADV_TDS)
AGD: Guidance documents	Operational user guidance (AGD_OPE)
	Preparative user guidance (AGD_PRE)
ALC: Life-cycle support	CM capabilities (ALC_CMC)
	CM scope (ALC_CMS)
	Delivery (ALC_DEL)
	Development security (ALC_DVS)
	Flaw remediation (ALC_FLR)
	Life-cycle definition (ALC_LCD)
	Tools and techniques (ALC_TAT)
ASE: Security Target evaluation	Conformance claims (ASE_CCL)
	Extended components definition (ASE_ECD)
	ST introduction (ASE_INT)
	Security objectives (ASE_OBJ)
	Security requirements (ASE_REQ)
	Security problem definition (ASE_SPD)
	TOE summary specification (ASE_TSS)
ATE: Tests	Coverage (ATE_COV)
	Depth (ATE_DPT)
	Functional tests (ATE_FUN)
	Independent testing (ATE_IND)
AVA: Vulnerability assessment	Vulnerability analysis (AVA_VAN)

Table 1 Assurance family breakdown and mapping

9.1 Class APE: Protection Profile evaluation

151 Assurance class APE: Protection Profile evaluation defines requirements for the evaluation of an PP to demonstrate that the PP is sound and internally consistent, and, if the PP is based on one or more PPs or packages, that the PP is a correct instantiation of these PPs and packages.

9.1.1 PP introduction (APE_INT)

152 The PP introduction describes the TOE in a narrative way.

9.1.2 Conformance claims (APE_CCL)

153 Conformance claims describes how the Protection Profile conforms to CC Part 2 and CC Part 3, to Protection Profiles and to packages.

9.1.3 Security problem definition (APE_SPD)

154 The security problem definition defines the problem addressed by the TOE, the operational environment of the TOE and the development environment of the TOE.

9.1.4 Security objectives (APE_OBJ)

155 The security objectives are a concise statement of the intended response to the security problem.

9.1.5 Extended components definition (APE_ECD)

156 Extended security requirements are requirements that are not based on components from CC Part 2 or CC Part 3, but are based on extended components: components defined by the PP author.

157 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 could not have been clearly expressed using existing CC Part 2 or CC Part 3 components.

9.1.6 Security requirements (APE_REQ)

158 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.

9.2 Class ASE: Security Target evaluation

159 Assurance class ASE: Security Target evaluation defines requirements for the evaluation of an ST, to demonstrate that the ST is sound and internally consistent, and, if the ST is based on one or more PPs or packages, that the ST is a correct instantiation of these PPs and packages.

9.2.1 ST introduction (ASE_INT)

160 The ST introduction describes the TOE in a narrative way on three levels of abstraction.

9.2.2 Conformance claims (ASE_CCL)

161 Conformance claims describes how the Security Target conforms to CC Part 2 and CC Part 3, to Protection Profiles and to packages.

9.2.3 Security problem definition (ASE_SPD)

162 The security problem definition defines the problem addressed by the TOE, the operational environment of the TOE and the development environment of the TOE.

9.2.4 Security objectives (ASE_OBJ)

163 The security objectives are a concise statement of the intended response to the security problem.

9.2.5 Extended components definition (ASE_ECD)

164 Extended components are defined wherever it is impossible to clearly express requirements using only components from CC Part 2 and/or CC Part 3.

9.2.6 Security requirements (ASE_REQ)

165 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.

9.2.7 TOE summary specification (ASE_TSS)

166 The TOE Summary specification allows evaluators and potential consumers of the TOE to understand how the TOE meets its SFRs.

9.3 Class ADV: Development

167 The purpose of the Development class is to provide evidence about the TOE. Without the knowledge about the TOE that is gained from this information, there could be no useful vulnerability analysis or testing conducted upon the TOE (as described in the AVA and ATE classes).

9.3.1 Architectural design (ADV_ARC)

168 The information presented for the architectural design of the TOE is related to the information contained in other decomposition documentation (functional specification and TOE design documentation) provided for the TSF, but presents the design in a manner that supports architectural

arguments (e.g., the TSP cannot be compromised; the TSF provides security domains consistent with its SFRs; the TSF cannot be bypassed).

9.3.2 Functional specification (ADV_FSP)

169 The information presented in the functional specification describes the interfaces through which the TSF services are invoked. At the lower levels of assurance, there is an effort to reduce the amount of information that must be supplied by requiring only the most security-critical information.

9.3.3 Implementation representation (ADV_IMP)

170 The implementation representation of the TOE (and, at higher levels, the implementation itself) is made available so that it can be analysed by the evaluator 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). At higher levels, the implementation representation is demonstrated to be transformable to the implementation that is used for testing.

171 The implementation representation captures the detailed internal workings of the TSF. This may be software source code, firmware source code, hardware diagrams and/or chip specifications.

9.3.4 TSF internals (ADV_INT)

172 The internal structure of the TSF can aid or hamper understandability of the implementation representation. Source code that conforms to coding standards, that exhibit a minimum of interactions, and that is written in modules each with a single purpose, is much easier to understand than poorly-structured code with unnecessary or loosely-defined interactions.

9.3.5 Security policy modelling (ADV_SPM)

173 A formal security model precisely describes important aspects of security and their relationship to the behaviour of the TSF. Formalism helps to prove mathematically the thoroughness of the security.

9.3.6 TOE design (ADV_TDS)

174 The design description provides a further-refined description of the TSF from that presented in the functional specification. The functional specification provides a description of *what* the TSF does at its interface; the design description provides more insight into the TSF by describing *how* the TSF works in order to perform the functions supporting the SFRs. At lower assurance levels, complete details relating to all portions of the TSF are not required. As the desired assurance increases, more detail is made available so that analysis can be performed that supports the assurance claims being made.

9.4 Class AGD: Guidance documents

175 Assurance class AGD: Guidance documents defines requirements directed at the understandability, coverage and completeness of the preparative and operational documentation provided by the developer. This documentation, which provides information for all user roles, is an important factor in the secure preparation and operation of the TOE.

9.4.1 Operational user guidance (AGD_OPE)

176 Requirements for operational user guidance help ensure that all types of users are able to operate the TOE in a secure manner (e.g. the usage constraints assumed by the PP or ST must be clearly explained and illustrated). It should be excluded that the TOE can be used in a manner that is insecure but that the user of the TOE would reasonably believe to be secure. Operational user guidance is the primary vehicle available to the developer for providing the TOE users with the necessary background and specific information on how to correctly use the TOE's protection functions.

177 Operational user guidance must do two things. First, it needs to explain what the security functionality accessible by the user does and how it is to be used, so that users are able to consistently and effectively protect their information. Second, it needs to explain the user's role in maintaining the TOE's security.

9.4.2 Preparative user guidance (AGD_PRE)

178 Preparation requires that the delivered copy of the TOE is accepted, configured and activated by the user to exhibit the protection properties as needed during operation of the TOE. The preparative procedures provide confidence that the user will be aware of the TOE configuration parameters and how they can affect the TSF.

9.5 Class ALC: Life-cycle support

179 Assurance class ALC: Life-cycle support defines requirements for assurance through the adoption of a well defined life-cycle model for all the steps of the TOE development, including flaw remediation procedures and policies, correct use of tools and techniques and the security measures used to protect the development environment.

180 Configuration management (CM) helps to ensure that the integrity of the TOE is preserved, by preventing unauthorised modifications, additions, or deletions to the TOE, thus providing assurance that the TOE and documentation used for evaluation are the ones prepared for distribution.

181 The delivery procedures define requirements for the measures, procedures, and standards concerned with secure delivery of the TOE, ensuring that the security protection offered by the TOE is not compromised during the transfer to the user.

9.5.1 CM capabilities (ALC_CMC)

182 Configuration management capabilities define the characteristics of the configuration management system.

9.5.2 CM scope (ALC_CMS)

183 Configuration management scope indicates the TOE items that need to be controlled by the configuration management system.

9.5.3 Delivery (ALC_DEL)

184 Delivery covers the procedures used to maintain security during transfer of the TOE to the user, both on initial delivery and as part of subsequent modification. It includes special procedures or operations required to demonstrate the authenticity of the delivered TOE. Such procedures and measures are the basis for ensuring that the security protection offered by the TOE is not compromised during transfer. While compliance with the delivery requirements cannot always be determined when a TOE is evaluated, it is possible to evaluate the procedures that a developer has developed to distribute the TOE to users.

9.5.4 Development security (ALC_DVS)

185 Development security covers the physical, procedural, personnel, and other security measures used in the development environment. It includes physical security of the development location(s) and controls on the selection and hiring of development staff.

9.5.5 Flaw remediation (ALC_FLR)

186 Flaw remediation ensures that flaws discovered by the TOE consumers will be tracked and corrected while the TOE is supported by the developer. While future compliance with the flaw remediation requirements cannot be determined when a TOE is evaluated, it is possible to evaluate the procedures and policies that a developer has in place to track and repair flaws, and to distribute the repairs to consumers.

9.5.6 Life-cycle definition (ALC_LCD)

187 Life-cycle definition establishes that the engineering practises used by a developer to produce the TOE include the considerations and activities identified in the development process and operational support requirements. Confidence in the correspondence between the requirements and the TOE is greater when security analysis and the production of evidence are done on a regular basis as an integral part of the development process and operational support activities. It is not the intent of this component to dictate any specific development process.

9.5.7 Tools and techniques (ALC_TAT)

188 Tools and techniques addresses the need to define the development tools being used to analyse and implement the TOE. It includes requirements concerning the development tools and implementation dependent options of those tools.

9.6 Class ATE: Tests

189 Assurance class ATE: Tests states testing requirements that demonstrate that the TOE matches its design descriptions as provided in the ADV: Development class.

9.6.1 Coverage (ATE_COV)

190 Coverage deals with the completeness of the functional tests performed by the developer on the TOE. It addresses the extent to which the TSF is tested.

9.6.2 Depth (ATE_DPT)

191 Depth deals with the level of detail to which the developer tests the TSF. Testing is based upon increasing depth of information derived from analysis of the TSF representations.

9.6.3 Functional tests (ATE_FUN)

192 Functional testing establishes that the tests performed by the developer are performed and documented correctly.

9.6.4 Independent testing (ATE_IND)

193 Independent testing specifies the degree to which the testing of the TSF must be performed by a party other than the developer (e.g. a third party). This family adds value by the introduction of tests that are not part of the developer's tests.

9.7 Class AVA: Vulnerability assessment

194 Assurance class AVA: Vulnerability assessment defines requirements directed at the identification of exploitable vulnerabilities. Specifically, it addresses those vulnerabilities introduced in the construction, operation, misuse, or incorrect configuration of the TOE.

9.7.1 Vulnerability analysis (AVA_VAN)

195 Vulnerability analysis consists of the identification of flaws potentially introduced in the different refinement steps of the development. It results in the definition of penetration tests through the collection of the necessary information concerning: (1) the completeness of the TSF (does the TSF counter all the postulated threats?), (2) the dependencies between all SFRs and (3) whether any of the SFRs can be undermined through unexpected behaviour of the TOE. These potential vulnerabilities are assessed through

penetration testing to determine whether they could, in practise, be exploitable to compromise the security of the TOE.

196 The characteristics of different levels of attack potential are discussed in CEM Annex B.3, When is attack potential used.

9.8 Class ACO: Composition

197 Assurance class ACO: Composition defines requirements of the information necessary to ensure that two or more components, which have themselves been the subject of evaluation, can be integrated in a secure manner.

198 The ACO: Composition assurance requirements will be applied to the composed TOE to:

a) determine that the required assurance is provided by the base component;

b) determine that the base component and dependent component are compatible; and

c) search for any vulnerabilities introduced through composing the base and dependent components into a single composed TOE entity.

9.8.1 Composition rationale (ACO_COR)

199 The Composition rationale (ACO_COR) family is used to determine whether or not the appropriate assurance measures have been applied to the base for successful integration in the composed TOE. That is, the SARs claimed by the base component are consistent with the SARs in the assurance package for the composed TOE. (e.g. if the assurance package for the composed TOE included ACO_DEV.3 Detailed evidence of design, a base component that was evaluated against ADV_FSP.2 Security-enforcing functional specification would not have had the appropriate assurance measures applied, as insufficient design evidence would have been examined.)

200 The Composition rationale (ACO_COR) family calls for evidence that the appropriate assurance is provided, without being specific about how this is achieved. If the appropriate evidence is not available, then it may be necessary to report an assessment of the residual risk to assist consumers of the composed TOE (e.g. accreditors). This report would need to identify the change to the base component that may have an effect on the assurance gained during the original evaluation, along with any known effects.

9.8.2 Development evidence (ACO_DEV)

201 Development evidence (ACO_DEV) provides details of the base component interfaces and internals in increasing levels of detail, mirroring the level of detail provided by Reliance of dependent component (ACO_REL). The application of these two families will provide the specifications of security

services from each perspective of the TSF making the call and the TSF servicing the call.

202 Having the two descriptions then allows a determination to be made, as part of the Development evidence (ACO_DEV) activities (ACO_DEV.*.2E actions), that these two descriptions are consistent.

9.8.3 Reliance of dependent component (ACO_REL)

203 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 the base component evaluation because the service in the base component was not considered security-relevant, either because of the inherent purpose of the service (e.g., adjust type font) or because associated CC SFRs are not being claimed in the base component's ST (e.g. the login interface when no FIA: Identification, Authentication and Binding SFRs are claimed). These interfaces into the base component are often viewed as functional interfaces when evaluating the base component, and are in addition to the security interfaces (TSFI) considered in the functional specification.

9.8.4 Base TOE testing (ACO_TBT)

204 The family Base TOE testing (ACO_TBT) states testing requirements that demonstrate that the base component interfaces match the design descriptions as provided in the development information (Development evidence (ACO_DEV)). Testing evidence is to be provided for all base component interfaces used by the dependent component, as identified in Reliance of dependent component (ACO_REL).

205 The testing of the composed TOE is performed during the dependent component evaluation activities, as the composed TOE configuration is (one of) the configurations used for satisfaction of the ATE: Tests requirements.

9.8.5 Composition vulnerability analysis (ACO_VUL)

206 The vulnerability analysis in Composition vulnerability analysis (ACO_VUL) includes determination of two different aspects of resistance by the composed TOE, namely:

a) Residual vulnerabilities in the base and dependent components remain unexploitable in the operational environment of the composed TOE;

b) The composed TOE is resistant to attackers with a given level of attack potential.

10 Class APE: Protection Profile evaluation

207 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.

208 Figure 7 shows the families within this class, and the hierarchy of components within the families.

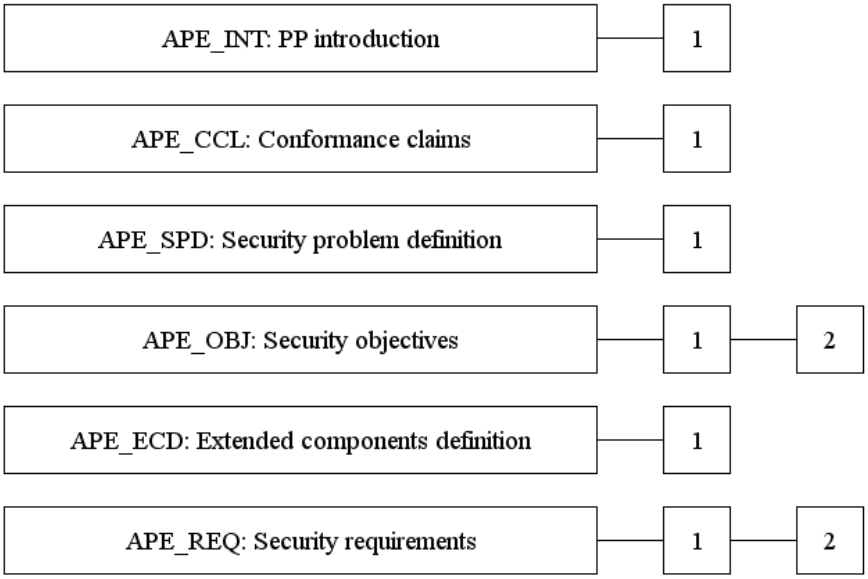

Figure 7 - APE: Protection Profile evaluation class decomposition

10.1 PP introduction (APE_INT)

Objectives

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

210 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.

APE_INT.1 PP introduction

Dependencies: No dependencies.

Developer action elements:

APE_INT.1.1D **The developer shall provide a PP introduction.**

Content and presentation elements:

APE_INT.1.1C **The PP introduction shall contain a PP reference and a TOE overview.**

APE_INT.1.2C **The PP reference shall uniquely identify the PP.**

APE_INT.1.3C **The TOE overview shall summarise the usage and major security features of the TOE.**

APE_INT.1.4C **The TOE overview shall identify the TOE type.**

APE_INT.1.5C **The TOE overview shall identify any non-TOE hardware/software/firmware available to the TOE.**

Evaluator action elements:

APE_INT.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

10.2 Conformance claims (APE_CCL)

Objectives

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

APE_CCL.1 Conformance claims

Dependencies: APE_INT.1 PP introduction
APE_ECD.1 Extended components definition
APE_REQ.1 Stated security requirements

Developer action elements:

APE_CCL.1.1D **The developer shall provide a conformance claim.**

APE_CCL.1.2D **The developer shall provide a conformance claim rationale.**

APE_CCL.1.3D **The developer shall provide a conformance statement.**

Content and presentation elements:

APE_CCL.1.1C **The conformance claim shall contain a CC conformance claim that identifies the version of the CC to which the PP claims conformance.**

APE_CCL.1.2C **The CC conformance claim shall describe the conformance of the PP to CC Part 2 as either CC Part 2 conformant or CC Part 2 extended.**

APE_CCL.1.3C **The CC conformance claim shall describe the conformance of the PP to CC Part 3 as either CC Part 3 conformant or CC Part 3 extended.**

APE_CCL.1.4C **The CC conformance claim shall be consistent with the extended components definition.**

APE_CCL.1.5C **The conformance claim shall identify all PPs and security requirement packages to which the PP claims conformance.**

APE_CCL.1.6C **The conformance claim shall describe any conformance of the PP to a package as either package-conformant or package-augmented.**

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.**

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.**

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.**

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.**

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.**

Evaluator action elements:

APE_CCL.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

10.3 Security problem definition (APE_SPD)

Objectives

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

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

APE_SPD.1 Security problem definition

Dependencies: No dependencies.

Developer action elements:

APE_SPD.1.1D **The developer shall provide a security problem definition.**

Content and presentation elements:

APE_SPD.1.1C **The security problem definition shall describe the threats.**

APE_SPD.1.2C **All threats shall be described in terms of a threat agent, an asset, and an adverse action.**

APE_SPD.1.3C **The security problem definition shall describe the OSPs.**

APE_SPD.1.4C **The security problem definition shall describe the assumptions about the operational environment of the TOE.**

Evaluator action elements:

APE_SPD.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

10.4 Security objectives (APE_OBJ)

Objectives

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

215 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, its development environment, and its operational environment is clearly defined, and that the security objectives are internally consistent.

Component levelling

216 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 and security objectives for the development environment.

APE_OBJ.1 Security objectives for the operational environment

Dependencies: No dependencies.

Developer action elements:

APE_OBJ.1.1D **The developer shall provide a statement of security objectives.**

Content and presentation elements:

APE_OBJ.1.1C **The statement of security objectives shall describe the security objectives for the operational environment.**

Evaluator action elements:

APE_OBJ.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

APE_OBJ.2 Security objectives

Dependencies: APE_SPD.1 Security problem definition

Developer action elements:

APE_OBJ.2.1D The developer shall provide a statement of security objectives.

APE_OBJ.2.2D **The developer shall provide a security objectives rationale.**

Content and presentation elements:

APE_OBJ.2.1C **The statement of security objectives shall describe the security objectives for the TOE, the security objectives for the development environment and the security objectives for the operational environment.**

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.**

APE_OBJ.2.3C **The security objectives rationale shall trace each security objective for the development environment back to threats countered by that security objective and OSPs enforced by that security objective.**

APE_OBJ.2.4C **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.**

APE_OBJ.2.5C The security objectives **rationale** shall **demonstrate that** the security objectives **counter all threats.**

APE_OBJ.2.6C **The security objectives rationale shall demonstrate that the security objectives enforce all OSPs.**

APE_OBJ.2.7C **The security objectives rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.**

Evaluator action elements:

APE_OBJ.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

10.5 Extended components definition (APE_ECD)

Objectives

217 Extended security requirements are requirements that are not based on components from CC Part 2 or CC Part 3, but are based on extended components: components defined by the ST author.

218 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 could not have been clearly expressed using existing CC Part 2 or CC Part 3 components.

APE_ECD.1 Extended components definition

Dependencies: No dependencies.

Developer action elements:

APE_ECD.1.1D **The developer shall provide a statement of security requirements**

APE_ECD.1.2D **The developer shall provide an extended components definition.**

Content and presentation elements:

APE_ECD.1.1C **The statement of security requirements shall identify all extended security requirements.**

APE_ECD.1.2C **The extended components definition shall define an extended component for each extended security requirement.**

APE_ECD.1.3C **The extended components definition shall describe how each extended component is related to the existing CC components, families, and classes.**

APE_ECD.1.4C **The extended components definition shall use the existing CC components, families, classes, and methodology as a model for presentation.**

APE_ECD.1.5C **The extended components shall consist of measurable and objective elements such that compliance or noncompliance to these elements can be demonstrated.**

Evaluator action elements:

APE_ECD.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

APE_ECD.1.2E **The evaluator *shall confirm* that no extended component can be clearly expressed using existing components.**

10.6 Security requirements (APE_REQ)

Objectives

219 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.

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

Component levelling

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

APE_REQ.1 Stated security requirements

Dependencies: APE_ECD.1 Extended components definition

Developer action elements:

APE_REQ.1.1D The developer shall provide a security requirements rationale.

Content and presentation elements:

APE_REQ.1.1C The statement of security requirements shall describe the SFRs and the SARs.

APE_REQ.1.2C The statement of security requirements shall identify all operations on the security requirements.

APE_REQ.1.3C All operations shall be performed correctly.

APE_REQ.1.4C Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

Evaluator action elements:

APE_REQ.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

APE_REQ.1.2E The evaluator *shall confirm* that the statement of security requirements is internally consistent.

APE_REQ.2 Derived security requirements

Dependencies: APE_OBJ.1 Security objectives for the operational environment
APE_ECD.1 Extended components definition

Developer action elements:

APE_REQ.2.1D The developer shall provide a security requirements rationale.

Content and presentation elements:

APE_REQ.2.1C The statement of security requirements shall describe the SFRs and the SARs.

APE_REQ.2.2C The statement of security requirements shall identify all operations on the security requirements.

APE_REQ.2.3C All operations shall be performed correctly.

APE_REQ.2.4C Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

APE_REQ.2.5C **The security requirements rationale shall trace each SFR back to the security objectives for the TOE.**

APE_REQ.2.6C **The security requirements rationale shall demonstrate that the SFRs meet all security objectives for the TOE.**

APE_REQ.2.7C **The security requirements rationale shall trace each SAR back to the security objectives for the development environment.**

APE_REQ.2.8C **The security requirements rationale shall demonstrate that the SARs meet all security objectives for the development environment.**

Evaluator action elements:

APE_REQ.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

APE_REQ.2.2E The evaluator ***shall confirm*** that the statement of security requirements is internally consistent.

11 Evaluation assurance levels

222 The Evaluation Assurance Levels (EALs) provide an increasing scale that balances the level of assurance obtained with the cost and feasibility of acquiring that degree of assurance. The CC approach identifies the separate concepts of assurance in a TOE at the end of the evaluation, and of maintenance of that assurance during the operational use of the TOE.

223 It is important to note that not all families and components from CC Part 3 are included in the EALs. This is not to say that these do not provide meaningful and desirable assurances. Instead, it is expected that these families and components will be considered for augmentation of an EAL in those PPs and STs for which they provide utility.

11.1 Evaluation assurance level (EAL) overview

224 Table 2 represents a summary of the EALs. The columns represent a hierarchically ordered set of EALs, while the rows represent assurance families. Each number in the resulting matrix identifies a specific assurance component where applicable.

225 As outlined in the next section, seven hierarchically ordered evaluation assurance levels are defined in the CC for the rating of a TOE's assurance. They are hierarchically ordered inasmuch as each EAL represents more assurance than all lower EALs. The increase in assurance from EAL to EAL is accomplished by substitution of a hierarchically higher assurance component from the same assurance family (i.e. increasing rigour, scope, and/or depth) and from the addition of assurance components from other assurance families (i.e. adding new requirements).

226 These EALs consist of an appropriate combination of assurance components as described in chapter 8 of this CC Part 3. More precisely, each EAL includes no more than one component of each assurance family and all assurance dependencies of every component are addressed.

227 While the EALs are defined in the CC, it is possible to represent other combinations of assurance. Specifically, the notion of "augmentation" allows the addition of assurance components (from assurance families not already included in the EAL) or the substitution of assurance components (with another hierarchically higher assurance component in the same assurance family) to an EAL. Of the assurance constructs defined in the CC, only EALs may be augmented. The notion of an "EAL minus a constituent assurance component" is not recognised by the standard as a valid claim. Augmentation carries with it the obligation on the part of the claimant to justify the utility and added value of the added assurance component to the EAL. An EAL may also be augmented with extended assurance requirements.

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		**1**	1	1	1	1	1
	ADV_FSP	**1**	**2**	**3**	**4**	**5**	5	**6**
	ADV_IMP				**1**	1	**2**	2
	ADV_INT					**2**	**3**	**4**
	ADV_SPM						**1**	1
	ADV_TDS		**1**	**2**	**3**	**4**	**5**	**6**
Guidance documents	AGD_OPE	**1**	1	1	1	1	1	1
	AGD_PRE	**1**	1	1	1	1	1	1
Life-cycle support	ALC_CMC	**1**	**2**	**3**	**4**	4	**5**	5
	ALC_CMS	**1**	**2**	**3**	**4**	**5**	5	5
	ALC_DEL		**1**	1	1	1	1	1
	ALC_DVS			**1**	1	1	**2**	2
	ALC_FLR							
	ALC_LCD				**1**	**2**	2	**3**
	ALC_TAT				**1**	**2**	**3**	3
Security Target evaluation	ASE_CCL	**1**	1	1	1	1	1	1
	ASE_ECD	**1**	1	1	1	1	1	1
	ASE_INT	**1**	1	1	1	1	1	1
	ASE_OBJ	**1**	**2**	2	2	2	2	2
	ASE_REQ	**1**	**2**	2	2	2	2	2
	ASE_SPD		**1**	1	1	1	1	1
	ASE_TSS	**1**	1	1	1	1	1	1
Tests	ATE_COV		**1**	**2**	2	2	**3**	3
	ATE_DPT			**1**	1	**2**	2	**3**
	ATE_FUN		**1**	1	1	1	**2**	2
	ATE_IND	**1**	**2**	2	2	2	2	**3**
Vulnerability assessment	AVA_VAN	**1**	**2**	2	**3**	**4**	**5**	5

Table 2 Evaluation assurance level summary

11.2 Evaluation assurance level details

228 The following sections provide definitions of the EALs, highlighting differences between the specific requirements and the prose characterisations of those requirements using bold type.

11.3 Evaluation assurance level 1 (EAL1) - functionally tested

Objectives

229 EAL1 is applicable where some confidence in correct operation is required, but the threats to security are not viewed as serious. It will be of value where independent assurance is required to support the contention that due care has been exercised with respect to the protection of personal or similar information.

230 EAL1 requires only a limited security target. It is sufficient to simply state the SFRs that the TOE must meet, rather than deriving them from threats, OSPs and assumptions through security objectives.

231 EAL1 provides an evaluation of the TOE as made available to the customer, including independent testing against a specification, and an examination of the guidance documentation provided. It is intended that an EAL1 evaluation could be successfully conducted without assistance from the developer of the TOE, and for minimal outlay.

232 An evaluation at this level should provide evidence that the TOE functions in a manner consistent with its documentation.

