ASSERT4SOA

Advanced Security Service cERTificate for SOA

CP - STREP - Grant No. 257351

Report on the identified certificational requirements

Deliverable D.7.4

Project Title	ASSERT4SOA – Advanced Security Service cERTificate for SOA
Grant Number	257351
Project Type	CP - STREP

Title of Deliverable	Report on the identified certificational requirements
Subtitle of Deliverable	-
Deliverable Number	D.7.4
Dissemination Level	Consortium
Internal Rev. No.	-
Contractual Delivery Date	30 September 2013 (M36)
Actual Delivery Date	30 September 2013 (M36)
Contributing WPs	WP7
Editor(s)	Renato Menicocci (FUB)
Author(s)	Franco Guida (FUB), Vittorio Bagini (FUB), Alessandro Riccardi (FUB), Samuel Paul Kaluvuri (SAP), Antonino Sabetta (SAP), Hristo Koshutanski (UMA)
Reviewer(s)	Michele Bezzi (SAP), Valentina Di Giacomo (ENG)

Executive Summary

The objective of this document is to summarize the most significant topics targeted during the continuous consultancy done by FUB under Task 7.4 (Continuous Consultancy about 'Certificational' Requirements) of WP7 (Requirements, specification and validation of ASSERT4SOA) [DoW]. The consultancy went on for the whole project duration and addressed a number of project partner requests regarding specific topics in the security certification domain.

This consultancy activity brought to the identification of some sort of recommendations which correspond to the above *certificational requirements* (a neologism introduced in [DoW]). These recommendations have not been formalized as such, but they have been exploited to define requirements for the ASSERT4SOA Framework and effectively drive its implementation from a certification world perspective.

This document concentrates on 10 consultancy cases which seem to be the most significant ones and bring to the construction of complete and well-structured written responses. The nature of the consultancy cases ranges from clarification about ICT current security certification best practices to concrete analysis of a specific ASSERT4SOA problem with the provision of a possible solution.

The document reports the consultancy cases and additionally, for some of these cases, it describes the exploitation of the consultancy within ASSERT4SOA.

Contents

1. Introduction 8
1.1 Scope and objectives 8
1.1.1 Consultancy case report format 9
2. Consultancy Reports 10
2.1 Introduction 10
2.2 Case: Composition in Common Criteria 11
2.3 Case: Formal methods in Common Criteria 15
2.4 Case: Notion of time in Common Criteria 20
2.5 Case: Incremental certification in Common Criteria 22
2.6 Case: SOA and Common Criteria (I) 26
2.7 Case: Assurance Requirements and Policies 28
2.8 Case: SOA and Common Criteria (II) 34
2.9 Case: ASSERT Revocation Status 35
2.10 Case: ASSERT Issuer Competence 39
2.11 Case: Examples of concrete ASSERT Profiles 42
3. Consultancy Exploitation 49
4. Conclusions 53
5. Appendix A: CC Terminology 54
6. Appendix B: Introduction to CC Security Requirements 58
7. Appendix C: Selected Assumptions and Requirements from [D7.1] 60
8. Appendix D: ASSERT Issuer Competence Support by UMA 62
9. Bibliography 72

Chapter 1

Introduction

1.1 Scope and objectives

Based on its expertise on standard approaches to software security certification and especially on Common Criteria, FUB acted as an on–demand consultant for the whole ASSERT4SOA project duration. Within this consultancy activity, FUB answered to specific requests from other partners regarding various aspects of the ICT security certification ecosystem, as required in the description of Task 7.4 (Continuous Consultancy about 'Certificational' Requirements) of WP7 (Requirements, specification and validation of ASSERT4SOA) [DoW]. Notice that the consultancy has allowed comparing the evolution of the ASSERT4SOA Framework to the real world of ICT security certification.

The results of the consultancy activity contain some sort of recommendations, which, in a sense, correspond to the expected *certificational requirements*. Even though these recommendations have not been formalized as such, they have been initially exploited as an input to [D7.1] for the definition of the Framework requirements related to ASSERT language, ASSERT management, and verification of ASSERT validity (within Service Discovery). Moreover, after D7.1 delivery (M9, M12), the above recommendations have been exploited as driving principles for the implementation of the Framework.

This document reports 10 consultancy cases which seem to be the most significant ones and which reached the "critical mass" to lead to the production of complete and well-structured responses to given requests. Some of these requests (see Sections 2.2-2.5 and 2.7) can be classified as requests for clarification about how the certification world addressed specific aspects, some others (see Section 2.6 and 2.8) can be seen as requests for analysis of how much the ASSERT4SOA concepts are aligned with and exploitable in a specific certification context, still some others (see Sections 2.9-2.11) are deep and

detailed analysis of specific ASSERT4SOA problems along with the corresponding possible solutions.

Consultancy cases are presented in Chapter 2 by using the specific format given in Section 1.1.1. In Chapter 2, where suitable, the reader is referred to Appendixes A and B for basic material related to Common Criteria, to Appendix C for selected contents of [D7.1], and to Appendix D for the contents of an internal document by UMA on the management of credentials of ASSERT Issuers. Chapter 3 reports, for some of the reported consultancy cases and directly from the requesting partners, how the results of the consultancy have been exploited in the project. Finally, Chapter 4 contains the concluding remarks.

1.1.1Consultancy case report format

A consultancy case report starts with a title and goes on with a description, which is formatted as follows.

Request

A summary of the request for consultancy.

Requesting Partner

The partner(s) of the consortium requesting the consultancy.

Response

The whole response to the request.

Chapter 2

Consultancy Reports

2.1 Introduction

This chapter reports 10 consultancy cases produced by FUB following as many requests coming from the remaining ASSERT4SOA partners. The received requests could be classified as follows:

- *Common Criteria Base Knowledge:* The requests reported in Sections 2.2-2.5 and 2.7 asked to clarify how the certification world addressed specific aspects. The corresponding responses were tutorials on specific topics based on Common Criteria, which are the main reference for ICT security certification;
- *Common Criteria and Services:* Those reported in Sections 2.6 and 2.8 were requests for analysis of how the certification world could address SOA based systems. The corresponding responses were studies of how specific aspects of Common Criteria certification could be adapted to services, possibly exploiting some ASSERT4SOA concepts.
- *ASSERT4SOA Processes:* Those reported in Sections 2.9-2.11 were requests for analysis of specific ASSERT4SOA problems and the corresponding responses were the requested analysis along with proposals of possible solutions for the specific problems addressed.

Each consultancy case is reported according to the structure presented in Section 1.1.1.

Notice that, in the descriptions of the consultancy cases of Sections 2.2-2.8, some Common Criteria (CC) terms and concepts are used. The reader is referred to Appendix A for the used CC terminology and to Appendix B for an introduction to CC security requirements. Also, in the descriptions of the relevant consultancy cases, and in Appendix A and B as well, an underlined element signals the (first) occurrence of a (connected) term which is defined in Appendix A. Moreover, the descriptions of the consultancy cases of Sections 2.9 and 2.10 refer to some assumptions and requirements for the ASSERT4SOA Framework [D7.1] which, for convenience, are reported in Appendix C. Finally,

the description of the consultancy case of Section 2.10 refers to an internal document by UMA whose contents, for convenience, are reported in Appendix D.

2.2 Case: Composition in Common Criteria

Request

How do Common Criteria face certification of a system made of different components when the certification of the system as a whole is not practicable?

Requesting Partner

SIT

Response

An overview of the approaches proposed within the Common Criteria environment to solve this problem has been presented in [ICCC2010] and is summarized in the following.

Definition of the problem

The problem of interest may be defined by the following assumptions:

- The system to be certified has a known number of components;
- Each component is completely specified from both a functional and a security point of view;
- A description of the Security Objectives of both the system and its environment is available;
- For some reasons (*System evaluation issues*) the certification of the system as a whole is not practicable.

In [ICCC2010] some possible system evaluation issues (which are not recalled here) are identified together with possible approaches for solving them.

Alternative approaches to system evaluation

The most significant approaches to the problem considered in [ICCC2010] are

- TOE (Target Of Evaluation) composition
  - Approach supported by CC v3.1 and defined by [CCpart3] and [CEM];
- Composite product evaluation
  - Approach supported by CC v2.x and v3.1 and defined by the smart card evaluation supporting document [CCDB-2007-09-01];
- Cross evaluation of components
  - Approach of two Protection Profiles included in [CWA 14169].

These 3 approaches will be recalled in the following.

TOE composition

CC v3.1 provide support for the evaluation of a composed TOE that is the combination of two components, where the dependent component relies on the services provided by the base component. Composed TOE evaluation aims at gaining confidence in the security functions of the composed TOE based on the

successful evaluation of the security functions of its components under the following assumptions:

- The evaluation of the base component is completed before the evaluation of the composed TOE starts;
- The evaluation of the dependent component is completed before the evaluation of the composed TOE is also completed;
- The sponsor of the composed TOE evaluation is the same of the dependent component evaluation.

CC v3.1 define specific tools for composed TOE evaluation, namely:

- The ACO (Composition) class of SARs (Security Assurance Requirements);
- The CAPs (Composed Assurance Packages).

ACO requirements focus analysis on interfaces between dependent and base components. ACO requirements are also assembled with other SARs to constitute the CAPs, which provide an assurance scale for composed TOEs different from the standard EALs (Evaluation Assurance Levels). An alternative to EAL is needed because the dependent component developer does not usually have access to the information necessary to perform an evaluation of both the dependent and base components at EAL2 or above.

The ACO activities require the following inputs:

- Dependent component evaluation evidence;
- Base component;
- Specifications and test documentation of the base component interfaces required by the dependent component to provide its security functions;
- Base component guidance documentation;
- Residual vulnerabilities in the base component, as reported during the base component evaluation (this is required for the ACO_VUL (Composition vulnerability analysis) activities).

The following issues have been pointed out in [ICCC2010] about the TOE composition approach:

- Composed TOEs evaluated with CAP are not easily comparable between each other and to TOEs evaluated using the EAL packages;
- Stepwise (i.e., iterated) composition has not been analyzed;
- The base component documentation required by the ACO activities is not always completely available to the dependent component developer;
- According to the current CCRA Web site, no CAP certification has been issued and no supporting documentation for CAP evaluations is available;
- The maximum attack potential considered for CAP evaluations is Enhanced-Basic.

Composite product evaluation

This approach has been defined in [CCDB-2007-09-01] for smart cards and similar devices. It takes into account a *composite TOE* that includes both HW and SW parts and, for the purpose of evaluation, is divided in 2 components:

- A *Platform* that includes all the HW parts and possibly some SW parts of the composite TOE;
- An *Application* that includes the remaining SW parts of the composite TOE.

For instance, if the composite TOE is a smart card, in one case the platform may consist only of an integrated circuit and the application may consist only

of a card operating system; in another case, the platform may include both an integrated circuit and an operating system and the application may include any additional SW.
It is assumed that the platform has previously been evaluated and certified at some EAL, whereas the application has not necessarily been evaluated.
Security mechanisms provided by the platform are managed by the application and therefore the platform could be considered as the base component and the application as the dependent one, but the TOE composition approach is not used because:

- It does not allow direct comparison with a similar system certified after a single evaluation;
- It is not applicable if resistance to an attack potential higher than Enhanced-Basic is required.

Both these points are serious issues if the TOE is a smart card or a similar device.
[CCDB-2007-09-01] defines CC evaluation of the composite TOE at an EAL that is assumed to be equivalent to the EAL of the platform or lower. For this purpose, ad hoc refinements to some CC v2.x and v3.1 SARs are defined so that the approach is applicable to each EAL.
This approach has several pros against TOE composition, namely

- Higher assurance may be obtained
  - The composite TOE obtains an EAL certification and may in principle resist to any attack potential;
- One evaluation is required instead of two
  - Since the platform has been previously evaluated, EAL evaluation of the composite TOE only is needed whereas the TOE composition approach would require EAL evaluation of the application and CAP evaluation of the system;
- The evaluation process is easier to iterate
  - An evaluated composite TOE can be the platform for a new composite TOE.

On the other hand, the [CCDB-2007-09-001] approach can be applied only under some assumptions, not required by the TOE composition approach, about the nature of the TOE and the cooperation between the different entities involved by the certification process (see [ICCC2010] for details).

Cross evaluation of components

With this approach, assuming that the TOE is made of only two components A and B, two separate evaluations of the single components are performed, preferably with the same assurance level. In each single evaluation, the component B is considered to be part of the Operational Environment (OE) of the component A and vice versa.
The CC evaluation activities refer to the OE by means of Assumptions and Security Objectives for the OE, which are both defined in the Security Target (ST). CC do not require an instance of the OE as an input for evaluation activities, but the model alone of OE derivable from the OE Security Objectives could be insufficient to perform some OE-related evaluation activities (e.g., the evaluator is required to check consistency between the Security Objectives for the OE and the test environment provided during functional testing, independent testing and vulnerability assessment activities). Furthermore, CC

do not provide the end user with means to check if its OE is consistent with the one defined in ST (via Security Objectives for OE).
The component B may be included in the OE of A by executing the following steps when writing specific parts of the ST of A:

- In the Assumptions, B is declared to be part of the OE;
- In the TOE overview, B is defined and its interaction with A is described;
- In the Security Objectives for the OE, B is declared to be in charge of specific Security Objectives.

This approach has several effects. For instance, all the evaluation activities performed on A that must take into account the OE should consider the component B present in the OE. For this purpose, in the ST of A, the characterization of B should be completed by SFRs (Security Functional Requirements) for B that (are also useful to) put constraints in the evaluation of B.
Clearly, the requirement of having both an evaluated version of B when evaluating A and an evaluated version of A when evaluating B can be directly satisfied only in the case where a simultaneous evaluation of A and B is undertaken. Thus, a correct (and possibly partial) implementation of B could be acceptable for the intended purpose. The correctness of such an implementation of B should be assured by suitable tests and, if such tests must be performed during the evaluation of A, suitable evaluation activities must be introduced.
Furthermore, a link between the respective STs of A and B must be established (e.g., by means of suitable Protection Profiles). In particular, to allow cross evaluation of A and B it should be assured that Security Objectives for the TOE in the ST of A include Security Objectives for the OE attributed to A in the ST of B and vice versa (this is extended to SFRs, if SFRs for the OE[1] are included).
The cross evaluation approach requires a deeper analysis since, e.g., it misses the assurance that the evaluation could provide on the effects of the real interaction between components A and B.
A significant example of the cross evaluation approach is provided by the Type 1 and Type 2 Protection Profiles defined by [CWA 14169], where 3 Types of Secure signature-creation device (SSCD) are considered:

- SSCD Type 1: Key pair generation;
- SSCD Type 2: Signature creation;
- SSCD Type 3: Key pair generation and Signature creation.

For each SSCD Type, a PP conform to CC v2.1 has been evaluated.
The cross evaluation of SSCDs of Types 1 and 2 may be used by designers who decide to employ dedicated devices (SSCD Type 1) for (resource-consuming) key pair generation.
In the Type 1 PP, Type 2 SSCD is part of the OE and vice versa. Type 1 and 2 PPs establish a link between conformant STs, i.e., Security Objectives (and SFRs) for the TOE in the Type 1 PP include Security Objectives (and SFRs) for the OE attributed to the Type 1 SSCD in the Type 2 PP and vice versa. By

[1] SFRs for the OE are not defined in CC v3.1, but in [CCpart1v2.3] it is explicitly considered the optional statement in a ST or PP of security requirements (including SFRs) for the IT environment (i.e., in the current CC terminology, the IT part of the OE).

consequence, cross evaluation is possible and Type 1 and 2 SSCDs are separately evaluated (both at EAL 4+).

2.3 Case: Formal methods in Common Criteria

Request

Use of formal methods in Common Criteria certification

Requesting Partner

SIT

Response

FUB consultancy on the use of formal methods in CC certification is reported in the following.

Preliminary remarks

The use of formal methods is mandatory only at the CC Evaluation Assurance Levels EAL6 and EAL7 [CCpart3]. Guidance for the evaluation activities that are mandatory only at EAL6 and EAL7 is not provided by [CEM]. As for evaluation activities that involve formal methods, it must be pointed out that:

- CC provide only some general instructions and a list of possible tools (see "Supplementary CC material on formal methods");
- Any additional guidance should be provided by the single evaluation schemes;
- The Italian Certification Body (OCSI - Organismo di Certificazione della Sicurezza Informatica) has not provided so far any guidance;
- Other evaluation schemes provide specific guidance. For instance, BSI provides CEM-style guidance for some of the relevant activities (see "Use of formal methods in ADV_SPM (Security policy modelling) and BSI guidance").

Formal methods and testing activities

It must be stressed that testing activities are required by CC evaluation even if formal methods are used (testing activities are mandatory at every EAL).

CC definitions of informal, semiformal and formal specifications

Whereas only formal specifications, notations, etc. will be extensively addressed in this consultancy case, also *semiformal* and *informal* specifications, notations, etc. will be mentioned. For the sake of clarity, the CC texts from [CCpart3] that respectively define informal, semiformal and formal specifications are reported in the following.

