

Methodology for Components above EAL4

Currently, the CEM provides methodology only for those components that are contained within EALs 1-4. While use of components contained within EALs 5 and higher is allowed, there is currently no mutually-recognized methodology for them. Therefore, CCEVS is providing high-level interim guidance for the use of these components as they are used within evaluations until a more structured methodology (e.g. with CEM-like workunits) is available.

In addition, some of these requirements will be addressed - in whole or in part - by the NSA. Any actions to be performed by the CCTL are likewise identified on this page.

For evaluations including these components, validators are asked to pay particular attention in these areas, and keep CCEVS apprised if anything even remotely questionable comes up, so that we can clarify the interim guidance if necessary. As other components above EAL4 are included in evaluations, causing other such interim guidance to be generated, these will be collected together in this list and made available in an effort to maintain consistency.

Components beyond EAL1-4

ACM_AUT.2	ADV_IMP.2	ADV_SPM.3	ATE_FUN.2
ACM_CAP.5	ADV_IMP.3	ALC_DVS.2	ATE_IND.3
ACM_SCP.3	ADV_INT.1	ALC_FLR.*	AVA_CCA.1
ADO_DEL.3	ADV_INT.2	ALC_LCD.2	AVA_CCA.2
ADV_FSP.3	ADV_INT.3	ALC_LCD.3	AVA_CCA.3
ADV_FSP.4	ADV_LLD.2	ALC_TAT.2	AVA_MSU.3
ADV_HLD.3	ADV_RCR.2	ALC_TAT.3	AVA_VLA.3
ADV_HLD.4	ADV_RCR.3	ATE_COV.3	AVA_VLA.4
ADV_HLD.5	ADV_SPM.2	ATE_DPT.2	AMA_*
		ATE_DPT.3	

ACM

ACM_AUT.2 (*Performed by CCTL*)

(From *Configuration Management Evaluation Guidance for High Robustness Systems*, by Michael E. Gross, NAVAL POSTGRADUATE SCHOOL, Monterey, California, March 2004)

ACM_AUT.2.1C

- The evaluator shall check the CM plan for a description of the automated measures to control access to the TOE implementation representation.
- The evaluator shall examine the automated access control measures to determine that they are effective in preventing unauthorised modification of the TOE implementation representation.
- The evaluator shall check the CM plan for a description of the automated measures to control access to the configuration items.
- The evaluator shall examine the automated access control measures to determine that they are effective in preventing unauthorized modification of configuration items.

- The evaluator reviews the configuration management documentation to identify those individuals or roles authorized to make changes to the TOE implementation representation. For example, once it is under configuration management, access to an element of the implementation representation may only be allowed for the individual who performs the software integration role. Similarly, the evaluator reviews the CM documentation to identify those individuals or roles authorized to access a configuration item.
- The evaluator should exercise the automated access control measures to determine whether they can be bypassed by an unauthorised role or user. This determination need only comprise a few basic tests. Tests should cover the TOE implementation representation and configuration items.

ACM_AUT.2.2C

- The evaluator shall check the CM documentation for automated means to support generation of the TOE from its implementation representation.
 - In this work unit the term generation applies to those processes adopted by the developer to progress the TOE from its implementation to a state ready to be delivered to the end customer.
 - The evaluator should verify the existence of automated generation support procedures within the CM documentation.
- The evaluator shall examine the automated generation procedures to determine that they can be used to support generation of the TOE.
 - The evaluator determines that by following the generation procedures a TOE would be generated that reflects its implementation representation. The customer can then be confident that the version of the TOE delivered for installation implements the TSP as described in the ST. For example, in a software TOE this may include checking that the automated generation procedures help to ensure that all source files and related libraries that are relied upon to enforce the TSP are included in the compiled object code.
 - It should be noted that this requirement is only to provide support. For example, an approach that placed Unix makefiles under configuration management should be sufficient to meet the aim, given that in such a case automation would have made a significant contribution to accurate generation of the TOE. Automated procedures can assist in identifying the correct configuration items to be used in generating the TOE.

ACM_AUT.2.3C

- The evaluator shall check that the CM plan includes information on the automated tools used in the CM system.

ACM_AUT.2.4C

- The evaluator shall examine the information relating to the automated tools provided in the CM plan to determine that it describes how they are used.
 - The information provided in the CM plan provides the necessary detail for a user of the CM system to be able to operate the automated tools correctly in order to maintain the integrity of the TOE. For example, the information provided may include a description of:
 - the functionality provided by the tools;
 - how this functionality is used by the developer to control changes to the implementation representation;
 - how this functionality is used by the developer to support generation of the TOE.

- The evaluator shall examine the CM system to determine that the automated tools and procedures described in the CM plan are used.
 - This work unit may be viewed as an additional activity to be carried out in parallel with the evaluator's examination into the use of the CM system required by ACM_CAP. The evaluator looks for evidence that the tools and procedures are in use. This should include a visit to the development site to witness operation of the tools and procedures, and an examination of evidence produced through their use.
 - For guidance on site visits see Annex B.5.

ACM_AUT.2.5C

- The evaluator shall check the CM documentation for a description of the automated means by which changes can be ascertained between the TOE and its preceding version.
- The evaluator shall examine the CM system to ensure it has the capability to provide an automated means to ascertain the changes between the TOE and its preceding version.
 - The evaluator should verify the existence of the automated means to ascertain changes procedures within the CM documentation.
 - Previous TOE versions may or may not exist. Regardless, evaluators should exercise the automated mechanisms within the CM system or verify the capability exists.

ACM_AUT.2.6C

- The evaluator shall check that the CM documentation includes information on the automated means to identify all other configuration items that are affected by the modification of a given configuration item.
- The evaluator shall examine the automated mechanisms used by the CM system to identify all other configuration items that are affected by the modification of a given configuration item.
 - The CM documentation should state how the automated means will identify the changes to configuration items that are affected by the modification of a given configuration item. CM systems may differ on how this information is presented.

ACM_CAP.5 *(Performed by CCTL)*

(From *Configuration Management Evaluation Guidance for High Robustness Systems*, by Michael E. Gross, NAVAL POSTGRADUATE SCHOOL, Monterey, California, March 2004)

ACM_CAP.5.1C

- The evaluator shall check that the version of the TOE provided for evaluation is uniquely referenced.
 - The evaluator should use the developer's CM system to validate the uniqueness of the reference by checking the configuration list to ensure that the configuration items are uniquely identified. Evidence that the version provided for evaluation is uniquely referenced may be incomplete if only one version is examined during the evaluation, and the evaluator should look for a referencing system that is capable of supporting unique references (e.g. use of numbers, letters or dates). However, the absence of any reference will normally lead to a fail verdict against this requirement unless the evaluator is confident that the TOE can be uniquely identified.
 - The evaluator should seek to examine more than one version of the TOE (e.g. during rework following discovery of a vulnerability), to check that the two versions are referenced differently.

ACM_CAP.5.2C

- The evaluator shall check that the TOE provided for evaluation is labelled with its reference.
 - The evaluator should ensure that the TOE contains a unique reference such that it is possible to distinguish different versions of the TOE. This could be achieved through labelled packaging or media, or by a label displayed by the operational TOE. This is to ensure that it would be possible for consumers to identify the TOE (e.g. at the point of purchase or use).
 - The TOE may provide a method by which it can be easily identified. For example, a software TOE may display its name and version number during the start up routine, or in response to a command line entry. A hardware or firmware TOE may be identified by a part number physically stamped on the TOE.
- The evaluator shall check that the TOE references used are consistent.
 - If the TOE is labelled more than once then the labels have to be consistent. For example, it should be possible to relate any labelled guidance documentation supplied as part of the TOE to the evaluated operational TOE. This ensures that consumers can be confident that they have purchased the evaluated version of the TOE, that they have installed this version, and that they have the correct version of the guidance to operate the TOE in accordance with its ST. The evaluator can use the configuration list that is part of the provided CM documentation to verify the consistent use of identifiers.
 - The evaluator also verifies that the TOE reference is consistent with the ST.
 - For guidance on consistency analysis see Annex B.3.

ACM_CAP.5.3C

- The evaluator shall check that the CM documentation provided includes a configuration list.
 - A configuration list identifies the items being maintained under configuration control.
- The evaluator shall check that the CM documentation provided includes a CM plan.
- The evaluator shall check that the CM documentation provided includes an acceptance plan.
- The evaluator shall check that the CM documentation provided includes integration procedures.

ACM_CAP.5.4C

- The evaluator shall examine the configuration list to determine that it identifies the configuration items that comprise the TOE.
 - The minimum scope of configuration items to be covered in the configuration list is given by ACM_SCP.

ACM_CAP.5.5C

- The evaluator shall examine the method of identifying configuration items to determine that it describes how configuration items are uniquely identified.

ACM_CAP.5.6C

- The evaluator shall check that the configuration list uniquely identifies each configuration item.
 - The configuration list contains a list of the configuration items that comprise the TOE, together with sufficient information to uniquely identify which version of each item has been used (typically a version number). Use of this list will enable the evaluator to check that the correct configuration items, and the correct version of each item, have been used during the evaluation.

ACM_CAP.5.7C

- The evaluator shall examine the CM plan to determine that it describes how the CM system is used to maintain the integrity of the TOE configuration items.
 - The descriptions contained in a CM plan may include:
 - all activities performed in the TOE development environment that are subject to configuration management procedures (e.g. creation, modification or deletion of a configuration item);
 - the roles and responsibilities of individuals required to perform operations on individual configuration items (different roles may be identified for different types of configuration item (e.g. design documentation or source code));
 - the procedures that are used to ensure that only authorized individuals can make changes to configuration items;
 - the procedures that are used to ensure that concurrency problems do not occur as a result of simultaneous changes to configuration items;
 - the evidence that is generated as a result of application of the procedures. For example, for a change to a configuration item, the CM system might record a description of the change, accountability for the change, identification of all configuration items affected, status (e.g. pending or completed), and date and time of the change. This might be recorded in an audit trail of changes made or change control records;
 - the approach to version control and unique referencing of TOE versions (e.g. covering the release of patches in operating systems, and the subsequent detection of their application).