Assurance components

233 **EAL1 provides a basic level of assurance by a limited security target and an analysis of the SFRs in that ST using a functional and interface specification and guidance documentation, to understand the security behaviour.**

234 **The analysis is supported by a search for potential vulnerabilities in the public domain and independent testing (functional and penetration) of the TSF.**

235 **EAL1 also provides assurance through unique identification of the TOE and of the relevant evaluation documents.**

236 **This EAL provides a meaningful increase in assurance over unevaluated IT.**

Assurance Class	Assurance components
ADV: Development	ADV_FSP.1 Basic functional specification
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.1 Labelling of the TOE
	ALC_CMS.1 TOE CM coverage
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.1 Security objectives for the operational environment
	ASE_REQ.1 Stated security requirements
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_IND.1 Independent testing - conformance
AVA: Vulnerability assessment	AVA_VAN.1 Vulnerability survey

Table 3 EAL1

11.4 Evaluation assurance level 2 (EAL2) - structurally tested

Objectives

237 EAL2 requires the co-operation of the developer in terms of the delivery of design information and test results, but should not demand more effort on the part of the developer than is consistent with good commercial practise. As such it should not require a substantially increased investment of cost or time.

238 EAL2 is therefore applicable in those circumstances where developers or users require a low to moderate level of independently assured security in the absence of ready availability of the complete development record. Such a situation may arise when securing legacy systems, or where access to the developer may be limited.

Assurance components

239 **EAL2** provides assurance by a **full** security target and an analysis of the SFRs in that ST, using a functional and interface specification, guidance documentation **and a basic description of the architecture of the TOE,** to understand the security behaviour.

240 The analysis is supported by independent testing of the TSF, **evidence of developer testing based on the functional specification, selective independent confirmation of the developer test results, and a vulnerability analysis (based upon the functional specification, TOE design, architectural design and guidance evidence provided) demonstrating resistance to penetration attackers with a basic attack potential.**

241 **EAL2** also provides assurance through **use** of **a configuration management system** and **evidence** of **secure delivery procedures.**

242 This EAL **represents** a meaningful increase in assurance **from EAL1 by requiring developer testing, a vulnerability analysis (in addition to the search of the public domain), and independent testing based upon more detailed TOE specifications.**

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.2 Security-enforcing functional specification
	ADV_TDS.1 Basic design
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.2 Use of a CM system
	ALC_CMS.2 Parts of the TOE CM coverage
	ALC_DEL.1 Delivery procedures
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.1 Evidence of coverage
	ATE_FUN.1 Functional testing
	ATE_IND.2 Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.2 Vulnerability analysis

Table 4 EAL2

11.5 Evaluation assurance level 3 (EAL3) - methodically tested and checked

Objectives

243 EAL3 permits a conscientious developer to gain maximum assurance from positive security engineering at the design stage without substantial alteration of existing sound development practices.

244 EAL3 is applicable in those circumstances where developers or users require a moderate level of independently assured security, and require a thorough investigation of the TOE and its development without substantial re-engineering.

Assurance components

245 **EAL3** provides assurance by a full security target and an analysis of the SFRs in that ST, using a functional and interface specification, guidance documentation, and a **an architectural** description of the **design** of the TOE, to understand the security behaviour.

246 The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification **and TOE design,** selective independent confirmation of the developer test results, and a vulnerability analysis (based upon the functional specification, TOE design, architectural design and guidance evidence provided) demonstrating resistance to penetration attackers with a basic attack potential.

247 **EAL3** also provides assurance through **the** use of **development environment controls, TOE** configuration management, and evidence of secure delivery procedures.

248 This EAL represents a meaningful increase in assurance from **EAL2** by requiring **more complete** testing **coverage** of the **security functions** and **mechanisms and/or procedures that provide some confidence that the** TOE **will not be tampered with during development.**

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.3 Functional specification with complete summary
	ADV_TDS.2 Architectural design
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.3 Authorisation controls
	ALC_CMS.3 Implementation representation CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.1 Identification of security measures
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.2 Analysis of coverage
	ATE_DPT.1 Testing: basic design
	ATE_FUN.1 Functional testing
	ATE_IND.2 Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.2 Vulnerability analysis

Table 5 EAL3

11.6 Evaluation assurance level 4 (EAL4) - methodically designed, tested, and reviewed

Objectives

249 EAL4 permits a developer to gain maximum assurance from positive security engineering based on good commercial development practises which, though rigorous, do not require substantial specialist knowledge, skills, and other resources. EAL4 is the highest level at which it is likely to be economically feasible to retrofit to an existing product line.

250 EAL4 is therefore applicable in those circumstances where developers or users require a moderate to high level of independently assured security in conventional commodity TOEs and are prepared to incur additional security-specific engineering costs.

Assurance components

251 **EAL4** provides assurance by a full security target and an analysis of the SFRs in that ST, using a functional and **complete** interface specification, guidance documentation, a description of the **basic modular** design of the TOE, **and a subset of the implementation,** to understand the security behaviour. **Assurance is additionally gained through an informal model of the TOE security policy.**

252 The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification and TOE design, selective independent confirmation of the developer test results, **evidence of a developer search for vulnerabilities,** and a vulnerability analysis (based upon the functional specification, TOE design, **implementation representation,** architectural design and guidance evidence provided) demonstrating resistance to penetration attackers with **an extended-basic** attack potential.

253 **EAL4** also provides assurance through the use of development environment controls **and additional** TOE configuration management **including automation,** and evidence of secure delivery procedures.

254 This EAL represents a meaningful increase in assurance from **EAL3** by requiring more **design description, a subset** of the **implementation,** and **improved** mechanisms and/or procedures that provide confidence that the TOE will not be tampered with during development **or delivery.**

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.4 Complete functional specification
	ADV_IMP.1 Implementation representation of the TSF
	ADV_TDS.3 Basic modular design
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.4 Production support, acceptance procedures and automation
	ALC_CMS.4 Problem tracking CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.1 Identification of security measures
	ALC_LCD.1 Developer defined life-cycle model
	ALC_TAT.1 Well-defined development tools
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.2 Analysis of coverage
	ATE_DPT.1 Testing: basic design
	ATE_FUN.1 Functional testing
	ATE_IND.2 Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.3 Focused vulnerability analysis

Table 6 EAL4

11.7 Evaluation assurance level 5 (EAL5) - semiformally designed and tested

Objectives

255 EAL5 permits a developer to gain maximum assurance from security engineering based upon rigorous commercial development practises supported by moderate application of specialist security engineering techniques. Such a TOE will probably be designed and developed with the intent of achieving EAL5 assurance. It is likely that the additional costs attributable to the EAL5 requirements, relative to rigorous development without the application of specialised techniques, will not be large.

256 EAL5 is therefore applicable in those circumstances where developers or users require a high level of independently assured security in a planned development and require a rigorous development approach without incurring unreasonable costs attributable to specialist security engineering techniques.

Assurance components

257 **EAL5** provides assurance by a full security target and an analysis of the SFRs in that ST, using a functional and complete interface specification, guidance documentation, a description of the design of the TOE, and the implementation, to understand the security behaviour. Assurance is additionally gained through **a formal** model of **select TOE security policies and a semiformal presentation of** the **functional specification and** TOE **design. A modular TSF design is also required.**

258 The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification, TOE design, selective independent confirmation of the developer test results, and **an independent** vulnerability analysis demonstrating resistance to penetration attackers with **a moderate** attack potential.

259 **EAL5** also provides assurance through the use of **a** development environment controls, and **comprehensive** TOE configuration management including automation, and evidence of secure delivery procedures.

260 This EAL represents a meaningful increase in assurance from **EAL4** by requiring **semiformal** design **descriptions,** the **entire** implementation, **a more structured (and hence analysable) architecture,** and improved mechanisms and/or procedures that provide confidence that the TOE will not be tampered with during development.

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.5 Complete semi-formal functional specification with additional error information
	ADV_IMP.1 Implementation representation of the TSF
	ADV_INT.2 Full modular decomposition
	ADV_TDS.4 Semiformal modular design
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.4 Production support, acceptance procedures and automation
	ALC_CMS.5 Development tools CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.1 Identification of security measures
	ALC_LCD.2 Standardised life-cycle model
	ALC_TAT.2 Compliance with implementation standards
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.2 Analysis of coverage
	ATE_DPT.2 Testing: modular design
	ATE_FUN.1 Functional testing
	ATE_IND.2 Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.4 Methodical vulnerability analysis

Table 7 EAL5

11.8 Evaluation assurance level 6 (EAL6) - semiformally verified design and tested

Objectives

261 EAL6 permits developers to gain high assurance from application of security engineering techniques to a rigorous development environment in order to produce a premium TOE for protecting high value assets against significant risks.

262 EAL6 is therefore applicable to the development of security TOEs for application in high risk situations where the value of the protected assets justifies the additional costs.

Assurance components

263 **EAL6** provides assurance by a full security target and an analysis of the SFRs in that ST, using a functional and complete interface specification, guidance documentation, the design of the TOE, and the implementation to understand the security behaviour. Assurance is additionally gained through a formal model of select TOE security policies and a semiformal presentation of the functional specification and TOE design. A modular **and layered** TSF design is also required.

264 The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification, TOE design, selective independent confirmation of the developer test results, and an independent vulnerability analysis demonstrating resistance to penetration attackers with a **high** attack potential.

265 **EAL6** also provides assurance through the use of a **structured** development **process, development** environment controls, and comprehensive TOE configuration management including **complete** automation, and evidence of secure delivery procedures.

266 This EAL represents a meaningful increase in assurance from **EAL5** by requiring **more comprehensive analysis, a structured representation of** the implementation, more **architectural structure (e.g. layering), more comprehensive independent vulnerability analysis,** and improved **configuration management and** development **environment controls.**

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.5 Complete semi-formal functional specification with additional error information
	ADV_IMP.2 Implementation of the TSF
	ADV_INT.3 Reduction of complexity through layering
	ADV_SPM.1 Formal TOE security policy model
	ADV_TDS.5 Complete semiformal modular design
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.5 Advanced support
	ALC_CMS.5 Development tools CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.2 Sufficiency of security measures
	ALC_LCD.2 Standardised life-cycle model
	ALC_TAT.3 Compliance with implementation standards - all parts
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.3 Rigorous analysis of coverage
	ATE_DPT.2 Testing: modular design
	ATE_FUN.2 Ordered functional testing
	ATE_IND.2 Independent testing - sample
AVA: Vulnerability assessment	AVA_VAN.5 Advanced methodical vulnerability analysis

Table 8 EAL6

11.9 Evaluation assurance level 7 (EAL7) - formally verified design and tested

Objectives

267 EAL7 is applicable to the development of security TOEs for application in extremely high risk situations and/or where the high value of the assets justifies the higher costs. Practical application of EAL7 is currently limited to TOEs with tightly focused security functionality that is amenable to extensive formal analysis.

Assurance components

268 **EAL7** provides assurance by a full security target and an analysis of the SFRs in that ST, using a functional and complete interface specification, guidance documentation, the design of the TOE, and **a structured presentation of** the implementation to understand the security behaviour. Assurance is additionally gained through a formal model of select TOE security policies and a semiformal presentation of the functional specification and TOE design. A modular, **layered** and **simple** TSF design is also required.

269 The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification, TOE design **and implementation representation, complete** independent confirmation of the developer test results, and an independent vulnerability analysis demonstrating resistance to penetration attackers with a high attack potential.

270 **EAL7** also provides assurance through the use of a structured development process, development environment controls, and comprehensive TOE configuration management including complete automation, and evidence of secure delivery procedures.

271 This EAL represents a meaningful increase in assurance from **EAL6** by requiring more comprehensive analysis **using formal representations** and **formal correspondence,** and **comprehensive testing.**

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Architectural Design with domain separation and non-bypassability
	ADV_FSP.6 Complete semi-formal functional specification with additional formal specification
	ADV_IMP.2 Implementation of the TSF
	ADV_INT.4 Minimisation of complexity
	ADV_SPM.1 Formal TOE security policy model
	ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.5 Advanced support
	ALC_CMS.5 Development tools CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.2 Sufficiency of security measures
	ALC_LCD.3 Measurable life-cycle model
	ALC_TAT.3 Compliance with implementation standards - all parts
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.3 Rigorous analysis of coverage
	ATE_DPT.3 Testing: implementation representation
	ATE_FUN.2 Ordered functional testing
	ATE_IND.3 Independent testing - complete
AVA: Vulnerability assessment	AVA_VAN.5 Advanced methodical vulnerability analysis

Table 9 EAL7

12 Composition assurance packages

272 The Composed Assurance Packages (CAPs) provide an increasing scale that balances the level of assurance obtained with the cost and feasibility of acquiring that degree of assurance for composed TOEs.

273 It is important to note that there are only a small number of families and components from CC Part 3 included in the CAPs. This is due to their nature of building upon evaluation results of previously evaluated entities (base components and dependent components), and is not to say that these do not provide meaningful and desirable assurances.

12.1 Composition assurance package (CAP) overview

274 Table 10 represents a summary of the CAPs. The columns represent a hierarchically ordered set of CAPs, while the rows represent assurance families. Each number in the resulting matrix identifies a specific assurance component where applicable.

275 As outlined in the next section, three hierarchically ordered composition assurance packages are defined in the CC for the rating of a composed TOE's assurance. They are hierarchically ordered inasmuch as each CAP represents more assurance than all lower CAPs. The increase in assurance from CAP to CAP is accomplished by substitution of a hierarchically higher assurance component from the same assurance family (i.e. increasing rigour, scope, and/or depth) and from the addition of assurance components from other assurance families (i.e. adding new requirements). These increases result in greater analysis of the composition to identify the impact on the evaluation results gained for the individual component TOEs.

276 These CAPs consist of an appropriate combination of assurance components as described in chapter 8 of this CC Part 3. More precisely, each CAP includes no more than one component of each assurance family and all assurance dependencies of every component are addressed.

277 The CAPs only consider resistance against an attacker with an attack potential up to extended-basic. This is due to the level of design information that can be provided through the ACO_DEV components, as apposed to the ADV approach, which subsequently affects the rigour of vulnerability analysis that can be performed.

Assurance class	Assurance Family	Assurance Components by Composition Assurance Level		
		CAP-A	CAP-B	CAP-C
Composition	ACO_COR	**1**	1	1
	ACO_DEV	**1**	**2**	**3**
	ACO_REL	**1**	**2**	**3**
	ACO_TBT	**1**	1	1
	ACO_VUL	**1**	**2**	**3**
Life-cycle support	ALC_CMC	**1**	1	1
	ALC_CMS	**2**	2	2
	ALC_DEL			
	ALC_DVS			
	ALC_FLR			
	ALC_LCD			
	ALC_TAT			
Security Target evaluation	ASE_CCL	**1**	1	1
	ASE_ECD	**1**	1	1
	ASE_INT	**1**	1	1
	ASE_OBJ	**1**	**2**	2
	ASE_REQ	**1**	**2**	2
	ASE_SPD		**1**	1
	ASE_TSS	**1**	1	1

Table 10 Composition assurance level summary

12.2 Composition assurance package details

278 The following sections provide definitions of the CAPs, highlighting differences between the specific requirements and the prose characterisations of those requirements using bold type.

12.3 Composition assurance level A (CAP-A) - Structurally composed

Objectives

279 CAP-A is applicable when a composed TOE is integrated and confidence in the correct security operation of the resulting composite is required. This requires the cooperation of the developer of the dependent TOE in terms of delivery of design information and test results from the dependent TOE evaluation, without requiring the involvement of the base TOE developer.

280 CAP-A is therefore applicable in those circumstances where developers or users require a low to moderate level of independently assured security in the absence of ready availability of the complete development record.

Assurance components

281 **CAP-A provides assurance by analysis of a security target for the composed TOE. The SFRs in the composed TOE ST are analysed using the outputs from the evaluations of the component TOEs (e.g. ST, guidance documentation) and a specification for the interfaces between the component TOEs in the composed TOE to understand the security behaviour.**

282 **The analysis is supported by independent testing of the interfaces of the base TOE that are relied upon by the dependent TOE, as described in the reliance information, evidence of developer testing based on the reliance information, development information and composition information, and selective independent confirmation of the developer test results. The analysis is also supported by a vulnerability analysis of the composed TOE by the evaluator.**

283 **CAP-A also provides assurance through unique identification of the composed TOE (i.e. IT TOE and guidance documentation), and evidence of secure delivery procedures.**

Assurance Class	Assurance components
ACO: Composition	ACO_COR.1 Composition rationale
	ACO_DEV.1 Functional Description
	ACO_REL.1 Basic reliance information
	ACO_TBT.1 Interface testing
	ACO_VUL.1 Composition vulnerability review
ALC: Life-cycle support	ALC_CMC.1 Labelling of the TOE
	ALC_CMS.2 Parts of the TOE CM coverage
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.1 Security objectives for the operational environment
	ASE_REQ.1 Stated security requirements
	ASE_TSS.1 TOE summary specification

Table 11 CAP-A

12.4 Composition assurance level B (CAP-B) - Methodically composed

Objectives

284 CAP-B permits a conscientious developer to gain maximum assurance from understanding, at a subsystem level, the affects of interactions between component TOEs integrated in the composed TOE, whilst minimising the demand of involvement of the base TOE developer.

285 CAP-B is applicable in those circumstances where developers or users require a moderate level of independently assured security, and require a thorough investigation of the composed TOE and its development without substantial re-engineering.

Assurance components

286 **CAP-B** provides assurance by analysis of a **full** security target for the composed TOE. The SFRs in the composed TOE ST are analysed using the outputs from the evaluations of the component TOEs (e.g. ST, guidance documentation), a specification for the interfaces between the component TOEs **and the high-level TOE design contained** in the composed **development information** to understand the security behaviour.

287 The analysis is supported by independent testing of the interfaces of the base TOE that are relied upon by the dependent TOE, as described in the reliance information **(now also including high-level TOE design),** evidence of developer testing based on the reliance information, development information and composition information, and selective independent confirmation of the developer test results. The analysis is also supported by a vulnerability analysis of the composed TOE by the evaluator **demonstrating resistance to attackers with basic attack potential.**

288 **CAP-B** also provides assurance through **confirming the use** of **development environment controls,** TOE **configuration management,** and evidence of secure delivery procedures **from the component TOE evaluations.**

289 **This CAP represents a meaningful increase in assurance from CAP-A by requiring more complete testing coverage of the security functions and mechanisms and/or procedures that provide some confidence that the TOE will not be tampered with during development.**

Assurance Class	Assurance components
ACO: Composition	ACO_COR.1 Composition rationale
	ACO_DEV.2 Basic evidence of design
	ACO_REL.2 Reliance information
	ACO_TBT.1 Interface testing
	ACO_VUL.2 Composition vulnerability analysis
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.1 Labelling of the TOE
	ALC_CMS.2 Parts of the TOE CM coverage
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification

Table 12 CAP-B

12.5 Composition assurance level C (CAP-C) - Methodically composed, tested and reviewed

Objectives

290 CAP-C permits a developer to gain maximum assurance from positive analysis of the interactions between the components of the composed TOE, which, though rigorous, do not require full access to all evaluation evidence of the base TOE.

291 CAP-C is therefore applicable in those circumstances where developers or users require a moderate to high level of independently assured security in conventional commodity composed TOEs and are prepared to incur additional security-specific engineering costs.

Assurance components

292 **CAP-C** provides assurance by analysis of a full security target for the composed TOE. The SFRs in the composed TOE ST are analysed using the outputs from the evaluations of the component TOEs (e.g. ST, guidance documentation), a specification for the interfaces between the component TOEs and the TOE design contained in the composed development information to understand the security behaviour.

293 The analysis is supported by independent testing of the interfaces of the base TOE that are relied upon by the dependent TOE, as described in the reliance information (now including TOE design), evidence of developer testing based on the reliance information, development information and composition information, and selective independent confirmation of the developer test results. The analysis is also supported by a vulnerability analysis of the composed TOE by the evaluator demonstrating resistance to attackers with **extended-basic** attack potential.

294 **CAP-C** also provides assurance through confirming the use of development environment controls **and additional** TOE configuration management **including automation,** and evidence **that** secure delivery procedures **have not been compromised during composition.**

295 This CAP represents a meaningful increase in assurance from **CAP-B** by requiring more **design description.**

Assurance Class	Assurance components
ACO: Composition	ACO_COR.1 Composition rationale
	ACO_DEV.3 Detailed evidence of design
	ACO_REL.3 Detailed reliance information
	ACO_TBT.1 Interface testing
	ACO_VUL.3 Extended-basic Composition vulnerability analysis
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.1 Labelling of the TOE
	ALC_CMS.2 Parts of the TOE CM coverage
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification

Table 13 CAP-C

13 Assurance classes, families, and components

296 The next seven chapters provide the detailed requirements, of each of the assurance components, grouped by class and family.

14 Class ASE: Security Target evaluation

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

298 Figure 8 shows the families within this class, and the hierarchy of components within the families.

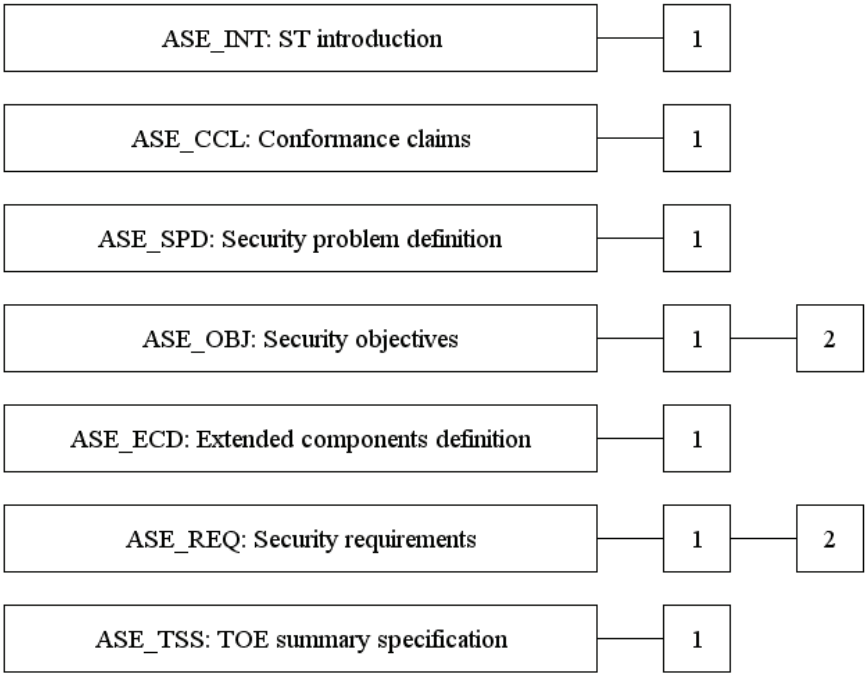

Figure 8 - ASE: Security Target evaluation class decomposition

14.1 ST introduction (ASE_INT)

Objectives

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

300 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.

ASE_INT.1 ST introduction

Dependencies: No dependencies.

Developer action elements:

ASE_INT.1.1D **The developer shall provide an ST introduction.**

Content and presentation elements:

ASE_INT.1.1C **The ST introduction shall contain an ST reference, a TOE reference, a TOE overview and a TOE description.**

ASE_INT.1.2C **The ST reference shall uniquely identify the ST.**

ASE_INT.1.3C **The TOE reference shall identify the TOE.**

ASE_INT.1.4C **The TOE overview shall summarise the usage and major security features of the TOE.**

ASE_INT.1.5C **The TOE overview shall identify the TOE type.**

ASE_INT.1.6C **The TOE overview shall identify any non-TOE hardware/software/firmware required by the TOE.**

ASE_INT.1.7C **The TOE description shall describe the physical scope and boundaries of the TOE.**

ASE_INT.1.8C **The TOE description shall describe the logical scope and boundaries of the TOE.**

Evaluator action elements:

ASE_INT.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ASE_INT.1.2E **The evaluator *shall confirm* that the TOE reference, the TOE overview, and the TOE description are consistent with each other.**

14.2 Conformance claims (ASE_CCL)

Objectives

301 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.

ASE_CCL.1 Conformance claims

Dependencies: ASE_INT.1 ST introduction
ASE_ECD.1 Extended components definition
ASE_REQ.1 Stated security requirements

Developer action elements:

ASE_CCL.1.1D **The developer shall provide a conformance claim.**

ASE_CCL.1.2D **The developer shall provide a conformance claim rationale.**

Content and presentation elements:

ASE_CCL.1.1C **The conformance claim shall contain a CC conformance claim that identifies the version of the CC to which the ST and the TOE claim conformance.**

ASE_CCL.1.2C **The CC conformance claim shall describe the conformance of the ST to CC Part 2 as either CC Part 2 conformant or CC Part 2 extended.**

ASE_CCL.1.3C **The CC conformance claim shall describe the conformance of the ST to CC Part 3 as either CC Part 3 conformant or CC Part 3 extended.**

ASE_CCL.1.4C **The CC conformance claim shall be consistent with the extended components definition.**

ASE_CCL.1.5C **The conformance claim shall identify all PPs and security requirement packages to which the ST claims conformance.**

ASE_CCL.1.6C **The conformance claim shall describe any conformance of the ST to a package as either package-conformant or package-augmented.**

ASE_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.**

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 PPs for which conformance is being claimed.**

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 PPs for which conformance is being claimed.**

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 PPs for which conformance is being claimed.**

Evaluator action elements:

ASE_CCL.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

14.3 Security problem definition (ASE_SPD)

Objectives

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

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

ASE_SPD.1 Security problem definition

Dependencies: No dependencies.

Developer action elements:

ASE_SPD.1.1D **The developer shall provide a security problem definition.**

Content and presentation elements:

ASE_SPD.1.1C **The security problem definition shall describe the threats.**

ASE_SPD.1.2C **All threats shall be described in terms of a threat agent, an asset, and an adverse action.**

ASE_SPD.1.3C **The security problem definition shall describe the OSPs.**

ASE_SPD.1.4C **The security problem definition shall describe the assumptions about the operational environment of the TOE.**

Evaluator action elements:

ASE_SPD.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

14.4 Security objectives (ASE_OBJ)

Objectives

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

305 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, its development environment, and its operational environment is clearly defined, and that the security objectives are internally consistent.

Component levelling

306 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 and security objectives for the development environment.

ASE_OBJ.1 Security objectives for the operational environment

Dependencies: No dependencies.

Developer action elements:

ASE_OBJ.1.1D **The developer shall provide a statement of security objectives.**

Content and presentation elements:

ASE_OBJ.1.1C **The statement of security objectives shall describe the security objectives for the operational environment.**

Evaluator action elements:

ASE_OBJ.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ASE_OBJ.2 Security objectives

Dependencies: ASE_SPD.1 Security problem definition

Developer action elements:

ASE_OBJ.2.1D The developer shall provide a statement of security objectives.

ASE_OBJ.2.2D **The developer shall provide a security objectives rationale.**

Content and presentation elements:

ASE_OBJ.2.1C **The statement of security objectives shall describe the security objectives for the TOE, the security objectives for the development environment and the security objectives for the operational environment.**

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.**

ASE_OBJ.2.3C **The security objectives rationale shall trace each security objective for the development environment back to threats countered by that security objective and OSPs enforced by that security objective.**

ASE_OBJ.2.4C **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.**

ASE_OBJ.2.5C The security objectives **rationale** shall **demonstrate that** the security objectives **counter all threats.**

ASE_OBJ.2.6C **The security objectives rationale shall demonstrate that the security objectives enforce all OSPs.**

ASE_OBJ.2.7C **The security objectives rationale shall demonstrate that the security objectives for the operational environment uphold all assumptions.**

Evaluator action elements:

ASE_OBJ.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

14.5 Extended components definition (ASE_ECD)

Objectives

307 Extended security requirements are requirements that are not based on components from CC Part 2 or CC Part 3, but are based on extended components: components defined by the ST author.

308 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 could not have been clearly expressed using existing CC Part 2 or CC Part 3 components.

ASE_ECD.1 Extended components definition

Dependencies: No dependencies.

Developer action elements:

ASE_ECD.1.1D The developer shall provide a statement of security requirements.

ASE_ECD.1.2D The developer shall provide an extended components definition.

Content and presentation elements:

ASE_ECD.1.1C The statement of security requirements shall identify all extended security requirements.

ASE_ECD.1.2C The extended components definition shall define an extended component for each extended security requirement.

ASE_ECD.1.3C The extended components definition shall describe how each extended component is related to the existing CC components, families, and classes.

ASE_ECD.1.4C The extended components definition shall use the existing CC components, families, classes, and methodology as a model for presentation.