- *An informal specification is written as prose in natural language.* […] *An informal specification is not subject to any notational or special restrictions other than those required as ordinary conventions for that language (e.g. grammar and syntax). While no notational restrictions apply, the informal specification is also required to provide defined*

meanings for terms that are used in a context other than that accepted by normal usage.

- *The difference between semiformal and informal documents is only a matter of formatting or presentation: a semiformal notation includes such things as an explicit glossary of terms, a standardised presentation format, etc. A semiformal specification is written to a standard presentation template. The presentation should use terms consistently if written in a natural language. The presentation may also use more structured languages/diagrams (e.g. data-flow diagrams, state transition diagrams, entity-relationship diagrams, data structure diagrams, and process or program structure diagrams). Whether based on diagrams or natural language, a set of conventions must be used in the presentation. The glossary explicitly identifies the words that are being used in a precise and constant manner; similarly, the standardised format implies that extreme care has been taken in methodically preparing the document in a manner that maximises clarity. It should be noted that fundamentally different portions of the TSF* [TOE Security Functionality, see "Identification of CC evaluation activities that involve formal methods"] *may have different semiformal notation conventions and presentation styles (as long as the number of different "semiformal notations" is small); this still conforms to the concept of a semiformal presentation.*
- *A formal specification is written in a notation based upon well-established mathematical concepts, and is typically accompanied by supporting explanatory (informal) prose. These mathematical concepts are used to define the syntax and semantics of the notation and the proof rules that support logical reasoning. The syntactic and semantic rules supporting a formal notation should define how to recognise constructs unambiguously and determine their meaning. There needs to be evidence that it is impossible to derive contradictions, and all rules supporting the notation need to be defined or referenced.*

Identification of CC evaluation activities that involve formal methods

Security Assurance Requirements (SARs) that involve formal methods belong to 3 different families of the class ADV (Development). Requirements of this class pertain to the documentation of different aspects of the operation of the TOE Security Functionality (TSF). The TSF consists of all the parts of the Target Of Evaluation (TOE) that contribute to enforce the Security Functional Requirements (SFRs) claimed for the TOE in the Security Target (ST).

In particular, the 3 families of SARs that involve formal methods respectively correspond to 3 different representations of the TSF (some details are given below):

1. In the family ADV_SPM (Security policy modeling) the TSF is represented by a formal model of the TOE security policy;
2. In the family ADV_FSP (Functional specification) the TSF is represented by its functional specification;
3. In the family ADV_TDS (TOE design) the TSF is represented by the TOE design.

For each TSF representation, the corresponding SARs require that:

- The developer provides (as evaluation evidence) some documents that specify the relevant TSF representation and possibly its relationships with other representations;
- Each document provided by the developer includes specific content;
- The evaluator confirms that the developer has provided documents that meet the respective requirements.

Use of formal methods in ADV_SPM (Security policy modelling) and BSI guidance

In practice, the TOE security policy is generally a set of different security policies with different scopes (e.g., access control, information flow, audit, and authentication) that are all to be enforced by the TOE.

For CC evaluation at any level of assurance, the TOE security policy is specified at least at two levels of detail:

1. An informal specification of the TOE security policy is provided by the set of Security Objectives for the TOE claimed in the Security Target (ST);
2. A semiformal specification of the TOE security policy is provided by the set of Security Functional Requirements (SFRs) claimed for the TOE in the ST (under the assumption, which is verified during the evaluation, that such SFRs meet all the Security Objectives for the TOE).

The assurance about the TOE security policy may be increased by an additional formal specification, at least of some of the security policies enforced by the TOE (it is generally recognized that not all security policies can be formally modeled for all TOEs).

The assurance requirements for the formal specification of the TOE security policy are standardized by the ADV_SPM family, which consists of the unique SAR ADV_SPM.1 (Formal TOE security policy model).

In ADV_SPM.1 it is required in particular that

- The developer provides a formal model for one or more security policies enforced by the TOE (selected by the developer) such that
  - The model is in a formal style (supported by explanatory text) and identifies the security policies that are modeled;
  - The model identifies the relevant parts of SFRs that make up each modeled policy;
  - For each modeled policy, the model provides a definition of TOE security and a formal proof that the TOE cannot reach a state that is not secure;
- The developer provides a formal proof of correspondence between the model above and the formal functional specification (this requirement holds only if a formal functional specification is provided, which is required by ADV_FSP.6 (see "Use of formal methods in ADV_FSP (Functional specification)"). The correspondence shows that the functional specification is consistent and complete with respect to the model;
- The evaluator confirms that the developer has provided documents that meet the requirements detailed above.

Notice that ADV_SPM.1:

- Is mandatory at EAL6 and EAL7;

- May be an augmentation (additional SAR) for lower EALs.

BSI provides CEM-style guidance for ADV_SPM.1 evaluation activities in [AIS 34].
BSI has also proposed, in its Guideline [BSIGFSP], a standard terminology for both the constituting parts of the TOE security policy and the corresponding formal model, since such terms are not standardized by CC.
The BSI terminology, which is used in the [AIS 34] guidance to ADV_SPM.1 activities, is the following:

- *Security Principles* (informally called also *rules*) are the general rules of operation of the TOE security policy, derived from the analysis of the security objectives and stated in the SFRs;
- *Security Characteristics* (informally called also *practices*) are all the definitions, attributes, actions etc. necessary for the realization of the general rules of operation of the TOE security policy, formulated in the Security Principles.

In a formal model of the TOE security policy:

- Security Principles are represented by *Security Properties*;
- Security Characteristics are represented by *Security Features*.

Use of formal methods in ADV_FSP (Functional specification)

The functional specification is a TSF representation in terms of the TSF Interfaces (TSFIs). TSFIs are means by which external entities (or subjects in the TOE but outside of the TSF) exchange data with the TSF and invoke services from the TSF.
For any level of assurance, SARs of the ADV_FSP family require in particular that the developer provides:

- A functional specification that describes the TSFIs in terms of their respective purpose, method of use, (input and output) parameters, actions and messages;
- A tracing from the functional specification to the SFRs, i.e., an indication of which TSFI is used to invoke each SFR.

The level of formality (informal, semiformal, formal) of the required documents and the detail of information they have to provide on the TSFIs increase with the assurance level. The highest assurance level corresponds to the SAR ADV_FSP.6 (Complete semi-formal functional specification with additional formal specification), which is the only SAR involving formal methods for the ADV_FSP family.
In ADV_FSP.6 it is required in particular that:

- The developer provides a formal presentation of the functional specification. The formal presentation describes the TSFIs using a formal style (supported by informal explanatory text where appropriate);
- The evaluator confirms that the developer has provided a functional specification that meets the requirement above.

Notice that ADV_FSP.6:

- Is mandatory at EAL7 only;
- May be an augmentation (additional SAR) for lower EALs.

Use of formal methods in ADV_TDS (TOE design)

The TOE design is a representation of how the TSF implements the SFRs. In the TOE design the TSF is described using two levels of decomposition:

1. A Subsystem is a description (at a high level) of how a part of the TOE provides a specific functionality. A subsystem is further decomposed either into lower-level subsystems, or into modules (see below);
2. A Module is the lowest level of description in the TOE design, i.e., a developer should be able to implement the part of the TOE represented by a module with no further design decision.

For any level of assurance, SARs of the ADV_TDS family require in particular that the developer provides:

- A TOE design that describes the TSF subsystems and modules in terms of their respective behavior, interfaces and interactions with other subsystems and modules;
- A mapping from the TSFIs of the functional specification to the lowest level of decomposition available in the TOE design.

The level of formality (informal, semiformal, formal) of the required documents and the detail of information they have to provide on the TSF subsystems and modules increase with the assurance level. The highest assurance level corresponds to the SAR ADV_TDS.6 (Complete semiformal modular design with formal high-level design presentation), which is the only SAR involving formal methods for the ADV_TDS family.

In ADV_TDS.6 it is required in particular that:

- The developer provides a formal specification of the TSF subsystems. The formal specification describes the TSF subsystems using a formal style (supported by informal explanatory text where appropriate);
- The developer provides a proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification (the latter is required by ADV_FSP.6, see "Use of formal methods in ADV_FSP (Functional specification)"; notice that ADV_TDS.6 relies on the presence of ADV_FSP.6 or, in CC terminology, ADV_FSP.6 is a dependency of ADV_TDS.6). The proof demonstrates that all the behavior described in the TOE design is a correct and complete refinement of the TSFI that invoked it;
- The evaluator confirms that the developer has provided documents that meet the requirements detailed above.

Notice that ADV_TDS.6:

- Is mandatory at EAL7 only;
- May be an augmentation (additional SAR) for lower EALs.

Supplementary CC material on formal methods

The following material is provided in Annex A.5 of [CCPart3].

General instructions for the evaluator:

- The evaluator should examine the formal systems used by the developer to make sure that they provide *formal specifications* of the requested kind (see "CC definitions of informal, semiformal and formal specifications");

- If the developer uses a formal system which is already accepted by the relevant evaluation scheme, the evaluator can rely on the level of formality and strength of such a system and focus on its instantiation to the TOE specification (in terms of security policies, TSFI, subsystems, ...) and to the derivation of the relevant proofs;
- The evaluator should examine the explanatory text that accompanies each formal specification to make sure that it is supportive.

List of some specific tools (specification languages and theorem provers) by which formal systems may be implemented:

- Z specification language;
- ACL2;
- Isabelle;
- B method.

Use of formal methods in actual evaluations (some examples)

The BSI Guideline [BSIGFSP] includes also the description of an evaluation conformant to [ITSEC] and involving formal security policies, where:

- The TOE is a signature application for smart cards;
- The tool used is Verification Support Environment (VSE), which is authorized by BSI.

A CC evaluation where formal methods are applied not only to Security policy modeling, but also to functional specification and TOE design (notice that this evaluation is conformant to CC version 2.3 and therefore uses a set of SARs that is slightly different from the one described in this consultancy case) is summarized as follows:

- The TOE is the Java Card System of Usimera Protect V1.0 card on SLE88CFX4000P developed by Gemalto;
- The tool used is Coq;
- The relevant Security Target and Certification Report are available at <www.commoncriteriaportal.org/products>;
- A description is presented in [FM2008].

2.4 Case: Notion of time in Common Criteria

Request

The ontology needs to have input about how the notion of time is included into CC certificates, especially in the security properties.

Requesting Partner

Consortium

Response

FUB consultancy on the notion of time in CC certification is reported in the following. First, the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) defined by CC that deal with time are recalled;

then, some examples of specific time formats used in actual CC evaluations are presented.

Notion of time in CC Security Functional Requirements

According to [CCpart2], these are some families of Security Functional Requirements that deal with time in some sense:

- FTA_TAH (TOE access history) deals with date and time for session establishment;
- FTA_SSL (session locking and termination) supports the specification of time interval of user inactivity;
- FTA_LSA (limitation on scope of selectable attributes) supports the specification of time constraints for session establishment with the TOE;
- FAU_GEN (security audit data generation) deals with date and time for the events to be audited;
- FTA_TSE (TOE session establishment) supports the specification of ranges of time in term of time-of-day, day-of-week, calendar dates;
- FIA_AFL (authentication failures) supports the specification of actions that could include time constraints (e.g., disabling time);
- FPT_STM (time stamps) defines requirements for reliable time stamps generation and it is mandatory for FAU_GEN and FMT_SAE.

Notion of time in CC Security Assurance Requirements

As for [CCpart3], these are some Security Assurance Requirements (SARs) that deal with time:

- The requirement ALC_CMC.5 (Advanced support) (from the family ALC_CMC (life-cycle support)) foresees a CM (Configuration Management) system that supports the audit of all the changes to the TOE including, among others, date and time in the audit trail;
- ALC_FLR (Flaw remediation) support the specification of the procedures (including the response time) that the developer has defined for flaw remediation.

Even if, as general rule, this kind of information regarding SARs should only be available for the evaluator, in some cases the developer could publish it to promote its product or service.

Use of specific time format in actual evaluations

[CCpart2] does not define a specific format for time attributes but some examples could be found in the Security Target (ST) of already certified products.

The Tivoli Access Manager for Operating System ST [TAMOS ST] defines in the login policy (connected to family FMT (Security Management) of Security Functional Requirements) the following time formats:

- "day-range:time-range[:utc|local]" for defining time-of-a-day login restrictions;
- "YYYY-MM-DD[-hh[:mm[:ss]]]" for defining holiday login restrictions.

A precise time value could be included into an SFR that foresees generic parameters, as in the following assignment action for FPT_TDC.1 SFR in [GD-PP]:

- *FPT_TDC.1.2(1) – The TSF shall use [UTC time format] when interpreting the TSF data from another trusted IT product.*

Each CC Certification Report, Security Target or Protection Profile (PP) is full of temporal references for issue date, date of references, date for recognition agreement statements, etc. As an example, [CIM-PP] states that the format of the date of a PP should be "*yyyymmdd*".

2.5 Case: Incremental certification in Common Criteria

Request

We would like to know if CC manages incremental certification, that is, if it is possible to certify evolving software without the need of re-certify software from scratch.

Requesting Partner

UNIMI

Response

Incremental CC certification is managed by Assurance Continuity activities. The minimal technical requirements for such activities to be mutually recognized by the CCRA signatory nations are set by [2004-02-009], whereas each nation may set further requirements for its own implementation of assurance continuity.
In the following, we give a summary of both the general framework defined by [2004-02-009] (see "Assurance continuity") and the specific approach to assurance continuity taken by the Italian Certification Body (OCSI - Organismo di Certificazione della Sicurezza Informatica) (see "Italian Scheme approach").

Assurance continuity

Assurance continuity activities are applied to a *changed TOE* that results from any change to a *certified TOE* (or to its environment), with the purpose of avoiding re-certification from scratch of the changed TOE. Assurance continuity results either in *maintenance* of the certificate already issued for the certified TOE or in *re-evaluation* of the changed TOE. In both cases, assurance continuity aims at reusing as much as possible the results of the earlier evaluation performed for the certified TOE.
Assurance continuity activities involve the following entities:

- the developer;
- the evaluation authority (the same under which the evaluation of the certified TOE was conducted);
- an evaluator (at least in case of re-evaluation).

The developer starts the assurance continuity process by performing an *impact analysis*, i.e., an analysis of the impact of each change upon the assurance that has been gained in the certified TOE. Impact analysis includes identifying the evaluation evidence that has to be updated to reflect the changes, and also

regression testing of the changed code, to be sure that it works when incorporated into the TOE. Impact analysis results in an *Impact Analysis Report (IAR)* provided by the developer to the evaluation authority.
The minimum content of the IAR is defined by [2004-02-009] and consists in:

- An *Introduction* that contains configuration control identifiers for the IAR itself, the changed TOE, the certified TOE, the Certification Report (CR), Evaluation Technical Report (ETR) and Security Target (ST) of the certified TOE, and the developer;
- A *Description of the Changes* to the certified TOE and to its development environment;
- The *Affected developer evidence*, i.e., a list of the items of the developer evidence affected by the changes;
- A *Description of the developer evidence modifications* that briefly describes the required modifications to the affected items of the developer evidence;
- *Conclusions* where the developer reports whether each change has minor or major impact on assurance and whether the overall impact of changes on assurance is minor or major, providing a rationale for each estimated impact.

Minor and major impact of the single change is defined as follows:

- *Minor impact* – The change does not affect the assurance gained in the certified TOE to the extent that new evaluation activities are required for the changed TOE;
- *Major impact* – The change affects the assurance gained in the certified TOE to the extent that new evaluation activities are required for the changed TOE.

In [2004-02-009] no fixed method for identifying major or minor impact is provided but a general guideline is given, with examples of exceptions.
The overall impact of changes on assurance is minor only if all the changes reported in the IAR have minor impact.
Based on the received IAR and possibly on consultation with the developer, the evaluation authority decides whether the impact of each reported change is minor or major. If all the changes reported in the IAR have minor impact (and if re-evaluation is not chosen based on additional factors, see below), the evaluation authority recognizes the changed TOE as a *maintained TOE*, for which the assurance gained in the certified TOE still applies and no additional certification is needed; otherwise, if any reported change has major impact, then re-evaluation is necessary.
It must be noted that:

- Maintenance is not intended to provide assurance about the resistance of the TOE to new vulnerabilities or attack methods discovered since the date of the initial certificate. Such assurance can only be gained through re-evaluation;
- The evaluation authority may choose between maintenance and re-evaluation based also on additional factors beyond impact analysis (e.g., appearance of new vulnerabilities or attack methods (see the previous point), elapsed time since certification);

- The developer may also opt directly for re-evaluation without creating an IAR, but re-evaluation is generally more efficient if the results of an impact analysis are provided as an input by the developer.

An evaluation authority provides maintenance by extending each certificate on its Certified Products List with a *maintenance addendum* that is the list of maintained TOEs for that certificate. If a new maintained TOE is recognized, the relevant maintenance addendum is updated and a *Maintenance Report* for the maintained TOE is also published on the Certified Products List, as an addendum to the relevant Certification Report. The Maintenance Report is derived from the complete IAR, which is generally shared only between the developer and the evaluation authority. Even if not specified by [2004-02-009], an evaluation authority may also publish on its Certified Products List a *Maintenance Security Target* for the maintained TOE (several examples are available on the CC website).