ACM_CAP.5.8C

- The evaluator shall check the CM documentation to ascertain that it includes the CM system records identified by the CM plan.
 - The output produced by the CM system should provide the evidence that the evaluator needs to be confident that the CM plan is being applied, and also that all configuration items are being maintained by the CM system as required by ACM_CAP.4.9C. Example output could include change control forms, or configuration item access approval forms.
- The evaluator shall examine the evidence to determine that the CM system is being used as it is described in the CM plan.
 - The evaluator should select and examine a sample of evidence covering each type of CM-relevant operation that has been performed on a configuration item (e.g. creation, modification, deletion, reversion to an earlier version) to confirm that all operations of the CM system have been carried out in line with documented procedures. The evaluator confirms that the evidence includes all the information identified for that operation in the CM plan. Examination of the evidence may require access to a CM tool that is used. The evaluator may choose to sample the evidence.
 - For guidance on sampling see Annex B.2.
 - Further confidence in the correct operation of the CM system and the effective maintenance of configuration items may be established by means of interview with selected development staff. In conducting such interviews, the evaluator should aim to gain a deeper understanding of how the CM system is used in practice as well as to confirm that the CM procedures are being applied as described in the CM documentation. Note that such interviews should complement rather than replace the examination of documentary evidence, and may not be necessary if the documentary evidence alone satisfies the requirement. However, given the wide scope of the CM plan it is possible that some aspects (e.g. roles and responsibilities) may not be clear from the CM plan and records alone. This is one case where clarification may be necessary through interviews.
 - It is expected that the evaluator will visit the development site in support of this activity.
 - For guidance on site visits see Annex B.5.

ACM_CAP.5.9C

- The evaluator shall check that the configuration items identified in the configuration list are being maintained by the CM system.
 - The CM system employed by the developer should maintain the integrity of the TOE. The evaluator should check that for each type of configuration item (e.g. high-level design or source code modules) contained in the configuration list there are examples of the evidence generated by the procedures described in the CM plan. In this case, the approach to sampling will depend upon the level of granularity used in the CM system to control CM items. Where, for example, 10,000 source code modules are identified in the configuration list, a different sampling strategy should be applied compared to the case in which there are only 5, or even 1. The emphasis of this activity should be on ensuring that the CM system is being operated correctly, rather than on the detection of any minor error.
 - For guidance on sampling see Annex B.2.

ACM_CAP.5.10C

- The evaluator shall examine the CM access control measures described in the CM plan to determine that they are effective in preventing unauthorised access to the configuration items.
 - The evaluator may use a number of methods to determine that the CM access control measures are effective. For example, the evaluator may exercise the access control measures to ensure that the procedures could not be bypassed. The evaluator may use the outputs generated by the CM system procedures and already examined as part of the work unit 4:ACM_CAP.4-13. The evaluator may also witness a demonstration of the CM system to ensure that the access control measures employed are operating effectively.
 - The developer will have provided automated access control measures as part of the CM system and as such their suitability may be verified under the component ACM_AUT.1

ACM_CAP.5.11C

- The evaluator shall check the CM documentation for procedures for supporting the generation of the TOE.
 - In this work unit the term generation applies to those processes adopted by the developer to progress the TOE from implementation to a state acceptable for delivery to the end customer.
 - The evaluator verifies the existence of generation support procedures within the CM documentation. The generation support procedures provided by the developer may be automated, and as such their existence may be verified under the component ACM_AUT.1.2C.
- The evaluator shall examine the TOE generation procedures to determine that they are effective in helping to ensure that the correct configuration items are used to generate the TOE.
 - The evaluator determines that by following the generation support procedures the version of the TOE expected by the customer (i.e. as described in the TOE ST and consisting of the correct configuration items) would be generated and delivered for installation at the customer site. For example, in a software TOE this may include checking that the procedures ensure that all source files and related libraries are included in the compiled object code.
 - The evaluator should bear in mind that the CM system need not necessarily possess the capability to generate the TOE, but should provide support for the process that will help reduce the probability of human error.

ACM_CAP.5.12C

- The evaluator shall examine the acceptance procedures to determine that they describe the acceptance criteria to be applied to newly created or modified configuration items.
 - An acceptance plan describes the procedures that are to be used to ensure that the constituent parts of the TOE are of adequate quality prior to incorporation into the TOE. The acceptance plan should identify the acceptance procedures to be applied:
 - at each stage of the construction of the TOE (e.g. module, integration, system);
 - to the acceptance of software, firmware and hardware components;
 - to the acceptance of previously evaluated components.
 - The description of the acceptance criteria may include identification of:
 - developer roles or individuals responsible for accepting such configuration items;
 - any acceptance criteria to be applied before the configuration items are accepted (e.g. successful document review, or successful testing in the case of software, firmware or hardware).

ACM_CAP.5.13C

- The evaluator shall check that the integration procedures describe how the CM system is applied in the TOE manufacturing process.

ACM_CAP.5.14C

- The evaluator shall check that the CM documentation describes configuration item acceptance procedures.
- The evaluator shall verify configuration item acceptance procedures in the CM system perform as described by the CM documentation.
 - The CM system should prevent the person who developed the configuration item from being able to accept it.

ACM_CAP.5.15C

- The evaluator shall verify that the CM documentation clearly identifies all configuration items that comprise the TSF.
- The evaluator shall verify that the configuration items that comprise the TSF are clearly identified in the CM system.

ACM_CAP.5.16C

- The evaluator shall verify the CM documentation describes the CM systems audit capabilities.
 - The CM system at a minimum will track all TOE modifications to include: the originator, date, and time in the audit trail.
- The evaluator shall verify the CM system's TOE audit capabilities.
 - The audit portion of the CM system shall provide at a minimum the originator, date and time in the audit trail.

ACM_CAP.5.17C

- The evaluator shall verify that the CM documentation describes how the CM system will identify the master copy of all material used to generate the TOE.
- The evaluator shall verify that the CM system identifies the master copy of all material used to generate the TOE.

ACM_CAP.5.18C

- The evaluator shall verify that the CM documentation describes how the use of the CM system and development security measures will only allow authorized changes to be made to the TOE.

ACM_CAP.5.19C

- The evaluator shall verify that the CM documentation describes how the use of the integration procedures will ensure that the generation of the TOE is performed correctly in the authorized manner.

ACM_CAP.5.20C

- The evaluator shall verify that the CM documentation sufficiently describes how the CM system will ensure that the person responsible for accepting a configuration item into the CM system is not the person who developed it.

ACM_CAP.5.21C

- The evaluator shall verify the CM documentation provides adequate justification that the acceptance procedures provide adequate and appropriate review for all changes made to configuration items.

ALC_DVS Dependency

- The evaluator shall confirm that CM-specific security measures from the TSF are implemented and documented as necessary in the CM documentation.

ACM_SCP.3 *(Performed by CCTL)*

(From *Configuration Management Evaluation Guidance for High Robustness Systems*, by Michael E. Gross, NAVAL POSTGRADUATE SCHOOL, Monterey, California, March 2004)

ACM_SCP.3.1C

- The evaluator shall check that the configuration list includes the minimum set of items required by the CC to be tracked by the CM system.
 - The list should include the following as a minimum:
 - all documentation required to meet the target level of assurance;
 - test software (if applicable);
 - the TOE implementation representation (i.e. the components or subsystems that compose the TOE). For a software-only TOE, the implementation representation may consist solely of source code; for a TOE that includes a hardware platform, the implementation representation may refer to a combination of software, firmware and a description of the hardware (or a reference platform).
 - the documentation used to record details of reported security flaws associated with the implementation (e.g. problem status reports derived from a developer's problem reporting database).

- development tools (e.g. programming languages and compilers) and related documentation (e.g. Information pertaining to TOE generation items (such as compiler options, installation/generation options, and build options).
- The evaluator shall confirm the level of documentation necessary for ensuring proper visibility of the development tools and related documentation throughout the entire CM documentation.
- The evaluator shall confirm the development tools and related documentation listed in the CM documentation is resident in the CM system. In performance of this action item, the evaluator does not need independent access to determine the sufficiency of correspondence between the system configuration list and documentation.

ACM_SCP.3.2C

- The evaluator shall examine the CM documentation to determine that the procedures describe how the status of each configuration item can be tracked throughout the lifecycle of the TOE.
 - The procedures may be detailed in the CM plan or throughout the CM documentation. The information included should describe:
 - how each configuration item is uniquely identified, such that it is possible to track versions of the same configuration item;
 - how configuration items are assigned unique identifiers and how they are entered into the CM system;
 - the method to be used to identify superseded versions of a configuration item;
 - the method to be used for identifying and tracking configuration items through each stage of the TOE development and maintenance lifecycle (i.e. requirements specification, design, source code development, through to object code generation and on to executable code, module testing, implementation and operation); method used for assigning the current status of the configuration item at a given point in time and for tracking each configuration item through the various levels of representation at the development phase (i.e. source code development, through to object code generation and on to executable code, module testing and documentation);
 - the method used for identifying and tracking flaws relative to configuration items throughout the development lifecycle;
 - the method used for identifying correspondence between configuration items such that if one configuration item is changed it can be determined which other configuration items will also need to be changed.
 - The analysis of the CM documentation for some of this information may have been satisfied by work units detailed under ACM_CAP.

ADO

ADO_DEL.3 (Performed by CCTL)

<To be written>

ADV

ADV_FSP.3 *(Performed by CCTL)*

ADV_FSP.3.1C (The functional specification shall describe the TSF and its external interfaces using a semiformal style, supported by informal, explanatory text where appropriate.)

ADV_FSP.3-1 The evaluator shall examine the functional specification to determine that it is presented in a semi-formal format, supported by informal, explanatory text where appropriate.

A semi-formal presentation is characterized by a standardized format with a well defined syntax that reduces ambiguity that may occur in informal presentations.

The intent of the semi-formal format is to enhance the reader's ability to understand the information presented. When describing the effects or exceptions of an interface, informal text will most likely be necessary.

For the purposes of this activity, the evaluator should ensure that each interface description—including its purpose and method of use statement—is formatted in a structured, consistent manner and uses common terminology. For sets of interfaces that are largely related (examples include kernel APIs, network protocols, PCI bus signals, Windows Registry entries), the format and terminology is consistent in order for the presentation to be considered semi-formal.

ADV_FSP.3.2C (The functional specification shall be internally consistent)

ADV_FSP.3-2 The evaluator shall examine the functional specification to determine that it is internally consistent.

The evaluator validates the functional specification by ensuring that the descriptions of the interfaces making up the TSFI are consistent with the descriptions of the functions of the TSF.

ADV_FSP.3.3C (The functional specification shall describe the purpose and method of use of all external TSF interfaces, providing complete details of all effects, exceptions and error messages.)