ASE_ECD.1.5C The extended components shall consist of measurable and objective elements such that compliance or noncompliance to these elements can be demonstrated.

Evaluator action elements:

ASE_ECD.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

ASE_ECD.1.2E The evaluator *shall confirm* that no extended component can be clearly expressed using existing components.

14.6 Security requirements (ASE_REQ)

Objectives

309 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.

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

Component levelling

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

ASE_REQ.1 Stated security requirements

Dependencies: ASE_ECD.1 Extended components definition

Developer action elements:

ASE_REQ.1.1D The developer shall provide a security requirements rationale.

Content and presentation elements:

ASE_REQ.1.1C The statement of security requirements shall describe the SFRs and the SARs.

ASE_REQ.1.2C The statement of security requirements shall identify all operations on the security requirements.

ASE_REQ.1.3C All assignment and selection operations shall be completed.

ASE_REQ.1.4C All operations shall be performed correctly.

ASE_REQ.1.5C Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

Evaluator action elements:

ASE_REQ.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

ASE_REQ.1.2E The evaluator *shall confirm* that the statement of security requirements is internally consistent.

ASE_REQ.2 Derived security requirements

Dependencies: ASE_OBJ.1 Security objectives for the operational environment
ASE_ECD.1 Extended components definition

Developer action elements:

ASE_REQ.2.1D The developer shall provide a security requirements rationale.

Content and presentation elements:

ASE_REQ.2.1C The statement of security requirements shall describe the SFRs and the SARs.

ASE_REQ.2.2C The statement of security requirements shall identify all operations on the security requirements.

ASE_REQ.2.3C All assignment and selection operations shall be completed.

ASE_REQ.2.4C All operations shall be performed correctly.

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.

ASE_REQ.2.6C **The security requirements rationale shall trace each SFR back to the security objectives for the TOE.**

ASE_REQ.2.7C **The security requirements rationale shall demonstrate that the SFRs meet all security objectives for the TOE.**

ASE_REQ.2.8C **The security requirements rationale shall trace each SAR back to the security objectives for the development environment.**

ASE_REQ.2.9C **The security requirements rationale shall demonstrate that the SARs meet all security objectives for the development environment.**

Evaluator action elements:

ASE_REQ.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ASE_REQ.2.2E The evaluator ***shall confirm*** that the statement of security requirements is internally consistent.

14.7 TOE summary specification (ASE_TSS)

Objectives

312 The TOE summary specification enables evaluators and potential consumers to understand how the TOE meets its SFRs.

313 Evaluation of the TOE summary specification is necessary to determine whether all SFRs have been adequately addressed, and whether the TOE summary specification is consistent with other narrative descriptions of the TOE.

ASE_TSS.1 TOE summary specification

Dependencies: ASE_INT.1 ST introduction
ASE_REQ.1 Stated security requirements

Developer action elements:

ASE_TSS.1.1D **The developer shall provide a TOE summary specification.**

Content and presentation elements:

ASE_TSS.1.1C **The TOE summary specification shall describe how the TOE meets each SFR.**

Evaluator action elements:

ASE_TSS.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ASE_TSS.1.2E **The evaluator *shall confirm* that the TOE summary specification is consistent with the TOE overview and the TOE description.**

15 Class ADV: Development

314 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.

315 The Development class encompasses six 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)
- 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 TSP model and for correspondence mappings between the TSP, the TSP 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)

316 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 Architectural design (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 components in the families are designed so that more assurance can be gained as the components hierarchically increase.

317 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.

318 Figure 9 indicates the relationships between the various TSF representations and the security objectives for the TOE and SFRs that they are intended to address. 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 the TOE summary specification.

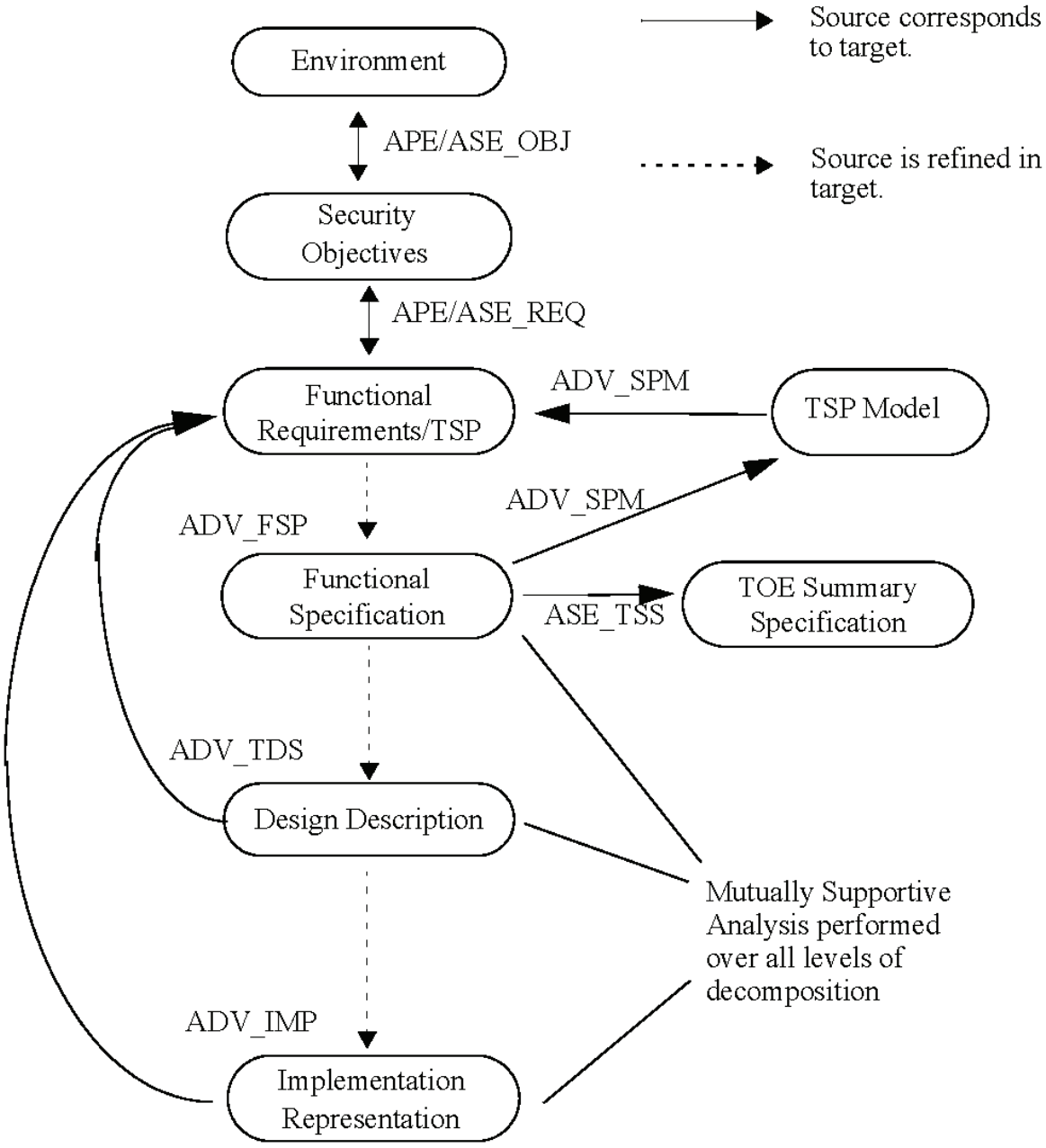

Figure 9 - Relationships between TOE representations and requirements

319 The requirements for all other correspondence shown in Figure 9 are defined in the ADV class. The ADV_SPM family defines the requirements for formally modelling selected security functionality of the TOE, and providing correspondence between the functional specification and the TSP 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 TSP to the functional requirements, the combination of which helps to ensure that the TSP has been addressed. All decompositions must completely and accurately reflect all other decompositions (i.e., be mutually supportive). 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.

320 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.

321 The TOE security policy (TSP) is the set of rules that regulate how resources are managed, protected and distributed within a TOE, expressed by the TOE security functional requirements. The developer is not explicitly required to provide a TSP, as the TSP is expressed by the TOE security functional requirements.

322 The TOE security functionality (TSF) consists of all parts of the TOE that have to be relied upon for enforcement of the TSP. The TSF includes both functions that directly enforce the TSP, and also those functions that, while not directly enforcing the TSP, contribute to the enforcement of the TSP in a more indirect manner, including functions that have the capability to cause the TSP to be violated. This includes portions of the system that are invoked on TOE start-up that are responsible for putting the TSF in its initial secure state.

323 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.

324 One over-riding notion is that, as more information becomes available, greater assurance can be obtained that the security functions 1) are correctly implemented; 2) cannot be compromised; 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).

325 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 system are more *interesting* than others) applies to other families, the criteria are expressed differently in order to obtain the assurance required.

326 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 system 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 components that are termed *SFR-enforcing*. Such components play a direct role in implementing any SFR on the TOE. Such SFRs are not limited to the access control requirements, but 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*.

327 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.

328 The architecture family (Architectural design (ADV_ARC)) provides for requirements and analysis of the TOE based on properties of TSF trusted initialisation, self-protection, and non-bypassability. Such requirements are

at least as important, if not more important, than SFRs because if these requirements are not met, 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.

329 The difference in analysis of the implementation of SFRs (SFR-enforcing functionality) and the implementation of somewhat fundamental security proprieties 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 (Architectural design (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.

330 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. It is acceptable for the module-level documentation to co-exist with the implementation representation, as long as there exists a procedure to extract that documentation from the implementation representation. The concept is that the module-level documentation should stand on its own, and not depend on the implementation representation for additional detail.

331 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.

332 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. Dutch, English, French, German). 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.

333 The difference between semiformal and informal documents is only a matter of formatting/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*.

334 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.

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

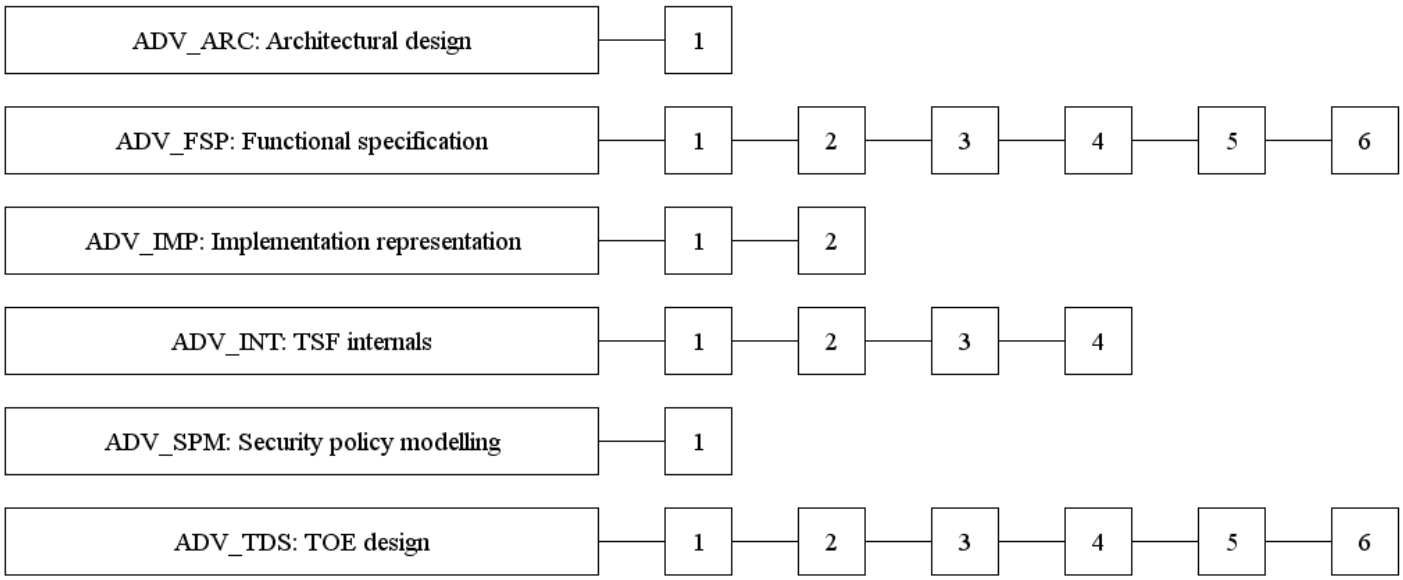

Figure 10 - ADV: Development class decomposition

15.1 Architectural design (ADV_ARC)

Objectives

336 The objective of this family is for the developer to provide an architectural design of the architectural properties of the TSF at the level of detail of the most detailed evidence presented for the SFR-enforcing elements of the TOE. 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.

Component levelling

337 This family contains only one component.

Application notes

338 The notions of self-protection, domain isolation, and non-bypassability are distinct from security functionality expressed in Part 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.

339 The overall approach used in this family is for the developer to provide a TSF that meets the above-mentioned properties, and provide evidence (in the form of documentation) that can be analysed to show that the properties are indeed met. 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.

340 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 architectural design 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 architectural design document.

341 This family consists of requirements for an architectural design that describes the self-protection, domain isolation, non-bypassability principles, including a description of how these principles are supported by the parts of the TOE that are used for TSF initialization.

342 Additional information on the architectural notions of self-protection, domain isolation, and non-bypassability can be found in (A.1, ADV_ARC: Supplementary material on architectural characteristics).

ADV_ARC.1 Architectural Design with domain separation and non-bypassability

Dependencies: ADV_FSP.1 Basic functional specification
ADV_TDS.1 Basic design

Developer action elements:

ADV_ARC.1.1D The developer shall provide the architectural design of the TSF.

Content and presentation elements:

ADV_ARC.1.1C The descriptive information contained in the architectural design shall be at a level of detail commensurate with the description of the SFR-enforcing abstractions described in the TOE design document.

ADV_ARC.1.2C The architectural design shall describe the security domains maintained by the TSF consistently with the SFRs.

ADV_ARC.1.3C The architectural design shall describe how the TSF initialisation process is secure.

ADV_ARC.1.4C The architectural design shall demonstrate that the TSF protects itself from interference and tampering.

ADV_ARC.1.5C The architectural design shall demonstrate that the TSF prevents bypass of the SFR-enforcing functionality.

Evaluator action elements:

ADV_ARC.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

15.2 Functional specification (ADV_FSP)

Objectives

343 This family levies requirements upon the functional specification, which describes the TSF interfaces (TSFI). The TSFI consist of all means for users to invoke a service from the TSF (by supplying data that is processed by the TSF) and the corresponding responses to those service invocations. It does *not* describe how the TSF processes those service requests, nor does it describe the communication when the TSF invokes services from its IT environment. These are addressed by the TOE design (ADV_TDS) and Reliance of dependent component (ACO_REL) families, respectively.

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

- ADV_ARC, where the description of the TSFI may be used to gain better understanding of how the TSF is protected against bypass and/or corruption;
- ATE, where the description of the TSFI is an important input for both developer and evaluator testing;
- AVA, where the description of the TSFI is used to search for vulnerabilities.

Component levelling

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

Application notes

346 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.

347 A common example of this is the case where the means of invoking given services is constrained to only those users with access to specific permissions, but whose associated guidance documentation instructs such users not to invoke the given services. It can be rightfully presumed that those services will not be invoked, and so those means are not part of the TSFI. For example, suppose a TOE has a command-line interface through which users with access to the “audit administrator” permissions might use to administer the audit functionality of the TOE, or they could use a GUI-based tool that essentially translates the GUI-based check boxes, text boxes, etc., into scripts that invoke the command-line utilities. However, suppose that these users were directed in their guidance to use only the GUI-based

tool in administering the TOE. Because the command-line interface is inaccessible to users without access to the "audit administrator" permissions, and users with access to the "audit administrator" permissions are instructed not to use it, the command-line interface is not considered to be part of the TSFI and does not have to be documented. Because users with access to the "audit administrator" privilege use the GUI tool, it is part of the TSFI and would have to be documented.

348 The remainder of these Application Notes discusses the process for determining the TSFI, some different types of TSFI along with an example, the detail that is used in describing the TSFI once they are determined, and a comparison of the components in this family.

15.2.1 Determining the TSFI

349 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 TSP enforcement 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 isolation, and non-bypassability (see Architectural design (ADV_ARC)).

350 Once the TSF has been defined, the TSFI are identified. The TSFI consist of all means for users to invoke a service from the TSF (by supplying data that is processed by the TSF) and the corresponding responses to those service invocations. These service invocations and responses are means of crossing the TSF boundary. While many of these are readily apparent, others might not be as obvious. The following discussions illustrate the application of the TSFI definition in different contexts. For additional examples based upon TOE types, see A.2, ADV_FSP: Examples of Identifying the TSFI

15.2.1.1 Electrical interfaces

351 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.

15.2.1.2 Network protocol stack

352 In a firewall appliance that filters traffic at multiple layers of the network protocol stack, each layer of the stack interprets the single stream of received

bits and recognises the fields of its own protocol header. Therefore, the TSF boundary exists at each layer of the stack.

353 The functional specification in this case 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.

354 Because an adversary could attempt communication at any of these layers, all of them are considered TSFI and would therefore have to be described. The description would likely reference published communication standards (telnet, FTP, TCP, etc.) that are used, noting which user-defined options are chosen.

15.2.1.3 Single network layer

355 In contrast to a firewall appliance, where an adversary could supply input to all communication layers, an application firewall performs filtering at only the application layer (such as FTP commands). In this case, the TSF boundary is between the application layer and the underlying layer. Therefore, only the application layer is exposed to potential adversaries (i.e. is the TSFI), so only application-layer behaviour would have to be described.

15.2.1.4 Hardware interfaces

356 Hardware interfaces exist as well. Functions provided by the BIOS of various devices may be visible through a “wrapper” interface, for example, the IOCTLs in a Unix operating system. If the TOE includes a hardware device (e.g., a network interface card), the bus interface signals, as well as the interface seen at the network port, must be considered TSFI. Switches that can change the behaviour of the hardware are also TSFI.

15.2.2 Describing the TSFI

357 Once the TSFI are determined in accordance with the previous discussions, the next issue is describing them. The following sections describe determining the security-relevance of a set of TSFI, and then the required details for the TSFI.

15.2.2.1 Security-Relevance of the Interfaces

358 Some interfaces (or operations available through the interfaces) making up the TSFI are more critical and require more analysis than other interfaces. If an operation 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 operations and results, some of which may be SFR-enforcing and some of which may not.

359 As described earlier in 15.2.1, interfaces to (or operations available through an interface relating to) operations that SFR-enforcing functionality depends on, but need only to function correctly in order for the security policies of the TOE to be preserved, are termed *SFR supporting* . Interfaces to operations on which SFR-enforcing functionality has no dependence are termed *SFR non-interfering*.

360 It should be noted that in order for an interface to be SFR supporting or SFR non-interfering it must have *no* SFR-enforcing operations or results. In contrast, an SFR-enforcing interface may have SFR-supporting operations (for example, the ability to set the system clock may be an SFR-enforcing operation of an interface, but if that same interface is used to display the system date that operation may only be 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.

361 As more information about the TSFI 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.

362 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 focus for 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 artifacts, such as detailed interface and design documentation) to provide only the necessary evidence without undue cost.

15.2.2.2 Detail about the Interfaces

363 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.

364 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.)

365 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 was a Unix command shell, *ls*, *mv* and *cp* would be interactions for that interface. For each interaction it needs to be described 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).

366 *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.

367 A *parameter description* tells what the parameter is in some meaningful way. For instance, the interface *foo(i)* could be described as having “parameter i which is an integer”; this is not an acceptable parameter description. A description such as “parameter i is an integer that indicates the number of users currently logged in to the system” is required.

368 *Operations* of an interface describe what the interface does, and can be categorised as *regular* operations, and *SFR-related* operations. Regular operations are descriptions of what the interface does. The amount of information provided for this description is dependant on the complexity of the interface. The SFR-related operations that need to be described in a functional specification are those that are visible at any external interface (for instance, audit activity caused by the invocation of an interface (assuming audit requirements are included in the ST) should be described, even though the result of that operation is generally not visible through the invoked interface). Depending on the parameters of an interface, there may be many different operations able to be invoked through the interface (for instance, an API might have the first parameter be a “subcommand”, and the following parameters be specific to that subcommand. The IOCTL API in some Unix systems is an example of such an interface).

369 The *error messages* generated by the TSF can be the direct result of invoking a TSFI (e.g. an API call that returns an error); an error that is easily tied to a TSFI (e.g. setting a parameter in a configuration that is error-checked when read, returning an immediate notification); or an error that is not easily tied to a TSFI (e.g. setting a parameter that, in combination with certain system

states, generates an error condition that occurs at a later time. An example might be resource exhaustion of a system resource (e.g., disk space) due to setting a parameter to too low of a value).

370 Errors can take many forms, depending on the interface being described. For an API, the interface itself may return an error code; set a global error condition, or set a certain parameter with an error code. For a configuration file, an incorrectly configured parameter may cause an error message to be written to a log file. For a hardware PCI card, an error condition may raise a signal on the bus, or trigger an exception condition to the CPU.

371 Errors (and the associated error messages) come about through the invocation of an interface. The processing that occurs in response to the interface invocation may encounter error conditions, which trigger (through an implementation-specific mechanism) an error message to be generated. In some instances this may be a return value from the interface itself; in other instances a global value may be set and checked after the invocation of an interface. It is likely that a TOE will have a number of low-level error messages that may result from fundamental resource conditions, such as "disk full" or "resource locked". While these error messages may map to a large number of TSFI, at higher levels of assurance they could be used to detect instances where detail from an interface description has been omitted. For instance, a TSFI that produces a "disk full" message, but has no obvious description of why that TSFI should cause an access to the disk in its description of operations, might cause the evaluator to examine other evidence (Architectural design (ADV_ARC), TOE design (ADV_TDS)) related that TSFI to determine if the description is complete and accurate.

372 It is also possible that the implementation of the TSF contains "guard" code with associated error messages; for instance, a check for a condition that should not logically occur, but will generate an error message should the condition be detected. In an operational TOE, these error messages should never been seen at the interface.

15.2.3 Components of this Family

373 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.

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

375 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 TSFI. Additionally, for the SFR-enforcing TSFI the

developer has to describe the SFR-enforcing interactions and error messages resulting form those interactions.

376 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 interactions and error messages to show that they are not SFR-enforcing. Further, the developer must now document all of the error messages resulting from the invocation of SFR-enforcing TSFI.

377 At ADV_FSP.4 Complete functional specification, the developer needs to provide complete documentation of all TSFI.

378 At ADV_FSP.5 Complete semi-formal functional specification with additional error information, the developer needs to provide complete documentation of all TSFI using a semi-formal style.

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

380 The ADV_FSP.*.2E elements within this family provide a direct correspondence between the TOE security functional requirements and the functional specification. In the cases where the ST contains such functional requirements as Residual information protection (FPT_RIP), whose functionality may not manifest itself at the TSFI, it is expected that the evaluator do not considered this a deficiency in the functional specification.

ADV_FSP.1 Basic functional specification

Dependencies: No dependencies.

Developer action elements:

ADV_FSP.1.1D **The developer shall provide a functional specification.**

Content and presentation elements:

ADV_FSP.1.1C **The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.**

ADV_FSP.1.2C **The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.**

ADV_FSP.1.3C **The functional specification shall provide rationale for the implicit categorisation of interfaces as SFR-non-interfering.**

Evaluator action elements:

ADV_FSP.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ADV_FSP.1.2E **The evaluator *shall determine* that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.**

ADV_FSP.2 Security-enforcing functional specification

Dependencies: ADV_TDS.1 Basic design

Developer action elements:

ADV_FSP.2.1D The developer shall provide a functional specification.

Content and presentation elements:

ADV_FSP.2.1C **The functional specification shall completely represent the TSF.**

ADV_FSP.2.2C The functional specification shall describe the purpose and method of use for **all** TSFI.

ADV_FSP.2.3C The functional specification shall identify **and describe** all parameters associated with each TSFI.

ADV_FSP.2.4C **For SFR-enforcing TSFIs, the** functional specification shall **describe the SFR-enforcing operations** associated with **the** TSFI.

ADV_FSP.2.5C **For SFR-enforcing TSFIs, the functional specification shall describe error messages resulting from processing associated with the SFR-enforcing operations.**

Evaluator action elements:

ADV_FSP.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.2.2E The evaluator ***shall determine*** that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.

ADV_FSP.3 Functional specification with complete summary

Dependencies: ADV_TDS.1 Basic design

Developer action elements:

ADV_FSP.3.1D The developer shall provide a functional specification.

Content and presentation elements:

ADV_FSP.3.1C The functional specification shall completely represent the TSF.

ADV_FSP.3.2C The functional specification shall describe the purpose and method of use for all TSFI.

ADV_FSP.3.3C The functional specification shall identify and describe all parameters associated with each TSFI.

ADV_FSP.3.4C For SFR-enforcing TSFIs, the functional specification shall describe the SFR-enforcing operations associated with the TSFI.

ADV_FSP.3.5C For SFR-enforcing TSFIs, the functional specification shall describe error messages resulting from **invocation of** the **TSFI.**

ADV_FSP.3.6C **The functional specification shall summarise the non-SFR-enforcing operations associated with each TSFI.**

Evaluator action elements:

ADV_FSP.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.3.2E The evaluator ***shall determine*** that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.

ADV_FSP.4 Complete functional specification

Dependencies: ADV_TDS.1 Basic design

Developer action elements:

ADV_FSP.4.1D The developer shall provide a functional specification.

Content and presentation elements:

ADV_FSP.4.1C The functional specification shall completely represent the TSF.

ADV_FSP.4.2C The functional specification shall describe the purpose and method of use for all TSFI.

ADV_FSP.4.3C The functional specification shall identify and describe all parameters associated with each TSFI.

ADV_FSP.4.4C The functional specification shall **describe all** operations associated with each TSFI.

ADV_FSP.4.5C **The functional specification shall describe all error messages that may result from an invocation of each TSFI.**

Evaluator action elements:

ADV_FSP.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.4.2E The evaluator ***shall determine*** that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.

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

Developer action elements:

ADV_FSP.5.1D The developer shall provide a functional specification.

Content and presentation elements:

ADV_FSP.5.1C The functional specification shall completely represent the TSF.

ADV_FSP.5.2C **The functional specification shall describe the TSFI using a semi-formal style.**

ADV_FSP.5.3C The functional specification shall describe the purpose and method of use for all TSFI.

ADV_FSP.5.4C The functional specification shall identify and describe all parameters associated with each TSFI.

ADV_FSP.5.5C The functional specification shall describe all operations associated with each TSFI.

ADV_FSP.5.6C The functional specification shall describe all error messages that may result from an invocation of each TSFI.

ADV_FSP.5.7C **The functional specification shall describe all error messages that do not result from an invocation of a TSFI.**

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.**

Evaluator action elements:

ADV_FSP.5.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.5.2E The evaluator ***shall determine*** that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.

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

Dependencies: ADV_TDS.1 Basic design

Developer action elements:

ADV_FSP.6.1D The developer shall provide a functional specification.

ADV_FSP.6.2D **The developer shall provide a formal presentation of the functional specification of the TSF.**

Content and presentation elements:

ADV_FSP.6.1C The functional specification shall completely represent the TSF.

ADV_FSP.6.2C The functional specification shall describe the TSFI using a semi-formal style.

ADV_FSP.6.3C The functional specification shall describe the purpose and method of use for all TSFI.

ADV_FSP.6.4C The functional specification shall identify and describe all parameters associated with each TSFI.

ADV_FSP.6.5C The functional specification shall describe all operations associated with each TSFI.

ADV_FSP.6.6C The functional specification shall describe all error messages that may result from an invocation of each TSFI.

ADV_FSP.6.7C The functional specification shall **describe all** error **messages** contained in the TSF implementation **that are** not **otherwise described in the functional specification.**

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.

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.**

Evaluator action elements:

ADV_FSP.6.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.6.2E The evaluator ***shall determine*** that the functional specification is an accurate and complete instantiation of the TOE security functional requirements.

15.3 Implementation representation (ADV_IMP)

Objectives

381 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 is analysable 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 chip specifications. At higher levels, the implementation representation is demonstrated to be transformable to the implementation that is used for testing.

Component levelling

382 The components in this family are levelled on the basis of the degree of confidence obtained that the implementation representation can be transformed to the implementation used for testing.