As for re-evaluation, [2004-02-009] does not describe it in details, but stresses that it is performed in the context of an earlier evaluation, therefore reusing any results from that evaluation that still apply. In particular, the parts of the certified TOE that remain unchanged may be identified by the impact analysis. Then all the evaluation activities already performed on the unchanged parts of the TOE need not be performed again in re-evaluation, thereby maximizing the amount of results from the earlier evaluation that can be reused.

Italian Scheme approach

OCSI has defined a Certificates Management Scheme (CMS) to provide a way to extend the validity of a CC Certificate, which can be lost for any change to the certified TOE or to its environment. Significant examples of such changes are the following:

- discovery of new vulnerabilities, which were not known during the vulnerability analysis of the evaluation process;
- inclusion of new functionalities into the TOE;
- update of important sections of the ST (e.g., new security functionalities, changes in security problem definition).

The CMS defined by OCSI is applied to a system certification up to the Evaluation Assurance Level EAL4 only if the Sponsor of the evaluation voluntary joins the CMS before the starting of certification process. In this case, at the end of certification process, OCSI issues a *Live Certificate* (opposite to a *Static Certificate*, for which no maintenance is provided). A live certificate is composed of a CC standard certificate and a set of *addenda*, which are issued by OCSI on the basis of additional evaluation activities to extend the validity of the standard certificate. Addenda may be issued during the period the sponsor chooses to join to CMS, which can be within 1 and 3 years.

A TOE with a live certificate is continuously monitored according to a *Maintenance Plan* submitted by the sponsor and approved by OCSI. In particular:

- TOE correct operation is periodically checked;
- new exploitable vulnerabilities are identified and patched;
- evaluation activities updated is periodically performed, resulting in addenda to the certificate.

In order to schedule and perform this continuous monitoring, each Sponsor joining to the CMS shall indicate:

- a Responsible for the Certificate Management (RCM) who is an evaluator accredited by OCSI, in charge of monitoring the TOE according to the maintenance plan;
- a Security Evaluation Facility (SEF) that is in charge of performing the evaluation activities required for the issuing of the certification addenda.

Note that, although formally each modification either to the TOE or to the related certification evidences would invalidate the certification, in case of a TOE within the CMS the actions performed by the RCM are devoted to reduce the TOE exposure to new vulnerabilities during the *intra-addendum* period between the issue of two subsequent addenda. The certificate however has its full formal validity only at the end of certification process or at the end of each addendum set of activities.

To use an electronic metaphor, the validity of a common CC certificate is like a capacitor discharge: just after the issuing of the certificate the assurance level is the highest, then it decreases over time; on the other hand a CC certificate within the CMS, is like a capacitor discharge with recharging each time an addendum is issued: the assurance level is the highest at the time of the certification but also at the time of each addendum.

A variety of modifications could be performed to a TOE or to its certification documents. OCSI discriminates between major changes and minor changes.

Major changes are substantial modifications to the TOE or to its certification documents that should lead to a new certification and so cannot be performed within the CMS: the CMS process for that TOE is suspended by OCSI. Some examples of major changes could be:

- important changes to the security problem of the Security Target (ST);
- changes to the set of Security Functional Requirements (SFRs);
- raising of the Evaluation Assurance Level.

On the other hand, all minor changes can be managed within the CMS. Examples of minor changes for a TOE can be:

- rewording of phrases into the ST;
- changes to the TOE environment software/hardware with no relevant impact to the TOE interfaces;
- changes to the development environment, such as updating of the configuration management tool;
- software installation for the TOE or its environment, if it does not imply a substantial modification of the threats in the ST.

Finally, it's important to stress that the CMS does not guarantee that the TOE maintains its assurance level all the time, but it guarantees that the TOE is continuously monitored according to a plan approved by the OCSI.

In the same way of Security Assurance Requirements (SARs), also CMS foresees different level of monitoring (then assurance) based on frequency of checks, accuracy of checks, actors that perform checks.

2.6 Case: SOA and Common Criteria (I)

Request

General problem: possible extensions of Common Criteria to apply these to SOA based systems.

Requesting Partner

SAP

Response

Applying Common Criteria to SOA based systems raises specific problems due to several aspects, such as:

- The dynamic nature of the SOA environment;
- The fact that the consumer of a service does not control the environment in which the service operates (Operational Environment).

A contribution concerning the specific problem of the assurance on the Operational Environment (OE) of a service (related to the second aspect listed above) has been included in [ICCC2012] and is summarized in the following.

Assurance on Operational Environment

The Operational Environment (OE) of a CC certified product (Target Of Evaluation (TOE)) is only defined by the Security Objectives (SOs) for the OE stated in the Security Target (ST). It may be divided into:

- IT Operational Environment (HW/SW/FW components that realize the IT part of the SO for the OE);
- Non-IT Operational Environment (Security procedures and physical measures put in place to realize the non-IT part of the SO for the OE).

In CC evaluation the OE is assumed to be a 100% correct instantiation of the SOs for the OE [CCpart1] and the correctness verification of the OE is left to the consumer.

In traditional software provisioning, the consumer of a CC certified software product (e.g., the IT department of an organization) deploys such product in an IT infrastructure that is controlled by the consumer itself. In such a scenario, the consumer is able to implement an OE that is consistent with the ST of the product by configuring the IT infrastructure (i.e., the IT OE) and the IT related processes (i.e., the non-IT OE).

However, in SOA based software provisioning, the consumer searches for a service aimed at performing a certain task and made available by an external provider and then uses a thin client (application) to consume that particular service. In this context, the service consumer:

- Does not own the IT infrastructure of the service provider;
- Cannot control IT related processes put in place by the service provider.

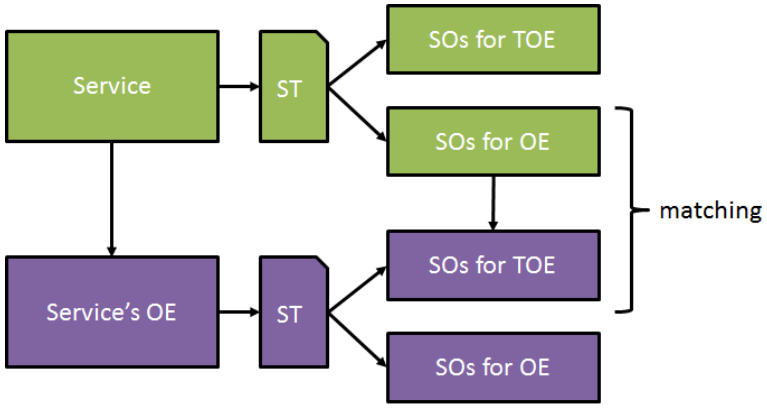
Therefore, when CC certification is applied to services, a new approach to gain assurance on the OE is needed, since the traditional approach cannot provide the required assurance over the OE.

In the following it will be assumed that CC certified services are available with the respective STs (at the moment, this assumption cannot be fulfilled).

Proposed solution

In [ICCC2012] a twofold solution for gaining assurance on the IT and the non-IT part of the OE of a service has been proposed.

Assurance for the IT part of the OE of a CC certified service could be provided by CC certification if the IT infrastructure of the service provider includes CC certified products. In this case, it would be sufficient that the IT part of the Security Objectives for the OE in the ST of the service matches the Security Objectives (SOs) for the TOE in the STs of the products in the environment as shown in the following figure (where the OE is assumed to include only one CC certified product). A similar approach is already adopted in some CC certifications, when a certified TOE B is declared to be part of the OE of another TOE A and to be in charge of specific Security Objectives for the OE of A (see "Cross evaluation of components" in Section 2.2).



In order to determine the above matching, a comparison between SOs from different STs is needed, but such a comparison is generally difficult since SOs are defined in natural language and there is no common vocabulary. The comparison would be simplified if SOs were represented in a structured form. In [ICCC2012] a structure for SOs is proposed and its use for translating and comparing SOs from different existing STs is illustrated by some examples. An example of storage service is also presented, for which structured SOs for the TOE and the OE are derived from structured threats. For this purpose, a structure for threats is proposed which directly comes from [CCpart1].

Providing assurance on the IT part of the OE by writing SOs (and Threats) in a structured form is part of a more general issue of CC extension to SOA. In fact, in a SOA environment the assurance about the services should be automatically processable and for this purpose the ST of a product intended for use in SOA should be written in a form that is structured as much as possible.

Assurance for the non-IT part of the OE of a service, which cannot be based on CC evaluation, could exploit the fact that major cloud service providers have an ISO 27001 certification [ISO27001], which may include the IT related processes they put in place. ISO 27001 allows an organization to define its control

objectives that are evaluated by the Authorized Certification Authorities. Since the results of the certification (i.e., the specific control objectives) are not disclosed publicly, to provide assurance for the non-IT part of the OE of a service, a competent authority should verify that the relevant CC Security Objectives are backed up by corresponding control objectives in the ISO 27001 certificate. Some examples of mapping between CC SO for OE and ISO 27001 Control Objectives are presented in [ICCC2012].

2.7 Case: Assurance Requirements and Policies

Request

Is there available some real examples of assurance requirements and policies, which are used in the traditional software selection process (e.g., in the military and/or governmental fields)? In other words, do you have some examples of assurance requirements and policies used in the Common Criteria certification process and/or used by customers to select software on the basis of their Common Criteria certificates?

Requesting Partner

UNIMI

Response

An overview of assurance requirements defined in different contexts is given in the following with a focus on software testing. First, (quality) assurance requirements explicitly defined for the selection of software to be used by organizations as NASA, ESA and EUROCONTROL are presented. Then, the (security) assurance requirements defined by Common Criteria for both functional and penetration tests are recalled. It is also briefly discussed how Common Criteria can be used to get a third party proof about the (quality) assurance requirements satisfied by a given product. It must be remarked that NASA, ESA. EUROCONTROL and standard Common Criteria assurance requirements are all very high-level and do not mandate specific tests to be performed, so differing from the approach taken in ASSERT4SOA, where the relevant assurance requirements for services (defined in user queries to which ASSERT-E certificates have to correspond) are defined by using a detailed hierarchy of test types (see e.g., [D4.1]). We have found only two examples of detailed assurance requirements related to testing (see below): the NIST CAVP tests for cryptographic algorithms (functional tests) and the security tests for evaluating the OSSTMM RAV (non-functional tests).

NASA Software Assurance Standard

The Technical Standard NASA-STD-8739.8 provides specific requirements for software assurance defined as *the planned and systematic set of activities that ensure that software life cycle processes and products conform to requirements, standards, and procedures* [NASA-STD-8739.8] including the disciplines of

Software Quality, Software Safety, Software Reliability, Software Verification and Validation (V&V) and Independent Verification and Validation (IV&V).
Specific requirements must be implemented by the *Acquirer* (defined as *the entity or individual who specifies the requirements and accepts the resulting software products*), the *Provider* (defined as *the entity or individual that designs, develops, implements, tests, operates, and maintains the software products*) and the *IV&V Facility*.
Some form of tailoring of the requirements is also foreseen. In particular, the rigor of the implementation of the requirements depends on the *class* that is assigned to the relevant software product based on its criticality at the beginning of the assurance process.
The NASA-STD-8739.8 requirements are very high-level and generally state that suitable procedures are followed throughout the software life cycle. This is true also for test-related requirements, some examples of which are quoted below.

7.1.1 Product assurance shall be performed to assure that:
[…]
7.1.1.5 Formal and acceptance software testing are witnessed by software assurance personnel to verify satisfactory completion and outcome.
7.1.1.6 Lower level testing results and software development folders are updated, audited, and/or reviewed for completeness.
[…]
7.1.1.9 The software is verified (e.g., tested, analyzed, measured) for compliance with functional and performance requirements.
[…]
7.3.1 Software assurance shall assure that fault tolerance and redundancy have been specified, implemented correctly, and verified by testing.

No specific test is mandated by NASA-STD-8739.8, which explicitly considers functional tests and some non-functional ones as performance and reliability (see the text emphasized examples above) but no penetration tests.

ESA Software Product Assurance

ECSS-Q-ST-80C Space product assurance – Software product assurance [ECSS-Q-ST-80C] has been developed by ECSS (European Cooperation for Space Standardization), which is a cooperative effort of the ESA with some national space agencies and European industry associations. ECSS-Q-ST-80C may be considered the ESA counterpart of NASA-STD-8739.8 and the two standards show some similarities.
ECSS-Q-ST-80C sets requirements for *software product assurance* defined as *the totality of activities, standards, controls and procedures in the lifetime of a software product which establishes confidence that the delivered software product, or software affecting the quality of the delivered product, conforms to customer requirements*.
Most of the ECSS-Q-ST-80C requirements are addressed to the software supplier, which has to provide evidence of the compliance with the requirements.

ECSS-Q-ST-80C also specifies tailoring of the requirements based on software criticality and defines software categories for this purpose.
The ECSS-Q-ST-80C requirements are very high-level and generally state that suitable procedures are followed throughout the software life cycle. This is true also for test-related requirements, some examples of which are quoted below.

6.3.5.1
a. Testing shall be performed in accordance with a strategy for each testing level (i.e. unit, integration, validation against the technical specification, validation against the requirements baseline, acceptance), which includes:
1. the types of tests to be performed;
NOTE For example: functional, boundary, performance, and usability tests.
2. the tests to be performed in accordance with the plans and procedures;
3. the means and organizations to perform assurance function for testing and validation.

7.2.3.2
a. The test documentation shall cover the test environment, tools and test software, personnel required and associated training requirements.

No specific test is mandated by ECSS-Q-ST-80C, which explicitly considers functional tests and some non-functional ones as performance and usability (see the text emphasized above) but no penetration tests.

EUROCONTROL Safety Assessment

EUROCONTROL provides a process for Safety Assessment for Air Navigation Systems which includes guidelines for definition and verification of SW assurance levels (SWAL). A SWAL establishes a level of confidence that the overall software lifecycle has been conducted in a sufficiently disciplined manner to limit the likelihood of development errors that could impact safety during operations [SAM].
Depending on the SWAL the activities performed during the assessment are conducted at a different depth of analysis:

- SWAL1 at executable level (analysis, verification and evidence performed looking at compilers/linkers output);
- SWAL2 at source code level;
- SWAL3 at architectural design;
- SWAL4 at software requirements level.

Although the whole methodology does not mandate specific tests to be performed, the SWALs have obviously impact also on number, depth and coverage of the tests: the lower the SWAL the higher is the effort to be put into testing activities.

Security Assurance Requirements about Tests in CC

CC define Security Assurance Requirements (SARs) in [CCpart3] for both functional and penetration tests. For the former kind of tests, CC define the security assurance class ATE (Tests) for assessing that the security functions of the Target Of Evaluation (TOE) behave as described in the Security Target (ST) and in the design documentation; for the latter, CC define the security

assurance class AVA (Vulnerability Assessment) for determining whether potential vulnerabilities identified by the evaluator could allow attackers to violate the Security Functional Requirements (SFRs) of the TOE.
In order to allow the evaluator to perform the activities defined in the ATE class, the developer should produce evidence about the functional tests performed on the security functions of the TOE. In particular, the developer should:

- Produce evidence about tests performed, test results and test documentation (ATE_FUN assurance family);
- Produce evidence about the test coverage of the interfaces of the TOE security functions (ATE_COV assurance family);
- Produce evidence about the level of detail of the tests performed on the TOE (ATE_DPT assurance family).

The evidence produced for ATE_FUN SAR family provides assurance that developer has performed and documented correctly the tests for the security functions of the TOE. The test documentation should include information such as test plans and expected and actual test results.
The evidence produced for ATE_COV SAR family provides assurance about the correspondence between the tests described in the test documentation and the TOE security functions described in the development documentation. The deeper is the analysis of the coverage provided by the developer, the higher is the resulting assurance.
The evidence produced for ATE_DPT SAR family provides assurance about the test depth. The deeper the level of detail of the tests performed by the developer (from the TOE security functions interfaces to the source code), the higher is the resulting assurance.
Apart from prescribing the provision of the evidence described before, the CC do not mandate the developer to perform specific tests or to adopt specific test strategies/approaches/policies.
The evidence presented by the developer is the input material for the evaluation activities defined by the CC, which include:

- The analysis of the evidence produced by the developer as result of ATE_FUN, ATE_COV and ATE_DPT families requirements;
- The actual execution of functional tests of the TOE security functions (ATE_IND assurance family);
- The vulnerability analysis and the execution of penetration tests to ensure that the potential vulnerabilities of the TOE discovered cannot be exploited in its operational environment (AVA_VAN assurance family).