ADV_FSP.3-3 The evaluator shall examine the functional specification to determine that it identifies all of the external TOE security function interfaces.

The term external refers to that which is visible to the user. External interfaces to the TOE are either direct interfaces to the TSF or interfaces to non-TSF portions of the TOE. However, these non-TSF interfaces might have eventual access to the TSF. These external interfaces that directly or indirectly access the TSF

collectively make up the TOE security function interface (TSFI). For further guidance on the TSFI, see the EAL4 CEM methodology for ADV_FSP.2.

ADV_FSP.3-4 The evaluator shall examine the functional specification to determine that it describes all of the external TOE security function interfaces.

For a TOE that has no threat of malicious users (i.e. FPT_PHP, FPT_RVM, and FPT_SEP are rightfully excluded from its ST), the only interfaces that are described in the functional specification (and expanded upon in the other TSF representation descriptions) are those to and from the TSF. The absence of FPT_PHP, FPT_RVM, and FPT_SEP presumes there is no concern for any sort of bypassing of the security features; therefore, there is no concern with any possible impact that other interfaces might have on the TSF.

On the other hand, if the TOE has a threat of malicious users or bypass (i.e. FPT_PHP, FPT_RVM, and FPT_SEP are included in its ST), all external interfaces are described in the functional specification, but only to the extent that the effect of each is made clear: interfaces to the security functions are completely described, while other interfaces are described only to the extent that it is clear that the TSF is inaccessible through the interface. The inclusion of FPT_PHP, FPT_RVM, and FPT_SEP implies a concern that all interfaces might have some effect upon the TSF. Because each external interface is a potential TSF interface, the functional specification must contain a description of each interface in sufficient detail so that an evaluator can determine whether the interface is security relevant.

Some architectures lend themselves to readily provide this interface description in sufficient detail for groups of external interfaces. For example, a kernel architecture is such that all calls to the operating system are handled by kernel programs; any calls that might violate the TSP must be called by a program with the privilege to do so. All programs that execute with privilege must be included in the functional specification. Any program external to the kernel that executes without privilege is incapable of affecting the TSP (i.e. such programs are interfaces of type a, rather than b in Figure 8.1) and may, therefore, be excluded from the functional specification. It is worth noting that, while the evaluator's understanding of the interface description can be expedited in cases where there is a kernel architecture, such an architecture is not necessary.

ADV_FSP.3-5 The evaluator shall examine the presentation of the TSFI to determine that it adequately and correctly describes the complete behavior of the TOE at each external interface describing effects, exceptions and error messages.

In order to assess the adequacy and correctness of an interface's presentation, the evaluator uses the functional specification, the TOE summary specification of the ST, and the user and administrator guidance to assess the following factors:

- a) All security relevant user input parameters (or a characterization of those parameters) should be identified. For completeness, parameters outside of direct user control should be identified if they are usable by administrators.
- b) Complete security relevant behavior described in the reviewed guidance should be reflected in the description of semantics in the functional specification. This should include an identification of the behavior in terms of events and the effect of each event. For example, if an operating system provides a rich file system interface, where it provides a different error code for each reason why a file is not opened upon request, the functional specification should explain that a file is either opened upon request, or else that the request is denied, along with a listing of the reasons why the open request might be denied (e.g. access denied, no such file, file is in use by another user, user is not authorized to open the file after 5pm, etc.). It would be insufficient for the functional specification merely to explain that a file is either opened upon request, or else that an error code is returned. The description of the semantics should include how the security requirements apply

- to the interface (e.g. whether the use of the interface is an auditable event and, if so, the information that can be recorded).
- c) All interfaces are described for all possible modes of operation. If the TSF provides the notion of privilege, the description of the interface should explain how the interface behaves in the presence or absence of privilege.
 - d) The information contained in the descriptions of the security relevant parameters and syntax of the interface should be consistent across all documentation.

Verification of the above is done by reviewing the functional specification and the TOE summary specification of the ST, as well as the user and administrator guidance provided by the developer. For example, if the TOE were an operating system and its underlying hardware, the evaluator would look for discussions of user-accessible programs, descriptions of protocols used to direct the activities of programs, descriptions of user-accessible databases used to direct the activities of programs, and for user interfaces (e.g. commands, application program interfaces) as applicable to the TOE under evaluation; the evaluator would also ensure that the processor instruction set is described.

This review might be iterative, such that the evaluator would not discover the functional specification to be incomplete until the design, source code, or other evidence is examined and found to contain parameters or error messages that have been omitted from the functional specification.

ADV_FSP.3.4C (The functional specification shall completely represent the TSF.)

ADV_FSP.3-6 The evaluator shall examine the functional specification to determine that the TSF is fully represented.

In order to assess the completeness of the TSF representation, the evaluator consults the TOE summary specification of the ST, the user guidance, and the administrator guidance. None of these should describe security functions that are absent from the TSF presentation of the functional specification.

ADV_FSP.3.5C (The functional specification shall include rationale that the TSF is completely represented.)

ADV_FSP.3-7 The evaluator shall examine the functional specification to determine that it contains a convincing argument that the TSF is completely represented by the functional specification.

The evaluator determines that there is a convincing argument that there are no interfaces of the TSFI that are missing from the functional specification. This may include a description of the procedure or methodology that the developer used to ensure that all external interfaces are covered. The argument would prove inadequate if, for example, the evaluator discovers commands, parameters, error messages, or other interfaces to the TSF in other evaluation evidence, yet absent from the functional specification.

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.3-8 The evaluator shall examine the functional specification to determine that it is a complete instantiation of the TOE security functional requirements.

To ensure that all ST security functional requirements are covered by the functional specification, the evaluator may construct a map between the TOE summary specification and the functional specification. Such a map might be already provided by the developer as evidence for meeting the correspondence (ADV_RCR.2) requirements, in which case the evaluator need only verify the completeness of this

mapping, ensuring that all security functional requirements are mapped onto applicable TSFI presentations in the functional specification.

ADV_FSP.3-9 The evaluator shall examine the functional specification to determine that it is an accurate instantiation of the TOE security functional requirements.

For each interface to a security function with specific characteristics, the detailed information in the functional specification must be exactly as it is specified in the ST. For example, if the ST contains user authentication requirements that the password length must be eight characters, the TOE must have eight-character passwords; if the functional specification describes six-character fixed length passwords, the functional specification would not be an accurate instantiation of the requirements.

For each interface in the functional specification that operates on a controlled resource, the evaluator determines whether it returns an error code that indicates a possible failure due to enforcement of one of the security requirements; if no error code is returned, the evaluator determines whether an error code should be returned. For example, an operating system might present an interface to OPEN a controlled object. The description of this interface may include an error code that indicates that access was not authorized to the object. If such an error code does not exist, the evaluator should confirm that this is appropriate (because, perhaps, access mediation is performed on READs and WRITEs, rather than on OPENs).

ADV_FSP.4 *(Performed by CCTL and NSA)*

<To be written>

ADV_HLD.3 *(Performed by CCTL)*

ADV_HLD.3.1C (The presentation of the high-level design shall be semiformal.)

ADV_HLD.3-1 The evaluator shall examine the high-level design to determine that it is semiformal in style and contains all necessary semiformal information and informal explanatory text where necessary.

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

ADV_HLD.3.2C (The high-level design shall be internally consistent.)

ADV_HLD.3-2 The evaluator shall examine the presentation of the high-level design to determine that it is internally consistent.

The evaluator validates the subsystem interface specifications by ensuring that the interface specifications are consistent with the description of the purpose of the subsystem.

ADV_HLD.3.3C (The high-level design shall describe the structure of the TSF in terms of subsystems.)

ADV_HLD.3-3 The evaluator shall examine the high-level design to determine that the TSF is described in terms of subsystems.

With respect to the high-level design, the term subsystem refers to large, related units (such as memory-management, file-management, process-management).

Breaking a design into the basic functional areas aids in the understanding of the design.

The primary purpose for examining the high-level design is to aid the evaluator's understanding of the TOE. The developer's choice of subsystem definition, and of the grouping of TSFs within each subsystem, are an important aspect of making the high-level design useful in understanding the TOE's intended operation. As part of this work unit, the evaluator should make an assessment as to the appropriateness of the number of subsystems presented by the developer, and also of the choice of grouping of functions within subsystems. The evaluator should ensure that the decomposition of the TSF into subsystems is sufficient for the evaluator to gain a high-level understanding of how the functionality of the TSF is provided.

The subsystems used to describe the high-level design need not be called "subsystems", but should represent a similar level of decomposition. For example, the design may be decomposed using "layers" or "managers".

ADV_HLD.3.4C (The high-level design shall describe the security functionality provided by each subsystem of the TSF.)

ADV_HLD.3-4 The evaluator shall examine the high-level design to determine that it describes the security functionality of each subsystem.

The security functional behavior of a subsystem is a description of what the subsystem does. This should include a description of any actions that the subsystem may be directed to perform through its functions and the effects the subsystem may have on the security state of the TOE (e.g. changes in subjects, objects, security databases).

ADV_HLD.3.5C (The high-level design shall identify any underlying hardware, firmware, and/or software required by the TSF with a presentation of the functions provided by the supporting protection mechanisms implemented in that hardware, firmware, or software.)

ADV_HLD.3-5 The evaluator shall check the high-level design to determine that it identifies all hardware, firmware, and software required by the TSF.

If the ST contains no security requirements for the IT environment, this work unit is not applicable and is therefore considered to be satisfied.

If the ST contains the optional statement of security requirements for the IT environment, the evaluator compares the list of hardware, firmware, or software required by the TSF as stated in the high-level design to the statement of security requirements for the IT environment to determine that they agree. The information in the ST characterizes the underlying abstract machine on which the TOE will execute.

ADV_HLD.3-6 The evaluator shall examine the high-level design to determine that it includes a presentation of the functions provided by the supporting protection mechanisms implemented in the underlying hardware, firmware, or software.

If the ST contains no security requirements for the IT environment, this work unit is not applicable and is therefore considered to be satisfied.

The presentation of the functions provided by the underlying abstract machine on which the TOE executes need not be at the same level of detail as the presentation of functions that are part of the TSF. The presentation should explain how the TOE uses the functions provided in the hardware, firmware, or software that implement the security requirements for the IT environment that the TOE is dependent upon to support the TOE security objectives.