Application notes

383 Source code or hardware diagrams and/or chip specifications 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.

384 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 be provided so that the implementation representation can be accurately determined.

385 The implementation representation is manipulated by the developer in form that it 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 evaluator's analysis efforts.

386 Even given this most detailed description of the TSF, there is still a question of whether the actual implementation that is tested reflects what is contained in the implementation representation. Some amount of additional assurance can be obtained in this area by requiring that the implementation representation provided to the evaluator be transformed to an instantiation of the implementation. This instantiation can then be compared to the instantiation used for testing (for example, in testing associated with the ATE: Tests class) to gain confidence that the analysis using the implementation representation is valid.

387 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 by taking a version of the implementation representation that is used by the TOE developer 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.

ADV_IMP.1 Implementation representation of the TSF

Dependencies: ADV_TDS.3 Basic modular design
ALC_TAT.1 Well-defined development tools

Developer action elements:

ADV_IMP.1.1D The developer shall make available the implementation representation for the entire TSF.

ADV_IMP.1.2D **The developer shall provide a mapping between the TOE design description and the sample of the implementation representation.**

Content and presentation elements:

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.**

ADV_IMP.1.2C **The implementation representation shall be in the form used by the development personnel.**

ADV_IMP.1.3C **The mapping between the TOE design description and the sample of the implementation representation shall demonstrate their correspondence.**

Evaluator action elements:

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.**

ADV_IMP.2 Implementation of the TSF

Dependencies: ADV_TDS.3 Basic modular design
ALC_TAT.1 Well-defined development tools
ALC_CMC.5 Advanced support

Developer action elements:

ADV_IMP.2.1D The developer shall make available the implementation representation for the entire TSF.

ADV_IMP.2.2D The developer shall provide a mapping between the TOE design description and the implementation representation.

Content and presentation elements:

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.

ADV_IMP.2.2C The implementation representation shall be in the form used by the development personnel.

ADV_IMP.2.3C The mapping between the TOE design description and the implementation representation shall demonstrate their correspondence.

Evaluator action elements:

ADV_IMP.2.1E 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.

ADV_IMP.2.2E **The evaluator *shall determine* that the implementation representation, when transformed to the implementation using the developer-provided tools and instructions, is identical to the implementation used in testing activities.**

15.4 TSF internals (ADV_INT)

Objectives

388 This family addresses the internal structure of the TSF software, since software written without adhering to sound software engineering principles has historically been found to contain flaws and vulnerabilities. Hardware, on the other hand, by its very nature, has required a modular design and implementation approach in order to facilitate high data rates, testability, and reliability. Correcting a bug found in an integrated circuit or board is time-consuming and expensive compared to correcting a software problem; therefore substantially more care is taken in ensuring the hardware design and implementation is well understood and tested before the implementation is realised. Another reason why this family solely addresses software is that these requirements help to facilitate the developer's understanding of the software; this helps to ensure that any maintenance activities (e.g. fixes, added functionality) do not adversely impact the software's ability to enforce the SFRs. This relevance to maintenance does not have the same potential impact on hardware components.

Component levelling

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

390 At the next level, the requirements for modular decomposition are placed on the entire TSF. Layering and minimisation are then introduced in the next two higher components, as well as requirements for dealing with duplicate or unused code.

391 The ADV_INT.1 allows for fewer exceptions to satisfying the specified metrics than ADV_INT.2. This is because ADV_INT.1 does not require the metrics to be applied to the entire TSF and the intent is that the critical code corresponding to the assigned SFRs will satisfy the more stringent metrics. ADV_INT.3 contains the notion of limited exceptions with respect to modularity and is more liberal regarding limiting interactions between layers (i.e., focus is limiting interactions to adjacent layers, whereas ADV_INT.4 desires limiting interactions period).

Application notes

392 Requirements are presented for modular decomposition, layering, and minimisation of the amount of non-SFR-enforcing functionality within the TSF. These requirements, when applied to the internal structure of the TSF, should result in improvements that aid both the developer and the evaluator

in understanding the TSF, and also provides the basis for designing and evaluating test suites. Further, improving understandability of the TSF should assist the developer in simplifying its maintainability. The principal goal achieved by inclusion of the requirements from this class in a PP/ST is understandability of the TSF.

393 Modular design aids in achieving understandability by clarifying what dependencies a module has on other modules (*coupling*), by including in a module only tasks that are strongly related to each other (*cohesion*), and by illuminating the design of a module by using internal structuring and reduced complexity. 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.

394 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.

395 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.

396 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.

397 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 class Architectural design (ADV_ARC).)

398 Several of the elements within the components for this family refer to the TSF internals description. The TSF internals description is at a similar level of abstraction as the ADV_TDS.3 Basic modular design component (modular design description), in that it is concerned with the modules of the

TSF. Whereas the modular design description describes the design of the modules of the TSF, the purpose of the TSF internals description is to provide evidence of modular decomposition, layering, and minimisation of complexity of the TSF, as applicable. Both the modular design description (ADV_TDS.3 Basic modular design and higher if applicable) and the implementation representation are required to be in compliance with the TSF internals description, to provide assurance that these TSF representations possess the required modular decomposition, layering, and minimisation of complexity.

399 The modules identified in the TSF internals description are the same as the modules identified in the modular design description.

ADV_INT.1 Partial modular decomposition

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

Objectives

400 The objective of this component is to provide a means for requiring portions of the TSF to satisfy more rigorous modularity metrics without requiring the entire TSF to adhere to these metrics. The intent is that the entire TSF has been designed and implemented with sound engineering principles, but allowing for less desirable forms of cohesion within modules and coupling between modules without the need for justification.

Application notes

401 This component requires the PP or ST author to fill in an assignment with the SFRs that are felt to be critical to the TOE and therefore their resulting design and implementation require stricter metrics for modularity.

402 The requirements in this component refer to the "Assigned SFR-enforcing" portions of the TSF. These are the portions of the TSF that implement the functionality that directly enforces the SFRs assigned in **ADV_INT.1.4D**. Further explanation on the use of this component and the Assignment can be found in A.3, ADV_INT.1: Subset Modularity.

Developer action elements:

ADV_INT.1.1D **The developer shall design and implement the TSF using modular decomposition.**

ADV_INT.1.2D **The developer shall use sound software engineering principles to achieve the modular decomposition of the TSF.**

ADV_INT.1.3D **The developer shall design the Assigned SFR-enforcing modules such that they exhibit good internal structure and are not overly complex.**

ADV_INT.1.4D **The developer shall design modules that implement the [assignment: *list of SFRs*] such that they exhibit only functional, sequential, communicational, or temporal cohesion, with limited exceptions.**

ADV_INT.1.5D **The developer shall design the Assigned SFR-enforcing modules such that they exhibit only call or common coupling, with limited exceptions.**

ADV_INT.1.6D **The developer shall implement the Assigned SFR-enforcing modules using coding standards that result in good internal structure that is not overly complex.**

ADV_INT.1.7D **The developer shall provide a TSF internals description.**

Content and presentation elements:

ADV_INT.1.1C **The TSF internals description shall identify the Assigned SFR-enforcing and non-Assigned SFR-enforcing modules.**

ADV_INT.1.2C **There shall be a one-to-one correspondence between the modules identified in TSF internals description and the modules identified in the modular design description.**

ADV_INT.1.3C **The TSF internals description shall provide a justification for the designation of non-Assigned SFR-enforcing modules that interact with the Assigned SFR-enforcing module(s).**

ADV_INT.1.4C **The TSF internals description shall describe the process used for modular decomposition.**

ADV_INT.1.5C **The TSF internals description shall describe how the TSF design is a reflection of the modular decomposition process.**

ADV_INT.1.6C **The TSF internals description shall provide a justification, on a per Assigned SFR-enforcing module basis, of any deviations from the coding standards.**

ADV_INT.1.7C **The TSF internals description shall include a coupling analysis that describes intermodule coupling for the Assigned SFR-enforcing modules.**

ADV_INT.1.8C **The TSF internals description shall include a cohesion analysis that describes the types of cohesion for the Assigned SFR-enforcing modules.**

ADV_INT.1.9C **The TSF internals description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by Assigned SFR-enforcing modules, other than those permitted.**

Evaluator action elements:

ADV_INT.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ADV_INT.1.2E **The evaluator *shall perform* a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of Assigned SFR-enforcing modules.**

ADV_INT.1.3E **The evaluator *shall perform* a complexity analysis for a subset of Assigned SFR-enforcing modules.**

ADV_INT.2 Full modular decomposition

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

Objectives

403 In this component, the objective is for the developer to demonstrate that modular decomposition has been used in the design of the entire TSF.

Developer action elements:

ADV_INT.2.1D The developer shall design and implement the TSF using modular decomposition.

ADV_INT.2.2D The developer shall use sound software engineering principles to achieve the modular decomposition of the TSF.

ADV_INT.2.3D The developer shall design the **TSF** modules such that they exhibit good internal structure and are not overly complex.

ADV_INT.2.4D The developer shall design **all TSF** modules such that they exhibit only functional, sequential, communicational, or temporal cohesion, with exceptions.

ADV_INT.2.5D The developer shall design **all TSF** modules such that they exhibit only call or common coupling, with exceptions.

ADV_INT.2.6D The developer shall implement **TSF** modules using coding standards that result in good internal structure that is not overly complex.

ADV_INT.2.7D The developer shall provide a TSF internals description.

Content and presentation elements:

ADV_INT.2.1C There shall be a one-to-one correspondence between the modules identified in TSF internals description and the modules identified in the modular design description.

ADV_INT.2.2C The TSF internals description shall describe the process used for modular decomposition.

ADV_INT.2.3C The TSF internals description shall describe how the TSF design is a reflection of the modular decomposition process.

ADV_INT.2.4C The TSF internals description shall provide a justification, on a **per-module** basis, of any deviations from the coding standards.

ADV_INT.2.5C The TSF internals description shall include a coupling analysis that describes intermodule coupling for **all TSF** modules.

ADV_INT.2.6C The TSF internals description shall include a cohesion analysis that describes the types of cohesion for **all TSF** modules.

ADV_INT.2.7C The TSF internals description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by modules **of the TSF,** other than those permitted.

Evaluator action elements:

ADV_INT.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_INT.2.2E The evaluator ***shall perform*** a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of SFR-enforcing, **SFR-supporting, and SFR non-interfering TSF** modules.

ADV_INT.2.3E The evaluator ***shall perform*** a complexity analysis for a subset of SFR-enforcing, **SFR-supporting, and SFR non-interfering TSF** modules.

ADV_INT.3 Reduction of complexity through layering

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

Objectives

404 The objective of this component is to introduce the notion of a layered software architecture to facilitate the understanding of the TSF. Having a layered architecture allows the developer and evaluator to analyse and understand the TSF in more digestible pieces. A layered architecture with well defined interactions between layers allows for analysis of individual layers and the dependencies that exist on other layers of the system. This is desirable, since attempting to understand the interactions and dependencies of all the modules in the TSF may be more difficult than conceptualising the TSF from a layered architecture that provides a context for collections of modules.

405 As this component is hierarchical to the modularity requirement, a layered design that satisfies this requirement will also satisfy the modularity requirements. While one might argue that a layered architecture could provide some benefit without requiring modularity it is felt that without

having a modular TSF, it would be difficult to ascertain with a sufficient degree of confidence that layers were communicating with one another in a well-defined manner (e.g., functions in one layer could be communicating with a function in another layer through a global variable that would not be documented as a communication occurring between the layers).

406 This component also eliminates the allowance of temporal cohesion with *limited* exceptions to facilitate a more desirable form of modularity. In addition, the developer must identify any redundant and unused code that may exist to add to a better understanding of the TSF architecture.

Application notes

407 This component requires the developer to identify all the interactions that take place between layers. These interactions include calls that are made directly to a module in another layer, as well as other communication that may take place between modules in different layers (e.g., setting a flag, writing to a global variable). In order to fully understand how a layer is designed and its role in the TSF it is critical that all the interactions be understood.

408 A well understood property of a properly layered architecture is that layers are only dependent on the layer directly below it in the layering hierarchy. Layers are typically defined by identifying the services a layer requires and those that a layer provides. Modules within a layer do not necessarily have to interact with one another, they may only interact with modules in a layer adjacently above - providing services, or with modules in a layer adjacently below - requesting services.

409 There may be instances where there is a need to have a module in a layer initiate a call to a layer above. This is an undesirable property, because this could lead to mutual dependencies or circular interactions. This type of interaction is not strictly prohibited, but requires the developer to provide a justification as to why this interaction is necessary.

410 Ideally, layers of a sound software architecture would lend themselves to independent testing; that is one could test the interfaces presented by a layer without requiring layers that are higher in the hierarchy (i.e., one could test the interfaces at the highest layer - the TSFI, strip off a layer and continue to test the interfaces presented at the next layer in the hierarchy, and so on). This would require that no dependencies or calls to higher layers in the hierarchy are present which may not be practical in all instances.

Developer action elements:

ADV_INT.3.1D The developer shall design and implement the TSF using modular decomposition.

ADV_INT.3.2D The developer shall use sound software engineering principles to achieve the modular decomposition of the TSF.

ADV_INT.3.3D The developer shall design the TSF modules such that they exhibit good internal structure and are not overly complex.

ADV_INT.3.4D **The developer shall design all TSF modules such that they exhibit only functional, sequential, or communicational cohesion, with limited exceptions.**

ADV_INT.3.5D The developer shall design all TSF modules such that they exhibit only call or common coupling, with **limited** exceptions.

ADV_INT.3.6D The developer shall implement **the** TSF modules using coding standards that result in good internal structure that is not overly complex.

ADV_INT.3.7D The developer shall design **and structure the** TSF **in a layered fashion** that **minimises interactions except between adjacent layers of the design.**

ADV_INT.3.8D **The developer shall design and structure the TSF such that interactions between layers are initiated from a higher layer in the hierarchy down to the next layer in the hierarchy with exceptions.**

ADV_INT.3.9D The developer shall provide a TSF internals description.

Content and presentation elements:

ADV_INT.3.1C There shall be a one-to-one correspondence between the modules identified in TSF internals description and the modules identified in the modular design description.

ADV_INT.3.2C The TSF internals description shall describe the process used for modular decomposition.

ADV_INT.3.3C The TSF internals description shall describe how the TSF design is a reflection of the modular decomposition process.

ADV_INT.3.4C The TSF internals description shall provide a justification, on a per-module basis, of any deviations from the coding standards.

ADV_INT.3.5C The TSF internals description shall include a coupling analysis that describes intermodule coupling for all TSF modules.

ADV_INT.3.6C The TSF internals description shall include a cohesion analysis that describes the types of cohesion for all TSF modules.

ADV_INT.3.7C The TSF internals description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by modules of the TSF, other than those permitted.

ADV_INT.3.8C **The TSF internals description shall describe the layering architecture and shall describe the services that each layer provides.**

ADV_INT.3.9C **The TSF internals description shall describe the methodology used to determine the layering architecture.**

ADV_INT.3.10C **The TSF internals description shall identify all modules associated with each layer of the TSF.**

ADV_INT.3.11C **The TSF internals description shall identify all interactions between layers of the TSF.**

ADV_INT.3.12C **The TSF internals description shall provide a justification of interactions that are initiated from a lower layer to a higher layer.**

ADV_INT.3.13C **The TSF internals description shall identify all modules of the TSF that contain unused or redundant code.**

Evaluator action elements:

ADV_INT.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_INT.3.2E The evaluator ***shall perform*** a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of SFR-enforcing, SFR-supporting, and SFR non-interfering TSF modules.

ADV_INT.3.3E The evaluator ***shall perform*** a complexity analysis for a subset of SFR-enforcing, SFR-supporting, and SFR non-interfering TSF modules.

ADV_INT.4 Minimisation of complexity

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

Objectives

411 This component is focused on minimising the complexity of the TSF's implementation. While the primary focus of this family is to require the developer to design and implement a TSF that is well understood by the development staff, this component also requires the design and implementation to be easily understood by an independent third-party in an efficient manner.

Application notes

412 While the previous component requires the identification of un-necessary code, this component requires that un-necessary code be eliminated from the implementation, be it unused or redundant code, or code that is un-necessary to implement the SFRs.

413 Additionally, this component requires that interactions between layers be minimised. ADV_INT.3 requires that interactions be minimised between adjacent layers, while this component requires that interactions between even adjacent layers be minimised. The intent is that an architecture with layers that provide the necessary functionality within the layer without relying on

other layers (even those below in the hierarchy) provides a more comprehensible design/implementation and more easily understood.

414 This component also introduces the notion of reducing/eliminating "mutual dependencies", which pertains to both modules in a single layer and 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.

Developer action elements:

ADV_INT.4.1D The developer shall design and implement the TSF using modular decomposition.

ADV_INT.4.2D The developer shall use sound software engineering principles to achieve the modular decomposition of the TSF.

ADV_INT.4.3D The developer shall design the TSF modules such that they exhibit good internal structure and are not overly complex.

ADV_INT.4.4D **The developer shall design all modules such that they exhibit only functional, sequential, communicational, or temporal cohesion, with limited exceptions.**

ADV_INT.4.5D The developer shall design all TSF modules such that they exhibit only call or common coupling, with limited exceptions.

ADV_INT.4.6D The developer shall implement the TSF modules using coding standards that result in good internal structure that is not overly complex.

ADV_INT.4.7D The developer shall design and structure the TSF in a layered fashion that minimises interactions between **the** layers of the design.

ADV_INT.4.8D The developer shall design and structure the TSF such that interactions between layers are initiated from a higher layer in the hierarchy down to the next layer in the hierarchy with exceptions.

ADV_INT.4.9D The developer shall design and structure the **modules of the** TSF **such** that **they are simple enough to be analysed.**

ADV_INT.4.10D **The developer shall ensure that functions whose purpose are not relevant for enforcing or supporting the SFRs are excluded from the TSF modules.**

ADV_INT.4.11D The developer shall provide a TSF internals description.

Content and presentation elements:

ADV_INT.4.1C There shall be a one-to-one correspondence between the modules identified in TSF internals description and the modules identified in the modular design description.

ADV_INT.4.2C The TSF internals description shall describe the process used for modular decomposition.

ADV_INT.4.3C The TSF internals description shall describe how the TSF design is a reflection of the modular decomposition process.

ADV_INT.4.4C The TSF internals description shall provide a justification, on a per-module basis, of any deviations from the coding standards.

ADV_INT.4.5C The TSF internals description shall include a coupling analysis that describes intermodule coupling for all TSF modules.

ADV_INT.4.6C The TSF internals description shall include a cohesion analysis that describes the types of cohesion for all TSF modules.

ADV_INT.4.7C The TSF internals description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by modules of the TSF, other than those permitted.

ADV_INT.4.8C The TSF internals description shall describe the layering architecture and shall describe the services that each layer provides.

ADV_INT.4.9C The TSF internals description shall describe the methodology used to determine the layering architecture.

ADV_INT.4.10C The TSF internals description shall identify all modules associated with each layer of the TSF.

ADV_INT.4.11C The TSF internals description shall identify all interactions between layers of the TSF.

ADV_INT.4.12C The TSF internals description shall provide a justification of interactions that are initiated from a lower layer to a higher layer.

ADV_INT.4.13C The TSF internals description shall **justify** all modules of the TSF that contain unused or redundant code.

ADV_INT.4.14C **The TSF internals description shall show that mutual dependencies have been minimised, and justify those that remain.**

ADV_INT.4.15C **The TSF internals description shall describe how the entire TSF has been structured to minimise complexity.**

ADV_INT.4.16C **The TSF internals description shall justify the inclusion of any SFR non-interfering modules in the TSF.**

Evaluator action elements:

ADV_INT.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_INT.4.2E The evaluator ***shall perform*** a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of SFR-enforcing, SFR-supporting, and SFR non-interfering TSF modules.

ADV_INT.4.3E The evaluator ***shall perform*** a complexity analysis for a subset of SFR-enforcing, SFR-supporting, and SFR non-interfering TSF modules.

ADV_INT.4.4E **The evaluator *shall confirm* that the modules of the TSF are simple enough to be analysed.**

15.5 Security policy modelling (ADV_SPM)

Objectives

415 It is the objective of this family to provide additional assurance that the interfaces associated with the security functions in the functional specification enforce the policies in the TSP. This is accomplished via the development of a formal security policy model of the security functionality of the TSF, and establishing a correspondence between the functional specification and the security policy model.

Component levelling

416 This family contains only one component.

Application notes

417 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 sufficiently be 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 be used as precise design and implementation guidance, thereby providing increased assurance that the modelled security requirements are satisfied by the TOE. The precision of the model and resulting guidance is significantly improved by casting the model in a formal language.

418 The creation of a formal security policy model helps to identify and eliminate ambiguous, inconsistent, 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.

419 A formal model 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.

420 A model is merely an abstraction or simplification of reality, the purpose of which is to capture key aspects of the behaviour of that reality. An IT security policy is set of restrictions and properties that specify how information and computing resources are prevented from being used to violate an organisational security policy, accompanied by a persuasive set of engineering arguments showing that these restrictions and properties play a key role in the enforcement of the organisational security policy. A security policy model is a precise formal presentation of the important aspects of

security and their relationship to the behaviour of the IT system; it identifies the set of rules and practises that regulates how a system manages, protects, and otherwise controls the system resources.

421 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. The rules of these policies are not explicitly stated in any SFR; they are implicit in the sense that one must consider all of the related requirement components (e.g. those of the FAU: Security audit family to derive the implicit audit policy). **ADV_SPM.1.2D** contains an assignment for identifying the policies that are formally modelled.

ADV_SPM.1 Formal TOE security policy model

Dependencies: ADV_FSP.4 Complete functional specification

Developer action elements:

ADV_SPM.1.1D **The developer shall provide a formal TSP model.**

ADV_SPM.1.2D **The developer shall provide a formal proof of correspondence between the functional specification and the following policies of the TSP model: [assignment: *list of formally-stated policies in the TSP model*].**

ADV_SPM.1.3D **The developer shall provide a demonstration of correspondence between the functional specification and the following policies of the TSP model: [assignment: *list of policies that are provided semi-formally in the TSP model*].**

Content and presentation elements:

ADV_SPM.1.1C **The TSP model shall be in a formal style, supported by explanatory text as required, and identify the security policies of the TSF that are modelled.**

ADV_SPM.1.2C **The TSP model shall describe the rules and characteristics of all policies of the TSF that are modelled.**

ADV_SPM.1.3C **The correspondence shall show that the functional specification is consistent and complete with respect to the TSP model.**

ADV_SPM.1.4C **The correspondence between the TSP model and the functional specification shall be at the correct level of formality.**

Evaluator action elements:

ADV_SPM.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

15.6 TOE design (ADV_TDS)

Objectives

422 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 realized.

Component levelling

423 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.

Application notes

424 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.

425 This family uses two levels of decomposition: the *TOE component* 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 TOE component 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 TOE component may be further divided into lower-level TOE components, or into modules. Very complex TOEs might require several levels of TOE components in order to adequately convey a useful description of how the TOE works. Very simple TOEs, in contrast, might not require a TOE component level of description; the module might adequately describe how the TOE works.

426 This family, like Functional specification (ADV_FSP), differentiates *SFR-enforcing*, *SFR-supporting*, and *SFR-non-interfering* characteristics, applying them to TOE components and modules, as well as to their behaviour. For example, an SFR-enforcing module can have SFR-enforcing behaviour as well as non-SFR-enforcing behaviour. TOE components and modules, and “SFR-enforcing”, etc. are all further explained in greater detail in A.4, ADV_TDS: Components and Modules.

427 The general approach adopted for design documentation is that, as the level of assurance increases, the emphasis of description shifts from the general (TOE component 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 TOE component 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 TOE component level of description.

428 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).

429 In the requirements for this family, the term *interface* is used as the means of communication (between two TOE components 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 TOE components or modules are communicating.

ADV_TDS.1 Basic design

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.1.1D **The developer shall provide the design of the TOE.**

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.**

Content and presentation elements:

ADV_TDS.1.1C **The design shall describe the structure of the TOE in terms of components.**

ADV_TDS.1.2C **The design shall identify all components of the TSF.**

ADV_TDS.1.3C **The design shall describe each TSF component identified as non-SFR-enforcing in sufficient detail to determine that it is non-SFR-enforcing.**

ADV_TDS.1.4C **The design shall provide a high-level description of the SFR-enforcing behaviour of the SFR-enforcing components.**

ADV_TDS.1.5C **The design shall provide a description of the interactions between the SFR-enforcing components of the TSF and other components of the TSF.**

Evaluator action elements:

ADV_TDS.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ADV_TDS.1.2E **The evaluator *shall determine* that the design is an accurate and complete instantiation of all security functional requirements.**

ADV_TDS.2 Architectural design

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.2.1D The developer shall provide the design of the TOE.

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.

Content and presentation elements:

ADV_TDS.2.1C The design shall describe the structure of the TOE in terms of components.

ADV_TDS.2.2C The design shall identify all components of the TSF.

ADV_TDS.2.3C **The design shall describe each SFR non-interfering component of the TSF in detail sufficient to determine that it is SFR non-interfering.**

ADV_TDS.2.4C The design shall provide a **detailed** description of the SFR-enforcing behaviour of the SFR-enforcing components.

ADV_TDS.2.5C The design shall provide a high-level description of the **non-SFR-enforcing** behaviour of the SFR-enforcing components.

ADV_TDS.2.6C **The design shall provide a high-level description of the behaviour of the SFR-supporting components.**

ADV_TDS.2.7C The design shall provide a description of the interactions between the components of the TSF.

Evaluator action elements:

ADV_TDS.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.2.2E The evaluator ***shall determine*** that the design is an accurate and complete instantiation of all security functional requirements.

ADV_TDS.3 Basic modular design

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.3.1D The developer shall provide the design of the TOE.

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.

Content and presentation elements:

ADV_TDS.3.1C The design shall describe the structure of the TOE in terms of components.

ADV_TDS.3.2C **The design shall describe the TSF in terms of modules.**

ADV_TDS.3.3C The design shall identify all components of the TSF.

ADV_TDS.3.4C The design shall provide a description of **each component** of the **TSF.**

ADV_TDS.3.5C The design shall provide a description of the interactions between the components of the TSF.

ADV_TDS.3.6C **The design shall provide a mapping from the components of the TSF to the modules of the TSF.**

ADV_TDS.3.7C **The design shall identify and describe data that are common to more than one module, where any of the modules is an SFR-enforcing module.**

ADV_TDS.3.8C **The design shall describe each SFR-enforcing module in terms of its purpose, interfaces, return values from those interfaces, and called interfaces to other modules.**

ADV_TDS.3.9C **For each SFR-enforcing module, the** design shall **provide an algorithmic description detailed enough to represent** the TSF **implementation.**

ADV_TDS.3.10C The design shall describe each **non-SFR-enforcing module in terms** of **its purpose and interaction with other modules.**

Evaluator action elements:

ADV_TDS.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.3.2E The evaluator ***shall determine*** that the design is an accurate and complete instantiation of all security functional requirements.

ADV_TDS.4 Semiformal modular design

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.4.1D The developer shall provide the design of the TOE.

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.

Content and presentation elements:

ADV_TDS.4.1C The design shall describe the structure of the TOE in terms of components.

ADV_TDS.4.2C The design shall describe the TSF in terms of modules, **designating each module as either as SFR-enforcing, SFR-supporting, or SFR-non-interfering.**

ADV_TDS.4.3C The design shall identify all components of the TSF.

ADV_TDS.4.4C The design shall provide a description of each component of the TSF.

ADV_TDS.4.5C The design shall provide a description of the interactions between the components of the TSF.

ADV_TDS.4.6C The design shall provide a mapping from the components of the TSF to the modules of the TSF.

ADV_TDS.4.7C The design shall identify and describe data that are common to more than one module, where any of the modules is an SFR-enforcing **or SFR-supporting** module.

ADV_TDS.4.8C The design shall describe each SFR-enforcing **and SFR-supporting** module in terms of its purpose, interfaces, return values from those interfaces, and called interfaces to other modules.

ADV_TDS.4.9C For each SFR-enforcing **and SFR-supporting** module, the design shall provide an algorithmic description detailed enough to represent the TSF implementation.

ADV_TDS.4.10C The design shall describe each **SFR-non-interfering** module in terms of its purpose and interaction with other modules.