The CC also define 7 predefined sets of assurance requirements, the Evaluation Assurance Levels (EALs), which provide an increasing scale that balances the level of assurance obtained with the cost and feasibility of acquiring that degree of assurance [CCpart3].
The increase of assurance from lower EALs to higher EALs results in additional (or stricter) assurance requirements. The following table reports the evidence to be produced by the developer corresponding to the assurance requirements of the ATE class included in each EAL (AVA_VAN requirements are not significant for this table since AVA_VAN activities at any EAL require the developer only to provide the TOE suitable for penetration testing by the evaluator).

EAL	Evidence to be produced by the developer
EAL1	ATE_IND.1: The developer shall prepare the TOE suitable for independent testing by the evaluator.
EAL2	ATE_IND.2: The developer shall prepare the TOE suitable for functional testing by the evaluator (a sample of developer's tests). ATE_COV.1: The developer shall provide evidence of test coverage of the TOE security function (TSF) interfaces in the development documentation. ATE_FUN.1: The developer shall provide test documentation including test plans and expected and actual test result.
EAL3	ATE_IND.2: same as EAL2. ATE_COV.2: The developer shall provide an analysis of test coverage which provides a demonstration that the tests cover all the TSFIs in the development documentation. ATE_FUN.1: same as EAL2. ATE_DPT.1: The developer shall provide an analysis of the depth of testing which provides a demonstration that the tests cover the TSF subsystems.
EAL4	ATE_IND.2: same as EAL3. ATE_COV.2: same as EAL3. ATE_FUN.1: same as EAL3. ATE_DPT.1: same as EAL3.
EAL5	ATE_IND.2: same as EAL4. ATE_COV.2: same as EAL4. ATE_FUN.1: same as EAL4. ATE_DPT.3: The developer shall provide an analysis of the depth of testing which provides a demonstration that the tests cover all the TSF subsystem and modules.
EAL6	ATE_IND.2: same as EAL5. ATE_COV.3: The developer shall provide an analysis of test coverage which provides a demonstration that the tests completely cover all the TSFIs in the development documentation. ATE_FUN.2: The developer shall provide test documentation including test plans, expected and actual test result and an analysis of the test procedure ordering dependencies. ATE_DPT.3: same as EAL5.
EAL7	ATE_IND.3: The developer shall prepare the TOE suitable for functional testing by the evaluator (the whole set of developer's tests). ATE_COV.3: same as EAL6. ATE_FUN.2: same as EAL6. ATE_DPT.4: The developer shall provide an analysis of the depth of testing which provides a demonstration that the tests cover all the TSF subsystems and modules and the TSF implementation representation.

Notice that the EAL packages can be augmented by the developer with the addition of one or more SARs, which may be already defined in the standard CC catalogue or even introduced by the developer by a suitable extension. The augmentation, which must be indicated in the ST, provides additional assurance on a single product, but may decrease the comparability among levels of assurance provided by different products. In principle, the augmentation may be used to include in a CC certification non-CC assurance requirements defined with any level of detail (e.g., requirements from NASA-STD-8739.8 or from the NIST CAVP). The resulting CC certification would provide a third party proof that a product satisfies specific (quality) assurance requirements.

The NIST Cryptographic Algorithm Validation Program (CAVP)

The NIST Cryptographic Algorithm Validation Program (CAVP) provides an example of software assurance based on recommended test vectors. CAVP encompasses validation (functional) testing for software, firmware, or hardware implementations of cryptographic algorithms that are FIPS-approved and/or NIST-recommended (i.e., 1) specified in a FIPS or NIST Recommendation; or 2) adopted in a FIPS or NIST Recommendation; or 3) specified in a list of NIST-approved security functions). Cryptographic algorithm validation by CAVP is a prerequisite to the NIST Cryptographic Module Validation Program (CMVP) and cryptographic modules validated by CMVP are accepted by USA and Canada Federal Agencies for the protection of sensitive information.

All of the CAVP tests are handled by third-party laboratories that are accredited as Cryptographic and Security Testing (CST) Laboratories by the National Voluntary Laboratory Accreditation Program (NVLAP). Vendors interested in validation testing of their algorithm implementation may select any of the accredited laboratories.

A list of the algorithms for which the CAVP currently has validation testing is available on the NIST website [CAVP]. For each algorithm, the list includes links to:

- The cryptographic algorithm specification;
- The testing requirements for validating implementations;
- Sets of test vectors to be used with the testing requirements;
- The list of validated implementations.

The CAVP testing requirements are explicitly designed to test conformance to the specifications. By consequence, they are aimed to detect implementation flaws but not intentional attempts to misrepresent conformance. Thus, CAVP validation should not be interpreted as an evaluation of the overall product security.

OSSTMM RAV

With respect to penetration testing, a possible requirement for software provisioning could come directly from OSSTMM methodology [OSSTMM]: the Risk Assessment Value (RAV). The RAV is the result of a computation based on a number of values of three different risk categories: *operational security*, *controls* and *limitations*. The operational security takes into account the visibility of the system (a value representing the opportunity of an attack), the access to the system (a value counting the interactive access points) and the

trust (a value counting the points allowing unauthenticated interaction). On the other hand, the controls represent the security measures put in place in the system (e.g., authentication, confidentiality, integrity), while the limitations count known vulnerabilities and exposures of the system. The values of the risk categories are assigned during the security tests performed according to OSSTMM and are then combined to obtain the final RAV score.
An organization which wants to acquire software could define an acceptable range of values for the RAV score resulting from a vulnerability assessment using OSSTMM methodology.
This kind of assurance requirement could be a more fine-grained one as compared with the verdict of a CC certification, which is basically "pass/no-pass", even if graduated with different EALs.

2.8 Case: SOA and Common Criteria (II)

Request

General problem: possible extensions of Common Criteria to apply these to SOA based systems.

Requesting Partner

SAP

Response

As already stated in Section 2.6, applying Common Criteria to SOA based systems raises several problems due to the nature of the service domain.
A contribution concerning the specific problem of the assurance continuity for a certified service is summarized in the following.

Dynamic Service Landscape

The change in the nature of software delivery in the SOA paradigm presents significant challenges on providing security assurance about certified software (CC-TOE). In SOA, software is not delivered to the consumer; rather the service provider exposes an interface that is used to invoke the service. Hence neither the consumers nor the Certification Authorities have the means to verify that the running version of the specific service is the certified one. It is thus imperative that the CC certification provides strong assurance to consumers that the service instance being consumed is the certified version.
In addition, the service landscape is very dynamic since threats and vulnerabilities are discovered and evolve very rapidly over time. In such scenarios, certifying a service at a point in time does not provide the required assurance for the consumers since they would need to verify if the certified service at consumption time is affected by any new vulnerabilities or threats that have been discovered after the certification of the service.
Typically, service providers update their services (CC-TOE) accordingly to fix any flaws that are discovered or to counter new threats. Such updates to services, though contributing to the security of the service, technically put the

service in a potentially uncertified configuration. The CC assurance continuity paradigm described in Section 2.5 could be applied in such contexts, but the current practice cannot scale easily to scenarios where there are frequent updates.

Proposed Solution

To address the above issues, the proposed solution tries to define a more dynamic role for the entities involved in the certification process (certification authorities and evaluation labs) that involve service monitoring providing the consumers with more assurance on the exposure of the service in respect of new known vulnerabilities. We refer to this as a "Dynamic Certification Lifecycle".

Additionally, novel mechanisms are introduced, which integrate information from reliable sources, such as the "National Vulnerability Database" (NVD) [NVD] which is maintained by NIST and reports new vulnerabilities that are discovered in a product. This information is used during the certification lifecycle of a product.

In order to aid the Certification Authorities to adapt to this new certification lifecycle, several extensions to the current state of the art are needed. In particular, the extensions impact the software evaluation and certification maintenance processes. The proposed solution involves the use of a monitor residing on the service's Operational Environment, which gathers information about the service's deployment and in case it detects any change to the deployed service, it alerts the certification authorities and the service consumers. In addition, this monitor communicates with the central module controlled by the Certification Authorities that allows the service to be evaluated periodically against the latest relevant vulnerabilities. In this manner, the service consumer can gain assurance on the fact that the service does possess the certified security characteristics at any given point in time.

Notice that a presentation of the concept of Dynamic Service Landscape has been submitted at the International Common Criteria Conference 2013 [ICCC2013] and accepted as an "alternative presentation" (in the end, not given).

2.9 Case: ASSERT Revocation Status

Request

General problem: The project needs a consultancy about possible mechanisms to be used for checking the revocation status of an ASSERT.

Requesting Partner

UMA

Response

Revocation status of an ASSERT

Basically, revocation can be seen as the act of invalidating a certificate due to an asynchronous event before its expiration date. The need for checking the revocation status of an ASSERT arises from some assumptions and requirements for ASSERT4SOA [D7.1] related to the need of checking the validity of an ASSERT (see 8006.ASS_SEC, 4006.FUN, 4007.FUN, 4008.FUN, 4021.FUN, and 4030.FUN in Appendix C). Since the mechanisms used by the ASSERT4SOA Framework for checking the revocation status of an ASSERT depend on the mechanisms used by ASSERT Issuers for notifying the ASSERT status, in the following we refer to the last ones.

A consequence of 8006.ASS_SEC (see Appendix C) is that any mechanism used by ASSERT Issuers for notifying the ASSERT revocation status will have to consider the ASSERT Registry as the authoritative repository for such notification.
Note also that, according to 4008.FUN (see Appendix C), in addition to checking the ASSERT revocation status, optional checks based on user trust preferences may be performed. Information to be optionally checked includes (see 4008.FUN and 4021.FUN in Appendix C):

- The identity of the ASSERT Issuer;
- The service binding specified in the ASSERT;
- The credentials about the ASSERT Issuer competence.

Notice that the management of the credentials about ASSERT Issuer competence has been covered in the consultancy case reported in Section 2.10, where the use of Attribute Certificates [X509], is considered. Notice also that, in principle, revocation affects also ASSERT Issuer credentials (see "ASSERT4SOA scenario").
In the following, we review the main mechanisms used for notifying certificate revocation status information in the identity certificate context and then analyze the suitability of such mechanisms for the ASSERT4SOA context.

PKI Scenario

Certificate revocation is one of the main issues faced by Identity Certification paradigms, such as Public Key Infrastructure (PKI) [X509]. According to [RFC 5280], there are a number of events that could bring to revoke an X.509 Identity Certificate [X509] (e.g., private key compromise and/or Certification Authority compromise). In all these cases, the certificate should become not valid before its expiration date and this revocation status information should be distributed (certificate revocation notification) to all certificate users as soon as possible. Notice that the protection of the integrity of the revocation status information is one of the requirements for the relevant mechanisms. Notice also that the revocation status information should allow the determination of the instant in time when the certificate became invalid.
It is clear that, in a scenario with many users and many certificates, the amount of information to be transferred could become very large and each

approach defined for notifying the revocation status of a certificate shall face this kind of problem.
It is important to notice, however, that in real world certification scenarios the percentage of revoked identity certificates over the total number of issued certificates could be less than 10% (see for example [INFN CA]).
Currently, in PKI using X.509, the two main approaches used to notify the revocation status of certificates are the following:

- Certificate Revocation Lists (CRL) [RFC 5280];
- Online Certificate Status Protocol (OCSP) [RFC 2560].

CRL approach

In CRL approach the revocation status information is distributed by a CRL issuer, which in general is the Certification Authority (CA) itself. A CRL is a time-stamped list identifying revoked certificates (by their serial number) that is signed by a CRL issuer and made available in a public repository [RFC 5280]. A certificate user, after verifying the signature and the time validity of a certificate, should also acquire a "fresh" CRL and verify that the certificate is not in the CRL. CRLs are published periodically by CRL issuers and the requirements for freshness could vary according to users policies.
CRLs have the undisputed advantage of being distributed by the same means as certificates, without the need of a dedicated infrastructure with different capabilities. On the other hand, a user that needs to know the revocation status of a certificate has to download a whole CRL, which usually includes a much larger amount of information than it is actually needed. Another limitation of the basic CRL approach is the time granularity which is limited to the issue period of the CRL (e.g., once an hour, once a day, once a week).
To face these limitations and also the fact that lists are not so effective data structures, an improvement to base CRL approach has been defined: delta-CRL. A delta-CRL is a list of certificates whose revocation status has changed since the issuance of a given CRL. The (complete) CRL is called base CRL. Delta-CRLs are typically smaller than the base ones and so the time intervals between two issues can be shorter.
Another problem to be faced, when implementing an actual delta-CRL distribution system, is that many certificate users need the freshest base-CRL at the same time, causing peaks of requests at the time the base-CRL is published. To mitigate this problem, a possible solution is to over-issue a base-CRL, that is to issue it earlier than the scheduled time.

OCSP approach

In the OCSP approach [RFC 2560], to have a timelier revocation status information, a certificate user does not download any CRL, but issues a status request to a specific server called OCSP responder and suspends the validation of a certificate until a signed response is provided by the responder. The responder can either know the answer to the client's request (by caching it) or ask for it to its neighboring OCSP responders and so on until the process possibly involves the CA which is assumed to know the answer.
It is important to notice that this approach has the drawback of requiring the CA to reply to a huge number of requests, which can be reduced by caching the results of previous requests by the intermediate layers of OCSP responders.

Notice also that OCSP allows a CA to exploit the CRL concept. In fact, a CRL can be used as the ultimate source of revocation information for the OCSP Responders [CRP] (for examples of this, see [CWP]).

ASSERT4SOA scenario

The approach to be used in ASSERT4SOA for notifying the revocation status of ASSERTs should take care of both the relevant assumptions and requirements for the ASSERT4SOA Framework from [D7.1], the previous considerations about identity certificates revocation, and the specification of the following parameters, which should come from an actual ASSERT4SOA context:

1. The number of ASSERTs to be validated (i.e., the total number of ASSERTs being in the ASSERT registries at a given time) and the number of users that need to validate them (i.e., the number of requests arriving to ASSERT4SOA Framework at a given time);
2. The number of ASSERTs to be revoked;
3. The required freshness for the ASSERT revocation status notification.

For the first parameter, since actual figures are not available, let us assume that both the number of certified (ASSERTed) services and the number of their consumers are very large

As for the second parameter, we start from looking at the Common Criteria context for possible requirements about the revocation of ICT security certificates. In current Common Criteria schemes, certificate revocation (*withdrawal* in CC context) could occur under specific conditions, especially if the certificate is under an *assurance continuity* program (for details on CC assurance continuity, see Section 2.5 and [2004-02-009]). The actual list of reasons that could bring to certificate withdrawal can vary from one Common Criteria national scheme to another. As an example, the Italian scheme OCSI indicates a couple of possible reasons: certificate misuse and discovery of some new vulnerability (if the certificate is under the assurance continuity program). When considering the SOA context, which is much more dynamic than the ones typically addressed by Common Criteria, the number of reasons that could bring to changes to certified services (and consequently to certificate revocation and to the need of a new certificate for the service) could be very large (e.g., functional update and/or patching of newly discovered vulnerabilities). So, let us assume that the number of revoked ASSERTs could be very large (the possibility of compensating this by issuing ASSERTs with a very short time validity has not been explored in detail).

Thus, the only real variable seems to be the required freshness, which may be assumed to increase with the criticality of the ASSERTed service.

Based on the previous considerations, the notification of the revocation status of ASSERTs could be based on CRLs, in cases where requirements about notification freshness are not very strict, and, where the notification freshness is instead a priority, an OCSP-based approach should be adopted. As already announced above (see “Revocation status of an ASSERT”), some basic requirements for the implementation of the said approaches in the ASSERT4SOA Framework are

- CRL approach: each ASSERT Issuer should manage specific CRLs, to be included in (referenced from) the ASSERT registry contents;

- OCSP approach: OCSP Responders should ultimately refer to the ASSERT registry, whose contents include (reference to) ASSERTs revocation status notification (possibly in form of CRLs (see the OCSP approach exploiting CRLs sketched in "OCSP approach")).

Finally, since ASSERTs and ASSERT Issuer credentials are both subject to revocation, the use of common mechanisms for notifying/checking their revocation status could also be considered. Even though this aspect has not been fully analyzed, some considerations follow. Apart from the fact that the two contexts are different in nature (ASSERTs and ASSERT Issuers credentials would generally be revoked for different reasons and by different entities and their revocation would have different effects), it appears that, if we restrict the attention to revocation status notification/checking, then mechanisms suitable for ASSERTs should also be adaptable to ASSERT Issuers credentials. This is mainly based on the fact that for ASSERTs it is expected a revocation frequency larger than the one needed for ASSERT Issuer credentials. Further consideration of the revocation of ASSERT Issuer credentials is given in Section 2.10.

2.10 Case: ASSERT Issuer Competence

Request

General problem: The ASSERT language needs a consultancy about expressing and managing competences of ASSERT Issuers and especially about the applicability of ASSERT Profiles for these purposes.

Requesting Partner

UMA

Response

Starting point

The need for expressing and managing the competence of ASSERT Issuers arises from some requirements for ASSERT4SOA [D7.1] related to the need of checking the validity of an ASSERT (see 4006.FUN, 4008.FUN, and 4021.FUN in Appendix C).