The statement of security requirements for the IT environment may be abstract, particularly if it is intended to be capable of being satisfied by a variety of different combinations of hardware, firmware, or software. As part of the Tests activity, where the evaluator is provided with at least one instance of an underlying machine that is claimed to satisfy the security requirements for the IT environment, the evaluator can determine whether it provides the necessary security functions for the TOE. This determination by the evaluator does not require testing or analysis of the underlying machine; it is only a determination that the functions expected to be provided by it actually exist.

ADV_HLD.3.6C (The high-level design shall identify all interfaces to the subsystems of the TSF.)

ADV_HLD.3-7 The evaluator shall check that the high-level design identifies the interfaces to the TSF subsystems.

The high-level design includes, for each subsystem, the name of each of its entry points.

ADV_HLD.3.7C (The high-level design shall identify which of the interfaces to the subsystems of the TSF are externally visible.)

ADV_HLD.3-8 The evaluator shall check that the high-level design identifies which of the interfaces to the subsystems of the TSF are externally visible.

As discussed under work unit 4:ADV_FSP.2-3, external interfaces (i.e. those visible to the user) may directly or indirectly access the TSF. Any external interface that accesses the TSF either directly or indirectly is included in the identification for this work unit. External interfaces that do not access the TSF need not be included.

ADV_HLD.3.8C (The high-level design shall describe the purpose and method of use of all interfaces to the subsystems of the TSF, providing complete details of all effects, exceptions and error messages.)

ADV_HLD.3-9 The evaluator shall examine the high-level design to determine that it describes the interfaces to each subsystem in terms of their purpose and method of use, and provides details of effects, exceptions and error messages, as appropriate.

The high-level design should include descriptions in terms of the purpose and method of use for all interfaces of each subsystem. Such descriptions may be provided in general terms for some interfaces, and in more detail for others. In determining the level of detail of effects, exceptions and error messages that should be provided, the evaluator should consider the purposes of this analysis and the uses made of the interface by the TOE. For example, the evaluator needs to understand the nature of the interactions between subsystems to establish confidence that the TOE design is sound, and may be able to obtain this

understanding with only a general description of some of the interfaces between subsystems. In particular, internal subsystem entry points that are not called by any other subsystem would not normally require detailed descriptions.

Detailed descriptions would include details of any input and output parameters, of the effects of the interface, and of any exceptions or error messages it produces. In the case of external interfaces, the required description is probably included in the functional specification and can be referenced in the high-level design without replication.

ADV_HLD.3.9C (The high-level design shall describe the separation of the TOE into TSP-enforcing and other subsystems.)

ADV_HLD.3-10 The evaluator shall check that the high-level design describes the separation of the TOE into TSP-enforcing and other subsystems.

The TSF comprises all the parts of the TOE that have to be relied upon for enforcement of the TSP. Because 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, all TSP enforcing subsystems are contained in the TSF. Subsystems that play no role in TSP enforcement are not part of the TSF. An entire subsystem is part of the TSF if any portion of it is.

ADV_HLD.3.2E (The evaluator shall determine that the high-level design is an accurate and complete instantiation of the TOE security functional requirements.)

ADV_HLD.3-11 The evaluator shall examine the high-level design to determine that it is an accurate instantiation of the TOE security functional requirements.

The evaluator analyses the high-level design for each TOE security function to ensure that the function is accurately described. The evaluator also ensures that the function has no dependencies that are not included in the high-level design.

The evaluator also analyses the security requirements for the IT environment in both the ST and the high-level design to ensure that they agree. For example, if the ST includes TOE security functional requirements for the storage of an audit trail, and the high-level design stated that audit trail storage is provided by the IT environment, then the high-level design is not an accurate instantiation of the TOE security functional requirements.

The evaluator should validate the subsystem interface specifications by ensuring that the interface specifications are consistent with the description of the purpose of the subsystem.

ADV_HLD.3-12 The evaluator shall examine the high-level design to determine that it is a complete instantiation of the TOE security functional requirements.

To ensure that all ST security functional requirements are covered by the high-level design, the evaluator may construct a map between the TOE security functional requirements and the high-level design.

ADV_HLD.4 *(Performed by CCTL)*

<To be written>

ADV_HLD.5 *(Performed by CCTL and NSA)*

<To be written>

ADV_IMP.2 *(Performed by CCTL)*

ADV_IMP.2 applies to the entire TSF, rather than only the subset of the TSF required by ADV_IMP.1. Therefore, an evaluation including ADV_IMP.2 should use the methodology for ADV_IMP.1, but applied to the entire TSF.

The evaluator confirms the information provided meets the requirements of the additional content and presentation element (ADV_IMP.2.3C: "The implementation representation shall describe the relationships between all portions of the implementation") by checking the code to be sure that interactions among the portions of the code are identified. The "portions of code" in question are those portions that implement the modules that are identified in the low-level design.

The importance of having all of the implementation representation -- rather than only the subset -- becomes apparent not in the evaluation work associated with ADV_IMP itself, but in other components. For example, the correspondence between the description of interactions among the modules in the low-level design and the interactions of the portions of the code that implement these modules will be performed as part of ADV_RCR; this ADV_IMP action makes sure the necessary input for that activity is available. Similarly, with the entire implementation representation available, the vulnerability analysis is much more straightforward and lucrative because the evaluator can trace through the code without running into dead ends that would otherwise result from portions of code being unavailable.

ADV_IMP.3 *(Performed by CCTL)*

<To be written>

ADV_INT.1 *(Performed by CCTL)*

ADV_INT.1.1C (The architectural description shall identify the modules of the TSF.)

ADV_INT.1-1 The evaluator shall check the architectural description to determine that it identifies the modules of the TSF.

The architectural description identifies a modular structure of the TSF. The modules should be the same as those listed in the low-level design, in which case this work unit can be fulfilled by referring to the low-level design. A rationale must be provided for cases where the modules are not the same as those in the low-level design.

The architectural description provides an analysis of modularity based on module definition and classification as defined in the low-level design.

The analysis can be informal.

This work unit may be performed in conjunction with the work unit ADV_INT.1-2.

ADV_INT.1.2C (The architectural description shall describe the purpose, interface, parameters, and effects of each module of the TSF.)

ADV_INT.1-2 The evaluator shall examine the architectural description to determine whether it describes the purpose, interface, parameters, and effects of each module of the TSF.

The modules of the TSF should be defined in tabular form and for each module, the evaluator should document that the purpose, interface, parameters, and effects of each module of the TSF is defined. The consistency of the information with that contained in the low level design, the high-level design, and the functional specification should be checked.

This work unit may be performed in conjunction with the work unit ADV_INT.1-1 but with a focus on interfaces between modules.

ADV_INT.1.3C (The architectural description shall describe how the TSF design provides for largely independent modules that avoid unnecessary interactions.)

ADV_INT.1-3 The evaluator shall examine the architectural description to determine that it describes the modularity of the TSF design and structure and how it avoids unnecessary interactions between modules.

Modules interact by providing services to one another, or by cooperating in achieving an overall goal. Modules should be designed to include only related functions, to have a single, well-defined purpose, and to avoid unnecessary interactions with other modules. It is possible that some modules must be combined to implement a single security function and that dependencies in design and technical implementation may exist between modules that limit modularity.

The architectural description should include justification for the interactions that exist between modules. For example, a justification may be that elimination of an interaction results in excessive inefficiency, or in unreasonable storage requirements.

For software, modularity attributes include cohesion and coupling. Cohesion is a measure of the strength of relationship between the activities performed by a module. Cohesion ranges from the most desirable form, functional cohesion, to the least desirable form, coincidental cohesion.

Allowable forms of cohesion include functional, sequential, and communicational cohesion; coincidental cohesion is unacceptable. Temporal cohesion is acceptable only for special-purpose use, such as

initialization, shutdown, and error recovery. Logical cohesion may be acceptable, based on the argument for its justification.

Coupling is a measure of the degree of interaction between modules, and the strength of the dependencies between them. Coupling ranges from the most desirable form, call coupling, to the least desirable form, content coupling. In content coupling, which is unacceptable, one module has knowledge of, and uses, the internal content of another module.

Common coupling, which is intermediate between call and content coupling, involves the shared use of common system resources by two (or more) modules; global variables are an example of common coupling. The degree of common coupling can be assessed from analysis of the use made of common resources; such an analysis may be in the form of a matrix indicating which modules modify a resource, and which ones reference it. If multiple modules modify the same resource, there is probably an unacceptable amount of coupling.

Call trees, which diagram the calls made from one module to another, are useful analysis devices for determining when coupling exists between modules.

The evaluator should document module interactions in tabular form and define, for each interaction, whether appropriate justification was provided.

ADV_INT.1.2E (The evaluator shall determine that both the low-level design and the implementation representation are in compliance with the architectural description.)

ADV_INT.1-4 The evaluator shall examine the architectural description to determine that the low-level design is in compliance with the architectural description.

The architectural description is at a similar level of abstraction as the low-level design, in that it is concerned with the modules of the TSF. The evaluator performs a correspondence analysis between the architectural description and the low-level design, looking for consistency and accuracy between them. The evaluator ensures that each TSF module identified in the low level design is identical to a TSF module identified in the architectural description and vice versa.

ADV_INT.1-5 The evaluator shall examine the architectural description to determine that the implementation representation is in compliance with the architectural description.

The evaluator looks for consistency and accuracy between the architectural description and the implementation representation (source code and/or hardware drawings) of the TSF, to ensure that the provided implementation representation is a correct representation of the architectural description.

Correspondence information between low-level design and implementation may support this consistency check, but this consistency check is focused on the aspects, that the modularity concept outlined in the architectural description is not made ineffective by the way the implementation level is structured.

Successful completion of the previous work unit and of the ADV_RCR subactivity (which compares the low-level design with the implementation representation) and the ADV_IMP subactivity may satisfy this work unit.

ADV_INT.2 (*Performed by CCTL*)

<To be written>

ADV_INT.3 (*Performed by CCTL*)

<To be written>

ADV_LLD.2 (*Performed by CCTL*)

<To be written>

ADV_LLD.3 (*Performed by CCTL and NSA*)

<To be written>

ADV_RCR.2 (*Performed by CCTL*)

At EAL5, only the Functional Specification and High-Level Design are provided in a semiformal format. ADV_RCR.2 imposes a correspondence determination between these semiformal representations. For all remaining representations that are informal, there must likewise be a correspondence determination. The correspondence determination (at both the semiformal and informal levels) is done in accordance with the methodology for ADV_RCR.1.