Evaluator action elements:

ADV_TDS.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.4.2E The evaluator ***shall determine*** that the design is an accurate and complete instantiation of all security functional requirements.

ADV_TDS.5 Complete semiformal modular design

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.5.1D The developer shall provide the design of the TOE.

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.

Content and presentation elements:

ADV_TDS.5.1C The design shall describe the structure of the TOE in terms of components.

ADV_TDS.5.2C The design shall describe the TSF in terms of modules, designating each module as either as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

ADV_TDS.5.3C The design shall identify all components of the TSF.

ADV_TDS.5.4C The design shall provide a description of each component of the TSF.

ADV_TDS.5.5C The design shall provide a description of the interactions between the components of the TSF.

ADV_TDS.5.6C The design shall provide a mapping from the components of the TSF to the modules of the TSF.

ADV_TDS.5.7C The design shall identify and describe data that are common to more than one module.

ADV_TDS.5.8C The design shall describe each module in terms of its purpose, interfaces, return values from those interfaces, and called interfaces to other modules.

ADV_TDS.5.9C **The** design shall provide an algorithmic description **for each module** detailed enough to represent the TSF implementation.

Evaluator action elements:

ADV_TDS.5.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.5.2E The evaluator ***shall determine*** that the design is an accurate and complete instantiation of all security functional requirements.

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

Dependencies: No dependencies.

Developer action elements:

ADV_TDS.6.1D The developer shall provide the design of the TOE.

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.

ADV_TDS.6.3D **The developer shall provide a formal presentation of the high-level description of the TSF.**

Content and presentation elements:

ADV_TDS.6.1C The design shall describe the structure of the TOE in terms of components.

ADV_TDS.6.2C The design shall describe the TSF in terms of modules, designating each module as either as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

ADV_TDS.6.3C The design shall identify all components of the TSF.

ADV_TDS.6.4C The design shall provide a description of each component of the TSF.

ADV_TDS.6.5C The design shall provide a description of the interactions between the components of the TSF.

ADV_TDS.6.6C The design shall provide a mapping from the components of the TSF to the modules of the TSF.

ADV_TDS.6.7C The design shall identify and describe data that are common to more than one module.

ADV_TDS.6.8C The design shall describe each module in terms of its purpose, interfaces, return values from those interfaces, and called interfaces to other modules.

ADV_TDS.6.9C The design shall provide an algorithmic description for each module detailed enough to represent the TSF implementation.

ADV_TDS.6.10C **The formal presentation of the high-level description of the TSF shall describe the TSF using a formal style, supported by informal, explanatory text where appropriate.**

Evaluator action elements:

ADV_TDS.6.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.6.2E The evaluator ***shall determine*** that the design is an accurate and complete instantiation of all security functional requirements.

16 Class AGD: Guidance documents

430 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 includes the possibility of unintended incorrect configuration or handling of the TOE.

431 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.

432 Guidance documentation comprises, when included in the assurance requirements, the specific guidance resulting from the requirements in the Flaw remediation (ALC_FLR) family.

433 Textual and graphical representations of the guidance documents related terms can be found in Section 4.2 respectively in figure 1.

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

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

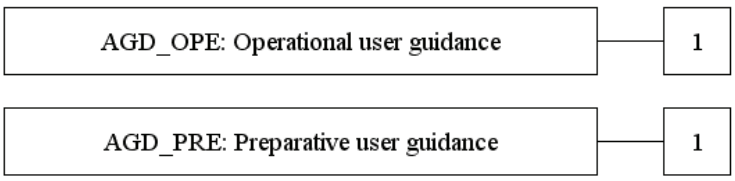

Figure 11 - AGD: Guidance documents class decomposition

16.1 Operational user guidance (AGD_OPE)

Objectives

436 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 configuring, 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, including 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.

437 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. This 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 functions, resulting in an undetected insecure state.

Component levelling

438 This family contains only one component.

Application notes

439 There may be different user roles or groups defined in the ST 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 user guidance. They may be roughly grouped in administrators and non-administrative users, more specific in 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.

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

441 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.

442 **AGD_OPE.1.4C** requires that the user guidance describe 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.

443 Misleading or unreasonable guidance may result in a user of the TOE believing that the TOE is secure when it is not.

444 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.

445 An example of unreasonable guidance would be a recommendation to follow a procedure that imposed an unduly onerous administrative burden.

AGD_OPE.1 Operational user guidance

Dependencies: ADV_FSP.1 Basic functional specification

Developer action elements:

AGD_OPE.1.1D **The developer shall provide the user guidance.**

Content and presentation elements:

AGD_OPE.1.1C **The 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.**

AGD_OPE.1.2C **The user guidance shall describe, for each user role, how to use the available interfaces provided by the TOE in a secure manner.**

AGD_OPE.1.3C **The 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.**

AGD_OPE.1.4C **The 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.**

AGD_OPE.1.5C **The 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.**

AGD_OPE.1.6C **The user guidance shall, for each user role, describe the security measures to be followed in order to fulfil the security objectives for the operational environment as described in the ST.**

AGD_OPE.1.7C **The user guidance shall be clear and reasonable.**

Evaluator action elements:

AGD_OPE.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

16.2 Preparative user guidance (AGD_PRE)

Objectives

446 Preparative procedures are useful for ensuring that the TOE has been accepted 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 operation in the user 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.

Component levelling

447 This family contains only one component.

Application notes

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

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

450 It might also be the case that the TOE is already installed by the time the evaluation starts. In this case it may be inappropriate to demand and analyse installation procedures.

451 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.

AGD_PRE.1 Preparative procedures

Dependencies: No dependencies.

Developer action elements:

AGD_PRE.1.1D **The developer shall provide the preparative procedures.**

Content and presentation elements:

AGD_PRE.1.1C **The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered TOE in accordance with developer's delivery procedures.**

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.**

Evaluator action elements:

AGD_PRE.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

AGD_PRE.1.2E **The evaluator shall apply the preparative procedures to confirm that the TOE can be prepared securely for operation.**

17 Class ALC: Life-cycle support

452 Life-cycle support is an aspect of establishing discipline and control in the processes of refinement of the TOE during its development and maintenance. 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 and maintenance activities.

453 In the product life-cycle it is distinguished whether the TOE is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the TOE is handed over to the user. This is also the point of transition from the ALC to the AGD class.

454 The ALC class consists of seven families. Life-cycle definition (ALC_LCD) is the high-level description of the TOE life-cycle; CM capabilities (ALC_CMC) a more detailed description of the management of the configuration items. CM scope (ALC_CMS) requires a minimum set of configuration items to be managed in the defined way. Development security (ALC_DVS) is concerned with the developer's physical security measures; Tools and techniques (ALC_TAT) with the development tools and implementation standards used by the developer; Flaw remediation (ALC_FLR) with the handling of security flaws. Delivery (ALC_DEL) defines the procedures used for the delivery of the TOE to the consumer. Delivery processes occurring during the development of the TOE are denoted rather as transports, and are handled in the context of integration and acceptance procedures in other families of this class.

455 Textual and graphical representations of several terms frequently used in this class and their relations can be found in Section 4.3 respectively in figure 1. Note: The rough life-cycle model presented there serves for the clarification of the used terms; developers may use different life-cycle models.

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

457 Figure 12 shows the families within this class, and the hierarchy of components within the families.

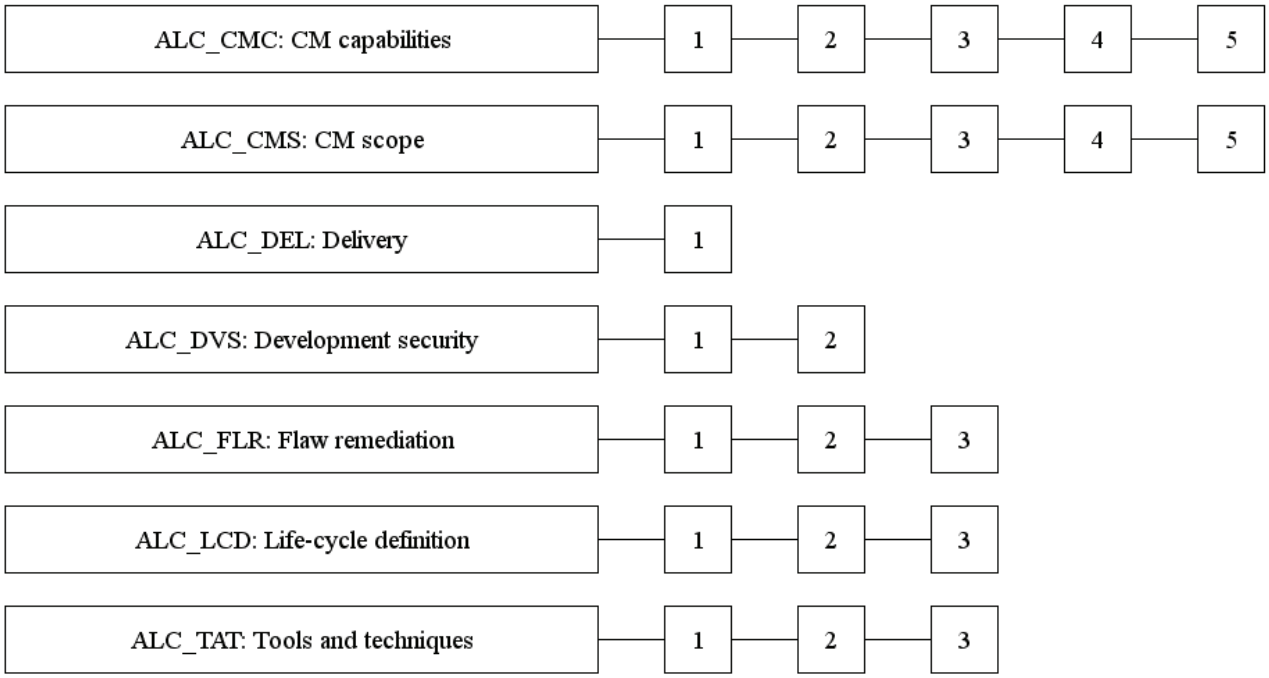

Figure 12 - ALC: Life-cycle support class decomposition

17.1 CM capabilities (ALC_CMC)

Objectives

458 Configuration management (CM) is one means for increasing assurance that the TOE meets the SFRs. CM establishes this by requiring discipline and control in the processes of refinement and modification of the TOE and the related information. CM systems are put in place to ensure the integrity of the portions of the TOE that they control, by providing a method of tracking any changes, and by ensuring that all changes are authorised.

459 The objective of this family is to require the developer's CM system to have certain capabilities. These address the likelihood that accidental or unauthorised modifications of the configuration items will occur. The CM system should ensure the integrity of the TOE from the early design stages through all subsequent maintenance efforts.

460 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 prove insufficient to prevent unauthorised modification, automated systems are less susceptible to human error or negligence.

461 The objectives of this family include the following:

a) ensuring that the TOE is correct and complete before it is sent to the consumer;

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

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

Component levelling

462 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.

Application notes

463 While it is desired that CM be applied from the early design stages and continue into the future, this family requires that CM be in place and in use prior to the end of the evaluation.

464 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.

465 For developers who have separate CM systems for development, production and/or the final product, it would be useful to document all of them. For

evaluation purposes, the separate CM systems should be regarded as parts of an overall CM system which is addressed in the criteria.

466 Similarly, if parts of the TOE are produced by different developers or at different sites, the CM systems being in use at the different places should be regarded as parts of an overall CM system which is addressed in the criteria. In this situation, integration aspects have also to be taken into account.

467 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 identify specific items that must be included in the configuration list, and hence covered by CM.

468 **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.

469 **ALC_CMC.3.7C** 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.

470 **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.

471 **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.

ALC_CMC.1 Labelling of the TOE

Dependencies: ALC_CMS.1 TOE CM coverage

Objectives

472 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.

Developer action elements:

ALC_CMC.1.1D **The developer shall provide the TOE.**

Content and presentation elements:

ALC_CMC.1.1C **The TOE shall be labelled with its reference.**

Evaluator action elements:

ALC_CMC.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_CMC.2 Use of a CM system

Dependencies: ALC_CMS.1 TOE CM coverage

Objectives

473 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.

474 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.

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

Developer action elements:

ALC_CMC.2.1D The developer shall provide the TOE.

ALC_CMC.2.2D **The developer shall provide the CM documentation.**

ALC_CMC.2.3D **The developer shall use a CM system.**

Content and presentation elements:

ALC_CMC.2.1C The TOE shall be labelled with its reference.

ALC_CMC.2.2C **The CM documentation shall describe the method used to uniquely identify the configuration items.**

ALC_CMC.2.3C **The CM system shall uniquely identify all configuration items.**

Evaluator action elements:

ALC_CMC.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMC.3 Authorisation controls

Dependencies: ALC_CMS.1 TOE CM coverage
ALC_DVS.1 Identification of security measures

Objectives

476 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.

477 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.

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

479 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.

Developer action elements:

ALC_CMC.3.1D The developer shall provide the TOE.

ALC_CMC.3.2D The developer shall provide the CM documentation.

ALC_CMC.3.3D The developer shall use a CM system.

Content and presentation elements:

ALC_CMC.3.1C The TOE shall be labelled with its reference.

ALC_CMC.3.2C The CM documentation shall describe the method used to uniquely identify the configuration items.

ALC_CMC.3.3C The CM system shall uniquely identify all configuration items.

ALC_CMC.3.4C **The CM system shall provide measures such that only authorised changes are made to the configuration items.**

ALC_CMC.3.5C **The CM documentation shall include a CM plan.**

ALC_CMC.3.6C **The CM plan shall describe how the CM system is used for the development of the TOE.**

ALC_CMC.3.7C **The evidence shall demonstrate that the CM system is operating in accordance with the CM plan.**

ALC_CMC.3.8C **The evidence shall demonstrate that all configuration items have been and are being maintained under the CM system.**

Evaluator action elements:

ALC_CMC.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMC.4 Production support, acceptance procedures and automation

Dependencies: ALC_CMS.1 TOE CM coverage
ALC_DVS.1 Identification of security measures
ALC_LCD.1 Developer defined life-cycle model

Objectives

480 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.

481 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.

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

483 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.

484 The purpose of acceptance procedures is to ensure that the parts of the TOE are of adequate quality. Acceptance procedures are an essential element in integration processes and in the life-cycle management of the TOE.

485 In development environments where the implementation representation 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 the objective of this component to ensure that the implementation representation is 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.

486 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.

Developer action elements:

ALC_CMC.4.1D The developer shall provide the TOE.

ALC_CMC.4.2D The developer shall provide the CM documentation.

ALC_CMC.4.3D The developer shall use a CM system.

Content and presentation elements:

ALC_CMC.4.1C The TOE shall be labelled with its reference.

ALC_CMC.4.2C The CM documentation shall describe the method used to uniquely identify the configuration items.

ALC_CMC.4.3C The CM system shall uniquely identify all configuration items.

ALC_CMC.4.4C The CM system shall provide **automated** measures such that only authorised changes are made to the configuration items.

ALC_CMC.4.5C **The CM system shall support the production of the TOE by automated means.**

ALC_CMC.4.6C The CM documentation shall include a CM plan.

ALC_CMC.4.7C The CM plan shall describe how the CM system is used for the development of the TOE.

ALC_CMC.4.8C **The CM plan shall describe the acceptance procedures.**

ALC_CMC.4.9C The evidence shall demonstrate that the CM system is operating in accordance with the CM plan.

ALC_CMC.4.10C The evidence shall demonstrate that all configuration items have been and are being maintained under the CM system.

Evaluator action elements:

ALC_CMC.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMC.5 Advanced support

Dependencies: ALC_CMS.1 TOE CM coverage
ALC_DVS.2 Sufficiency of security measures
ALC_LCD.1 Developer defined life-cycle model

Objectives

487 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.

488 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.

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

490 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.

491 The purpose of acceptance procedures is to ensure that the parts of the TOE are of adequate quality. Acceptance procedures are an essential element in integration processes and in the life-cycle management of the TOE.

492 In development environments where the implementation representation 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 the objective of this component to ensure that the implementation representation is 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.

493 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.

494 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.

495 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.

Developer action elements:

ALC_CMC.5.1D The developer shall provide the TOE.

ALC_CMC.5.2D The developer shall provide the CM documentation.

ALC_CMC.5.3D The developer shall use a CM system.

Content and presentation elements:

ALC_CMC.5.1C The TOE shall be labelled with its reference.

ALC_CMC.5.2C The CM documentation shall describe the method used to uniquely identify the configuration items.

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.**

ALC_CMC.5.4C The CM system shall uniquely identify all configuration items.

ALC_CMC.5.5C The CM system shall provide automated measures such that only authorised changes are made to the configuration items.

ALC_CMC.5.6C The CM system shall support the production of the TOE by automated means.

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.**

ALC_CMC.5.8C **The CM system shall clearly identify the configuration items that comprise the TSF.**

ALC_CMC.5.9C **The CM system shall support the audit of all changes to the TOE by automated means, including as a minimum the originator, date, and time in the audit trail.**

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.**

ALC_CMC.5.11C **The CM system shall be able to identify the version of the implementation representation from which the TOE is generated.**

ALC_CMC.5.12C The CM documentation shall include a CM plan.

ALC_CMC.5.13C The CM plan shall describe how the CM system is used for the development of the TOE.

ALC_CMC.5.14C The CM plan shall describe the acceptance procedures.

ALC_CMC.5.15C The evidence shall demonstrate that the CM system is operating in accordance with the CM plan.

ALC_CMC.5.16C The evidence shall demonstrate that all configuration items have been and are being maintained under the CM system.

Evaluator action elements:

ALC_CMC.5.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

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.**

17.2 CM scope (ALC_CMS)

Objectives

496 The objective of this family is to require 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.

Component levelling

497 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 in the ST; the parts of the TOE; the implementation representation; security flaws; and development tools and related information.

Application notes

498 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).

ALC_CMS.1 TOE CM coverage

Dependencies: No dependencies.

Objectives

499 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.

Application notes

500 **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).

Developer action elements:

ALC_CMS.1.1D **The developer shall provide a configuration list for the TOE.**

Content and presentation elements:

ALC_CMS.1.1C **The configuration list includes the following: the TOE itself; and the evaluation evidence required by the SARs in the ST.**

ALC_CMS.1.2C **The configuration list shall uniquely identify the configuration items.**

Evaluator action elements:

ALC_CMS.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_CMS.2 Parts of the TOE CM coverage

Dependencies: No dependencies.

Objectives

501 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 in the ST under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

Application notes

502 ALC_CMS.2.1C introduces the requirement that the parts that comprise the TOE (the deliverables) be included in the configuration list and hence be subject to the CM requirements of CM capabilities (ALC_CMC).

Developer action elements:

ALC_CMS.2.1D The developer shall provide a configuration list for the TOE.

Content and presentation elements:

ALC_CMS.2.1C The configuration list includes the following: the TOE itself; the evaluation evidence required by the SARs in the ST; **and the parts that comprise the TOE.**

ALC_CMS.2.2C The configuration list shall uniquely identify the configuration items.

ALC_CMS.2.3C **For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.**

Evaluator action elements:

ALC_CMS.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMS.3 Implementation representation CM coverage

Dependencies: No dependencies.

Objectives

503 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 in the ST under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

Application notes

504 **ALC_CMS.3.1C** introduces the requirement that the TOE implementation representation (as defined in the glossary) be included in the list of configuration items and hence be subject to the CM requirements of CM capabilities (ALC_CMC).

Developer action elements:

ALC_CMS.3.1D The developer shall provide a configuration list for the TOE.

Content and presentation elements:

ALC_CMS.3.1C The configuration list includes the following: the TOE itself; the evaluation evidence required by the SARs in the ST; the parts that comprise the TOE; **and the implementation representation.**

ALC_CMS.3.2C The configuration list shall uniquely identify the configuration items.

ALC_CMS.3.3C For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

Evaluator action elements:

ALC_CMS.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMS.4 Problem tracking CM coverage

Dependencies: No dependencies.

Objectives

505 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 in the ST under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

506 Placing security flaws under CM ensures that security flaw reports are not lost or forgotten, and allows a developer to track security flaws to their resolution.

Application notes

507 **ALC_CMS.4.1C** introduces the requirement that 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 previous security flaws and their resolution be maintained, as well as details regarding current security flaws.

Developer action elements:

ALC_CMS.4.1D The developer shall provide a configuration list for the TOE.

Content and presentation elements:

ALC_CMS.4.1C The configuration list includes the following: the TOE itself; the evaluation evidence required by the SARs in the ST; the parts that comprise the TOE; the implementation representation; **and security flaws.**

ALC_CMS.4.2C The configuration list shall uniquely identify the configuration items.

ALC_CMS.4.3C For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

Evaluator action elements:

ALC_CMS.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_CMS.5 Development tools CM coverage

Dependencies: No dependencies.

Objectives

508 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 in the ST under CM provides assurance that they have been modified in a controlled manner with proper authorisations.

509 Placing security flaws under CM ensures that security flaw reports are not lost or forgotten, and allows a developer to track security flaws to their resolution.

510 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.

Application notes

511 **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.

Developer action elements:

ALC_CMS.5.1D The developer shall provide a configuration list for the TOE.

Content and presentation elements:

ALC_CMS.5.1C The configuration list includes the following: the TOE itself; the evaluation evidence required by the SARs in the ST; the parts that comprise the TOE; the implementation representation; security flaws; **and development tools and related information.**

ALC_CMS.5.2C The configuration list shall uniquely identify the configuration items.

ALC_CMS.5.3C For each TSF relevant configuration item, the configuration list shall indicate the developer of the item.

Evaluator action elements:

ALC_CMS.5.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of

17.3 Delivery (ALC_DEL)

Objectives

512 The concern of this family is the secure transfer of the finished TOE from the development environment into the responsibility of the user.

513 The requirements for delivery call for system control and distribution facilities and procedures that detail the measures 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 objectives identified in the PP/ST relating to the security of the TOE during delivery.

Component levelling

514 This family contains only one component. An increasing level of protection is established by requiring commensurability of the delivery procedures with the assumed attack potential in the family Vulnerability analysis (AVA_VAN).

Application notes

515 Transports from subcontractors to the developer or between different development sites are not considered here, but in the family Development security (ALC_DVS).

516 The end of the delivery phase is marked by the transfer of the TOE into the responsibility of the user. This does not necessarily coincide with the arrival of the TOE at the user's location.

517 The delivery procedures could consider issues such as:

a) ensuring that the TOE received by the consumer corresponds precisely to the evaluated version of the TOE;

b) avoiding/detecting any tampering with the actual version of the TOE;

c) preventing submission of a false 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 is delivered;

e) avoiding/detecting the TOE being intercepted during delivery; and

f) avoiding the TOE being delayed or stopped during distribution.

ALC_DEL.1 Delivery procedures

Dependencies: No dependencies.

Developer action elements:

ALC_DEL.1.1D **The developer shall document procedures for delivery of the TOE or parts of it to the user.**

ALC_DEL.1.2D **The developer shall use the delivery procedures.**

Content and presentation elements:

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 user.**

Evaluator action elements:

ALC_DEL.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

17.4 Development security (ALC_DVS)

Objectives

518 Development security is concerned with physical, procedural, personnel, and other security measures that may be used in the development environment to protect the TOE and its parts. It includes the physical security of the development location and any procedures used to select development staff.

Component levelling

519 The components in this family are levelled on the basis of whether justification of the sufficiency of the security measures is required.

Application notes

520 This family deals with measures to remove or reduce threats existing at the developer's site.

521 The evaluator should determine whether there is a need for visiting the developer's site in order to confirm that the requirements of this family are met.

522 Although development security deals with the maintenance of the TOE and hence with aspects becoming relevant after the completion of the evaluation, the Development security (ALC_DVS) requirements specify only that the development security measures be in place at the time of evaluation. Furthermore, Development security (ALC_DVS) does not contain any requirements related to the sponsor's intention to apply the development security measures in the future, after completion of the evaluation.

523 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.

ALC_DVS.1 Identification of security measures

Dependencies: No dependencies.

Developer action elements:

ALC_DVS.1.1D **The developer shall produce development security documentation.**

Content and presentation elements:

ALC_DVS.1.1C **The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment.**

ALC_DVS.1.2C **The development security documentation shall provide evidence that these security measures are followed during the development and maintenance of the TOE.**

Evaluator action elements:

ALC_DVS.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_DVS.1.2E **The evaluator *shall confirm* that the security measures are being applied.**

ALC_DVS.2 Sufficiency of security measures

Dependencies: No dependencies.

Developer action elements:

ALC_DVS.2.1D The developer shall produce development security documentation.

Content and presentation elements:

ALC_DVS.2.1C The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment.

ALC_DVS.2.2C The development security documentation shall provide evidence that these security measures are followed during the development and maintenance of the TOE.

ALC_DVS.2.3C **The evidence shall justify that the security measures provide the necessary level of protection to maintain the confidentiality and integrity of the TOE.**

Evaluator action elements:

ALC_DVS.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_DVS.2.2E The evaluator ***shall confirm*** that the security measures are being applied.

17.5 Flaw remediation (ALC_FLR)

Objectives

524 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.

Component levelling

525 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.

Application notes

526 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.

527 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.

528 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) measures 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.

529 Once the evaluation of a TOE is complete, it is no longer the target for evaluation. Furthermore, any changes to this evaluated TOE result in the original evaluation results being no longer applicable to the changed version. The phrase release of the TOE used in this family therefore refers to a version of a product or system that is a release of a certified TOE, to which changes have been applied.

ALC_FLR.1 Basic flaw remediation

Dependencies: No dependencies.

Developer action elements:

ALC_FLR.1.1D **The developer shall provide flaw remediation procedures addressed to TOE developers.**

Content and presentation elements:

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.**

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.**

ALC_FLR.1.3C **The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.**

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.**

Evaluator action elements:

ALC_FLR.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_FLR.2 Flaw reporting procedures

Dependencies: No dependencies.

Objectives

530 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.

Developer action elements:

ALC_FLR.2.1D The developer shall provide flaw remediation procedures addressed to TOE developers.

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.**

ALC_FLR.2.3D **The developer shall provide flaw remediation guidance addressed to TOE users.**

Content and presentation elements:

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.

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.

ALC_FLR.2.3C The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

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.

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.**

ALC_FLR.2.6C **The procedures for processing reported security flaws shall ensure that any reported flaws are corrected and the correction issued to TOE users.**

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.**

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.**

Evaluator action elements:

ALC_FLR.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_FLR.3 Systematic flaw remediation

Dependencies: No dependencies.

Objectives

531 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.

Developer action elements:

ALC_FLR.3.1D The developer shall provide flaw remediation procedures addressed to TOE developers.

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.

ALC_FLR.3.3D The developer shall provide flaw remediation guidance addressed to TOE users.

Content and presentation elements:

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.

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.

ALC_FLR.3.3C The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

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.

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.

ALC_FLR.3.6C **The flaw remediation procedures shall include a procedure requiring timely responses for the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.**

ALC_FLR.3.7C The procedures for processing reported security flaws shall ensure that any reported flaws are corrected and the correction issued to TOE users.

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.

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.

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.**

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.**

Evaluator action elements:

ALC_FLR.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

17.6 Life-cycle definition (ALC_LCD)

Objectives

532 Poorly controlled development and maintenance of the TOE can result in a TOE that does not meet all of its SFRs. Therefore, it is important that a model for the development and maintenance of a TOE be established as early as possible in the TOE's life-cycle.

533 Using a model for the development and maintenance of a TOE does not guarantee that the TOE meets all of its SFRs. It is possible that the model chosen will be insufficient or inadequate and therefore no benefits in the quality of the TOE can be observed. Using a life-cycle model that has been approved by a group of experts (e.g. academic experts, standards bodies) improves the chances that the development and maintenance models will contribute to the TOE meeting its SFRs. The use of a life-cycle model including some quantitative valuation adds further assurance in the overall quality of the TOE development process.

Component levelling

534 The components in this family are levelled on the basis of increasing requirements for standardisation and measurability of the life-cycle model, and for compliance with that model.

Application notes

535 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.

536 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 Life-cycle definition (ALC_LCD), acceptance subsequent to internal transports in Development 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.

537 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.

538 A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the developer can supply information that

shows that the model appropriately minimises the danger the TOE not meeting its SFRs.