In [D1.2], FUB proposed to use an Attribute Certificate (AC) consistent with [X509] as a tool for the suitable management of credentials about the competence of ASSERT Issuers. Basic structures for the management of the relevant ACs in different cases have also been considered in [D1.2] (these structures may be seen as examples of Privilege Management Infrastructure (PMI) [X509] mentioned in the following). A specific structure, which has successively been refined and extended by further UMA work and related FUB consultancy (see "Further work on ASSERT Issuer Competence Support"), relies on the concept of ASSERT Attribute Authority (AAA). An AAA:

- First attests (by proper accreditation procedures) the competence of an entity to act as an ASSERT Issuer;
- Then recognizes (by signing a suitable AC) the entity as an Accredited ASSERT Issuer.

A number of AAAs may exist. Each AAA accredits ASSERT Issuers in the corresponding domain and is accredited in its turn by a Root ASSERT Attribute Authority (RAAA), which recognizes the AAA as such by signing a suitable AC. For this structure, the concept of revocation (see also Section 2.9), to allow an entity issuing ACs to stop the validity of an issued AC, has been basically considered as follows (notice that an AC can be revoked, like any other certificate, by signing a suitable data collection which allows to identify relevant information such as the revoked certificate and the revocation time). Namely, RAAA can revoke the ACs released for each AAA in the structure, whereas an AAA can revoke the ACs released for each ASSERT Issuer in the corresponding domain. Finally, the RAAA is self-accredited by a self-signed AC and its authorization to act so is established and revoked by suitable means. In [D1.2] FUB also showed, for the reference structure, how the different roles (ASSERT Issuer, AAA, RAAA) and their respective competences (see before) could be expressed by attributes consistent with [X509]. Moreover, some examples of attribute refinements have been given. In particular, it has been shown how an AC may be used to restrict the ASSERT Issuer to issue only ASSERTs of specified types (evidence-based (E), model-based (M), ontology-based (O)).

Further work on ASSERT Issuer Competence Support

Starting from [D1.2], UMA has proposed a refined approach for the management of credentials about the ASSERT Issuer Competence. Both the basic PMI structure and the related competence attributes described before (see "Starting point" above) structure have been further analyzed by UMA in an internal document (for convenience, the contents of this document are reported in Appendix D). As a result, an extended PMI with an arbitrary number of levels of AAAs between the RAAA and the ASSERT Issuers has been defined (see Figure 1 in "ASSERT Issuer Competence Support"). In such a structure, the AAAs at any level accredit AAAs in the immediately based on the following *delegation-based* approach. The AC of each AAA includes a set of *privileges*, where a privilege is the ability to issue a specific type (E, M, O) of ASSERTs. Privileges are represented in ACs by *hasCompetence* attributes consistent with [X509]. The AC also specifies the AAA ability to exercise a privilege and/or to delegate it to a lower-level AAA. The abilities to exercise and delegate a privilege are respectively represented in ACs by the *noAssertion* and *basicAttConstraints* extensions, both consistent with [X509]. The following restrictions to privilege delegation/exercise applies:

1. The RAAA does not exercise any privilege included in its AC;
2. An ASSERT Issuer does not delegate any privilege included in its AC.

On UMA request, the proposed delegation-based approach has been analyzed by FUB. The approach, which has been found very similar to the access rights management approach used in SQL databases [SQL], doesn't seem to be

directly suitable for the ASSERT4SOA context. In fact, the approach assumes that a given attribute can only be got by entities owning (at least) that attribute (if these are enabled to transfer that attribute). This means, e.g., that the ability to issue specified ASSERTs can be (possibly) certified only by entities that have the same ability. The effects of this assumption may be clarified by considering that the requirements to be accredited as an ASSERT Issuer can involve aspects which are not covered by the requirements to be accredited as an AAA. For example, to be accredited as an ASSERT Issuer, a specific structure involving evaluation labs could be required, whereas requiring the same structure for an AAA would sound strange. Other practical problems related to possible conflict of interests could also arise.

Temporal validity of an ASSERT

An aspect of the management of the ASSERT Issuer competence credentials is the effect of the loss of validity (for revocation or expiration) of an AC on the validity of other ACs and ultimately on the validity of an ASSERT. Clearly, if a validity period is specified for a certificate (as it is for ASSERTs and X509 ACs), this is not valid at a time not included in the validity period. Again, if a certificate is revoked (as ASSERTs and X509 ACs can be), this is no longer valid from the revocation time on. Notice that, if the relevant certificate is an AC, the expiration/revocation clearly affects the future use of the attributes whereas it is not immediately clear if/how the previous use of the same attributes is affected. Think about an ASSERT, with validity period *t1-t2* (for simplicity, let us assume that the issue time equals *t1*), issued on the basis of an AC with validity period *T1-T2* (for simplicity, let us assume that the issue time equals *T1*). Suppose that, at time t, the validity check of the ASSERT is executed. Apart from the clear cases where t is not included in *t1-t2* or the ASSERT has been revoked before *t*, the ASSERT validity check (if requested (see 4008.FUN and 4021.FUN in Appendix C) involves the validity check of the corresponding AC. The problem now is if the AC validity at time *t* is an actual condition for the ASSERT validity. Notice that, if AC has expired (or been revoked) at time *TT* before *t*, then it is no longer valid at time *t*. Should one derive from this that the relevant ASSERT is not valid at time *t*? Notice that, if AC was valid at time *t1*, then, at least in the period *t1-TT*, one could accept to consider the relevant ASSERT as valid. Moreover, in this case, one could even accept that the ASSERT continues to be valid until it expires (is revoked). So, to support the ASSERT validity at time *t*, one could reduce to ask the AC validity at time *t1*. Notice that the validity of an AC at a given instant in time, when not resolved directly by checking the expiration (or revocation) time, has to be resolved by checking another AC. It seems that the previous considerations can be applied for this check as well. Finally, notice that the said check could also be driven by user preferences (based on suitable policies).

Use of ASSERT Profile as Competence Profile

The FUB consultancy resulted also in considering the use of an ASSERT Profile [D1.3] to express ASSERT Issuer Competence. Recall that an ASSERT Profile defines a class of ASSERTs, formed by all ASSERTs satisfying the rules defined in the relevant ASSERT Profile. Each ASSERT in a given class is by definition

conformant to the ASSERT Profile determining the relevant class. An ASSERT Profile can be, for example, used to define some aspects of the ASSERTs which can be accepted as valid in a given context. Notice that the management of this kind of validity includes the ability to both define a suitable ASSERT Profile and recognize that a given ASSERT is conform to a given ASSERT Profile [D1.3]. It is quite natural to consider the possible use of an ASSERT Profile as a *Competence Profile* to express ASSERT Issuer Competences (the concept of Competence Profile has been first presented at the 10th ASSERT4SOA General Meeting).
When used as a Competence Profile, an ASSERT Profile defines all the ASSERTs that an ASSERT Issuer is recognized as able to issue. Practically, an ASSERT Profile would be included in an Attribute Certificate (AC) of an ASSERT Issuer to express the recognized ASSERT issuance capacity of this ASSERT Issuer. Notice that this is consistent with the preliminary analysis given in [D1.2] (see "Starting point" above), where, e.g., the type of ASSERT (E, M, O) appears to be a possible attribute. In fact, the concept of Competence Profiles generalizes (at some extent) the analysis in [D1.2], since it allows to tune the ASSERT issuance capacity by assigning determined values to determined fields of the ASSERTs that can be issued.
Notice that, in the cases said before (see "Starting point" above), verifying the validity of an ASSERT would include verifying the AC of the Issuer, which, in its turn, would include checking that the ASSERT is conform to the relevant Competence Profile.
It must be noted that the nature of an ASSERT Profile, which has not been initially conceived to be used as a Competence Profile, could cause some limitations in this use. Especially, it must be considered the case where a single Competence Profile cannot capture all the competences to be recognized to an ASSERT Issuer. As an example, consider the case where the issuance capacity has to include ASSERTs of type E and M, whereas an ASSERT Profile should allow to determine only one specified type (either E or M or O). In general, it could result that the expression of complex competences requires a number of Competence Profiles. As a consequence, the management of complex competences could require the management of a number of Competence Profiles to support the verification of the competence of a single ASSERT Issuer.

2.11 Case: Examples of concrete ASSERT Profiles

Request

General problem: The ASSERT language needs a consultancy about realistic ASSERT Profiles.

Requesting Partner

UMA

Response

Candidate ASSERT Profile: "AES-based NIST confidentiality"

A candidate ASSERT Profile has been defined starting from the NIST rules defined by the Cryptographic Algorithm Validation Program (CAVP) [CAVP] (already introduced in Section 2.7). Recall that CAVP encompasses validation testing for software, firmware, or hardware implementations of cryptographic algorithms that are FIPS-approved and/or NIST-recommended. Notice that FIPS-approved and/or NIST-recommended means: specified in a FIPS or NIST Recommendation; or adopted in a FIPS or NIST Recommendation, or specified in a list of NIST-approved security functions. Cryptographic algorithm validation by CAVP is a prerequisite to the NIST Cryptographic Module Validation Program (CMVP).

For this consultancy case, special attention has been paid to the CAVP validation tests defined for the block cipher AES when used in a mode of operation that is approved by NIST for confidentiality (a *mode of operation* of a block cipher is a construction based on a block cipher which specifies how the block cipher is used to execute encryption and decryption). As a result, it has been given a systematic description for an ASSERT Profile that would enforce the NIST rules to claim conformance to AES for confidentiality.

Informally, the rules to be implemented by such a profile, which has been called *AES-based NIST confidentiality*, are the following:

- The block cipher must be AES (specified by [FIPS 197]) (AES is one of the block ciphers approved by NIST [FIPS 140]);
- The mode of operation must be one of the 6 modes approved by NIST for confidentiality based on the approved block ciphers [BCM], i.e.: ECB, CBC, CFB, OFB, CTR (all specified by [NIST SP 800-38A]), XTS-AES (specified by [NIST SP 800-38E]);
- The functional tests must be: AESAVS (Advanced Encryption Standard Algorithm Validation Suite) [AESAVS] for the modes specified by [NIST SP 800-38A]; XTSVS (XTS-AES Validation System) [XTSVS] for XTS-AES specified by [NIST SP 800-38E]. Corresponding NIST Test Vectors for AESAVS and XTSVS are given in [CAVP].

AESAVS and XTSVS are part of the CAVP.

The successful completion of the tests contained within the AESAVS is required to claim conformance to the AES as specified in [FIPS 197] [AESAVS] (it must be noted that, instead of the CFB mode, the AESAVS validates 3 modes of operation that correspond to CFB with 3 specified values of the parameter *data segment*).

The successful completion of the tests contained within the XTSVS and the AESVS* is required to be validated as conforming to the XTS-AES algorithm standard [XTSVS] (*The XTS-AES algorithm validation process requires additional prerequisite testing of the underlying AES implementation via the AESVS).

Apart from rules related to functional tests, all rules allow a single choice (all these single choices determine a *mechanism*) in a defined set. In fact, there is an option which allows more than this, which in the following will be referred to as the *multiple choice option*.

For each mechanism, the set of functional tests to be performed for validation is uniquely specified by the relevant validation suite [AESAVS] or [XTSVS] and the corresponding test vectors retrievable from the NIST web site [CAVP]. There are no options here: for a specified mechanism, the set of functional tests to be executed and corresponding test vectors is fixed.
The actual rules for AES-based NIST confidentiality are defined as follows.

- Rule 0 (block cipher): The block cipher (part of the mechanism specification) must be AES;
- Rule 1 (mode of operation): The mode of operation must be in {ECB, CBC, CFB1, CFB8, CFB128, OFB, CTR, XTS-AES} [AESAVS, XTSVS], where
    - CFB1 is CFB where the parameter length of the data segment is 1 bit [AESAVS];
    - CFB8 is CFB where the parameter length of the data segment is 8 bits [AESAVS];
    - CFB128 is CFB where the parameter length of the data segment is 128 bits [AESAVS];
- Rule 2 (ciphering direction): The ciphering direction must be in {Encryption, Decryption} [AESAVS, XTSVS];
- Rule 3 (key size if not XTS-AES): If the selected mode is not XTS-AES, the key size must be in {128, 192, 256} [AESAVS];
- Rule 4 (key size if XTS-AES): If the selected mode is XTS-AES, the key size must be in {256, 512} [XTSVS]. Notice that
    - 256 bit key size is denoted XTS-AES-128 [XTSVS];
    - 512 bit key size is denoted XTS-AES-256 [XTSVS];
- Rule 5 (tweak format if XTS-AES): If the selected mode is XTS-AES, the tweak format (format of the tweak value input) must be in {128-bit hexadecimal string, Data Unit Sequence Number} [XTSVS], where
    - Data Unit Sequence Number is a decimal number in [0, 255] [XTSVS];
- Rule 6 (referenced validation if XTS-AES): If the selected mode is XTS-AES, a referenced validation based on AESAVS of the underlying AES implementation must be indicated and the following additional rules apply to the referenced validation [XTSVS]:
    - Rule 6.1 (referenced validation mode of operation if XTS-AES): In the referenced validation the mode of operation must be
        - In {ECB, CBC, CFB1, CFB8, CFB128, OFB, CTR} if the selected ciphering direction for XTS-AES is Encryption;
        - In {ECB, CBC} if the selected ciphering direction for XTS-AES is Decryption;
    - Rule 6.2 (referenced validation ciphering direction if XTS-AES): In the referenced validation the ciphering direction must be
        - Encryption if the selected ciphering direction for XTS-AES is Encryption;
        - Encryption and Decryption (*multiple choice option*) if the selected ciphering direction for XTS-AES is Decryption;
    - Rule 6.3 (referenced validation key size if XTS-AES): In the referenced validation the key size must be

    - 128 if the selected key size for XTS-AES is 256;
    - 256 if the selected key size for XTS-AES is 512;
- Rule 7 (functional tests): The functional tests must be the ones specified by NIST for the selected mechanisms (see above).

As for the implementation of these rules, there are several points where different options may be considered (examples are given below). First of all, it must be chosen how to enforce the relevant rules in the actual ASSERT Profile. This may be done by:

1. Representing rules completely within the ASSERT Profile (e.g., enforcing that a mechanism is specified by choosing given values for given parameters);
2. Representing rules partially within the ASSERT Profile (e.g., enforcing that a mechanism is specified by choosing, for given parameters, values which are referred to by links to external materials);
3. Using only links to external materials.

Each option has pros and cons to be considered. These are given in the following notes. If option 1 is selected:

- Note 1: Representing rules for functional tests would require to report in the ASSERT Profile the description of both tests to be executed and corresponding test vectors;
- Note 2: If NIST specifications are changed, a new ASSERT Profile has to be issued.

If option 2 is selected:

- Note 3: Note 2 still applies (partially, where links to external materials are not used);
- Note 4: It must be defined a way to specify which part(s) of a document contain(s) the relevant information.

If option 3 is selected:

- Note 5: Note 4 is even more significant here (it applies to all the information included in the ASSERT Profile).

Another point where different options are possible is how to manage the reference to the preliminary validation required for the XTS mode (referenced validation). An option is to include a link to an additional ASSERT, which must be itself compliant to the "AES-based NIST confidentiality" profile. This case is probably interesting for the ASSERT validation, since an ASSERT compliant with a profile refers to another ASSERT which, in its turn, is compliant with a specific set of rules of the same profile. Other possibilities could also be considered, e.g., using a link to the NIST list of validated implementations available in [CAVP].

Still another point to address is how to manage the *multiple choice option* (it appears from the rules above that multiple choices are allowed by NIST for mode of operation, key size, and ciphering direction and that for the XTS mode the concept of multiple choice applies also to the referenced validation). The problem is clearly related to the kind of rules which can be defined in an ASSERT Profile (which, in its turn, is related to the type of values admitted for an ASSERT field). It seems that, apart from rules allowing, for a given ASSERT field, a given value (e.g., key size = 128) or a single value in a given set (e.g., key size = 128 OR 256), the multiple choice option would require the ability of

using rules allowing the combination of values in a given set (e.g., key size = 128 OR 256 OR (128 AND 256)). Notice that the said ability would simplify the management of the certification of complex implementations where more than one mechanism is used. Notice also that, for such complex implementations, it is reasonable to consider, in view of a security certification, an additional configuration mechanism to be used for the static/dynamic selection of the single mechanism out of the available ones. Finally, notice that no rules are given by NIST for such configuration mechanism (in the addressed context).