ADV_RCR.3 (*Performed by CCTL and NSA*)

<To be written>

ADV_SPM.2 *(Performed by CCTL)*

<To be written>

ADV_SPM.3 *(Performed by CCTL and NSA)*

ADV_SPM.3.1C (The TSP model shall be formal.)

ADV_SPM.3-1 The evaluator shall examine the TOE security policy model to determine that it is written in a formal style.

A formal model is written with a restricted syntax with defined semantics based on 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 recognize 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. A formal proof of correspondence requires that well-established mathematical concepts be used to define the syntax and semantics of the formal notation and the proof rules that support logical reasoning. The security properties need to be expressible in the formal specification language, and these security properties need to be shown to be satisfied by the formal specification. Pointers and references to other documents may also be used.

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

ADV_SPM.3-2 The evaluator shall examine the TOE security policy model to determine that it contains all necessary informal explanatory text.

Supporting narrative descriptions are necessary for all parts of the model (for example, to make clear the meaning of any formal notation and how they are used) including the security properties and features.

ADV_SPM.3.2C (The TSP model shall describe the rules (principles) and characteristics of all policies of the TSP that can be modelled.)

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

The security policy is expressed by the collection of the functional security requirements in the ST. Therefore, to determine the nature of the security policy (and hence what policies must be modelled), the

evaluator analyses the ST functional requirements for those policies explicitly called for (e.g. by FDP_ACC and FDP_IFC, if included in the ST).

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

ADV_SPM.3-4 The evaluator shall examine the TOE security policy model to determine that all security policies represented by the security functional requirements claimed in the ST are modelled.

In addition to the explicitly-listed policies (see work unit ADV_SPM.3-3), the evaluator analyses the ST functional requirements for those policies implied by the other functional security requirement classes. For example, inclusion of FDP requirements (other than FDP_ACC and FDP_IFC) would need a description of the Data Protection policy being enforced; inclusion of any FIA requirements would necessitate that a description of the Identification and Authentication policies be present in the model; inclusion of FAU requirements need a description of the Audit policies; etc. While the other functional requirement families are not typically associated with what are commonly referred to as security policies, they nevertheless do enforce security policies (e.g. non-repudiation, reference mediation, privacy, etc.) that must be included in the TOE security policy model.

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

ADV_SPM.3-5 The evaluator shall examine the security policy model to determine that all security policies represented by the security functional requirements with dependencies on ADV_SPM are clearly articulated.

Particular attention needs to be paid to the presence of functional security components with a dependency (direct or indirect) upon ADV_SPM. In such cases, the corresponding security policy must be described sufficiently to reflect the details inherent to the functional requirements:

- a) FMT_MSA.2 contains the notion of secure values for security attributes. The corresponding security policy would include a definition of secure values for attributes. For example, a security policy that addresses the management of passwords would include a definition of the minimum acceptable password length in its description within the model. This is the security attribute whose value is checked in accordance with the FMT_MSA.2 requirement.
- b) FCS_CKM.* is dependent upon FMT_MSA.2; therefore its inclusion in the PP/ST would likewise need a similar level of detail about secure values for security attributes to be contained in the model.
- c) FMT_MTD.3 contains the notion of secure values for TSF data, which is similar to the requirements of FMT_MSA.2. Therefore, the inclusion of FMT_MTD.3 in the PP/ST would likewise need a similar level of detail about secure values for TSF data to be contained in the model.
- d) FPT_FLS.1 and FPT_RCV.* contain the notion of secure state. In order for these requirements to be meaningful, such that it can be determined whether a TOE meets them, a secure state must be identified, defined, or described in the model. This definition of secure state would consist of identifying the attributes that - alone or in combination - affect secure state, as well as identifying the permissible values those attributes may take.
- e) FRU_FLT.* is dependent upon FPT_FLS.1; therefore its inclusion in the PP/ST would likewise need an identification, definition, or description of a secure state to be contained in the model. This definition of secure state would consist of identifying the attributes that - alone or in

combination - affect secure state, as well as identifying the permissible values those attributes may take.

If the functional requirements do not include FCS_CKM.*, FMT_MSA.2, FMT_MTD.3, FPT_FLS.1, FPT_RCV.*, or FRU_FLT.*, this work unit is not applicable and is therefore considered to be satisfied.

ADV_SPM.3-6 The evaluator shall examine the rules and characteristics of the security policies to determine that the modelled security behaviour of the TOE is clearly articulated.

The policy model's *characteristics* describe the attributes of the TOE that come into consideration when enforcing its policy's rules; they are the definitions of secure values for TSF data (along with any combinational factors). The policy model's *rules* describe the TOE's behavior in enforcing the policy; the description of security rules are how the TOE enforces the values that the security-relevant attributes can take. Together the rules and characteristics describe the quality of the TOE security policy.

The TSP model's *properties* describe the TOE's behaviour in enforcing the principles of the policy. For example, a policy that is modelled on the basis of state transitions would include principles of its states, identify its initial state, and define what it means to be a secure state. The TSP model's *features* describe the attributes and conditions of the TOE that come into consideration when enforcing its policy's characteristics. For example, a policy that is modelled on the basis of state transitions would describe the necessary conditions to transform the TOE from one state to the next.

Together the properties and features formally describe the entire security posture of the TOE security policy.

In order to be considered a clear articulation, such a description should define the notion of security for the TOE, identify the security attributes of the entities controlled by the TOE, identify the TOE actions which change those attributes, and identify the enforcement of valid values for those attributes. For example, if a policy attempts to address data integrity concerns, the security policy model would:

- a) Define the notion of integrity for that TOE;
- b) Identify the types of data for which the TOE would maintain integrity;
- c) Identify the entities that could modify that data;
- d) Identify the rules that potential modifiers must follow to modify data.

As mentioned under the previous work unit, FPT_FLS.1 and FPT_RCV.* explicitly contain the notion of secure state. The model need not be state-based for either or both of these functional requirements to be claimed:

- If the model is state-based, the model would identify its states (including its initial state), and define what it means to be a secure state; the rules of the model would describe the necessary conditions to transform the TOE from one state to the next.
- If the model is not state-based, the characteristics of the model would include characteristics of what it means to be secure, in terms of properties that must hold true and values of attributes or data that must be set; these characteristics are described sufficiently to be able to determine that security is being maintained.

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

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

ADV_SPM.3.3C (The TSP model shall include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP that can be modelled.)

ADV_SPM.3-7 The evaluator shall examine the TOE security policy model rationale to determine that it formally proves the correspondence between the security properties and the security features.

The proof shall show that the security features enforce the security properties. To determine the enforcement, the evaluator considers the security properties and the security features and verifies that the arguments used in the proof are valid. The proof of correspondence between the security properties and the security features shall be formal.

ADV_SPM.3-8 The evaluator shall examine the TOE security policy model rationale to determine that it proves the internal consistency of the TOE security policy model.

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

Since the TOE security policy model is formal, the proof of its internal consistency shall be formal. It is recognized that a complete formal proof of the internal consistency of the TOE security policy model usually is not possible due to the fundamental nature of formal frameworks. Generally, it is sufficient to generate evidence using formal proofs based on the specific TOE security policy model that prove the internal consistency by means of a combination with generic arguments of the formal framework.

For additional guidance on consistency analysis see Annex B.3 of CEM part2.

ADV_SPM.3-9 The evaluator shall examine the TOE security policy model rationale to determine that the behaviour modelled is consistent with respect to policies described by the security policies (as articulated by the functional requirements in the ST).

The examination considers the informal relationships of the model. Hence the meaning of consistency reflects the conventional understanding in contrast to the internal consistency concept of the previous work unit.

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

Assurance is to be gained from an explicit and general statement of the policies underlying the TOE security functional requirements. The assurance gained is two-fold: collecting the description of each

security policy into a concise whole aids in understanding the details of the policies being enforced. Additionally, such a collected description makes it much easier to see any gaps or inconsistencies (which must be sought as part of the ADV_SPM.*.3C element), and provides a clear characterization of secure states (sought as part of the ADV_SPM.*.2C element).

The requirement for a security policy model is met by a clear statement of the security policy. The need for a separate model is not absolute, since for very straightforward policies, or those very clearly expressed in the ST, there may be no need for a separate model. However, this is often not the case. For example, audit requirements may be spread throughout the statement of TOE security functional requirements in the ST, whilst there is no clear model of the overall policy. Such a model would allow the detection of inconsistencies within the ST requirements that may otherwise pass undetected. Where a developer claims that the model requirements for some or all of the security policies are met by the ST, the evaluator needs to determine that this is the case by applying the requirements of the ADV_SPM.1 component: determining that the policy is clearly expressed, and that the model is consistent with the remainder of the ST.

The evaluator also examines whether the security policies are reflected within their formal counterparts of the TSP model.

For additional guidance on consistency analysis see Annex B.3 of CEM part 2.

ADV_SPM.3-10 The evaluator shall examine the TOE security policy model rationale to determine that the behaviour modelled is complete with respect to the policies described by the security policies (i.e. as articulated by the functional requirements in the ST).

In determining completeness of this rationale, the evaluator verifies that the security functional requirements are consistent with the security policies: each requirement is checked to see what affect it has on each security policy. For example, the FMT requirements will likely have effects on all policies: the evaluator would determine that there are no inconsistencies arising from these interrelationships.

The rationale should also provide a justification for any requirement that is not mapped to any policy.

Where a developer claims that the model requirements for some or all of the security policies are met by the ST, the evaluator needs to determine that this is the case by applying the requirements of the ADV_SPM.1 component: determining that the policy is clearly expressed, and that the model is complete with respect to the remainder of the ST. When evaluating this work-unit, the evaluator may draw upon the results of the evaluation of the completeness of the various portions of the ST.

ADV_SPM.3.4C (The demonstration of correspondence between the TSP model and the functional specification shall show that all of the security functions in the functional specification are consistent and complete with respect to the TSP model.)

ADV_SPM.3-11 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policy model to determine that it identifies all security functions described in the functional specification that implement a portion of the policy.

The degree of formality of the correspondence is the same as the degree of formality of the functional specification: the correspondence to an informal functional specification is informal; the correspondence to

a semiformal functional specification is semiformal (see ADV_SPM.3-13); and the correspondence to a formal functional specification is formal (see ADV_SPM.3-14).

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

ADV_SPM.3-12 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policy model to determine that the descriptions of the functions identified as implementing the TSP model are consistent with the descriptions in the functional specification.

The model's correspondence to the functional specification is informal where the functional specification is informal; the correspondence to a semiformal or formal functional specification is semiformal (see ADV_SPM.2-10).