539 A standardised life-cycle model is a model that has been approved by a group of experts (e.g. academic experts, standards bodies). Standardised life-cycle models are normally public, well accepted and common practise in a specific industry, but developer specific models are not excluded; the emphasis is on the expertise. Examples of standardised life-cycle models are the Waterfall Model, Rapid Prototyping, the V-Model and the Spiral Model.

540 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.

ALC_LCD.1 Developer defined life-cycle model

Dependencies: No dependencies.

Developer action elements:

ALC_LCD.1.1D **The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE.**

ALC_LCD.1.2D **The developer shall provide life-cycle definition documentation.**

Content and presentation elements:

ALC_LCD.1.1C **The life-cycle definition documentation shall describe the model used to develop and maintain the TOE.**

ALC_LCD.1.2C **The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.**

Evaluator action elements:

ALC_LCD.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_LCD.2 Standardised life-cycle model

Dependencies: No dependencies.

Developer action elements:

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 standardised life-cycle model.**

ALC_LCD.2.2D The developer shall provide life-cycle definition documentation.

Content and presentation elements:

ALC_LCD.2.1C The life-cycle definition documentation shall describe the model used to develop and maintain the TOE.

ALC_LCD.2.2C **The life-cycle definition documentation shall explain how the model is used to develop and maintain the TOE.**

ALC_LCD.2.3C **The life-cycle definition documentation shall demonstrate that the life-cycle model used by the developer is compliant with the standardised life-cycle model.**

ALC_LCD.2.4C **The life-cycle definition documentation shall explain why the model was chosen.**

ALC_LCD.2.5C The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.

Evaluator action elements:

ALC_LCD.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_LCD.3 Measurable life-cycle model

Dependencies: No dependencies.

Developer action elements:

ALC_LCD.3.1D The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE, that is based on a standardised **and measurable** life-cycle model.

ALC_LCD.3.2D The developer shall provide life-cycle definition documentation.

ALC_LCD.3.3D **The developer shall measure the TOE development using the standardised and measurable life-cycle model.**

ALC_LCD.3.4D **The developer shall provide life-cycle output documentation.**

Content and presentation elements:

ALC_LCD.3.1C **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 TOE development against the model.**

ALC_LCD.3.2C The life-cycle definition documentation shall explain how the model is used to develop and maintain the TOE.

ALC_LCD.3.3C The life-cycle definition documentation shall demonstrate that the life-cycle model used by the developer is compliant with the standardised **and measurable** life-cycle model.

ALC_LCD.3.4C The life-cycle definition documentation shall explain why the model was chosen.

ALC_LCD.3.5C The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE.

ALC_LCD.3.6C **The life-cycle output documentation shall provide the results of the measurements of the TOE development using the standardised and measurable life-cycle model.**

Evaluator action elements:

ALC_LCD.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

17.7 Tools and techniques (ALC_TAT)

Objectives

541 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.

Component levelling

542 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.

Application notes

543 There is a requirement for well-defined development tools. These are tools that have been shown to be applicable without the need for intensive further clarification. 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.

544 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.

545 In ALC_TAT.2 and ALC_TAT.3, implementation guidelines may be accepted as 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.

546 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 includes third party software, hardware, or firmware. The configuration list introduced in CM scope (ALC_CMS) requires that for each configuration item to indicate if it has been generated by the TOE developer or by third party developers.

ALC_TAT.1 Well-defined development tools

Dependencies: ADV_IMP.1 Implementation representation of the TSF

Developer action elements:

ALC_TAT.1.1D **The developer shall identify the development tools being used for the TOE.**

ALC_TAT.1.2D **The developer shall document the selected implementation-dependent options of the development tools.**

Content and presentation elements:

ALC_TAT.1.1C **All development tools used for implementation shall be well-defined.**

ALC_TAT.1.2C **The documentation of the development tools shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.**

ALC_TAT.1.3C **The documentation of the development tools shall unambiguously define the meaning of all implementation-dependent options.**

Evaluator action elements:

ALC_TAT.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ALC_TAT.2 Compliance with implementation standards

Dependencies: ADV_IMP.1 Implementation representation of the TSF

Developer action elements:

ALC_TAT.2.1D The developer shall identify the development tools being used for the TOE.

ALC_TAT.2.2D The developer shall document the selected implementation-dependent options of the development tools.

ALC_TAT.2.3D **The developer shall describe the implementation standards to be applied.**

Content and presentation elements:

ALC_TAT.2.1C All development tools used for implementation shall be well-defined.

ALC_TAT.2.2C The documentation of the development tools shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.

ALC_TAT.2.3C The documentation of the development tools shall unambiguously define the meaning of all implementation-dependent options.

Evaluator action elements:

ALC_TAT.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_TAT.2.2E **The evaluator *shall confirm* that the implementation standards have been applied.**

ALC_TAT.3 Compliance with implementation standards - all parts

Dependencies: ADV_IMP.1 Implementation representation of the TSF

Developer action elements:

ALC_TAT.3.1D The developer shall identify the development tools being used for the TOE.

ALC_TAT.3.2D The developer shall document the selected implementation-dependent options of the development tools.

ALC_TAT.3.3D The developer shall describe the implementation standards **for all parts of the TOE.**

Content and presentation elements:

ALC_TAT.3.1C All development tools used for implementation shall be well-defined.

ALC_TAT.3.2C The documentation of the development tools shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.

ALC_TAT.3.3C The documentation of the development tools shall unambiguously define the meaning of all implementation-dependent options.

Evaluator action elements:

ALC_TAT.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ALC_TAT.3.2E The evaluator ***shall confirm*** that the implementation standards have been applied.

18 Class ATE: Tests

547 The class "Tests" encompasses four families: Coverage (ATE_COV), Depth (ATE_DPT), Independent testing (ATE_IND) (e.g. functional testing performed by evaluators), and Functional tests (ATE_FUN). Testing provides assurance that the TSF meets its design descriptions (functional specification, TOE design, and implementation representation).

548 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.

549 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 (TOE design, implementation representation) is required.

550 Functional tests (ATE_FUN) addresses the performing of these 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.

551 Figure 13 shows the families within this class, and the hierarchy of components within the families.

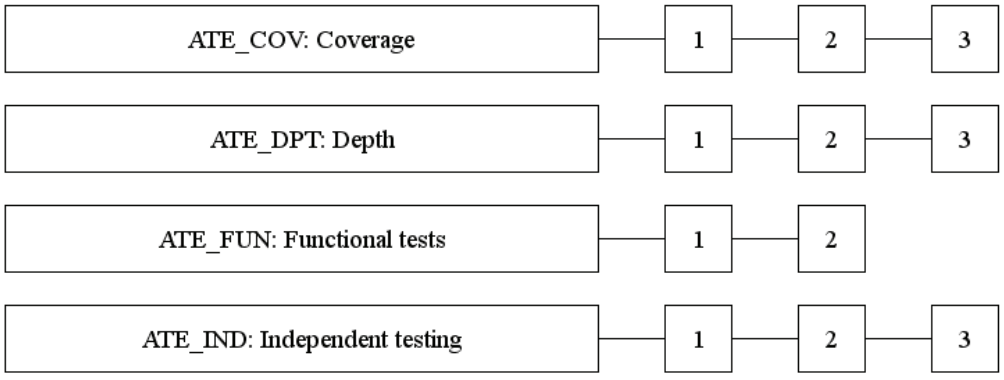

Figure 13 - ATE: Tests class decomposition

18.1 Coverage (ATE_COV)

Objectives

552 This family addresses those aspects of testing that deal with completeness of test coverage. That is, it addresses the extent to which the TSF is tested, and whether or not the testing is sufficiently extensive to demonstrate that the TSF operates in accordance with its functional specification.

Component levelling

553 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 TSF operates in accordance with its functional specification.

Application notes

ATE_COV.1 Evidence of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification
ATE_FUN.1 Functional testing

Objectives

554 In this component, the objective is to establish that the TSF has been tested against its functional specification. This is to be achieved through an examination of developer evidence of correspondence.

Application notes

555 In this component, the objective is to confirm that the developer performed some tests of some interfaces in the functional specification. The developer is required to show how tests in the test documentation correspond to interfaces in the functional specification. This can be achieved by a statement of correspondence, perhaps using a table.

Developer action elements:

ATE_COV.1.1D **The developer shall provide evidence of the test coverage.**

Content and presentation elements:

ATE_COV.1.1C **The evidence of the test coverage shall show the correspondence between the tests in the test documentation and the interfaces in the functional specification.**

Evaluator action elements:

ATE_COV.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ATE_COV.2 Analysis of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification
ATE_FUN.1 Functional testing

Objectives

556 In this component, the objective is to confirm that the developer performed some tests of all interfaces in the functional specification. This is to be achieved through an examination of developer evidence of correspondence.

Application notes

557 In this component the developer is required to show how tests in the test documentation correspond to interfaces 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.

Developer action elements:

ATE_COV.2.1D The developer shall provide **an analysis** of the test coverage.

Content and presentation elements:

ATE_COV.2.1C The **analysis** of the test coverage shall **demonstrate** the correspondence between the tests in the test documentation and the interfaces in the functional specification.

ATE_COV.2.2C **The analysis of the test coverage shall demonstrate that the correspondence between the interfaces in the functional specification and the tests in the test documentation is complete.**

Evaluator action elements:

ATE_COV.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ATE_COV.3 Rigorous analysis of coverage

Dependencies: ADV_FSP.2 Security-enforcing functional specification
ATE_FUN.1 Functional testing

Objectives

558 In this component, the objective is to confirm that the developer performed exhaustive tests of all interfaces in the functional specification.

Application notes

559 In this component the developer is required to show how tests in the test documentation correspond to interfaces 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 exhaustively test each interface in the functional specification.

560 The use of the term *exhaustive* in this component is meant to convey a degree of effort beyond simple verification that the interfaces behave as advertised. This additional work would include 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 of exceeded limits are expected to be checked, nor is it expected to verify that no access is granted to all possible Users other than A.

Developer action elements:

ATE_COV.3.1D The developer shall provide an analysis of the test coverage.

Content and presentation elements:

ATE_COV.3.1C The analysis of the test coverage shall demonstrate the correspondence between the tests in the test documentation and the interfaces in the functional specification.

ATE_COV.3.2C The analysis of the test coverage shall demonstrate that the correspondence between the interfaces in the functional specification and the tests in the test documentation is complete.

ATE_COV.3.3C **The analysis of the test coverage shall rigorously demonstrate that all interfaces in the functional specification have been exhaustively tested.**

Evaluator action elements:

ATE_COV.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

18.2 Depth (ATE_DPT)

Objectives

561 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 (TOE design, implementation representation).

562 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.

Component levelling

563 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.

Application notes

564 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 component 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.

565 At the highest component of this family, the testing is performed not only against the TOE design, but also against the implementation representation.

ATE_DPT.1 Testing: basic design

Dependencies: ADV_TDS.1 Basic design
ATE_FUN.1 Functional testing

Objectives

566 The TOE design's components descriptions of the TSF provide a high-level description of the internal workings of the TSF. Testing at the level of the TOE components (e.g. subsystems) provides assurance that the TSF subsystems have been correctly realised.

Developer action elements:

ATE_DPT.1.1D **The developer shall provide the analysis of the depth of testing.**

Content and presentation elements:

ATE_DPT.1.1C **The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the interfaces of TOE components in the TOE design.**

ATE_DPT.1.2C **The analysis of the depth of testing shall demonstrate that the correspondence between the component interfaces in the TOE design and the tests in the test documentation is complete.**

Evaluator action elements:

ATE_DPT.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ATE_DPT.2 Testing: modular design

Dependencies: ADV_TDS.3 Basic modular design
ATE_FUN.1 Functional testing

Objectives

567 The TOE design's components descriptions of the TSF provide a high-level description of the internal workings of the TSF. Testing at the level of the TOE components (e.g. subsystems) provides assurance that the TSF subsystems have been correctly realised.

568 The TOE design's modules descriptions of the TSF provide a description of the internal workings of the TSF. Testing at the level of the modules provides assurance that the TSF modules have been correctly realised.

Developer action elements:

ATE_DPT.2.1D The developer shall provide the analysis of the depth of testing.

Content and presentation elements:

ATE_DPT.2.1C The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the interfaces of TOE components **and modules** in the TOE design.

ATE_DPT.2.2C The analysis of the depth of testing shall demonstrate that the correspondence between the component interfaces in the TOE design and the tests in the test documentation is complete.

ATE_DPT.2.3C **The analysis of the depth of testing shall demonstrate that the correspondence between the module interfaces in the TOE design and the tests in the test documentation is complete.**

Evaluator action elements:

ATE_DPT.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ATE_DPT.3 Testing: implementation representation

Dependencies: ADV_TDS.3 Basic modular design
ADV_IMP.2 Implementation of the TSF
ATE_FUN.1 Functional testing

Objectives

569 The TOE design's components descriptions of the TSF provide a high-level description of the internal workings of the TSF. Testing at the level of the TOE components (e.g. subsystems) provides assurance that the TSF subsystems have been correctly realised.

570 The TOE design's modules descriptions of the TSF provide a description of the internal workings of the TSF. Testing at the level of the modules provides assurance that the TSF modules have been correctly realised.

571 The implementation representation of the TSF provides a detailed description of the internal workings of the TSF. Testing at the level of the implementation provides assurance that the TSF implementation has been correctly realised.

Developer action elements:

ATE_DPT.3.1D The developer shall provide the analysis of the depth of testing.

Content and presentation elements:

ATE_DPT.3.1C The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the interfaces of TOE components and modules in the TOE design **and the implementation representation.**

ATE_DPT.3.2C The analysis of the depth of testing shall demonstrate that the correspondence between the component interfaces in the TOE design and the tests in the test documentation is complete.

ATE_DPT.3.3C The analysis of the depth of testing shall demonstrate that the correspondence between the module interfaces in the TOE design and the tests in the test documentation is complete.

ATE_DPT.3.4C **The analysis of the depth of testing shall demonstrate that the TSF operated in accordance with its implementation representation.**

Evaluator action elements:

ATE_DPT.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

18.3 Functional tests (ATE_FUN)

Objectives

572 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.

573 This family contributes to providing assurance that the likelihood of undiscovered flaws is relatively small.

574 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).

Component levelling

575 This family contains two components, the higher requiring that ordering dependencies are analysed.

Application notes

576 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.

577 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.

ATE_FUN.1 Functional testing

Dependencies: ATE_COV.1 Evidence of coverage

Objectives

578 The objective is for the developer to demonstrate that the tests in the test documentation are performed and documented correctly.

Developer action elements:

ATE_FUN.1.1D **The developer shall test the TSF and document the results.**

ATE_FUN.1.2D **The developer shall provide test documentation.**

Content and presentation elements:

ATE_FUN.1.1C **The test documentation shall consist of test plans, expected test results and actual test results.**

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.**

ATE_FUN.1.3C **The expected test results shall show the anticipated outputs from a successful execution of the tests.**

ATE_FUN.1.4C **The actual test results shall be consistent with the expected test results.**

Evaluator action elements:

ATE_FUN.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ATE_FUN.2 Ordered functional testing

Dependencies: ATE_COV.1 Evidence of coverage

Objectives

579 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.

Application notes

580 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.

Developer action elements:

ATE_FUN.2.1D The developer shall test the TSF and document the results.

ATE_FUN.2.2D The developer shall provide test documentation.

Content and presentation elements:

ATE_FUN.2.1C The test documentation shall consist of test plans, expected test results and actual test results.

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.

ATE_FUN.2.3C The expected test results shall show the anticipated outputs from a successful execution of the tests.

ATE_FUN.2.4C The actual test results shall be consistent with the expected test results.

ATE_FUN.2.5C **The test documentation shall include an analysis of the test procedure ordering dependencies.**

Evaluator action elements:

ATE_FUN.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

18.4 Independent testing (ATE_IND)

Objectives

581 The objectives are to gain more assurance that the TSF meets its design representations by verifying the developer testing and the performing of additional tests by the evaluator.

Component levelling

582 Levelling is based upon the amount of test documentation, test support and the amount of evaluator testing.

Application notes

583 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. It may also take the form of the augmentation of the developer's functional tests, either to extend the scope or the depth of the developer's tests, or to test for obvious public domain security weaknesses that could be applicable to the TOE. 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.

584 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).

585 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.

586 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.

587 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.

588 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.

589 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.

ATE_IND.1 Independent testing - conformance

Dependencies: ADV_FSP.1 Basic functional specification
AGD_OPE.1 Operational user guidance
AGD_PRE.1 Preparative procedures

Objectives

590 In this component, the objective is to demonstrate that the TOE operates in accordance with its design representations and guidance documents.

Application notes

591 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.

Developer action elements:

ATE_IND.1.1D **The developer shall provide the TOE for testing.**

Content and presentation elements:

ATE_IND.1.1C **The TOE shall be suitable for testing.**

Evaluator action elements:

ATE_IND.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ATE_IND.1.2E **The evaluator *shall test* a subset of the TSF interfaces to confirm that the TSF operates as specified.**

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_FUN.1 Functional testing

Objectives

592 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.

Application notes

593 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.

594 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.

Developer action elements:

ATE_IND.2.1D The developer shall provide the TOE for testing.

Content and presentation elements:

ATE_IND.2.1C The TOE shall be suitable for testing.

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.**

Evaluator action elements:

ATE_IND.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ATE_IND.2.2E **The evaluator *shall execute* a sample of tests in the test documentation to verify the developer test results.**

ATE_IND.2.3E The evaluator ***shall test*** a subset of the TSF interfaces to confirm that the TSF operates as specified.

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_FUN.1 Functional testing

Objectives

595 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.

Application notes

596 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.

597 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.

Developer action elements:

ATE_IND.3.1D The developer shall provide the TOE for testing.

Content and presentation elements:

ATE_IND.3.1C The TOE shall be suitable for testing.

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.

Evaluator action elements:

ATE_IND.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ATE_IND.3.2E The evaluator ***shall execute*** **all** tests in the test documentation to verify the developer test results.

ATE_IND.3.3E The evaluator ***shall test*** the TSF to confirm that the TSF operates as specified.

19 Class AVA: Vulnerability assessment

598 The AVA: Vulnerability assessment class addresses the possibility of exploitable vulnerabilities introduced in the development or the operation of the TOE.

599 Figure 14 shows the families within this class, and the hierarchy of components within the families.

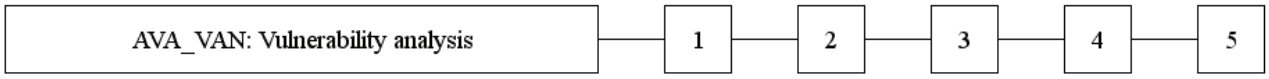

Figure 14 - AVA: Vulnerability assessment class decomposition

Application notes

600 If there are no Unobservability (FDP_UNO) or information flow policies (expressed through Access control (FDP_ACC)) requirements in the ST, it is not applicable to consider unintended channels of communication (e.g. covert channels) during the conduct of the vulnerability analysis. The lack of these requirements in the ST reflects that there are no objectives 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 illicit data signals.

601 The analysis for unintended channels of communication require significant levels of knowledge of the TOE and technology type. Therefore, this analysis would only be appropriate at higher levels of attack potential.

19.1 Vulnerability analysis (AVA_VAN)

Objectives

602 Vulnerability analysis is an assessment to determine whether potential vulnerabilities identified, during the evaluation of the construction 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 users to violate the TSP.

603 Vulnerability analysis deals with the threats that a user 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.

Component levelling

604 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.

AVA_VAN.1 Vulnerability survey

Dependencies: ADV_FSP.1 Basic functional specification
AGD_OPE.1 Operational user guidance
AGD_PRE.1 Preparative procedures

Objectives

605 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.

606 The evaluator performs penetration testing to demonstrate that the potential vulnerabilities cannot be exploited in its operational environment by an attacker with basic attack potential.

Developer action elements:

AVA_VAN.1.1D The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.1.1C The TOE shall be suitable for testing.

Evaluator action elements:

AVA_VAN.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.1.2E The evaluator *shall perform* a search of public domain sources to identify potential vulnerabilities in the TOE.

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.**

AVA_VAN.2 Vulnerability analysis

Dependencies: ADV_ARC.1 Architectural Design with domain separation and non-bypassability
ADV_FSP.1 Basic functional specification
ADV_TDS.1 Basic design
AGD_OPE.1 Operational user guidance
AGD_PRE.1 Preparative procedures

Objectives

607 A vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

608 The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE.

Developer action elements:

AVA_VAN.2.1D The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.2.1C The TOE shall be suitable for testing.

Evaluator action elements:

AVA_VAN.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.2.2E The evaluator ***shall perform*** a search of public domain sources to identify potential vulnerabilities in the TOE.

AVA_VAN.2.3E **The evaluator *shall perform* an independent vulnerability analysis of the TOE using the guidance documentation, functional specification and TOE design to identify potential vulnerabilities in the TOE.**

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.

AVA_VAN.3 Focused vulnerability analysis

Dependencies: ADV_ARC.1 Architectural Design with domain separation and non-bypassability
ADV_FSP.2 Security-enforcing 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

Objectives

609 A vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

610 The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE.

Developer action elements:

AVA_VAN.3.1D The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.3.1C The TOE shall be suitable for testing.

Evaluator action elements:

AVA_VAN.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.3.2E The evaluator ***shall perform*** a search of public domain sources to identify potential vulnerabilities in the TOE.

AVA_VAN.3.3E The evaluator ***shall perform*** an independent vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design **and implementation representation** to identify potential vulnerabilities in the TOE.

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 **extended-basic** attack potential.

AVA_VAN.4 Methodical vulnerability analysis

Dependencies: ADV_ARC.1 Architectural Design with domain separation and non-bypassability
ADV_FSP.2 Security-enforcing 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

Objectives

611 A methodical vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities.

612 The evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE.

Developer action elements:

AVA_VAN.4.1D The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.4.1C The TOE shall be suitable for testing.

Evaluator action elements:

AVA_VAN.4.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.4.2E The evaluator ***shall perform*** a search of public domain sources to identify potential vulnerabilities in the TOE.

AVA_VAN.4.3E The evaluator ***shall perform*** an independent, **methodical** vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design and implementation representation to identify potential vulnerabilities in the TOE.

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.

AVA_VAN.5 Advanced methodical vulnerability analysis

Dependencies: ADV_ARC.1 Architectural Design with domain separation and non-bypassability
ADV_FSP.2 Security-enforcing 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

Objectives

613 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.

Developer action elements:

AVA_VAN.5.1D The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.5.1C The TOE shall be suitable for testing.

Evaluator action elements:

AVA_VAN.5.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.5.2E The evaluator ***shall perform*** a search of public domain sources to identify potential vulnerabilities in the TOE.

AVA_VAN.5.3E The evaluator ***shall perform*** an independent, methodical vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design and implementation representation to identify potential vulnerabilities in the TOE.

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.

20 Class ACO: Composition

614 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.

615 Composition involves taking two or more evaluated IT entities and combining them for use, with no further development of either IT entity. The development of additional IT entities or consideration of additional environment entities is not included.

616 This approach does not provide an alternative approach for the evaluation of components, where all aspects of the component are developed by a single development organisation. Also, composition as discussed in ACO differs significantly from systems. Composition under ACO is the combination of two or more components such that a new product is formed that can be installed and integrated into any specific environment instance that meets the objectives for the environment. This contrasts with a system, which is developed and evaluated for a specific environment, taking the controls and procedures in the environment into consideration during the evaluation.

617 Further discussion of the definition and interactions within composed TOEs is provided in Annex B.

618 Figure 15 shows the families within this class, and the hierarchy of components within the families.

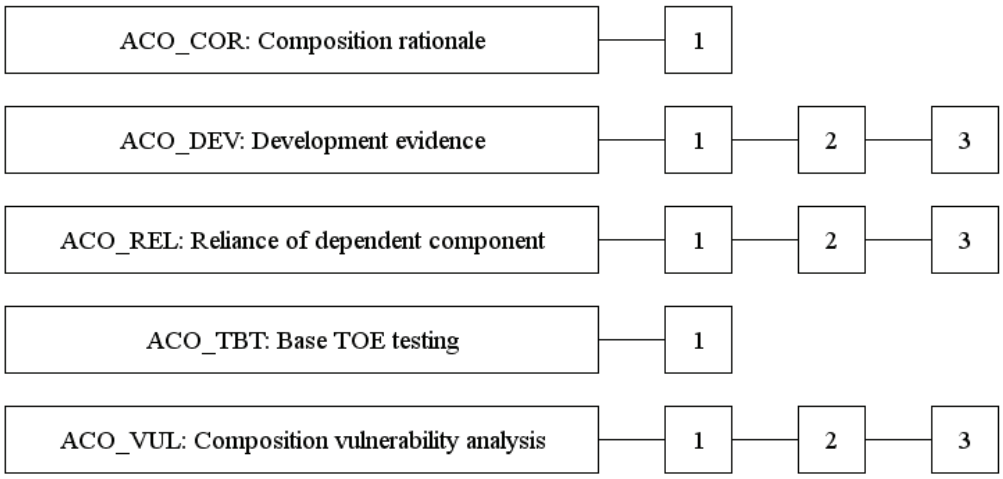

Figure 15 - ACO: Composition class decomposition

20.1 Composition rationale (ACO_COR)

Objectives

619 This family addresses the requirement to demonstrate that the base component can provide an appropriate level of assurance for use in composition.

Component levelling

620 There is only a single component in this family.

ACO_COR.1 Composition rationale

Dependencies: ACO_DEV.1 Functional Description
ALC_CMC.1 Labelling of the TOE
ACO_REL.1 Basic reliance information

Developer action elements:

ACO_COR.1.1D **The developer shall provide composition information for the base component.**

Content and presentation elements:

ACO_COR.1.1C **The composition information shall demonstrate that the base component, when configured as required to support the TSF of the dependent component, provides a level of assurance in that support functionality at least as high as that of the dependent component.**

Evaluator action elements:

ACO_COR.1.1E **The evaluator *shall confirm* that the information meets all requirements for content and presentation of evidence.**

20.2 Development evidence (ACO_DEV)

Objectives

621 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).

Component levelling

622 The components are levelled on the basis of increasing amounts of detail about the interfaces provided, and how they are implemented.

Application notes

623 The development information evidence includes a specification of the base component. This may be the evidence used to satisfy the 15 requirements in the base component evaluation, 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 functional dependency information provided for the dependent component.

624 The evidence required by Development evidence (ACO_DEV) is analogous to the evidence required by Functional specification (ADV_FSP), TOE design (ADV_TDS), and Implementation representation (ADV_IMP) families.

ACO_DEV.1 Functional Description

Dependencies: ACO_REL.1 Basic reliance information

Objectives

625 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.

626 Only the details of the interfaces are required, no additional details of the base component internals are necessary, as the internals of the base component were examined during the base component evaluation. The consistent scope of base component for both the component evaluation and

its use in the composed TOE evaluation is confirmed during the Composition rationale (ACO_COR) activity. Therefore, the results of the ADV_TDS.1 Basic design activity for the base component can be reused with no further analysis.

Developer action elements:

ACO_DEV.1.1D **The developer shall provide development information for the base component.**

Content and presentation elements:

ACO_DEV.1.1C **The development information shall describe the purpose of each interface of the base component used in the composed TOE.**

ACO_DEV.1.2C **The development information shall identify those interfaces of the base component that are provided to support the TSF of the dependent component.**

Evaluator action elements:

ACO_DEV.1.1E **The evaluator *shall confirm* that the information meets all requirements for content and presentation of evidence.**

ACO_DEV.1.2E **The evaluator *shall determine* that the interface description provided is consistent with the reliance information provided for the dependent component.**

ACO_DEV.2 Basic evidence of design

Dependencies: ACO_REL.1 Basic reliance information

Objectives

627 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.

628 Only the details of the interfaces are required, no additional details of the base component internals are necessary, as the internals of the base component were examined during the base component evaluation. The consistent scope of base component for both the component evaluation and its use in the composed TOE evaluation is confirmed during the Composition rationale (ACO_COR) activity. Therefore, the results of the ADV_TDS.2 Architectural design activity for the base component can be reused with no further analysis.

Developer action elements:

ACO_DEV.2.1D The developer shall provide development information for the base component.

Content and presentation elements:

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.