Extensions from "AES-based NIST confidentiality" ASSERT Profile

Starting from "AES-based NIST confidentiality" new profiles could be created, using a similar approach. Two examples have been considered in some detail:

1. Confidentiality Profiles;
2. AES Confidentiality/Authentication Profiles.

Generalization 1: Block cipher-based confidentiality by NIST

What done for AES could be repeated *mutatis mutandis* for TDEA (specified by [NIST SP 800-67]) and Skipjack (specified by [Skip]). Two specific profiles could be created by using the know-how from the AES confidentiality case.
In view of useful ASSERT Profiles, it could be interesting to analyze two different cases, which can be described as follows:

1. All the rules relative to AES, TDEA, and Skipjack are collected into a single ASSERT Profile "BlockCipher-based NIST confidentiality 1" where a preliminary rule allows a single* choice about the algorithm and then distinct sets of rules are to be applied, based on the initial choice (*A single choice is considered here to be consistent with the NIST CAVP context);
2. Three distinct ASSERT Profiles exist ("AES-based NIST confidentiality", "TDEA-based NIST confidentiality", and "Skipjack-based NIST confidentiality") and these are referenced in another ASSERT Profile ("BlockCipher-based NIST confidentiality 2") containing the preliminary rule said before and links to specific profiles depending on the initial choice.

In any case, rules will be applied to the same objects (block cipher, mode of operation, etc.) considered for AES-based Confidentiality. Informal examples of rules are the following:

- Generalized rule on block cipher: it must be one of those approved by NIST [FIPS 140], i.e.,
  - AES (specified by FIPS 197);
  - TDEA (specified by [NIST SP 800-67]);
  - Skipjack (specified by [Skip]);
- The rules on mode of operation, ciphering direction, key size, etc. will generally depend on the selected block cipher: for instance, if the selected block cipher is Skipjack, the ciphering direction must be decryption (Skipjack is allowed for legacy-use for decryption only [NIST SP 800-131A]);

- The rule on functional tests will include also those for TDEA and Skipjack, which (as those for AES) are part of the CAVP (all the corresponding NIST Test Vectors are given in [CAVP]):
    - TMOVS (Modes of Operation Validation System for TDEA) [TMOVS] with MMT (Multi-block Message Text) [MMT] for TDEA;
    - MOVS (Modes of Operation Validation System) [MOVS] for Skipjack.

Generalization 2: AES-based confidentiality and/or authentication by NIST

What done for AES Confidentiality could be repeated *mutatis mutandis* for modes of operation supporting Authentication and Confidentiality and Authentication Two specific profiles could be created by using the know-how from the AES confidentiality case.
In view of useful ASSERT Profiles, it could be interesting to analyze two different cases, which can be described as follows:

1. All the rules relative to all modes of operation of AES* are collected into a single ASSERT Profile ("AES-based NIST confidentiality/authentication 1") where a preliminary rule allows a multiple choice for the security property to be supported (confidentiality, authentication, confidentiality&authentication) and then distinct sets of rules are to be applied, based on the initial choice (*The modes for confidentiality are already covered in "AES-based NIST confidentiality");
2. Three distinct ASSERT Profiles exist ("AES-based NIST confidentiality", "AES-based NIST authentication", and "AES-based NIST confidentiality&authentication") and these are referenced in another ASSERT Profile "AES-based NIST confidentiality/authentication 2") containing the preliminary rule said before and links to specific profiles depending on the initial choice.

In any case, with respect to AES-based confidentiality, a rule will be given for an additional object: the security property to be enforced by AES. The related rule may be informally stated as follows:

- Rule on security property: it must be one of
    - Confidentiality;
    - Authentication;
    - Confidentiality and Authentication.

The rules on mode of operation, ciphering direction, key size, etc. will generally depend on the selected security property. Some of these rules are informally sketched in the following:

- Rules on mode of operation
    - If the selected security property is Confidentiality, the mode of operation must be one for AES-based confidentiality (see “Candidate ASSERT Profile: "AES-based NIST confidentiality" above);
    - If the selected security property is Authentication, the mode of operation must be CMAC (specified by [NIST SP 800-38B]) which is the only mode approved by NIST for authentication based on the approved block ciphers [BCM];

  - If the selected security property is Confidentiality and Authentication, the mode of operation must be one of CCM (specified by [NIST SP 800-38C]), GCM (specified by [NIST SP 800-38D]), KW, KWP (both specified by [NIST SP 800-38F]) (5 combined modes are approved by NIST for confidentiality and authentication based on the approved block ciphers [BCM] but TKW (also specified by [NIST SP 800-38F]) is based on TDEA);
- Rules for referenced validation for CMAC, CCM and GCM
  - If the selected mode is CMAC, a referenced validation based on AESAVS of the underlying AES implementation must be indicated and additional rules (to be defined) apply to the referenced validation [CMACVS];
  - If the selected mode is CCM, a referenced validation based on AESAVS of the underlying AES implementation must be indicated and additional rules (to be defined) apply to the referenced validation [CCMVS];
  - If the selected mode is GCM, a referenced validation based on AESAVS of the underlying AES implementation must be indicated and additional rules (to be defined) apply to the referenced validation [GCMVS];
- The rule on functional tests will include also those for CMAC, CCM and GCM, which are part of the CAVP (all the corresponding NIST Test Vectors are given in [CAVP])
  - CMACVS (CMAC Validation System) [CMACVS] for CMAC;
  - CCMVS (CCM Validation System) [CCMVS] for CCM;
  - GCMVS (Galois/Counter Mode (GCM) and GMAC Validation System) [GCMVS] for GCM.

Chapter 3

Consultancy Exploitation

The cases reported in Chapter 2 describe significant examples of topics addressed during the *Continuous Consultancy* offered along the whole project.

The responses provided to the consultancy requests have been exploited in several ways, including understanding in depth some specific topics related to ICT security certification and aligning the project choices with current real world ICT security certification approaches (mainly the Common Criteria one).

It results that especially the activities related to work packages WP1 (ASSERT model and language), WP4 (ASSERT-E: Evidence based certificate), and WP5 (ASSERT-M: Model based certificate) have taken advantage from the consultancy provided by FUB. The exploitation of the consultancy reports can be categorized as follows:

- consultancy reports exploited to refine the definition of the ASSERT language given in [D1.4];
- consultancy reports exploited to effectively drive the activities focused on the structure and handling of ASSERT4SOA evidence-based certificates and related to [D4.2] and [D4.3];
- consultancy reports exploited to suitably address some aspects of model–based certification [D5.1];
- consultancy reports exploited to assess the applicability of some ASSERT4SOA concepts in Common Criteria, disseminating the results in an international context, such as ICCC.

In the following, based also on specific feedback received by the involved project partners, we report example exploitations related to some of the described consultancy cases.

Case: Incremental certification in Common Criteria
The management of security certification of evolving services is a critical requirement especially in a scenario where services are subject to continuous

refinements. Uncontrolled changes in fact can easily invalidate existing certification results and require re-certification from scratch, with high costs and overheads on service providers. In this context, the need of supporting the incremental certification of evolving services and re-use, as much as possible, the certification evidence available from older certificates in the release of a new certificate arises. The consultancy on “incremental certification in Common Criteria” was therefore aimed to evaluate how Common Criteria manages incremental certification in the context of traditional software development. The reply to the above question clearly shows that, although some effort have been done in the context of Common Criteria, the proposed approach is still far from being usable in a service-based ecosystem. Following the above discussion, the consultancy has been exploited to prepare and develop a solution based on testing to incremental certification of services. The proposed solution has been presented in Deliverable D4.3 (M36) and has been the subject of the research paper “M. Anisetti, C.A. Ardagna, E. Damiani, "A Low-Cost Security Certification Scheme for Evolving Services," in *Proc. of the 19th IEEE International Conference on Web Services (ICWS 2012)*, Honolulu, HI, USA, June, 2012”. The paper presented at ICWS 2012 describes a solution for the management of incremental certification of evolving services and responds to the need of providing a low-cost certification scheme for evolving services, which does not prescribe re-certification from scratch. The paper presents a scheme that re-uses existing certificates and related evidence limiting the amount of new and additional certification activities. It concentrates on test-based evidence, and aims to minimize test generation and execution activities for certifying evolving services.

Case: Assurance Requirements and Policies

Certification and selection of software are processes traditionally managed by expert users with few (if any) automatic support by IT technologies. Service providers usually use certification schemes like Common Criteria to manually evaluate functional and non-functional properties of their software and to produce a human-readable certificate, while software procurement is managed by expert users that access human-readable certificates and software specifications to select among alternative software the best that fits their requirements. These processes are driven by assurance requirements and policies that in the first case represent the objectives of certification, while in the second the requirements on the software to be selected. In this context, it is always difficult to understand real assurance requirements and policies. The request for consultancy goes in this direction and aims to clarify a common ground for assurance requirements and policies. The main goal of this consultancy was to build on common practices in the definition of assurance requirements and policies, to the aim of defining a tool for the discovery and selection of services that fully address the needs of users specifying requirements on service certification. This consultancy has been exploited to define the matching and comparison solution and tool described in Deliverable D4.2. The proposed solution has also been presented in the research paper “M. Anisetti, C.A. Ardagna, E. Damiani, "Security Certification-Aware Service Discovery and Selection," in Proc. of the 5th IEEE International Conference on

Service-Oriented Computing and Applications (SOCA 2012), Taipei, Taiwan, December, 2012". The paper presented at SOCA 2012 describes a security-enhanced solution to the discovery and selection of services. This solution integrates a test-based certification scheme proving the security properties of services. In particular, it describes two algorithms enabling clients to select the service that best addresses their security requirements. It also presents a prototype implementation of the approach and an experimental evaluation of its performances.

Case: Applying Common Criteria to SOA

One of the very important aspects for SAP was to study the limitations of the current certification practices when applied to Service Oriented Architectures. In this regard, FUB expertise in Common Criteria certification was useful in the analysis of the CC certification approach and for proposing several extensions to the current certification practice that would enable CC certification to cater to SOA based products. This work on the extensions to the CC certification practice, i.e., assurance over operational environment and structured security target concept, has been successfully presented in the International Common Criteria Conference 2012 [ICCC2012].

Another important consultancy topic was about current CC certification approach to Assurance Continuity. The consultancy results were valuable to identify possible limitations to current approach and to develop the solutions to overcome these limitations which resulted in the concept of "Dynamic certification Lifecycle". This work has been submitted at the International Common Criteria Conference 2013 [ICCC2013] and accepted as an "alternative presentation" (in the end, not given).

Case: ASSERT Revocation Status

An important result of this consultancy case was the fact that certificate revocation is primarily defined for and used in the domain of PKI/PMI [X509] where user identity and authorization impersonation is of a primary concern. However, since ASSERT certificates are security assurance certificates for services, the notion of certificate misuse for user/service impersonation is not applicable.

The consultancy results were very useful to align on potential technologies taken from PKI/PMI domain such as Certificate Revocation Lists (CRLs) [RFC 5280] and Online Certificate Status Protocol (OCSP) [RFC 2560] that technologically enable the means to provide certificate revocation status information within the project framework, but *conceptually* care should be taken of when and how to model, define conditions under which ASSERT revocation is viable. The concluding remark of the consultancy case used in the ASSERT model [D1.4] was that given the domain of digital certificates for security assurance of services is very recent, the concept of certificate revocation is still subject to research to define the *best practices* under which revocation of such certificates is viable.

Case: ASSERT Issuer Competence

The consultancy of this aspect led UMA to accurately elaborate the issuer competence support as part of the ASSERT model and language v3 [D1.4]. Particularly, the adoption of X.509 attribute certificate standard [X509] as a means to express, encapsulate issuer competence information was a useful result that facilitated significantly the way issuer competence management and revocation are addressed by means of the X.509 standard. In that way, a privileged management infrastructure (PMI) can be used *transparently* to the accreditation support process as a means to handle more advanced management and delegation of issuer accreditation among different administrative domains.
Another important (very useful) result of the consultancy case was the definition of the concept of an issuer "*profile competence*". Particularly, the use of the ASSERT Profile concept to express and *limit* an entity's competence to issue ASSERTs conformant to a profile. In that way, by using competence profiles, one can achieve a *fine-grained* and flexible expression of issuer's competence on various aspects of certification. The results of the profile competence approach are reported in deliverable [D1.4].

Case: Examples of concrete ASSERT Profiles
The consultancy of this aspect let UMA further validate and verify the usefulness of the concept of ASSERT Profile on some widely recognized by security community requirements and evaluation activities on cryptographic algorithms. Particularly, UMA was given a systematic description for a possible profile defining AES-based confidentiality properties strictly adhering to the approved by NIST security requirements and validation test suits necessary to *claim* conformance to AES, and subsequently define any AES-based confidentiality properties.

Chapter 4

Conclusions

The consultancy offered by FUB according to Task 7.4 (Continuous Consultancy about 'Certificational' Requirements) of WP7 (Requirements, specification and validation of ASSERT4SOA) [DoW], provided, *on demand* and along the whole project duration, responses to specific questions coming from the remaining partners. Clearly, the continuous consultancy aimed at keeping the project evolution aligned with the real world of ICT security certification.

According to the feedback received from the remaining project partners, the objectives of the continuous consultancy have been reached. In fact, as reported in Chapter 3, the responses provided under Task 7.4 have effectively driven some ASSERT4SOA activities, such as the development of the solution for evidence-based certificates management and the definition of the third version of the ASSERT language. Moreover, some consultancy cases have been exploited to assess the usefulness and the applicability of some ASSERT4SOA concepts within the Common Criteria context.

Appendix A: CC Terminology

Definitions of some CC concepts used throughout this document are reported in the following with the respective sources (some definitions have been adapted or completed with notes for clarity).

- Adverse actions – actions performed by a threat agent on an asset [CCpart1]
- Assets – entities that the owner of the TOE presumably places value upon [CCpart1]
- Assumption – assumption made on the OE in order to be able to provide security functionality [CCpart1]
- Attack potential – measure of the effort to be expended in attacking a TOE, expressed in terms of an attacker's expertise, resources and motivation [CCpart1] (NOTE: The attack potential may be rated as Basic, Enhanced-Basic, Moderate or High [CEM])
- Augmentation – addition of one or more requirement(s) to a package [CCpart1] (NOTE: the package resulting from an augmentation is called augmented package)
- Base component – entity in a composed TOE, which has itself been the subject of an evaluation, providing services and resources to a dependent component [CCpart1]
- CCRA – Arrangement on the Recognition of Common Criteria Certificates in the field of IT Security [CCpart1]
- Class – set of CC families that share a common focus [CCpart1]
- Component – smallest selectable set of elements on which requirements may be based [CCpart1] (NOTE: requirements here must be intended as SFRs or SARs. The terms functional components and assurance components to denote SFRs and SARs, respectively, are also used in the CC)
- Composed Assurance Package (CAP) – assurance package consisting of requirements drawn from CC part3 (predominately from the ACO class), representing a point on the CC predefined composition assurance scale [CCpart1] (NOTE: Values in the CC predefined composition assurance scale range from CAP-A to CAP-C, in order of increasing assurance [CCpart3])
- Composed TOE – TOE comprised solely of two or more components that have been successfully evaluated [CCpart1]
- Configuration management – discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements [CCpart1]

- Dependency – relationship between components such that if a requirement based on the depending component is included in a PP, ST or package, a requirement based on the component that is depended upon must normally also be included in the PP, ST or package [CCpart1]
- Dependent component – entity in a composed TOE, which is itself the subject of an evaluation, relying on the provision of services by a base component [CCpart1]
- Developer – organisation responsible for the development of the TOE [CCpart1]
- Element – indivisible statement of a security need [CCpart1] (NOTE: SFRs and SARs are built upon functional elements and assurance elements, respectively. More details about this are given in Appendix B)
- Evaluation Assurance Level (EAL) – set of assurance requirements drawn from CC part3, representing a point on the CC predefined assurance scale, that form an assurance package [CCpart1] (NOTE: Values in the CC predefined assurance scale range from EAL1 to EAL7, in order of increasing assurance [CCpart3])
- Evaluation authority – body that sets the standards and monitors the quality of evaluations conducted by bodies within a specific community and implements the CC for that community by means of an evaluation scheme [CCpart1]
- Evaluation scheme – administrative and regulatory framework under which the CC is applied by an evaluation authority within a specific community [CCpart1]
- Evaluation Technical Report (ETR) – report that documents the overall verdict and its justification, produced by the evaluator and submitted to an evaluation authority [CEM]
- Extension – addition to an ST or PP of functional requirements not contained in CC part2 and/or assurance requirements not contained in CCpart3 [CCpart1] (NOTE: The requirement resulting from an extension is called extended requirement)
- Family – set of components that share a similar goal but differ in emphasis and rigour [CCpart1] (NOTE: Component here (see the relevant definition) is to be intended as SFR or SAR)
- Guidance documentation – documentation that describes the delivery, preparation, operation, management and/or use of the TOE [CCpart1]
- Module – the lowest level of description in the TOE design. A developer should be able to implement the part of the TOE represented by a module with no further design decision [CCpart3]
- Operational Environment (OE) – environment in which the TOE is operated [CCpart1]
- Organisational Security Policy (OSP) – set of policy rules, procedures or guidelines for an organization. A policy may pertain to a specific OE [CCpart1]
- Overall verdict – *pass or fail* statement issued by an evaluator with respect to the result of an evaluation [CEM]