To demonstrate consistency, the evaluator verifies that the functional specification correspondence shows that the functional specification's description of the interfaces to functions implementing each policy described in the model identify the same attributes and characteristics as the model and enforce the same rules as the model. The relationship between functions implementing a policy and the interfaces to those functions would also be contained in the High Level Design, if such is provided.

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

For additional guidance on consistency analysis see Annex B.3 of CEM part 2 [CEM].

ADV_SPM.3.5C (Where the functional specification is semiformal, the demonstration of correspondence between the TSP model and the functional specification shall be semiformal.)

ADV_SPM.3-13 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policy model to determine that it is presented in a semiformal style.

If the functional specification is not semiformal, this work unit is not applicable and is therefore considered to be satisfied. If the ST includes ADV_FSP.4 'Formal functional specification' then ADV_SPM.3.5C and this work unit are not applicable. Therefore in this case this work unit is considered to be satisfied; however, the work unit ADV_SPM.3-14 applies.

The functional specification correspondence demonstration described in work units ADV_SPM.3-11 and ADV_SPM.3-12 has to be semiformal.

A semiformal correspondence is one that results from a structured approach with a substantial degree of rigor (in terms of completeness and correctness), but is not as rigorous as a mathematical proof. Such a

semiformal correspondence limits the subjective interpretations of its terms, and so it provides less ambiguity than would exist in an informal correspondence.

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

ADV_SPM.3-14 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policy model to determine that it is in a formal style.

If the ST includes ADV_FSP.3 ‘Semiformal functional specification’ then ADV_SPM.3.6C and this work unit are not applicable. However, in this case the functional specification is considered to be semiformal and work unit ADV_SPM.3-14 applies.

The functional specification correspondence demonstration described in work units ADV_SPM.3-11 and ADV_SPM.3-12 has to be a formal proof.

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

ALC

ALC_DVS.2 *(Performed by CCTL)*

<To be written>

ALC_FLR.1,
ALC_FLR.2,
ALC_FLR.3
(Performed by CCTL)

See the ([Supplement on Flaw Remediation, version 1.1, Feb 2002, CEM-2001/0015R](#))

ALC_LCD.2 (Performed by CCTL)

ALC_LCD.2.1C (The life-cycle definition documentation shall describe the model used to develop and maintain the TOE.)

ALC_LCD.2-1 The evaluator shall examine the documented description of the life-cycle model used to determine that it covers the development and maintenance process.

A life-cycle model encompasses the procedures, tools and techniques used to develop and maintain the TOE. The description of the life-cycle model should include information on the procedures, tools and techniques used by the developer (e.g. for design, coding, testing, bug-fixing). It should describe overall management structure governing the application of the procedures (e.g. an identification and description of the individual responsibilities for each of the procedures required by the development and maintenance process covered by the life-cycle model).

ALC_LCD.2.2C (The life-cycle model shall provide for the necessary control over the development and maintenance of the TOE)

ALC_LCD.2-2 The evaluator shall examine the life-cycle model to determine that use of the procedures, tools and techniques described by the life-cycle model will make the necessary positive contribution to the development and maintenance of the TOE.

The information provided in the life-cycle model gives the evaluator assurance that the development and maintenance procedures adopted would minimize the likelihood of security flaws. For example, if the life-cycle model described the review process, but did not make provision for recording changes to components, then the evaluator may be less confident that errors will not be introduced into the TOE. The evaluator may gain further assurance by comparing the description of the model against an understanding of the development process gleaned from performing other evaluator actions relating to the TOE development (e.g. those actions covered under the ACM activity). Identified deficiencies in the life-cycle model will be of concern if they might reasonably be expected to give rise to the introduction of flaws into the TOE, either accidentally or deliberately.

The CC does not mandate any particular development approach, and each should be judged on merit. For example, spiral, rapid-prototyping and waterfall approaches to design can all be used to produce a quality TOE if applied in a controlled environment.

ALC_LCD.2.3C (The life-cycle definition documentation shall explain why the model was chosen.)

ALC_LCD.2-3 The evaluator shall examine the life-cycle definition documentation to determine that it explains why the model was chosen.

The life-cycle definition documentation should provide reasons for adoption of the chosen life-cycle model. Such reasons may include, for example, conformance to an organisational policy or to a security policy (also in the form of the Security Target), or may be in the form of benefits perceived to be attainable through use of the life-cycle model.

ALC_LCD.2.4C (The life-cycle definition documentation shall explain how the model is used to develop and maintain the TOE.)

ALC_LCD.2-4 The evaluator shall examine the life-cycle definition documentation to determine that it explains how the standardized model has been applied to the development and maintenance of the TOE.

Whereas the requirements of ALC_LCD.1 are confined to a description of the model used, this component requires the developer to explain how the standardized model has been applied to the TOE under evaluation. This explanation shall cover using the life-cycle model for development and maintenance of the TOE as well as adaptation of the standardized model to meet specific TOE or organisational requirements. Usage of the life-cycle model chosen shall be compliant with the configuration management (class ACM).

ALC_LCD.2-5 The evaluator shall examine the life-cycle definition for consistency with the Security Target of the TOE.

If the Security Target contains aspects of the TOE development and production, e.g. specific assumptions, policies and objectives for these life-cycle phases, consistency with the life-cycle definition documentation and the standardized model used has to be determined.

If the life-cycle model used for the TOE covers aspects after TOE delivery (e.g. the assurance family ALC_FLR is chosen), the life-cycle definition documentation has to be analysed for consistency between information given in the ST about the intended operating environment of the TOE and about the TOE's security objectives in comparison to the model for the portion of the life-cycle after the delivery of the TOE. If the assurance family ALC_FLR is chosen, the relevant information shall be considered in the context of the analysis.

ALC_LCD.2.5C (The life-cycle definition documentation shall demonstrate compliance with the standardised life-cycle model.)

ALC_LCD.2-6 The evaluator shall examine the life-cycle definition documentation to determine that it demonstrates that the life-cycle model used corresponds to the standardized model.

A standardized life-cycle model is a model that has been approved by some group of experts (e.g. academic experts, standards bodies). A formal approval for the standardized life-cycle model is not mandated, but it should be public, well accepted and used by a wider group of experts or e.g. being common practice in a specific industry.

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

The life-cycle definition documentation should relate aspects of the standardized model to the specific development and maintenance procedures in place for the TOE, such that conformance to the standardized model can be easily confirmed by the evaluator. The correspondence evidence may, for example, take the form of a mapping from detailed steps and organisation roles in the standardized model to individual development procedures and roles or personnel from the development environment.

Through completion of this work unit, the evaluator should gain a clear understanding of how the standardized model has been applied, and that it has been applied correctly.

ALC_LCD.3 *(Performed by CCTL)*

<To be written>

ALC_TAT.2 *(Performed by CCTL)*

ALC_TAT.2.1C (All development tools used for implementation shall be well-defined.)

ALC_TAT.2-1 The evaluator shall examine the development tool documentation provided to determine that all development tools are well-defined.

For example, a well-defined language, compiler or CAD system may be considered to be one that conforms to a recognized standard, such as the ISO standards. A well-defined language is one that has a clear and complete description of its syntax, and a detailed description of the semantics of each construct.

ALC_TAT.2.2C (The documentation of the development tools shall unambiguously define the meaning of all statements used in the implementation.)

ALC_TAT.2-2 The evaluator shall examine the documentation of development tools to determine that it unambiguously defines the meaning of all statements used in the implementation.

The development tool documentation (e.g. programming language specifications and user manuals) should cover all statements used in the implementation representation of the TOE, and for each such statement provide a clear and unambiguous definition of the purpose and effect of that statement. This work may be performed in parallel with the evaluator's examination of the implementation representation performed during the ADV_IMP sub-activity. The key test the evaluator should apply is whether or not the documentation is sufficiently clear for the evaluator to be able to understand the implementation representation. The documentation should not assume (for example) that the reader is an expert in the programming language used.

Reference to the use of a documented standard is an acceptable approach to meet this requirement, provided that the standard is available to the evaluator. Any differences from the standard should be documented.

The critical test is whether the evaluator can understand the TOE source code when performing source code analysis covered in the ADV_IMP sub-activity. However, the following checklist can additionally be used in searching for problem areas:

- a) In the language definition, phrases such as "the effect of this construct is undefined", and terms such as "implementation dependent" or "erroneous" may indicate ill-defined areas;
- b) Aliasing (allowing the same piece of memory to be referenced in different ways) is a common source of ambiguity problems;

- c) Exception handling (e.g. what happens after memory exhaustion or stack overflow) is often poorly defined.

Most languages in common use, however well designed, will have some problematic constructs. If the implementation language is mostly well defined, but some problematic constructs exist, then an inconclusive verdict should be assigned, pending examination of the source code.

The evaluator should verify, during the examination of source code, that any use of the problematic constructs does not introduce vulnerabilities. The evaluator should also ensure that constructs precluded by the documented standard are not used.

ALC_TAT.2.3C (The documentation of the development tools shall unambiguously define the meaning of all implementation dependent options.)

ALC_TAT.2-3 The evaluator shall examine the development tool documentation to determine that it unambiguously defines the meaning of all implementation-dependent options.

The documentation of software development tools should include definitions of implementation-dependent options that may affect the meaning of the executable code, and those that are different from the standard language as documented. Where source code is provided to the evaluator, information should also be provided on compilation and linking options used.

The documentation for hardware design and development tools should describe the use of all options that affect the output from the tools (e.g. detailed hardware specifications, or actual hardware).

ALC_TAT.2.2E (The evaluator shall confirm that the implementation standards have been applied.)

ALC_TAT.2-4 The evaluator shall examine aspects of the implementation process to determine that documented implementation standards have been applied.

This work unit requires the evaluator to analyse the provided implementation representation of the TOE to determine whether the documented implementation standards have been applied.

The evaluator should verify that constructs excluded by the documented standard are not used.

Additionally, the evaluator shall verify the developer's procedures which ensure the application of the defined standards within the design and implementation process of the TOE. Therefore, documentary evidence should be supplemented by visiting the development environment. A visit to the development environment will allow the evaluator to:

- a) observe the application of defined standards;
- b) examine documentary evidence of application of procedures describing the use of defined standards;
- c) interview development staff to check awareness of the application of defined standards and procedures.

A development site visit is a useful means of gaining confidence in the procedures being used. Any decision not to make such a visit should be determined in consultation with the overseer.