ACO_DEV.2.2C **The development information shall describe all parameters associated with each interface.**

ACO_DEV.2.3C **The development information shall describe all operations associated with each interface.**

ACO_DEV.2.4C **The development information shall describe the error messages resulting from processing associated with all operations.**

ACO_DEV.2.5C The development information shall identify those interfaces of the base component that are provided to support the TSF of the dependent component.

Evaluator action elements:

ACO_DEV.2.1E The evaluator ***shall confirm*** that the information meets all requirements for content and presentation of evidence.

ACO_DEV.2.2E The evaluator ***shall determine*** that the interface description provided is consistent with the **functional dependency** information provided for the dependent component.

ACO_DEV.3 Detailed evidence of design

Dependencies: ACO_REL.2 Reliance information

Objectives

629 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.

630 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.

Developer action elements:

ACO_DEV.3.1D The developer shall provide development information for the base component.

Content and presentation elements:

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.

ACO_DEV.3.2C The development information shall describe all parameters associated with each interface.

ACO_DEV.3.3C The development information shall describe all operations associated with each interface.

ACO_DEV.3.4C The development information shall describe the error messages resulting from processing associated with all operations.

ACO_DEV.3.5C **The development information shall describe the structure of the base component in terms of components.**

ACO_DEV.3.6C **The development information shall describe the architecture of those components that provide the interfaces of the base component that are relied upon to support the TSF of the dependent component.**

ACO_DEV.3.7C The development information shall identify those interfaces of the base component that are provided to support the TSF of the dependent component.

Evaluator action elements:

ACO_DEV.3.1E The evaluator ***shall confirm*** that the information meets all requirements for content and presentation of evidence.

ACO_DEV.3.2E The evaluator ***shall determine*** that the interface description provided is consistent with the functional dependency information provided for the dependent component.

20.3 Reliance of dependent component (ACO_REL)

Objectives

631 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.

632 This provides a description of the interface between the components that may not have been analysed during the component evaluations, as the interfaces were not TSFI of the component.

Component levelling

633 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.

Application notes

634 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 TSP. This will include all parts of the base component required to implement the base component SFRs.

635 The functional specification of this 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.

636 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. These are considered to be composition interfaces. The description of what a dependent component TSF relies upon of a base component is considered in Reliance of dependent component (ACO_REL).

637 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 the base component evaluation because the service in the base component was not

considered security-relevant, either because of the inherent purpose of the service (e.g., adjust type font) or because associated CC SFRs are not being claimed in the base component's ST (e.g. the login interface when no FIA: Identification, Authentication and Binding SFRs are claimed). These interfaces into the base component are often viewed as functional interfaces when evaluating the base component, and are in addition to the security interfaces (TSFI) considered in the functional specification.

638 In summary, the TSFI 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 are not explicitly considered during the component evaluation, are described by the reliance information provided to satisfy Reliance of dependent component (ACO_REL).

ACO_REL.1 Basic reliance information

Dependencies: ADV_FSP.2 Security-enforcing functional specification
ADV_TDS.1 Basic design

Developer action elements:

ACO_REL.1.1D The developer shall provide functional reliance information addressed to system integrators.

Content and presentation elements:

ACO_REL.1.1C The functional reliance information shall describe the functionality of the base component hardware, firmware and/or software that is relied upon by the dependent component TSF.

ACO_REL.1.2C The functional reliance information shall identify all interfaces through which the dependent component TSF requests services from the base component.

ACO_REL.1.3C The functional reliance information shall describe the purpose and method of use of each interface.

Evaluator action elements:

ACO_REL.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

ACO_REL.2 Reliance information

Dependencies: ADV_FSP.2 Security-enforcing functional specification
ADV_TDS.1 Basic design

Developer action elements:

ACO_REL.2.1D The developer shall provide functional reliance information addressed to system integrators.

Content and presentation elements:

ACO_REL.2.1C The functional reliance information shall describe the functionality of the base component hardware, firmware and/or software that is relied upon by the dependent component TSF.

ACO_REL.2.2C The functional reliance information shall identify all interfaces through which the dependent component TSF requests services from the base component.

ACO_REL.2.3C The functional reliance information shall describe the purpose and method of use of each interface.

ACO_REL.2.4C **The functional reliance information shall provide a parameter description for each interface.**

ACO_REL.2.5C **The functional reliance information shall describe the expected operations and results associated with each SFR-enforcing interface.**

ACO_REL.2.6C **The functional reliance information shall describe the error handling performed as a result of the dependent component TSF's use of each SFR-enforcing interface.**

Evaluator action elements:

ACO_REL.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ACO_REL.3 Detailed reliance information

Dependencies: ADV_FSP.4 Complete functional specification
ADV_TDS.3 Basic modular design
ADV_IMP.1 Implementation representation of the TSF

Developer action elements:

ACO_REL.3.1D The developer shall provide functional reliance information addressed to system integrators.

Content and presentation elements:

ACO_REL.3.1C The functional reliance information shall describe the functionality of the base component hardware, firmware and/or software that is relied upon by the dependent component TSF.

ACO_REL.3.2C The functional reliance information shall identify all interfaces through which the dependent component TSF requests services from the base component.

ACO_REL.3.3C The functional reliance information shall describe the purpose and method of use of each interface.

ACO_REL.3.4C The functional reliance information shall provide a parameter description for each interface.

ACO_REL.3.5C The functional reliance information shall describe the expected operations and results associated with **all interfaces.**

ACO_REL.3.6C The functional reliance information shall describe the error handling performed as a result of the dependent component TSF's use of **all interfaces.**

Evaluator action elements:

ACO_REL.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

20.4 Base TOE testing (ACO_TBT)

Objectives

639 This family requires that testing of the base component, as used in the composed TOE, is performed.

640 The dependent component is tested as used in the composed TOE, during the dependent component evaluation.

Component levelling

641 There is only a single component in this family.

Application notes

642 The test configurations used during the dependent component evaluation activities against the ATE: Tests components of the dependent component assurance package should include using the base component as a "platform". The Base TOE testing (ACO_TBT) family requires evidence of testing of the base component as used in the composed TOE.

643 The developer is to provide evidence of testing the base component interfaces used in the composition. The operation of base component interfaces would have been tested as part of the base component evaluation ATE: Tests activities. Therefore, if it was determined in Composition rationale (ACO_COR) that the base component is operating in accordance with the evaluated configuration, with all security functionality required by the dependent component included in the TSF, the evaluator action **ACO_TBT.1.1E** may be met through reuse of the base component ATE: Tests verdicts.

644 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 functional dependency information (Reliance of dependent component (ACO_REL) evidence).

645 The composed TOE will have been tested during the conduct of the ATE: Tests activities of the dependent component evaluation, as the configurations used for testing of the dependent component should have used the base component to satisfy the requirements for IT in the operational environment. If the base component was not used in the testing of the dependent component, or the configuration of either component varied, then the developer testing performed during the dependent component evaluation to satisfy the ATE: Tests requirements is to be repeated on the composed TOE.

ACO_TBT.1 Interface testing

Dependencies: ACO_REL.1 Basic reliance information

Objectives

646 The objective of this component is to ensure that each interface of the base component, on which the dependent component relies, is tested.

Developer action elements:

ACO_TBT.1.1D The developer shall provide the base component for testing.

ACO_TBT.1.2D The developer shall provide base component test documentation.

ACO_TBT.1.3D The developer shall test the base component and provide test results.

ACO_TBT.1.4D The developer shall provide dependent component test documentation.

ACO_TBT.1.5D The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the base component.

Content and presentation elements:

ACO_TBT.1.1C The base component shall be suitable for testing.

ACO_TBT.1.2C The base component test documentation shall consist of test plans, test procedure descriptions, expected test results and actual test results.

ACO_TBT.1.3C The test results from the developer execution of the tests shall demonstrate that the base component interface relied upon by the dependent component behaves as specified.

ACO_TBT.1.4C The dependent component test documentation shall demonstrate that the test configuration(s) included the base component as used in the composed TOE in the environment.

Evaluator action elements:

ACO_TBT.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

ACO_TBT.1.2E The evaluator *shall test* a subset of the interfaces to the base component to confirm that they operate as specified.

20.5 Composition vulnerability analysis (ACO_VUL)

Objectives

647 This family calls for an analysis of vulnerabilities that may be introduced as a result of the composition.

Component levelling

648 The components in this family are levelled on the basis of increasing scrutiny of vulnerability information from the public domain and independent vulnerability analysis.

Application notes

649 The developer will require details of any residual vulnerabilities reported during the base component evaluation. These may be gained from the base component developer or evaluation/certification reports from the base component evaluation. The developer will then perform an analysis of residual vulnerabilities in consideration of the composed TOE, operational environment.

650 Composition vulnerability analysis (ACO_VUL) is levelled in a manner consistent to that used in Vulnerability analysis (AVA_VAN), considering increasingly detailed information available for analysis and an increasing level of attack potential assumed for the attacker.

651 The assumptions and objectives for the component operational environment specified in each component ST are considered to determine that they are upheld in the composed TOE.

652 The consideration of composing the components will have to ensure that the vulnerability analysis for the base component has considered any applicable potential vulnerabilities identified in the public domain since the completion of the component evaluation. Therefore, the evaluator will perform a search of public domain sources.

653 Consideration of potential vulnerabilities in the dependent component is performed during the dependent component evaluation. That vulnerability analysis will be up to date for direct consideration in the composed TOE vulnerability analysis.

ACO_VUL.1 Composition vulnerability review

Dependencies: ACO_DEV.1 Functional Description

Developer action elements:

ACO_VUL.1.1D The developer shall provide the composed TOE for testing.

ACO_VUL.1.2D The developer shall provide composition vulnerability information for the composed TOE.

Content and presentation elements:

ACO_VUL.1.1C **The composition vulnerability information shall demonstrate that any residual vulnerabilities identified for the base component are not exploitable in the operational environment.**

ACO_VUL.1.2C **The composition vulnerability information shall demonstrate that any assumptions and objectives in the STs for the components are fulfilled by the other components.**

Evaluator action elements:

ACO_VUL.1.1E **The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.**

ACO_VUL.1.2E **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 operational environment.**

ACO_VUL.1.3E **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.**

ACO_VUL.2 Composition vulnerability analysis

Dependencies: ACO_DEV.2 Basic evidence of design

Developer action elements:

ACO_VUL.2.1D The developer shall provide the composed TOE for testing.

ACO_VUL.2.2D The developer shall provide composition vulnerability information for the composed TOE.

Content and presentation elements:

ACO_VUL.2.1C The composition vulnerability information shall demonstrate that any residual vulnerabilities identified for the base component are not exploitable in the operational environment.

ACO_VUL.2.2C The composition vulnerability information shall demonstrate that any assumptions and objectives in the STs for the components are fulfilled by the other components.

Evaluator action elements:

ACO_VUL.2.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ACO_VUL.2.2E 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 operational environment.

ACO_VUL.2.3E **The evaluator *shall perform* an independent vulnerability analysis of the composed TOE, using the guidance documentation, dependency information and composition information to identify potential vulnerabilities in the composed TOE.**

ACO_VUL.2.4E 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.

ACO_VUL.3 Extended-basic Composition vulnerability analysis

Dependencies: ACO_DEV.3 Detailed evidence of design

Developer action elements:

ACO_VUL.3.1D The developer shall provide the composed TOE for testing.

ACO_VUL.3.2D The developer shall provide composition vulnerability information for the composed TOE.

Content and presentation elements:

ACO_VUL.3.1C The composition vulnerability information shall demonstrate that any residual vulnerabilities identified for the base component are not exploitable in the operational environment.

ACO_VUL.3.2C The composition vulnerability information shall demonstrate that any assumptions and objectives in the STs for the components are fulfilled by the other components.

Evaluator action elements:

ACO_VUL.3.1E The evaluator ***shall confirm*** that the information provided meets all requirements for content and presentation of evidence.

ACO_VUL.3.2E 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 operational environment.

ACO_VUL.3.3E The evaluator ***shall perform*** an independent vulnerability analysis of the composed TOE, using the guidance documentation, dependency information and composition information to identify potential vulnerabilities in the composed TOE.

ACO_VUL.3.4E 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 **extended-basic** attack potential.

A Development (ADV) (normative)

654 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.1 ADV_ARC: Supplementary material on architectural characteristics

655 This section provides additional material on the architectural characteristics of self-protection, domain isolation, and non-bypassability of the TSF.

656 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. For TOEs that have dependencies on other IT entities, it is often the case that the TOE uses services/resources supplied by the other IT entities in order to perform its functions. In these cases, the TSF strictly speaking cannot protect itself, because it depends on the other IT entities to protect the services it uses. For the purposes of the architectural design document, the notion of *self-protection* applies only to the services provided by the TSF through its TSFI, and not to services provided by underlying IT entities that it uses.

657 Domain isolation is more complex, and refers to environments supplied by the TSF for the use by entities with limited security properties and access control rights (referred to as “untrusted entities” in the following example). An operating system TOE supplies a set of domains (address space, per-process environment variables) for use by untrusted processes (i.e. those processes 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 interfering (without brokering by the TSF) from another untrusted entity's domain.

658 Non-bypassability is a property that the security functions of the TSF (as specified by the SFRs) are always invoked 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).

659 The architectural design document presents the design information for mechanisms in the TSF related to self-protection, domain isolation, TSF initialisation, and non-bypassability. This type of information, like the decompositions for TOE design (ADV_TDS), describes how the TSF is implemented. The description, however, should be focused on providing information sufficient for the reader to determine that the TSF implementation or the security domains provided by the TSF are likely not to be compromised, and that the TSP enforcement mechanisms (that is, those that are implementing SFRs) are likely always being invoked. The level of this description is commensurate with the most detailed level of decomposition required for the SFR-enforcing portions of the TSF. For example, the architectural design document for a TOE with only an ADV_FSP.1 component requirement and no ADV_TDS.1 requirement would contain primarily configuration information that is visible from the external interface (protections on files that comprise the TSF, for instance). On the other hand, a TOE on which the ADV_TDS component was levied would have an architectural design document that contained more information on the internal design of the self-protection mechanisms (for instance, the memory management architecture).

660 The nature of the requirements for self-protection and domain isolation lend themselves to a design description much better than that for non-bypassability. Mechanisms that implement domain separation (e.g., memory management, protected processing modes provided by the hardware, etc.) can be identified and described. However, the property of non-bypassability is concerned with interfaces that bypass 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 TSP 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.

661 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 TSFI 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 non-SFR-enforcing 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 architectural design document.

662 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.

663 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 (usually 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.

664 With respect to self-protection and domain isolation, the design description should cover 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). The TSF might implement software protection constructs or coding conventions that contribute to implementing isolation of software domains, perhaps by delineating user address space from system address space.

665 For whichever means the TSF implements self-protection, domain isolation, and non-bypassability, the design description would be presented at the same level of 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 architectural design because none of the details of any internal workings of the TSF would be available.

666 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 components that make up the TSF, and there could be a description of how they work to implement self-protection, domain isolation, 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 architectural design 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.

667 If the available TOE design is more detailed (e.g. at the modular level), or the implementation representation is also available, then the description of the architectural design 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, are in place, parameter bounds checking, time of check/time of use checking), etc.

A.2 ADV_FSP: Examples of Identifying the TSFI

668 This section provides several examples that help to illustrate identification of the TSFI for various kinds of TOEs.

A.2.1 Example 1: stand-alone firewall

669 For a typical stand-alone firewall TSF (consisting of hardware and software, and where the TSF is the same as the TOE), the interfaces are:

a) The network interface with the LAN (the network cable socket)

b) The network interface with the WAN (the network cable socket)

c) An administrative interface for the firewall administrator (typically a graphical command shell)

d) Physical interfaces, through which attackers may physically alter the firewall or eavesdrop on its inner workings

e) A power interface, through which it draws electricity

A.2.2 Example 2: software firewall

670 For a typical software firewall TSF (a firewall application running on top of an OS and hardware, where the TSF is the same as the TOE and neither the OS nor the hardware is part of the TOE), the interfaces are:

a) The network interface with the LAN network stack provided by the underlying OS through which the TSF sends and receives IP packets

b) The network interface with the WAN network stack provided by the underlying OS through which the TSF sends and receives IP packets

c) An administrative interface for the firewall administrator (typically a GUI)

d) Interfaces through which the firewall can be accessed by other applications on the OS (deleting/modifying the firewall executable, direct reading/writing to the memory space of the firewall)

e) Interfaces through which the firewall obtains primitive functionality from the OS and hardware (executing machine code instructions, OS

APIs, such as creating/reading/writing/deleting files, graphical APIs etc.)

A.2.3 Example 3: A complex DBMS

671 Figure 16 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 boundary is identical to the TSF boundary. 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.

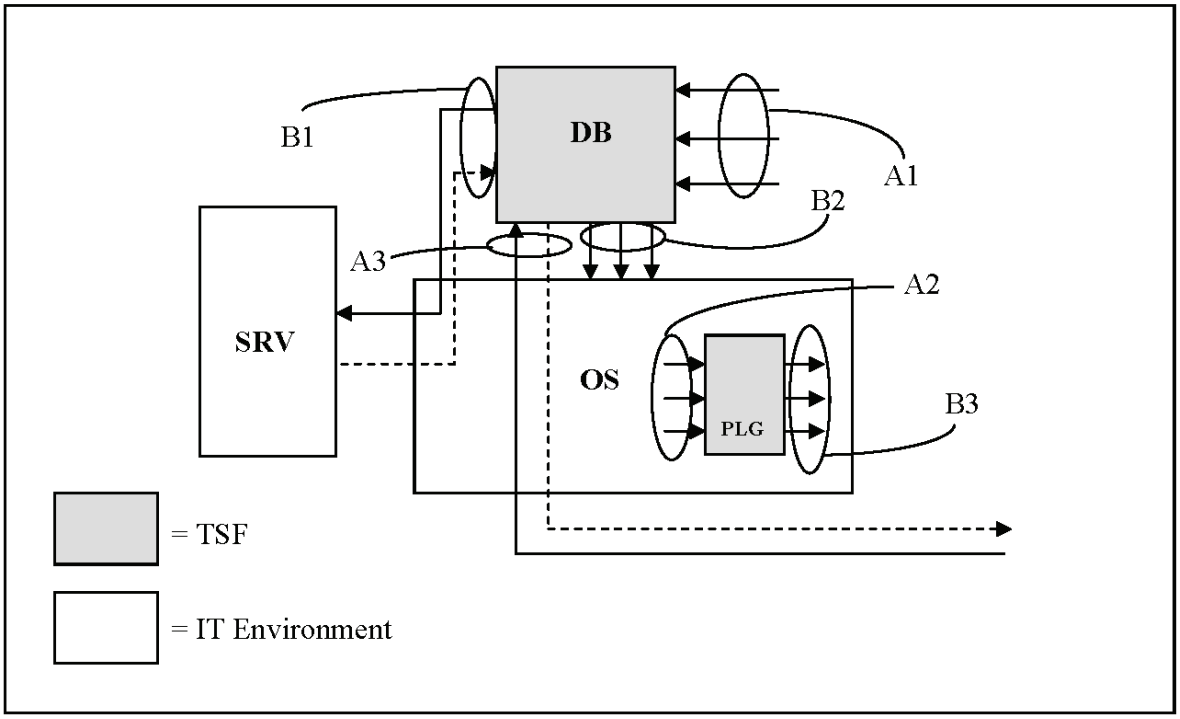

Figure 16 - Interfaces in a DBMS system

672 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.

673 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.

674 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.

675 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.

676 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 4: inaccessible interfaces

677 Examples interfaces that are not accessible (i.e., the ST security objectives for the operational environment may prevent access to these interfaces or limit access in such a way that they are practically inaccessible):

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 uninterruptible 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 state that “the OS and the hardware will provide a security domain for the application free from tampering by other programs”, interfaces through which the firewall can be accessed by other applications on the OS (deleting/modifying the firewall executable, direct reading/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/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.

A.3 ADV_INT.1: Subset Modularity

678 The component has limited value and would be suitable in cases where potentially-malicious users/subjects have limited or strictly controlled access to the TOE's TSFI or where there is another means of protection (e.g., domain isolation) that ensures the "modular" portions of the TSF cannot be adversely affected by the rest of the TSF (e.g., cryptographic functionality adheres to the strict modularity requirements and is isolated from the rest of the TSF).

679 This component may be particularly useful in specific situations where a developer has incorporated third-party software in the TOE that provides supporting functionality that does not directly enforce SFRs are crucial to the TOE and interactions with untrusted users/subjects is limited. For example, a firewall TOE may incorporate operating system functionality such as process management, memory management, file system management, etc. from a third-party that does not adhere to the modularity metrics levied in a higher level TSF internals (ADV_INT) component. Since this functionality may not be critical to the firewall enforcing its rule set and controlling the flow of network traffic it may be suitable to have only the critical SFR-enforcing software adhere to the stricter modularity requirements. The primary focus of the evaluation would be concerned with these critical modules and by satisfying the modularity metrics it will ensure the implementation of these modules is more easily understood and the developer has incorporated sound engineering principles in their design.

680 While this component may be suitable for some applications, there are instances where it would not make a lot of sense. For example, in a general purpose operating system it would not make sense to have only the access control functionality adhere to the modularity metrics, but not be isolated from the rest of the TSF. If a potentially-malicious users had access to a rich set of TSFI that may allow them to take advantage of a flaw in a portion of the software that does not adhere to the modularity metrics and may not be well understood by the developer, these users could adversely affect the access control software, even if it had satisfied the modularity requirements.

681 Figure 17 illustrates that the assigned SFR-enforcing portions of the TSF are a subset of the SFR-enforcing portions of the TSF. The non-SFR-enforcing portions of the TSF consist of the SFR-supporting and SFR-non-interfering modules.

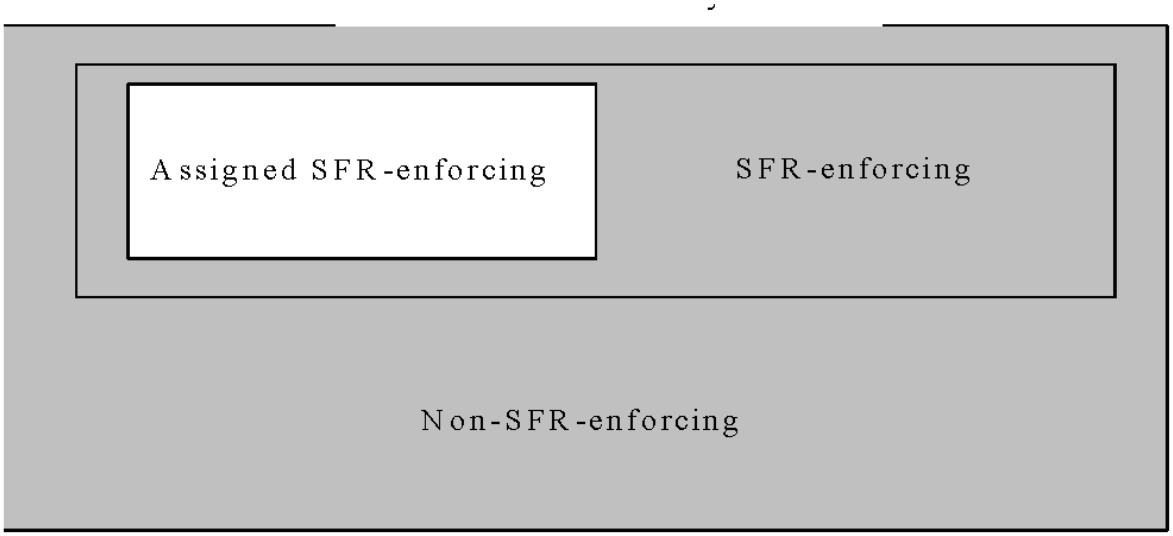

Figure 17 - Assigned SFR-enforcing modules may be a subset of the SFR-enforcing modules

682 The developer is required to identify the modules that are Assigned SFR-enforcing and implicitly the remaining modules, which will be non-Assigned SFR-enforcing. As stated earlier, the Assigned SFR-enforcing modules are those modules that interact with the module or modules that provide the TSFI for the assigned SFRs with justified exceptions. The justification of the non-Assigned SFR-enforcing modules (**ADV_INT.1.3C**) is required only for those modules that interact with Assigned SFR-enforcing modules and not for all non-Assigned SFR-enforcing modules. As depicted in Figure 18, if a TSFI has already been designated as non-Assigned SFR-enforcing then the designation of the modules interacting with the module providing the TSFI do not have to be justified (e.g., modules X, Y, Z). The justification of the designation is only necessary for the module(s) that interact with a module that provides a TSFI that is Assigned SFR-enforcing (e.g., modules D, E, F (since it is writing to a global variable that Module A is reading, but in this example, it is not an Assigned SFR-enforcing variable).

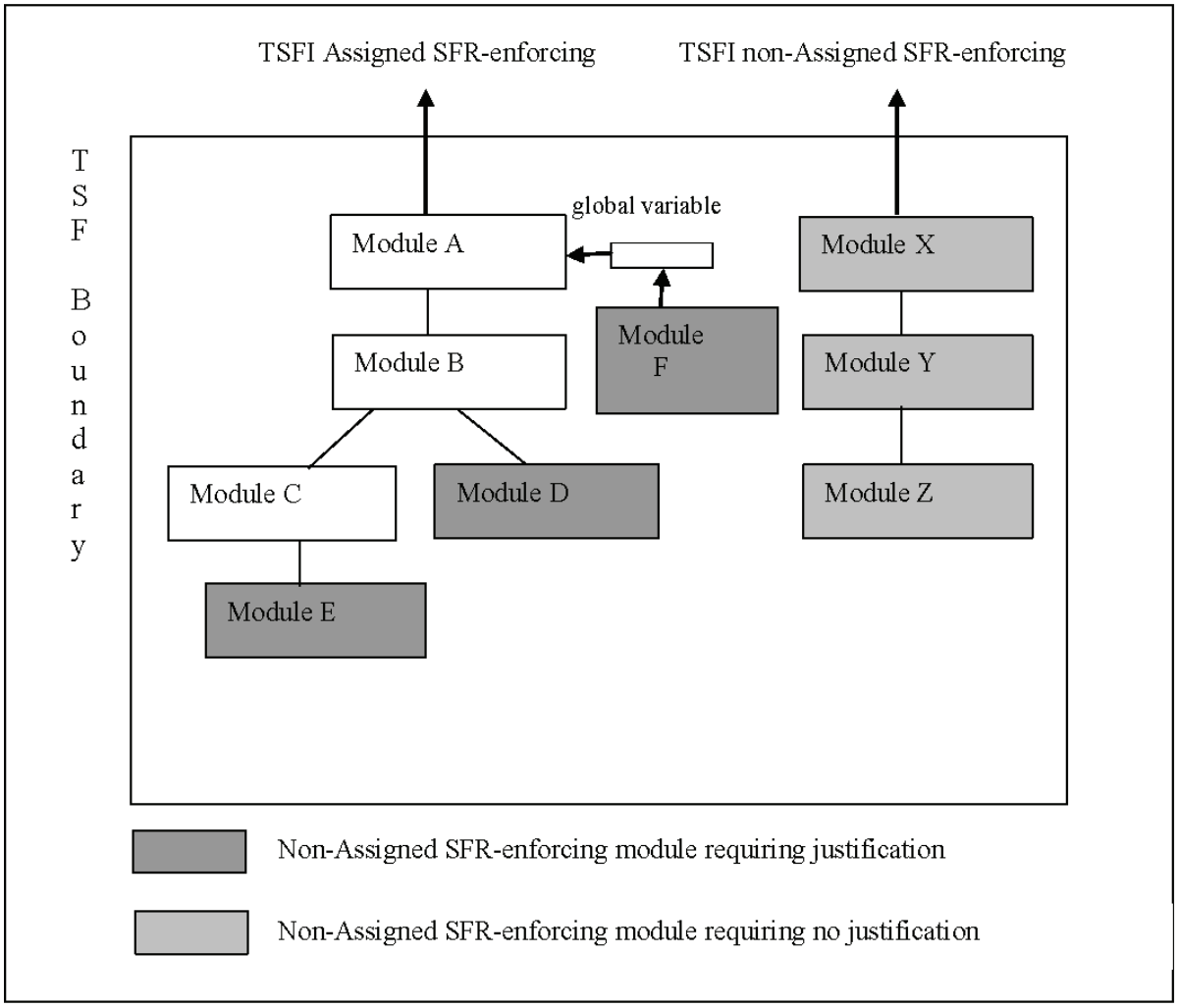

Figure 18 - Example of non-Assigned SFR-enforcing modules requiring justification

A.4 ADV_TDS: Components and Modules

683 This section provides additional guidance on the TDS family, and its use of the terms "component" 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 Components

684 Figure 19 shows that, depending on the complexity of the TSF, the design may be described in terms of components *and* modules (where components are at a higher level of abstraction than modules); or it may just be described in terms of one level of abstraction (e.g., *components* at lower assurance levels, *modules* at higher levels). In cases where a lower level of abstraction (modules) presented, requirements levied on higher-level abstractions (components) are essentially met by default. This concept is further elaborated in the discussion on components and modules below.