- Package – named set of either SFRs or SARs [CCpart1] (NOTE: The terms functional package and assurance package to denote packages of SFRs and SARs, respectively, are also used in the CC)
- Protection Profile (PP) – implementation-independent statement of security needs for a TOE type [CCpart1] (NOTE: In practice, a PP is intended as a template for any ST of a TOE of the considered type)
- Security Assurance Requirement (SAR) - specification of some evaluation activities that concur in providing assurance that the TOE meets the SFRs claimed in the ST [CCpart1] (NOTE: The semiformal notation to be used in SARs is standardized by [CCpart3], which also provides a catalogue of standard SARs; more details about SARs are in Appendix B)
- Security Functional Requirement (SFR) – specification (at a lower level with respect to a Security Objective) of some IT security measure [CCpart1] (NOTE: The semiformal notation to be used in SFRs is standardized by [CCpart2], which also provides a catalogue of standard SFRs; more details about SFRs are in Appendix B)
- Security Objective – statement of an intent to counter identified threats and/or satisfy identified organisational security policies and/or assumptions [CCpart1]
- Security Objective for the OE – informal definition of some IT or non-IT security measure that has to be implemented by the OE to counter identified threats and/or satisfy identified assumptions and/or OSPs [CCpart1]
- Security Objective for the TOE – informal definition of some IT security measure that has to be implemented by the TOE to counter identified threats and/or satisfy identified OSPs. A Security Objective is implemented by the TOE by means of a set of SFRs [CCpart1]
- Security Target (ST) – implementation-dependent statement of security needs for a specific identified TOE [CCpart1] NOTE: as stated in [CCpart1], the ST is provided by the developer to the evaluator and includes
    - ST introduction (including a ST reference and a TOE reference that respectively identify the ST and the TOE, a TOE overview and a TOE description)
    - Conformance claims (identifying the CC version and the possible packages and PPs to which the ST is conformant; as for packages, a ST is generally conformant to a (possibly augmented) EAL)
    - Security Problem Definition (it generally includes both Threats, OSPs and Assumptions)
    - Security Objectives claimed, including
        - Security Objectives for the TOE
        - Security Objectives for the OE
    - Extended component definition (optional: it defines the components possibly needed to define extensions of SFRs and/or SARs)
    - Security Requirements claimed for the TOE, including

    - Security Functional Requirements (SFRs)
    - Security Assurance Requirements (SARs)
  - TOE Summary Specification (it shows how the claimed SFRs are implemented by the TOE)
- Subsystem – description (at a high level) of how a part of the TOE provides a specific functionality. A subsystem is further decomposed either into lower-level subsystems, or into modules [CCpart3]
- Target Of Evaluation (TOE) – set of software, firmware and/or hardware, possibly accompanied by guidance [CCpart1]
- Threat – threat to be countered by the TOE, its OE, or a combination of the two. It is identified by a triple (adverse action, threat agent, asset) [CCpart1]
- Threat agent – entity that can adversely act on assets [CCpart1]
- TOE Security Functionality (TSF) – combined functionality of all hardware, software, and firmware of a TOE that must be relied upon for the correct enforcement of SFRs [CCpart1]
- TOE security policy (TSP) – a set of rules that regulate how assets are managed, protected and distributed within a TOE [CCpart1v2.3] (NOTE: A standard CC definition of TOE security policy is no longer provided in CC v3.1)
- TSF Interface – means by which external entities (or subjects in the TOE but outside the TSF) supply data to the TSF, receive data from the TSF and invoke services from the TSF [CCpart1]

Appendix B: Introduction to CC Security Requirements

Preliminary note

In this Appendix, the concepts of Security Functional Requirement (SFR) and Security Assurance Requirement (SAR) are introduced by explaining their structure and taxonomy (Notice that, it is Common Criteria practice to refer to SFR and SAR also as *functional components* and *assurance components*, respectively).

Structure and taxonomy of SFRs

A SFR is a set of more elementary requirements (functional elements) that, if further divided, would not yield a meaningful evaluation result.
SFRs are grouped into families, which in turn are grouped into classes. Each SFR/family/class is identified by a unique functional component/family/class name in natural language. In addition, short names of classes, families and single SFRs are defined as follows:

- The short name of a class of SFRs is FCL, where CL is a two-letter abbreviation of the class name;
- The short name of a family of SFRs of the class FCL is FCL_FAM, where FAM is a three-letter abbreviation of the family name;
- The single SFR of the family is identified by a number starting from 1, so that FCL_FAM is made of FCL_FAM.1, FCL_FAM.2, etc.

A unique short form of each functional element name is also provided as follows. The single element of FCL_FAM.1 is identified by an additional number starting from 1, so that FCL_FAM.1 is made of FCL_FAM.1.1, FCL_FAM.1.2, etc.
In this document, SFRs/families/classes are referred to by the short name, followed by the actual name in parentheses, e.g., FTA_TAH (TOE access history).

Structure and taxonomy of SARs

A SAR is a set of more elementary requirements (assurance elements). Each assurance element refers:

- Either to the developer (in this case, the assurance element (developer action element) defines activities to be performed by the developer);
- Or to content and presentation of the evaluation evidence to be provided by the developer (in this case, the assurance element (content and presentation element) defines the type of such evidence and the information it has to contain);
- Or to the evaluator (in this case, the assurance element (evaluator action element) defines activities to be performed by the evaluator).

SARs are grouped into families, which in turn are grouped into classes. Each SAR/family/class is identified by a unique assurance component/family/class name in natural language. In addition, short names of classes, families and single SARs are defined as follows:

- The short name of a class of SARs is ACL, where CL is a two-letter abbreviation of the class name;
- The short name of a family of SARs of the class ACL is ACL_FAM, where FAM is a three-letter abbreviation of the family name;
- The single SAR of the family ACL_FAM is identified by a number that corresponds to increasing assurance, starting from 1, so that ACL_FAM is made of ACL_FAM.1, ACL_FAM.2, etc.

A unique short form of each assurance element name is also provided as follows. The single element of ACL_FAM.1 is identified by an additional number starting form 1 plus a letter that denotes what the element refers to. Therefore ACL_FAM.1 is made of:

- Developer action elements ACL_FAM.1.1D, ACL_FAM.1.2D, etc.;
- Content and presentation elements ACL_FAM.1.1C, ACL_FAM.1.2C, etc.;
- Evaluator action elements ACL_FAM.1.1E, ACL_FAM.1.2E, etc.

In this document, SARs/families/classes are referred to by the short name, followed by the actual name in parentheses, e.g., ALC_CMC.5 (Advanced support).

Appendix C: Selected Assumptions and Requirements from [D7.1]

- Assumption 8006.ASS_SEC
  - Description: The results of the actions related to the ASSERT lifecycle management (including issuing and revocation), performed by the ASSERT Issuer, are stored in the ASSERT Registry;
  - Rationale: The ASSERT Registry must be considered as the authoritative repository of ASSERTs;

- Requirement 4006.FUN
  - Description: The Assert4Soa Framework SHOULD support the verification of credentials provided by ASSERT Issuers about their competence;
  - Rationale: This would support the Service-based application and the framework itself in establishing the trustworthiness of the ASSERT Issuer (e.g., to distinguish an issuer accredited by a suitable accreditation scheme);

- Requirement 4007.FUN
  - Description: The Assert4Soa Framework MUST include automated tools (or programmatic interfaces) for checking the revocation status of an ASSERT;
  - Rationale: The check of validity of an ASSERT must be available as an operation that can be included in an automated process;

- Requirement 4008.FUN
  - Description: In response to a service discovery query, the Assert4Soa Framework MUST include a service in the results if and only if the following conditions hold for each ASSERT corresponding to that service:
    - The ASSERT is not revoked;
    - The time of check is within the validity period specified in the ASSERT;
    - The ASSERT integrity is verified;
    - The properties specified in the ASSERT and the way these have been assessed satisfy the service discovery query;
    - Optionally, on user demand, the user trust preferences (see 4021.FUN) are verified;
  - Rationale: All the specified conditions support the reliability of the response provided to Service-based application. The user

trust preferences depend on 4021.FUN. Examples of checks performed according to user trust preferences are:
- The identity of the ASSERT Issuer is verified;
- The service binding specified in the ASSERT is verified;
- The credentials about the ASSERT Issuer competence are verified;

- Requirement 4021.FUN
  - Description: The query language MUST support the expression of conditions regarding all the different parts of an ASSERT;
  - Rationale: Different parts of service certificates (e.g., certificate type, producer, underlying evidence, status) may be important for the assessment of the certificate value. The above conditions include user trust preferences such as
    - Verification of the identity of the ASSERT Issuer;
    - Verification of the service binding;
    - Verification of the credentials about the ASSERT Issuer competence;

- Requirement 4030.FUN
  - Description: The Assert4Soa Framework SHALL allow an authenticated ASSERT Issuer to perform ASSERT management functions, such as the storage in ASSERT registries of results of actions related to the ASSERT lifecycle management (including issuing and revocation);
  - Rationale: Support the ASSERT Issuers in populating and updating the ASSERT registries.

Appendix D: ASSERT Issuer Competence Support by UMA

We report here the contents of the UMA internal document which we referred to in Section 2.10. Namely, the document is titled "ASSERT Issuer Competence Support Attribute Certificate, Language and Verifier Perspective" and is authored by: Rajesh Harjani, Hristo Koshutanski, and Antonio Maña.

Introduction

Regarding relevant cases presented by FUB in [D1.2], UMA revisits and refines the case where an ASSERT Attribute Authority (AAA) exists to attest (by proper accreditation procedures) the competence of an entity to act as an ASSERT Issuer (equivalent to say that AAA recognizes, by an Attribute Certificate (AC), the entity as an Accredited ASSERT Issuer).
UMA is in charge of the ASSERT Language and the ASSERT Verifier, both essential to the realization of the ASSERT Issuer Competence Verification support, identified by functional requirement **4006.FUN** and defined as part of users' preferences in **4008.FUN** [D7.1].
In the rest of the document AC abbreviates for attribute certificate and PKC for public-key certificate (also known as identity certificate).

Description of the structure for AC Management

For simplicity; we consider a **single** Root ASSERT Attribute Authority (RAAA), a number of subordinate ASSERT Attribute Authorities (AAAs) and a number of ASSERT Issuers with Certified ASSERT Attributes (AICAAs). Figure 1 shows the envisaged hierarchy.

RAAA is self-accredited by a self-signed AC (the authorization to act so is established by suitable trust relationship means). RAAA accredits first level subordinate AAA in the structure by signing the relevant AC.

We consider that the RAAA cannot exercise its privileges but only delegate them to subordinate AAAs. In turn, each AAA can exercise its privileges (for instance to issue ASSERTS) and can also delegate to other subordinates the right to be AAA (depending on the RAAA policy and implemented as shown below). The latter case is feasible if the AAA is given the right to delegate down attributes. We also consider that upon signing an AC, an AAA (or RAAA) first verifies (by out-of-hand means) the PKC of the entity being certified.

Prerequisite 1: It is the responsibility of the AAA, upon issuing an AC to an entity, to ensure that *(i)* the PKC of the entity being certified represents sufficient identity information of the entity, and *(ii)* the entity does hold the corresponding private key.

AAA accredits, in the corresponding domain, a number of AICAAs by signing the relevant ACs. AAA can also revoke the AC released for each AICAA in the corresponding domain.

An AICAA issues and manages ASSERTs, i.e. signs relevant data relative to the ASSERT lifecycle.

As shown in Figure 1, RAAA's private key is used to sign AAA_1's AC, AAA_1's private key would be used to sign AAA_2's AC, and so on until AAA_m's private key is used to sign AICAAs AC.

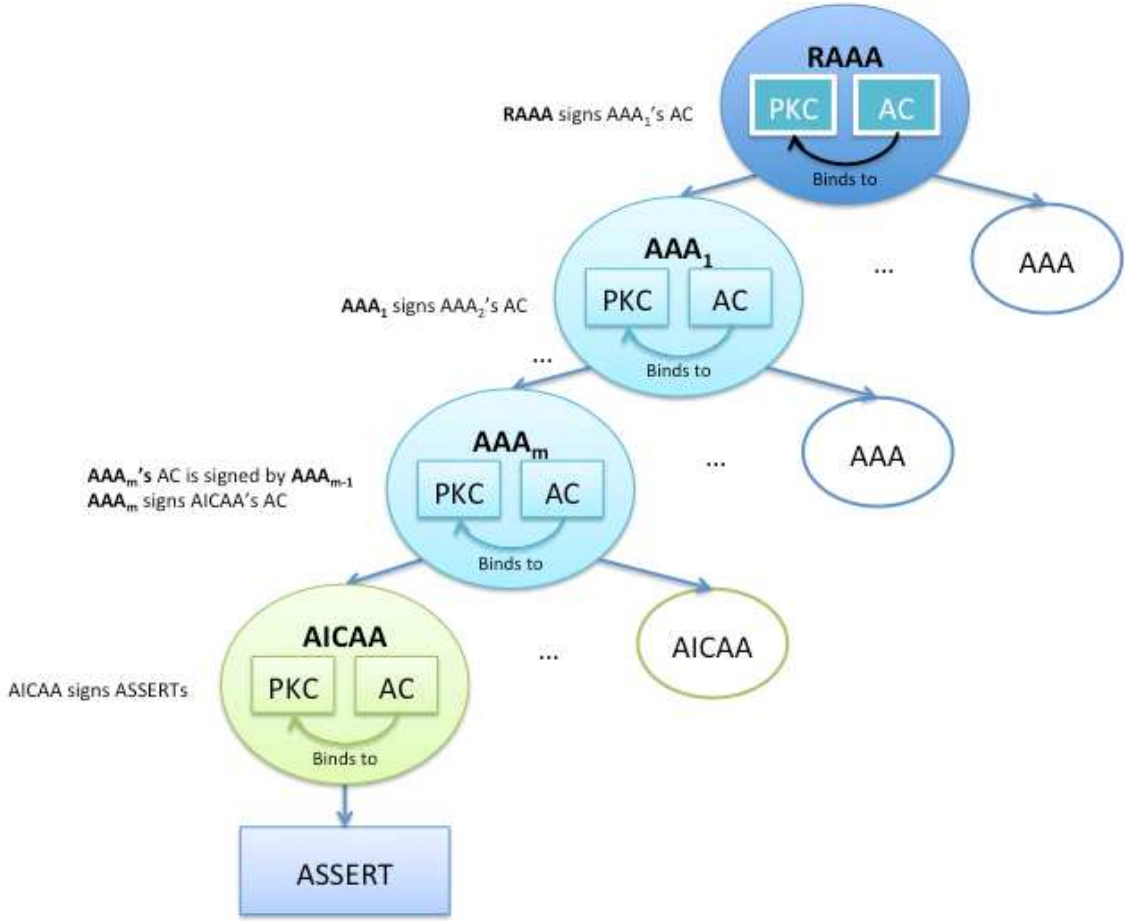


Figure 1. ASSERT4SOA Attribute Certificate Authority Hierarchy Specification

X.509 Attribute Certificate Competence Support

X.509 contains the definition of an AC given below. Details of all types can be found in [X509].

AttributeCertificate ::= SIGNED {AttributeCertificateInfo}

AttributeCertificateInfo ::= SEQUENCE

```
{
        version                 AttCertVersion, -- version is v2
        holder                          Holder,
        issuer                          AttCertIssuer,
        signature                       AlgorithmIdentifier,
        serialNumber            CertificateSerialNumber,
        attrCertValidityPeriod  AttCertValidityPeriod,
        attributes                      SEQUENCE OF Attribute,
        issuerUniqueID          UniqueIdentifier OPTIONAL,
        extensions                      Extensions OPTIONAL
}
```

We propose the following *Attribute* structure:

```
hasCompetence Attribute ::= {
        WITH SYNTAX AssertCompetenceStatement
        ID id-at-competence
}

AssertCompetenceStatement ::= SEQUENCE {
  ASSERTTypeSpecific ::= CHOICE OF {"ASSERT-E", "ASSERT-O", "ASSERT-M"},
  CertificationCriteria  ::=SEQUENCE {
        NameIDUTF8String,
        Version UTF8String OPTIONAL,
        Authority       UTF8String
   }
}
```

We have defined *hasCompetence* attribute structure as a sequence of *AssertCompetenceStatement* elements, each one identifying a particular certification criteria, and a type-specific ASSERT.

ASSERTTypeSpecific indicates the type-specific competence that the AICAA has for issuing ASSERTS. If empty, then the Verifier should interpret that the AICAA has competence to issue only non-type-specific ASSERTs (only general claims).

CertificationCriteria element is a triplet containing the same structure as the corresponding element in the ASSERT language. Each of these triplets is composed by:

- *NameID:* defines a name identifying (given) certification criteria.
- *Version:* if present it defines a version of the certification criteria used for the certification process.
- *Authority:* identifies the authoritative entity of the given certification criteria.

The attribute structure is defined according to the current ASSERT language v2 [D1.2] elements, in particular using the same names for representing competence attributes (*ASSERTTypeSpecific* and *CertificationCriteria*).

In the following we illustrate by example a possible structure of the attributes section of an AC. This example intends to show only the competence structure and not the semantic interpretation of the values.