The evaluator compares the provided implementation representation with the description of the applied implementation standards and verifies their use. At this level it is not required that the complete provided implementation representation of the TSF is based on implementation standards. It is only required that those implementation standards, referenced by the developer, are applied indeed. If the referenced implementation standards are not applied for at least parts of the provided implementation representation, this work unit fails.

This work unit may be performed in conjunction with the ADV_IMP.* sub-activities.

ALC_TAT.3 *(Performed by CCTL)*

<To be written>

ATE

ATE_COV.3 *(Performed by CCTL)*

<To be written>

ATE_DPT.2 *(Performed by CCTL)*

DPT.2 differs from DPT.1 in that the low-level design is also used in the depth analysis. The methodology is the same as that of DPT.1, with the following additional workunits:

ATE_DPT.2.1C (The depth analysis shall demonstrate that the tests identified in the test documentation are sufficient to demonstrate that the TSF operates in accordance with its high-level design and low-level design.)

ATE_DPT.2-2 The evaluator shall examine the depth of testing analysis for the mapping between the tests identified in the test documentation and the low-level design.

The depth of testing analysis identifies all modules described in the low-level design and provides a mapping of the tests to the modules. Correspondence may take the form of a table or matrix. In some cases

the mapping may be sufficient to show test correspondence. In other cases a rationale (typically prose) may have to supplement the mapping evidence provided by the developer.

All design details specified in the low-level design that map to and satisfy TOE security requirements are subject to testing and, hence, should be mapped to the test documentation. Figure 1.2 below displays a conceptual framework of the mapping between modules described in the low-level design and the tests outlined in the TOE's test documentation used to test them. Tests may involve one or multiple security functions depending on the test dependencies or the overall goal of the test being performed.

The focus of this examination is to have evidence, that all design details of the modules are adequately tested rather than showing that all modules and interfaces are only touched by test cases.

ATE_DPT.2-4 The evaluator shall examine the test procedures to determine that the test pre-requisites, test steps and expected result(s) adequately test each security function.

Guidance on this work unit, as it pertains to the high-level and low-level design, can be found in: (see CC Interpretation 074)

Application notes (see above), Verifying the adequacy of tests.

(*) In case of testing the high-level design the examination of the test procedures might make use of high-level design information on the structure and interaction of subsystems to get evidence that specific test cases (possibly activated from an external interface, see previous work unit) are adequate to cover a design detail of a subsystem or its interface as specified.

In case of testing the low-level design the examination of the test procedures might make use of low-level design information on the structure and interaction of modules to get evidence that specific test cases (possibly activated from an external interface, see previous work unit) are adequate to cover a design detail of a module or its interface as specified.

ATE_DPT.2-6 The evaluator shall check the depth of testing analysis to ensure that the TSF as defined in the low-level design is completely mapped to the tests in the test documentation.

The depth of testing analysis provides a complete statement of correspondence between the low-level design and the test plan and procedures. All modules and internal interfaces described in the low-level design have to be present in the depth of testing analysis. All the modules and internal interfaces present in the depth of testing analysis must have tests mapped to them in order for completeness to be claimed (see also above Application note, Verifying the adequacy of tests). As Figure 1.2 below displays, all the modules and internal interfaces have tests attributed to them and therefore complete depth of testing is depicted in this example. Incomplete coverage would be evident if a module or internal interface was identified in the depth of testing analysis and no tests could be attributed to it.

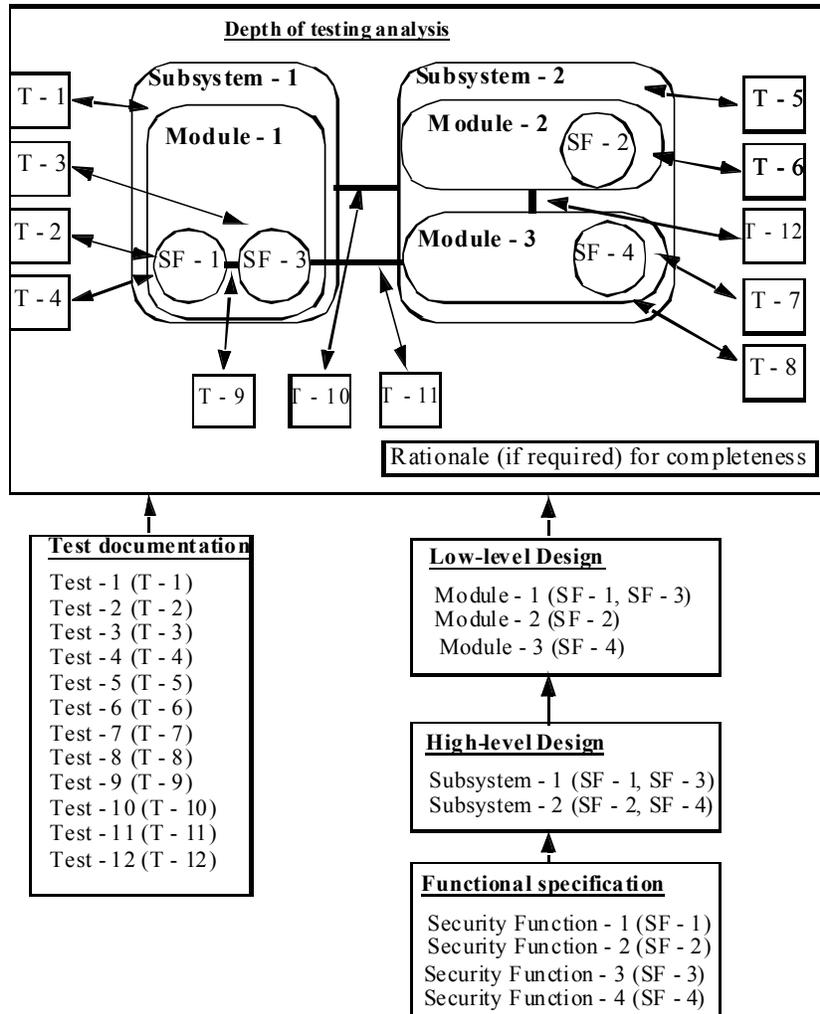


Figure 1: A conceptual framework of the depth of testing analysis on the module-level

ATE_DPT.3 (Performed by CCTL)

<To be written>

ATE_FUN.2 (Performed by CCTL)

ATE_FUN.2 adds a single requirement to ATE_FUN.1: there must be an analysis of any ordering dependencies of the tests. This explains any situation where a particular test must be run before another (e.g. where the output of one is used as an input to another),

or where one test must *not* be run before another. Any such relationships that exist among the tests must be explained.

ATE_IND.3 (*Performed by CCTL*)

While ATE_IND.2 requires the evaluator re-running a subset of the developer's tests, ATE_IND.3 requires the evaluator re-running all of the developer's tests. This is the only difference from IND.2

AVA

AVA_CCA.1 (*Performed by CCTL and NSA*)

(This set of guidance is derived from *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, National Computer Security Center, November 1993, NCSC_TG-030)

AVA_CCA.1.1C (The analysis documentation shall identify covert channels and estimate their capacity.)

AVA_CCA.1-1 The evaluator shall check the covert channel analysis documentation to determine that it identifies covert channels.

A covert channel is “a communication channel that allows a process to transfer information in a manner that violates the system’s security policy.” We introduce the term “bandwidth” here to denote the rate at which information is transmitted through a channel. This use of the term “bandwidth” is also related to the notion of “capacity.” The capacity of a channel is its maximum possible error-free information rate in bits per second.

Developers should show the identification method they use to be sound and reliable (e.g., repeatable). This implies, among other things, that independent evaluators can use the method on the same sources of covert channel information and get the same results. Otherwise, the identification evidence will lack credibility.

The documentation of each identified storage channel should consist of the variable the channel views/alters and the TSF primitives that alter or view the variable. Developers should distinguish potential covert channels from real ones.

The evaluator checks the covert channel analysis to determine that it identifies covert channels with reference to the information flow policies as defined in the ST by SFR of the FDP_IFC family:

- If ST does not define any SFR of the family FDP_IFC then the AVA_CCA.1 is not applicable.
- If ST defines a SFR of the family FDP_IFC then the evaluator check whether the covert channel analysis documentation provided by the developer address the information flow control policy.
- If the ST claims FDP_IFF.5 then the dependency on AVA_CCA.3 shall be fulfilled. Otherwise a reasonable rationale shall be provided.

When conducting the covert channel analysis, the evaluator needs to make sure that all resources (not just those defined in the applicable FDP_IFC/IFF components) are considered as part of the analysis; if they

determine that a resource can be used in a covert channel, yet its use doesn't circumvent the policy set forth by the rules in FDP_IFC/IFF, that covert channel can be ignored.

AVA_CCA.1-2 The evaluator shall check the covert channel analysis documentation to determine that it estimates the capacity of the covert channels.

Capacities are typically expressed in terms of bits per second or bits per operation which cause the illicit information flow.

Note if the ST claims FDP_IFF.3, FDP_IFF.4 or FDP_IFF.6, it shall define maximum capacity for types of illicit information flows by assignment. If the ST claims FDP_IFF.1, FDP_IFF.2 or FDP_IFF.5, any covert channel identified by these SFR with positive capacity will violate the SFP of these SFRs.

In measuring or estimating covert channel bandwidth, developers should consider the following factors:

- For maximum bandwidth, assume the channel is noiseless and the presence of other processes in the system does not delay the senders and receivers.
- The choice of informal estimation methods requires defining (and possibly justifying) assumptions about the coding method and, therefore, the distribution of 0s and 1s in all transmissions.
- Covert channel measurements should include the fastest primitives for altering, viewing, and setting up the transmission environment. Also, bandwidth measurements should involve the demonstrably fastest process (context) switch time.
- To determine bandwidth, derive the TSF primitives to measure from real scenarios of covert channel use. Take into account parameter and TSF state dependencies of each selected primitive (if any).
- Specify the measurement environment. This specification includes (1) the speed of the system components, (2) the system configuration, (3) the sizes of the memory and cache components, and (4) the system initialization. Document the sensitivity of the measurement results to configuration changes. (This documentation enables accreditors to assess the real impact of covert channels in different environments of use.)
- Sender-receiver synchronization time may be considered negligible and, therefore, ignored.
- Consider channel aggregation in bandwidth estimation.
- All measurements must be repeatable. For bandwidth estimation, developers must document measurements of each covert channel primitive and should include the bandwidth computation for each channel.