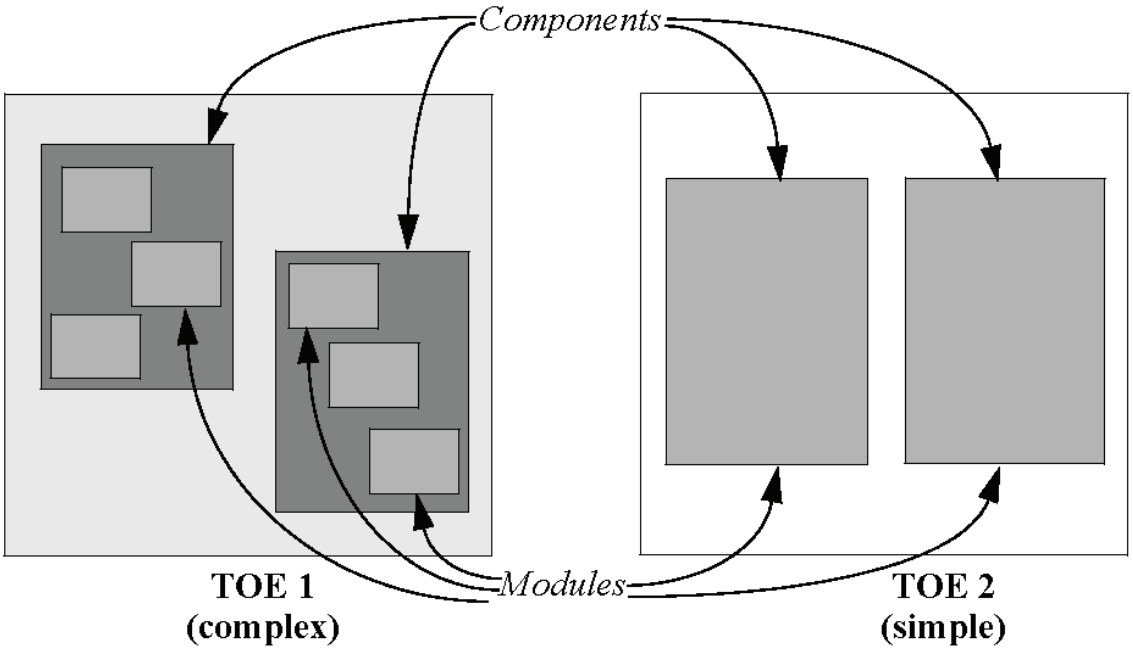

Figure 19 - Components and Modules

685 The developer is expected to describe the design of the TOE in terms of *components*. The term "component" was chosen to be specifically vague so that it could refer to units appropriate to the TOE (e.g., subsystems, modules). Components can even be uneven in scope, as long as the requirements for description of components are met.

686 The first use of components is to distinguish the TSF boundary; that is, the portions of the TOE that comprise the TSF. In general, a component 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 isolation (see A.1) where SFR-enforcing code is executed in one domain, then all components 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.

687 The second use of components 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). Components 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 component is determined by the degree to which that component is responsible for implementing an SFR.

688 An *SFR-enforcing* component is a component that provides mechanisms for enforcing an element of the TSP, or directly supports a component that is

responsible for enforcing the TSP. If a component provides (implements) an SFR-enforcing TSFI, then the component is SFR-enforcing.

689 Components can also be identified as *SFR-supporting* and *SFR-non-interfering*. An SFR-supporting component is one that is depended on by an SFR-enforcing component in order to implement an SFR, but does not play as direct a role as an SFR-supporting requirement. An SFR-non-interfering component is one that is not depended upon, in either a supporting or enforcing role, to implement an SFR.

A.4.2 Modules

690 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 components, modules describe the implementation in a level of detail that can serve as a guide to reviewing the implementation representation.

691 It is important to note that, depending on the TOE, modules and components 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 component 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 component 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 component descriptions if module descriptions exist. In TOEs that are sufficiently simple, a separate "component description" is not necessary; the requirements can be met through documentation provided by modules. For complex TOEs, the purpose of the component 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 19.

692 An SFR-enforcing module is a module that directly implements a security functional requirement (SFR) in the ST. Such modules will typically 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 components, 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.

693 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.

694 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 and used by the module, and 2) algorithmically identical to 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).

695 In the design, modules are described in detail in terms of the function they provide (the purpose); the interfaces they present; the return values from such interfaces; the interfaces (presented by other modules) they use; and an algorithmic description of how they provide their functionality.

696 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.

697 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 as to whether it is read or written (or both) by the module.

698 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.

699 By contrast, interfaces used by a module must be identified such that it can be determined which module is being invoked by the module being described. It must also be clear from the design description the algorithmic reason the invoking module is being called. For example, if Module A is being described, and it uses Module B's bubble sort routine, an inadequate algorithmic description would be “Module A invokes the *double_bubble()* interface in Module B to perform a bubble sort”. An adequate algorithmic description would be “Module A invokes the *double_bubble* routine with the list of access control entries; *double_bubble()* will return the entries sorted first on the username, then on the access_allowed field according the following rules...”. The detailed description of a module in the design must provide enough detail so that it is clear what effects Module A is expecting from the bubble sort interface. Note that one method of presenting these called interfaces is via a call tree, and then the algorithmic description can be included in the algorithmic description of the called module.

700 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.

701 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 a 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.

702 In the elements below, the labels (SFR-enforcing, SFR-supporting, and SFR-non-interfering) discussed for components 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 components and modules 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 artifacts, such as detailed interface and design documentation) to provide the necessary evidence without undue cost.

A.4.3 Levelling Approach

703 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 components 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.

704 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.

705 Table 14 summarises the information required at each of the family components for the architectural entities to be described.

	TSF Component			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.1 Basic design (informal presentation)	architecture, high-level description of SFR-Enf. behaviour, interactions	designation support[1]	designation support			
ADV_TDS.2 Architectural design (informal presentation)	architecture, detailed description of SFR-Enf. behaviour, high-level description of other behaviour interactions	architecture, high-level description of behaviour, interactions	designation support, interactions			
ADV_TDS.3 Basic modular design (informal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces[2], algorithmic[3]	interaction, purpose	interaction, purpose
ADV_TDS.4 Semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	interaction, purpose
ADV_TDS.5 Complete semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	common data, interfaces, algorithmic
ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation (semiformal presentation; additional formal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	common data, interfaces, algorithmic

[1] *designation support* means that only documentation sufficient to support the classification of the component / module is needed.
[2] *interfaces* means that the module description contains purpose, interfaces presented, and interfaces used.
[3] *algorithmic* means an algorithmic description of the entire module is provided.

Table 14 Description Detail Levelling

B Composition (ACO) (normative)

706 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 chapter 14.

B.1 Necessity for composed TOE evaluations

707 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.

708 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 above EAL2. Therefore, to compose components with assurance above EAL2 it is necessary to reuse the evaluation evidence and results of evaluations performed for the component developer.

709 It is intended that these 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 offers and dependent component for the services required. This will require iterations of the ACO components applying all requirements to each type of component peer.

710 The criteria are also intended to be more broadly applicable, stepwise, in more complex relationships, but this may require further interpretation.

711 The composed evaluation activities will take place at the same time as the dependent component evaluation. This is due to two factors:

a) 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 certification) with the understanding that the dependent component is undergoing certification 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, the testing of the dependent component using the base component in the environment is to form (one of) the test configurations during the dependent component evaluation, leaving ACO_TBT to consider the base component in this configuration.

b) 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.

712 In addition to the evaluation evidence for the dependent component, evaluation material from the base component evaluations 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.

713 The certification of the base component and dependent component are assumed to be complete by the time final verdicts are assigned for the ACO components.

B.2 Performing Security Target evaluation for a composed TOE

714 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.

715 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.

716 The composed TOE ST may reference 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.

717 During the conduct of the ASE evaluation activities for a composed TOE ST the evaluator determines that the component STs are accurately represented in the composed TOE ST. Also, the evaluator will need to determine that the dependencies of the dependent component on the IT environment are adequately fulfilled in the composed TOE.

718 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.

719 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 each security function described.

720 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.

721 Reuse of evaluation results from the evaluation of component STs can be made in the instances that the composed TOE ST directly refers out to the component STs. e.g. if the composed TOE ST refers out 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

722 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 TSP. This will include all parts of the base component required to implement the base component SFRs.

723 The TSFI of this base component represents 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 to external IT entities. However, the TSFI only includes those interfaces provided 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. It does not include calls made out by the TSF. 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 CC SFRs are not being claimed in the base component's ST (e.g. the login interface when no FIA: Identification, Authentication and Binding SFRs are claimed).

724 The functional interfaces provided by the base component are in addition to the security interfaces (TSFI), and are not required to be considered during the base component evaluation. These often include interfaces that are used by a dependent component to make calls to various services provided by the base component.

725 In summary, the TSFI only includes incoming interfaces; calls made into the TSF by external entities. Calls made by the TSF, outgoing interfaces, are not explicitly considered during the base component evaluation. This abstraction of the base component and the interfaces is shown in Figure 20 below.

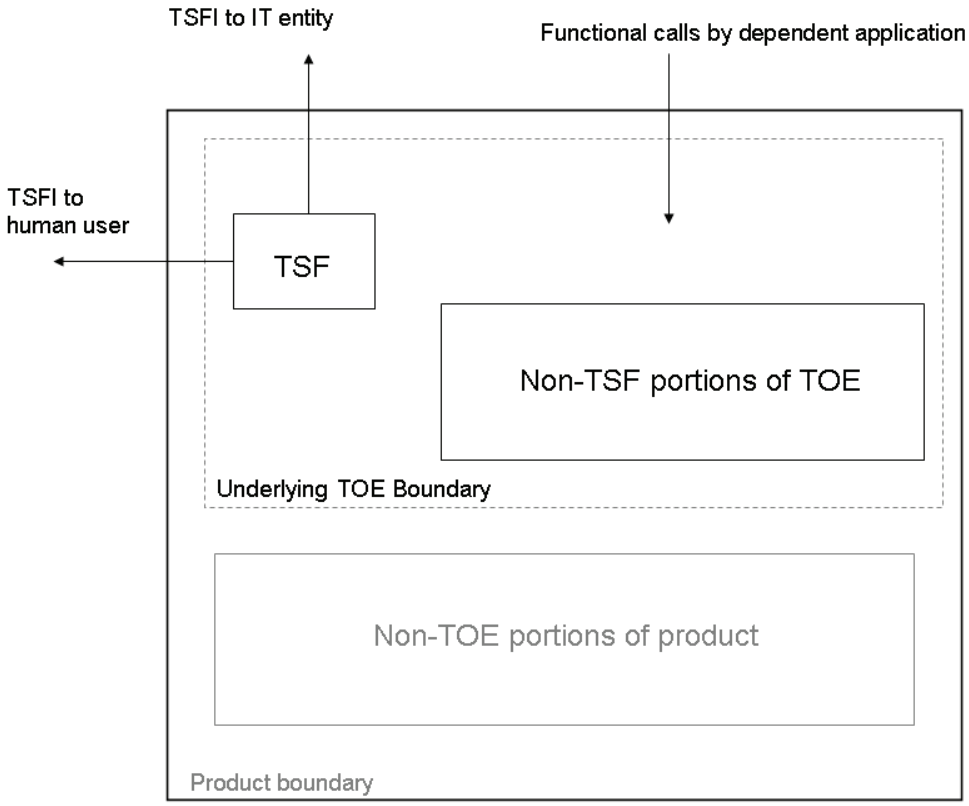

Figure 20 - Base component abstraction

726 The dependent component, which relies on the base component, is similarly defined: its SFR-related interfaces constitute its TSFI, while its non-SFR-related interfaces go un-investigated. (Note that in the hierarchical composed TOE scenario the dependent component is also installed on the base component.) However, a dependent component will have dependencies on the base environment, which in composition is provided by a base

component. These dependencies are expressed in the ST as security objectives for the environment.

727 Therefore, 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 SFRs. Such a service is outside the dependent component boundary and, hence, will not be analysed as part of the Functional specification (ADV_FSP) activities.

728 This abstraction of the dependent component and the interfaces is shown in Figure 21 below.

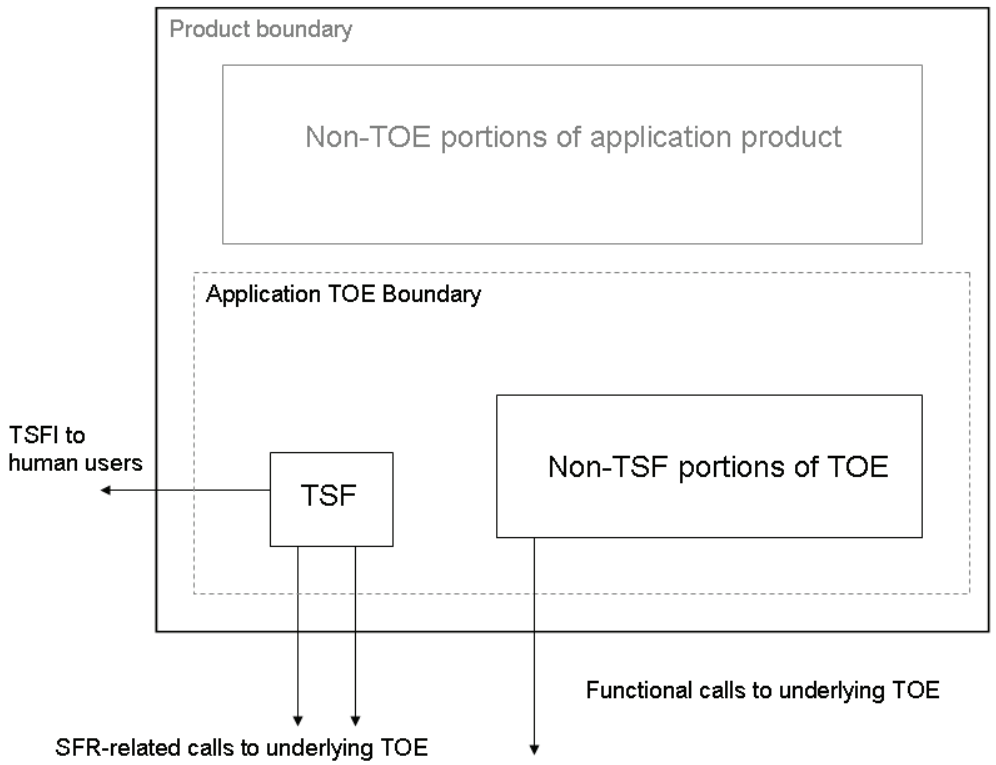

Figure 21 - Dependent component abstraction

729 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 is a TSFI of the base component and will therefore already be defined within the functional specification of the base component.

730 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 22), 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 IT environment are considered in the family Reliance of dependent component (ACO_REL).

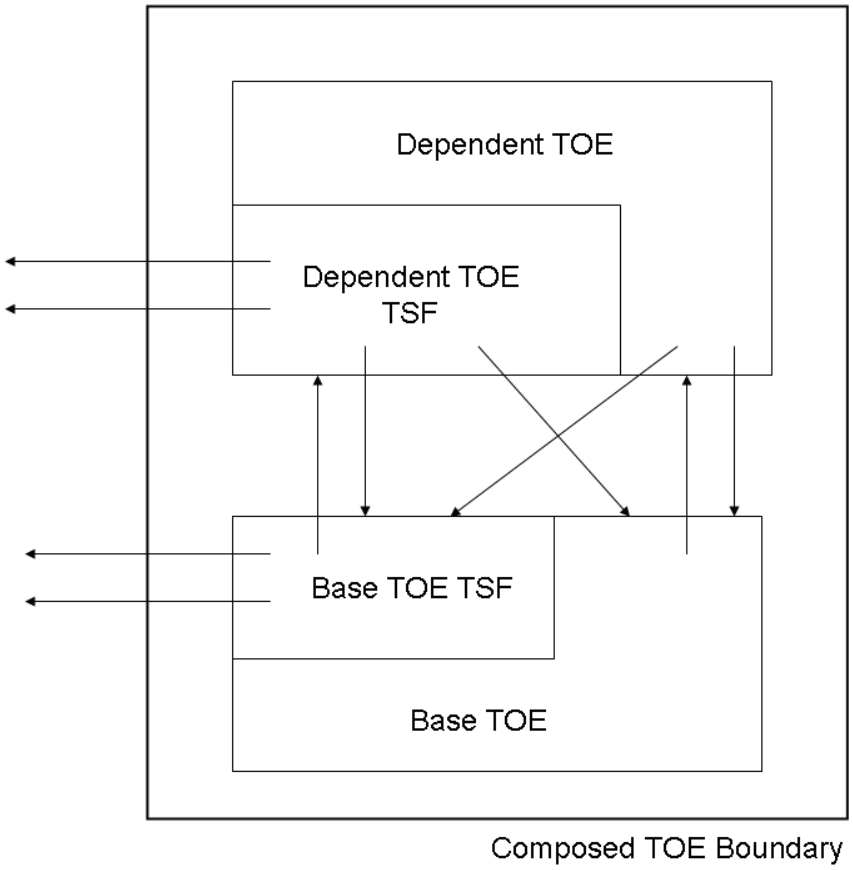

Figure 22 - Composed TOE abstraction

731 The non-TSF portion of the base component (solid grey in Figure 21) is drawn into the TSF of the composed TOE due to the dependencies the dependent component has on the base component to support the TSP 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.

732 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.

733 The possible interfaces are further described in the following diagram (Figure 23) and supporting text.

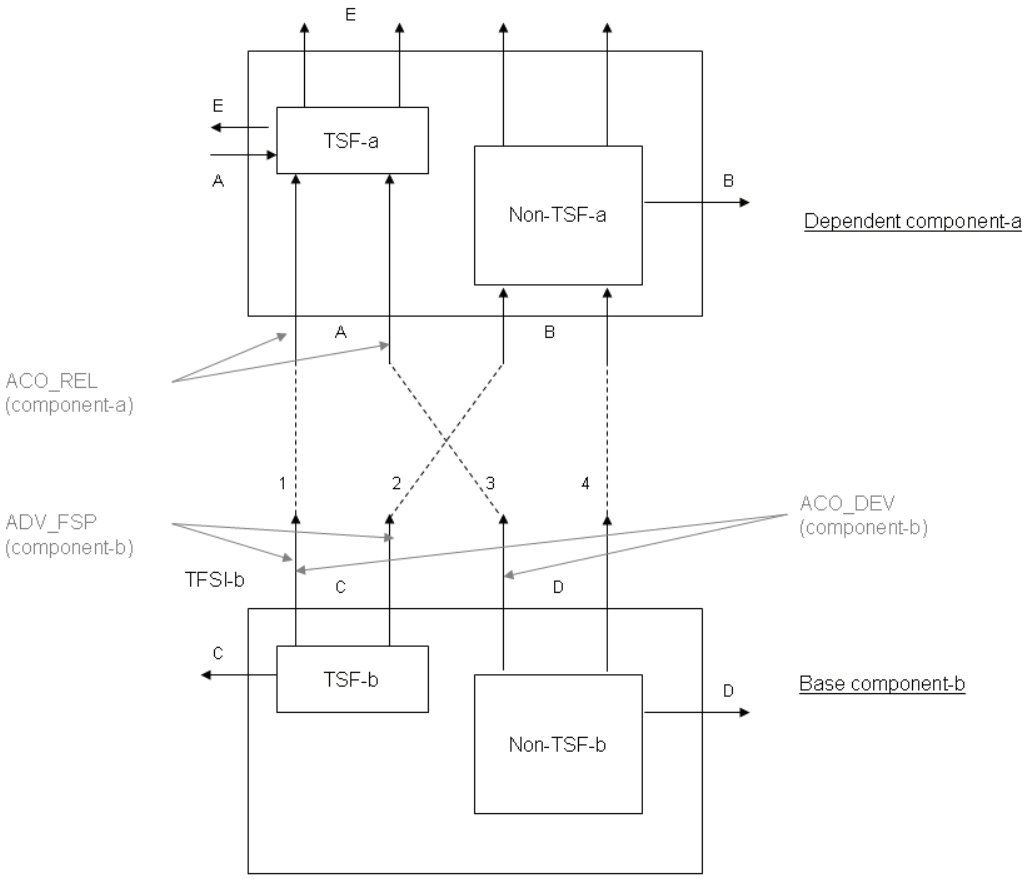

Figure 23 - Composed component interfaces

a) Arrows going *into* component-a (A and B) = calls out from dependent component to the environment;

b) Arrows coming *out* of component-b (C and D) = interfaces to services provided to things that call them;

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.

734 The following is a simplification, but explains the considerations that need to be made.

735 There are components a and b: the arrows coming *out* of TSF-a are services provided by TSF-a and are therefore TSFI(a); likewise, the arrows coming *out* of TSF-b ("C") are TSFI(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".

736 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-a and 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).

737 Note: if the kind of interaction described in case 3 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.

738 These interfaces 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.

C Cross reference of assurance component dependencies

(informative)

739 The dependencies documented in the components of chapters 10 and 14-20 are the direct dependencies between the assurance components.

740 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.

	ACO_DEV.1	ACO_DEV.2	ACO_DEV.3	ACO_REL.1	ACO_REL.2	ADV_FSP.2	ADV_FSP.4	ADV_IMP.1	ADV_TDS.1	ADV_TDS.3	ALC_CMC.1	ALC_CMS.1	ALC_TAT.1
ACO_COR.1	X			X		-			-		X	-	
ACO_DEV.1				X		-			-				
ACO_DEV.2				X		-			-				
ACO_DEV.3					X	-			-				
ACO_REL.1						X			X				
ACO_REL.2						X			X				
ACO_REL.3							X	X	-	X			-
ACO_TBT.1				X		-			-				
ACO_VUL.1	X			-		-			-				
ACO_VUL.2		X		-		-			-				
ACO_VUL.3			X		-	-			-				

Table 15 Dependency table for Class ACO: Composition

	ADV_FSP.1	ADV_FSP.4	ADV_IMP.1	ADV_TDS.1	ADV_TDS.3	ALC_CMC.5	ALC_CMS.1	ALC_DVS.2	ALC_LCD.1	ALC_TAT.1
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						
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_INT.4			X		X					X
ADV_SPM.1		X		-						
ADV_TDS.1										
ADV_TDS.2										
ADV_TDS.3										
ADV_TDS.4										
ADV_TDS.5										
ADV_TDS.6										

Table 16 Dependency table for Class ADV: Development

	ADV_FSP.1
AGD_OPE.1	X
AGD_PRE.1	

Table 17 Dependency table for Class AGD: Guidance documents

	ADV_IMP.1	ADV_TDS.3	ALC_CMS.1	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			
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_LCD.3							
ALC_TAT.1	X	-					-
ALC_TAT.2	X	-					-
ALC_TAT.3	X	-					-

Table 18 Dependency table for Class ALC: Life-cycle support

	APE_ECD.1	APE_INT.1	APE_OBJ.1	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 19 Dependency table for Class APE: Protection Profile evaluation

	ASE_ECD.1	ASE_INT.1	ASE_OBJ.1	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	

Table 20 Dependency table for Class ASE: Security Target evaluation

	ADV_FSP.1	ADV_FSP.2	ADV_FSP.4	ADV_IMP.1	ADV_IMP.2	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	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
ATE_DPT.2		-				-	X								-	X
ATE_DPT.3		-		-	X	-	X			-	-	-	-	-	-	X
ATE_FUN.1		-				-									X	-
ATE_FUN.2		-				-									X	-
ATE_IND.1	X							X	X							
ATE_IND.2	-	X				-		X	X						-	X
ATE_IND.3	-	-	X			-		X	X						-	X

Table 21 Dependency table for Class ATE: Tests

	ADV_ARC.1	ADV_FSP.1	ADV_FSP.2	ADV_IMP.1	ADV_TDS.1	ADV_TDS.3	AGD_OPE.1	AGD_PRE.1	ALC_TAT.1
AVA_VAN.1		X					X	X	
AVA_VAN.2	X	X			X		X	X	
AVA_VAN.3	X	-	X	X	-	X	X	X	-
AVA_VAN.4	X	-	X	X	-	X	X	X	-
AVA_VAN.5	X	-	X	X	-	X	X	X	-

Table 22 Dependency table for Class AVA: Vulnerability assessment

D Cross reference of PPs and assurance components

(normative)

741 Table 23 describes the relationship between PPs and the families and components of the APE class.

Assurance class	Assurance family	Assurance component	
		Low Assurance PP	PP
Protection Profile evaluation	APE_CCL	**1**	1
	APE_ECD	**1**	1
	APE_INT	**1**	1
	APE_OBJ	**1**	**2**
	APE_REQ	**1**	**2**
	APE_SPD		**1**

Table 23 PP assurance level summary

E Cross reference of EALs and assurance components

(normative)

742 Table 24 describes the relationship between the evaluation assurance levels and the assurance classes, families and components.

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		**1**	1	1	1	1	1
	ADV_FSP	**1**	**2**	**3**	**4**	**5**	5	**6**
	ADV_IMP				**1**	1	**2**	2
	ADV_INT					**2**	**3**	**4**
	ADV_SPM						**1**	1
	ADV_TDS		**1**	**2**	**3**	**4**	**5**	**6**
Guidance documents	AGD_OPE	**1**	1	1	1	1	1	1
	AGD_PRE	**1**	1	1	1	1	1	1
Life-cycle support	ALC_CMC	**1**	**2**	**3**	**4**	4	**5**	5
	ALC_CMS	**1**	**2**	**3**	**4**	**5**	5	5
	ALC_DEL		**1**	1	1	1	1	1
	ALC_DVS			**1**	1	1	**2**	2
	ALC_FLR							
	ALC_LCD				**1**	**2**	2	**3**
	ALC_TAT				**1**	**2**	**3**	3
Security Target evaluation	ASE_CCL	**1**	1	1	1	1	1	1
	ASE_ECD	**1**	1	1	1	1	1	1
	ASE_INT	**1**	1	1	1	1	1	1
	ASE_OBJ	**1**	**2**	2	2	2	2	2
	ASE_REQ	**1**	**2**	2	2	2	2	2
	ASE_SPD		**1**	1	1	1	1	1
	ASE_TSS	**1**	1	1	1	1	1	1
Tests	ATE_COV		**1**	**2**	2	2	**3**	3
	ATE_DPT			**1**	1	**2**	2	**3**
	ATE_FUN		**1**	1	1	1	**2**	2
	ATE_IND	**1**	**2**	2	2	2	2	**3**
Vulnerability assessment	AVA_VAN	**1**	**2**	2	**3**	**4**	**5**	5

Table 24 Evaluation assurance level summary

F Cross reference of CAPs and assurance components

(normative)

743 Table 25 describes the relationship between the composition assurance levels and the assurance classes, families and components.

Assurance class	Assurance Family	Assurance Components by Composition Assurance Level		
		CAP-A	CAP-B	CAP-C
Composition	ACO_COR	**1**	1	1
	ACO_DEV	**1**	**2**	**3**
	ACO_REL	**1**	**2**	**3**
	ACO_TBT	**1**	1	1
	ACO_VUL	**1**	**2**	**3**
Life-cycle support	ALC_CMC	**1**	1	1
	ALC_CMS	**2**	2	2
	ALC_DEL			
	ALC_DVS			
	ALC_FLR			
	ALC_LCD			
	ALC_TAT			
Security Target evaluation	ASE_CCL	**1**	1	1
	ASE_ECD	**1**	1	1
	ASE_INT	**1**	1	1
	ASE_OBJ	**1**	**2**	2
	ASE_REQ	**1**	**2**	2
	ASE_SPD		**1**	1
	ASE_TSS	**1**	1	1

Table 25 Composition assurance level summary