AVA_CCA.1.2C (The analysis documentation shall describe the procedures used for determining the existence of covert channels, and the information needed to carry out the covert channel analysis.)

AVA_CCA.1-3: For each covert channel identification method used the evaluator shall examine the analysis documentation to make certain it describes the procedures used for determining the existence of covert channels, and the information needed to carry out the covert channel analysis.

There are 4 different methods used for determining the existence of covert channels. They are listed below.

1. Syntactic Information-Flow Analysis

In all flow-analysis methods, one attaches information-flow semantics to each statement of a specification (or implementation) language. For example, a statement such as “a = b” causes information to flow from b

to a (denoted by $b \rightarrow a$) whenever b is not a constant. Various tools have been built to apply syntactic flow analysis to formal specifications.

2. Addition of Semantic Components to Information-Flow Analysis

This method for identification of potential storage channels is based on (1) The analysis of programming language semantics, code, and data structures used within the kernel, to discover variable alterability/visibility; (2) resolution of aliasing of kernel variables to determine their indirect alterability; and (3) information-flow analysis to determine indirect visibility of kernel variables.

[Reference: C.-R. Tsai, V. D. Gligor, and C. S Chandrasekaran, "A Formal Method for the Identification of Covert Storage Channels in Source Code," *IEEE Transactions on Software Engineering*, 16:6, pp. 569-580, June 1990. (Also in the *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 74-86, April 1987.)]

3. Shared Resource Matrix (SRM) Method

The SRM method for identifying covert channels requires the following four steps:

1. Analyze all TSF primitive operations specified formally or informally, or in source code.
2. Build a shared resource matrix consisting of user-visible TSF primitives as rows and visible/alterable TSF variables representing attributes of a shared resource as columns; mark each <TSF primitive, variable> entry by R or M depending on whether the attribute is read or modified. (This step assumes one has already determined variable visibility/alterability through the TSF interface.) Variables that can neither be viewed nor altered independently are lumped together and analyzed as a single variable.
3. Perform a transitive closure on the entries of the shared resource matrix. This step identifies all indirect reading of a variable and adds the corresponding entries to the matrix. A TSF primitive indirectly reads a variable y whenever a variable x, which the TSF primitive can read, can be modified by TSF functions based on a reading of the value of variable y. (Note that whenever the SRM method is applied to informal specifications of a TCB interface as defined in system reference manuals—and not to internal TSF specifications of each primitive, which may be unavailable—performing this step can only identify how processes outside the TSF can use information covertly obtained through the TSF interface. Therefore, whenever people using the SRM method treat the TSF as a black box, they can eliminate the transitive closure step since it provides no additional information about flows within the TSF specifications or code.)
4. Analyze each matrix column containing row entries with either an 'R' or an 'M'; the variable of these columns may support covert communication whenever a process may read a variable which another process can write and the security level of the former process does not dominate that of the latter. Analysis of the matrix entry leads to four possible conclusions:
 - If a legal channel exists between the two communicating processes (i.e., an authorized channel), this channel is of no consequence; label it "L".
 - If one cannot gain useful information from a channel, label it "N".
 - If the sending and receiving processes are the same, label the channel "S".
 - If a potential channel exists, label it "P".

[Reference: R. A. Kemmerer, "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels," *ACM Transactions on Computer Systems*, 1:3, pp. 256-277, August 1983.]

4. Noninterference Analysis

Noninterference analysis of a TSF requires one to view the TSF as an abstract machine. From the point of view of a user process, a TSF provides certain services when requested. A process' requests represent the abstract machine's inputs; the TSF responses (e.g., data values, error messages, or positive acknowledgements) are its outputs, and the contents of the TSF internal variables constitute its current state. Each input results in a (TSF) state change (if necessary) and an output. Each input comes from some particular process running at a particular security level, and each output is delivered only to the process that entered the input that prompted it.

AVA_CCA.1-4 The evaluator shall examine the covert channel analysis documentation to determine that it describes the information needed to conduct the covert channel analysis.

The information used in conducting the covert channel analysis comes from such sources as system reference manuals containing descriptions of TSF primitives, CPU and I/O processor instructions, their effects on system objects and registers, TSF parameters or instruction fields, and so on; the high-level design, low-level design, and functional specification; and TSF source code and processor-instruction (micro) code.

AVA_CCA.1.3C (The analysis documentation shall describe all assumptions made during the covert channel analysis.)

AVA_CCA.1-5: The evaluator shall examine the covert channel analysis documentation to determine that it describes the assumptions made during the covert channel analysis.

The analysis documentation includes all factors that affect the calculation or implementation of the covert channels. Such factors include the assumed noiselessness of the channel, the aggregation of the data variables used to implement the channel, and any encoding that is used to transmit information through the channel.

AVA_CCA.1.4C (The analysis documentation shall describe the method used for estimating channel capacity, based on worst-case scenarios.)

AVA_CCA.1-6: The evaluator shall examine the covert channel analysis documentation to determine that it describes the method used to estimate the capacity of the covert channels.

There are 2 methods that can be used for bandwidth estimation:

1. Information-Theory-Based Method for Channel-Bandwidth Estimation - one assumes the covert channels are noiseless, no processes other than the sender and receiver are present in the system during channel operation, and the sender-receiver synchronization takes a negligible amount of time. These assumptions are justified if the goal is the computation of the maximum attainable bandwidth.
2. Informal Method for Estimating Covert Channel Bandwidth - a simple formula for computing the maximum attainable bandwidth of a noise-less covert channel in absence of any spurious processes that would delay senders and receivers.

[Reference: C.-R. Tsai and V. D. Gligor, "A Bandwidth Computation Model for Covert Storage Channels and Its Applications," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 108-121, April 1988.]

AVA_CCA.1.5C (The analysis documentation shall describe the worst-case exploitation scenario for each identified covert channel.)

AVA_CCA.1-7: The evaluator shall examine the covert channel analysis documentation to determine that it describes the worst-case exploitation scenario for each covert channel.

Worst-case refers to the most efficient implementation of a channel (which yields the greatest bandwidth). While the theoretical worst-case bandwidth may not be realised in practice, the scenario producing such results is nevertheless described.

AVA_CCA.1.2E (The evaluator shall confirm that the results of the covert channel analysis show that the TOE meets its functional requirements.)

AVA_CCA.1-8: The evaluator shall examine the covert channel analysis documentation to determine that the TOE meets its functional requirements.

The evaluator confirms that the covert channel analysis shows that the TOE meets its functional requirements (AVA_CCA.1.2E) only for TOEs claiming FDP_IFF.1/FDP_IFC.1; these are the only functional requirements for which covert channels are meaningful. The policy quoted in these functional components is examined and the evaluator notes any covert channels that violate this policy, including the bandwidth of all such channels.

The FDP_IFF.1/FDP_IFC.1 requirements are verified to ensure that they take into account the covert channels as described in the covert channel analysis documentation.

If the ST claims FDP_IFF.4, the covert channel analysis confirms that the capacity of any found illicit information flow does not exceed the defined maximum capacity assigned for this type of illicit information flow in FDP_IFF.4.1, and does not discover any illicit information flow of types identified in FDP_IFF.4.2.

If the ST claims FDP_IFF.3, the covert channel analysis confirms that any illicit information flow that exceeds the defined maximum capacity is monitored. However, assignments in FDP_IFF.1 and FDP_IFF.2 might be written in such a way that covert channels would have an impact on them as well. In this case, if the covert channel analysis estimates a positive capacity of a type of illicit information flow, the conclusion, whether these functional requirements are met, depends on the decision, whether the relevant security objectives are actually reached.

Similarly, the audit requirements might also be affected by the presence of the covert channel analysis. For example, the use of resources that implement covert channels might be auditable events, in which case they would be expected to be listed in FAU_GEN.

Fulfillment of other functional requirements (e.g. FDP_ETC, FDP_ITC, FDP_ITT, FDP_ROL, FDP_UCT, FDP_UIT, also FMT_MSA, FPT_SEP) may likewise be affected by covert channels.

AVA_CCA.1.3E (The evaluator shall selectively validate the covert channel analysis through testing.)

AVA_CCA.1-9: The evaluator shall test a selection of the covert channels.

Testing of the covert channel analysis provides the opportunity to verify the identification, capacity estimation, elimination, monitoring, and exploitation scenarios of the covert channels that are identified and described in the covert channel analysis.

All channels identified during the covert channel analysis are included in the testing. Covert channel testing demonstrates that covert channel handling methods chosen by system designers work as intended. These methods include covert channel elimination, bandwidth limitation, and (ability to) audit. Testing is also useful to confirm that potential covert channels discovered in the system are in fact real channels. Furthermore, testing is useful when the handling method for covert channels uses variable bandwidth-reduction parameters (e.g., delays) that are settable by system administrators (e.g., by auditors).

AVA_CCA.1-10: The evaluator shall produce test documentation for the tests of covert channels, in sufficient detail to enable the tests to be repeatable.

The test documentation includes:

- identification of the covert channel the TOE being tested;
- instructions to connect and set-up all required test equipment as required to conduct the test;
- instructions to establish all test prerequisite initial conditions;
- instructions to stimulate the TSF;
- instructions for observing the behaviour of the TSF, among other for measurement of the channel capacity;
- descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results;
- instructions to conclude the test and establish the necessary post-test state for the TOE.

The intent of specifying this level of detail in the test documentation is to allow another evaluator to repeat the tests and obtain equivalent results.

AVA_CCA.2 (*Performed by CCTL and NSA*)

<To be written>

AVA_CCA.3 (*Performed by CCTL and NSA*)

<To be written>

AVA_MSU.3 (*Performed by CCTL*)

<To be written>

AVA_VLA.3,
AVA_VLA.4
(*Performed by CCTL and NSA*)

AVA_VLA.3 adds the requirement that the TOE be resistant to penetrations of medium attack potential. The is judged by NSA evaluators. However, the CCTL evaluators will still have work to contribute.

Evaluators are to apply the AVA_VLA.2 methodology contained within the CEM. The evaluator then provides the design documentation, ST, VLA.2 analysis, and draft ETR to CCEVS. CCEVS will provide these materials to NSA evaluators, who will augment the vulnerability analysis and penetration testing efforts.

AMA

**AMA_AMP.1,
AMA_CAT.1,
AMA_EVD.1,
AMA_SIA.1,
AMA_SIA.2**

These components have been replaced by the new approach to [Assurance Continuity](#).