Example:

```
hasCompetence                                                    ={
  assertCompetenceStatement ={
        ASSERTTypeSpecific = "ASSERT-E",
        CertificationCriteria = {
            NameID = "Common Criteria Part 3 conformant EAL 3 augmented by  ALC_FLR.3",
            Authority = "O=Common Criteria, C=US"
        }
   }
}

hasCompetence ={
 assertCompetenceStatement ={
        ASSERTTypeSpecific = "ASSERT-E",
        CertificationCriteria = {
            NameID = "ISO/IEC 27001:2005 - Information technology -- Security techniques
              – Information security management systems -- Requirements",
            Authority = "O=International Organization for Standardization, C=CH, L=Geneva"
        }
   }
}

hasCompetence ={
 assertCompetenceStatement ={
        ASSERTTypeSpecific = "ASSERT-M",
        CertificationCriteria = {
            NameID = "ISO/IEC 27001:2005 - Information technology -- Security techniques
              – Information security management systems -- Requirements",
            Authority = "O=International Organization for Standardization, C=CH, L=Geneva"
        }
   }
}
```

Delegation of ASSERT Competence Attributes

Let AAA_1 has a set A_1 of competence attributes *hasCompetence* (of type *Attribute*). This is the granularity of attributes we consider.

Definition 1 (Competence Attribute Equality): Let a_i and a_j be two competence attributes $a_i, a_j \in A_1$. We define a_i is equal to a_j as following:

$$a_i = a_j \Leftrightarrow$$
$$a_i.ASSERTTypeSpecific = a_j.ASSERTTypeSpecific \wedge$$
$$a_i.CertificationCriteria.NameID = a_j.CertificationCriteria.NameID \wedge$$
$$a_i.CertificationCriteria.Version = a_j.CertificationCriteria.Version \wedge$$
$$a_i.CertificationCriteria.Authority = a_j.CertificationCriteria.Authority$$

Definition 2 (Competence Attributes Delegation): We define a competence attribute delegation from AAA_1 holding a set A_1 to AAA_2 holding a set A_2 if and only if $A_2 \subseteq A_1$.

$$A_2 \subseteq A_1 \Leftrightarrow \forall a_j \in A_2, \exists a_i \in A_1, a_i = a_j.$$

Requirements for ASSERT Issuer Competence Credentials

AICAAs' attribute certificates need to be distinguishable from AAAs' attribute certificates to protect against AICAAs establishing themselves as AAAs without authorization.
Thus, in order to capture ASSERT issuance delegation capabilities we will be using the *basic attribute constraints* extension from the X.509 specification [X509]. This field indicates whether subsequent delegation of the privileges assigned in the certificate containing this extension is permitted. If so, a delegation path length constraint may also be specified.
This field is defined in [X509] as follows:

```
basicAttConstraints EXTENSION ::= {
        SYNTAX BasicAttConstraintsSyntax
        IDENTIFIED BY { id-ce-basicAttConstraints } }

BasicAttConstraintsSyntax ::= SEQUENCE {
        authority BOOLEAN DEFAULT FALSE,
        pathLenConstraint INTEGER (0..MAX) OPTIONAL }
```

The **authority** component indicates whether or not the holder is authorized to further delegate privilege. If authority is TRUE the holder is also an AAA and is authorized to further delegate privilege, dependent on relevant constraints. If authority is FALSE, the holder is an end-entity (AICAA) and is not authorized to delegate the privilege.
The **pathLenConstraint** component is meaningful only if authority is set to TRUE. It gives the maximum number of AAA certificates that may follow this certificate in a delegation path. Value 0 indicates that the subject of this certificate may issue certificates only to AICAAs and not to AAAs. If no **pathLenConstraint** field appears in any certificate of a delegation path, there is no limit to the allowed length of the delegation path. Note that the constraint takes effect beginning with the next certificate in the path. The constraint controls the number of AAA certificates between the AAA certificate containing the constraint and AICAA certificate. The constraint restricts the length of the segment of the delegation path between the certificate containing this extension and the AICAA certificate. It has no impact on the number of AAA certificates in the delegation path between the trust anchor and the certificate containing this extension. Therefore, the length of a complete delegation path may exceed the maximum length

of the segment constrained by this extension. The constraint controls the number of AAA certificates between the AAA certificate containing the constraint and the AICAA certificate. Therefore the total length of this segment of the path may exceed the value of the constraint by as many as two certificates. (This includes the certificates at the two endpoints of the segment plus the AAA certificates between the two endpoints that are constrained by the value of this extension.)
We consider this extension to be present in attribute certificates issued by AAAs, including RAAA, to other AAAs or to AICAAs.
If this extension is present in an attribute certificate, and authority is TRUE, the holder is authorized to issue subsequent attribute certificates delegating the contained privileges to other entities, but not public-key certificates.
We will use the *noAssertion* extension in order to indicate the case where the AC holder should not assert the privileges indicated in the attributes of the AC. This extension field can only be inserted into AAA ACs, and not into AICAA ACs. If present, this extension shall always be marked critical.

```
noAssertion EXTENSION ::= {
        SYNTAX NULL
        IDENTIFIED BY id-ce-noAssertion
}
```

Support by the ASSERT language: Attaching Competence Credentials.

UMA will add to the ASSERT Language v2 [D1.2] support (structure) to allow an ASSERT Issuer to refer to competence credentials together with the chain of AAAs' PKC and AC upper in a PMI hierarchy (See Figure 2).

Support by the ASSERT Verifier: Verification of ASSERT Issuer Competence.

The verification of ASSERT credential competence consists in three main steps:

1. *Trusted ASSERT Issuer Identity Verification* by verifying a chain of PKCs from the AICAA's PKC to a trusted RootCA's PKC.

2. *ASSERT Issuer Competence Verification* by verifying if the ASSERT Issuer's AC competence attributes values qualify for the issuance of this particular ASSERT.

3. *Trusted ASSERT Issuer Competence Verification* by verifying from the AC of the AICAA signing the ASSERT up to RAAA's AC, checking if each AC in the chain has appropriate attribute values to delegate ASSERT issuance.

Figure 2 illustrates the ASSERT issuer competence verification process. We consider that each PKC of the RAAA or AAA authorities comes from a different PKI hierarchy external to the ASSERT4SOA PKI and no trust relationships are established/verified/assumed for those PKCs in the framework. *Prerequisite 1* ensures and guarantees the correct behaviour of the considered ASSERT4SOA PMI model.

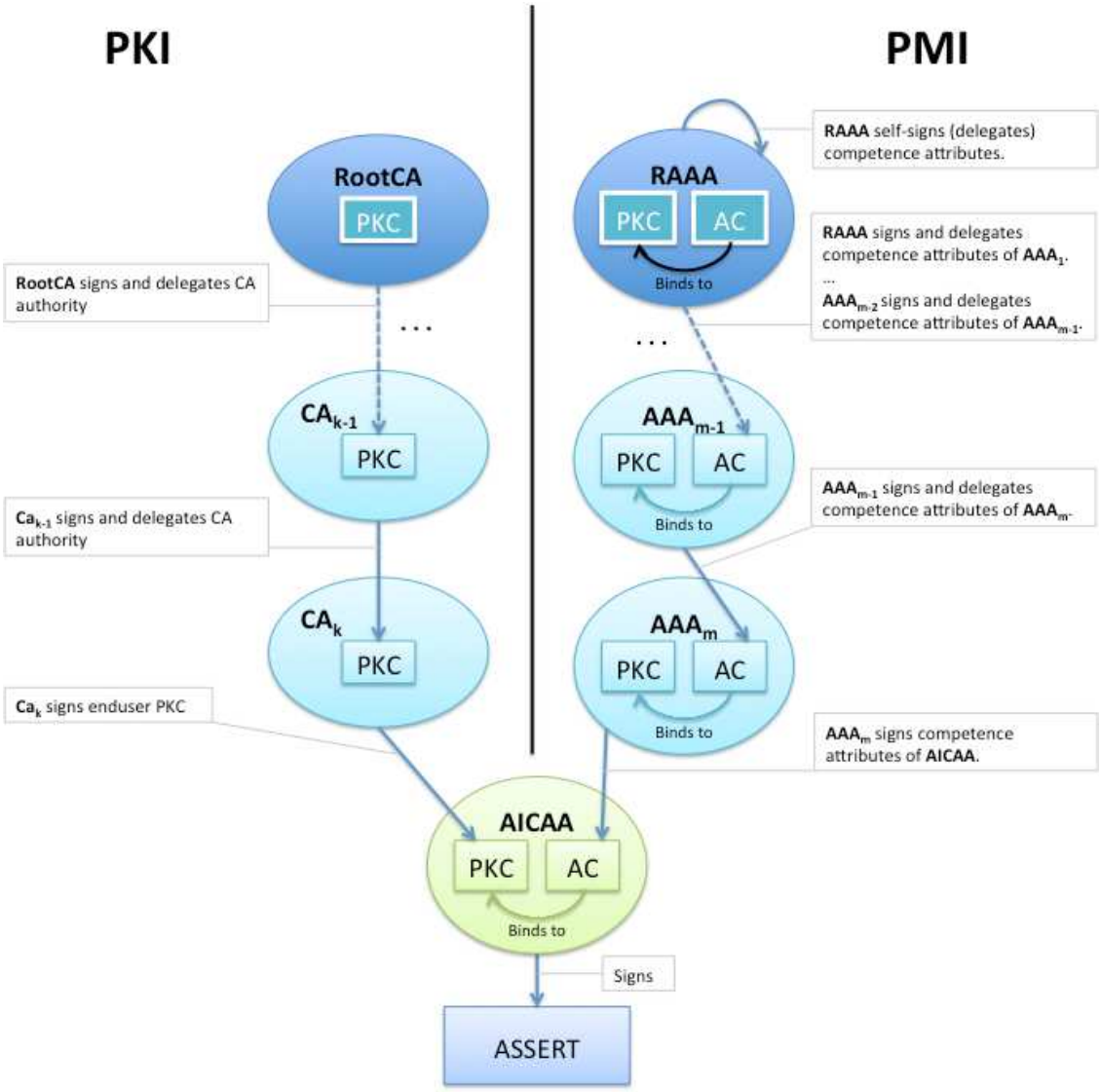


Figure 2. ASSERT Issuer Competence Certification Hierarchy

In the following we describe in detail each of the steps regarding issuer competence verification process:

1. Trusted ASSERT Issuer Identity Verification

   Verifies the identity of the AICAA's PKC according to a set of trusted CAs. This step also validates the integrity of the PKC, and its validity according to the current time of check.

   This step is defined as an optional user preference in 4008.FUN. If selected by the user, then the verifier will enforce it.

2. ASSERT Issuer Competence Verification (competence matching)

2.1. Verify whether the ASSERT Issuer's AC binds to the corresponding PKC or not.

According to X.509 [X509], there are several ways to set and verify binding between an AC and the corresponding entities PKC. The verifier will support some of those. It will be defined in a subsequent phase of the approach.

2.2. Verify if the ASSERT Issuer's AC competence attributes values qualify for the issuance of the particular ASSERT.

This can be resolved by matching the values identified in the ASSERT with the corresponding attributes values within the AC.
For instance, if the ASSERT being verified is E type-specific, then the *ASSERTTypeSpecific* attribute within ASSERT Issuer's AC should contain at least *ASSERT-E* value. If the ASSERT identifies a particular criteria then the corresponding value should also be present in the AC.

As mentioned before the **authority** component within the AC indicates whether or not the holder is authorized to further delegate privilege. Thus, we consider two options at this level: *(i)* if **authority** is FALSE we are in the case if an AICAA being the ASSERT Issuer, and *(ii)* if **authority** is TRUE we are in the case if an AAA being the ASSERT Issuer, and therefore should validate if the AAA's AC allows assertion – if **authority** is TRUE and *noAssertion* extension is **not** present in the AC then accept AAA acting as an ASSERT Issuer entity.

3. Trusted ASSERT Issuer Competence Verification

As Figure 2 illustrates, there must be a chain of AAAs with the last AAA_m, where $m \geq 1$, that signed the ASSERT Issuer Competence Credential, and with the first element that is the RAAA. We note that RAAA is the element in the chain with index 0, which means that the total of AAAs is m+1.

The verifier performs the following steps from bottom up, starting from the AICAA and ending with the RAAA:

3.1. Verify if the AICAA's AC is signed by AAA_m using AAA_m's PKC.

This step refers to validating the first link within the chain of trust.

3.1.1. Check if AAA_m's **authority** component value within the AC is TRUE. If not, the verification process ends at this point, as we do not trust AAA_m as Attribute Authority.

3.1.2. Validate AAA_m's AC is bound to AAA_m's PKC.

3.1.3. Check if the set A_{AICAA} of ASSERT Competence Attributes hold by AICAA is a subset of ASSERT Competence Attributes A_{AAAm} hold by AAA_m, i.e. check if the following holds $A_{AICAA} \subseteq A_{AAAm}$.

3.1.4. Check if the value of *pathLenConstraint* extension from AAA_m's AC is greater or equal that 0 (*pathLenConstraint* $\geq$0).

3.2. Verify if the AAA_m's AC is signed by AAA_{m-1} using AAA_{m-1}'s PKC.

This step refers to validating the second link within the chain of trust.

3.2.1. Check if AAA_{m-1}'s **authority** component value within the AC is TRUE. If not, the verification process ends at this point, as we do not trust AAA_{m-1} as Attribute Authority.

3.2.2. Validate AAA_{m-1}'s AC is bound to AAA_{m-1}'s PKC.

3.2.3. Check if the set A_{AAAm} of ASSERT Competence Attributes hold by AAA_m is a subset of ASSERT Competence Attributes A_{AAAm-1} hold by AAA_{m-1}, i.e. check if the following holds $A_{AAAm} \subseteq A_{AAAm-1}$.

3.2.4. Check if the value of *pathLenConstraint* extension from AAA_{m-1}'s AC is greater or equal that 1 (*pathLenConstraint* $\geq$1).

3.3. Generalizing step 3.2, we shall repeat step 3.2 for all AAA_{m-k} and AAA_{m-k-1} where $0 \leq k < m$. In case of k=0 we have exactly the step 3.2.

Verify if the AAA_{m-k}'s AC is signed by AAA_{m-k-1} using AAA_{m-k-1}'s PKC.

3.3.1. Check if AAA_{m-k-1}'s **authority** extension value within the AC is TRUE. If not, the verification process ends at this point, as we do not trust AAA_{m-k-1} as Attribute Authority.

3.3.2. Validate AAA_{m-k-1}'s AC is bound to AAA_{m-k-1}'s PKC.

3.3.3. Check if the set A_{AAAm-k} of ASSERT Competence Attributes hold by AAA_{m-k} is a subset of ASSERT Competence Attributes $A_{AAAm-k-1}$ hold by AAA_{m-k-1}, i.e. check if the following holds

$$A_{AAAm-k} \subseteq A_{AAAm-k-1}.$$

3.3.4. Check if the value of *pathLenConstraint* extension from AAA_{m-k-1}'s AC is greater or equal than *k+1*, i.e. *pathLenConstraint* $\geq$k+1.
In this step, we verify the delegation from the node AAA_{m-k-1} to node AAA_{m-k}, which means that we have already verified for AAA_{m-k} and below, i.e. already verified for k-number of **relations** between AAA nodes. That is why we need to have *pathLenConstraint* for AAA_{m-k-1} be greater than or equal to k+1.

3.4. We repeat step 3.3 until either of the two cases hold: (i) we reach the case where AAA_{m-k-1}'s AC is signed by the AAA_{m-k-1}'s PKC (self-signed AC); or (ii) no more AAAs are found that signed the AC of the last iteration verified.

In the first case, we check if the AAA_{m-k-1}'s PKC is within the trust store. If so return TRUE for the verification process. If not within the trust store then return FALSE.

In the second case, we check in within the trust store if there is a PKC (of RAAA) matching the signature of the last verified AAA's AC. If so return TRUE for the verification process, otherwise return FALSE.

In the second case we assume that the verifier knows (from a priori set up) the set of competence attributes that the RAAA in the trust store has authority for.

Bibliography

[2004-02-009] CCRA Supporting Document 2004-02-009 Assurance Continuity, available at:
http://www.commoncriteriaportal.org/files/supplements/2004-02-009.pdf.

[AESAVS]: The Advanced Encryption Standard Algorithm Validation Suite (AESAVS), available at:
http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf.

[AIS 34]: BSI Application Notes and Interpretation of the Scheme No. 34 (AIS 34): Evaluation Methodology for CC Assurance Classes for EAL5+ (CC v2.3 & v3.1) and EAL6 (CC v3.1) version 3, 03.09.2009.

[BCM]: NIST web page on Block Cipher Modes, available at:
http://csrc.nist.gov/groups/ST/toolkit/BCM.

[BSIGFSP]: BSI Guideline for the Development and Evaluation of formal security policy models in the scope of ITSEC and Common Criteria Version 2.0, December 4, 2007.

[CAVP]: NIST Cryptographic Algorithm Validation Program (CAVP), available at:
http://csrc.nist.gov/groups/STM/cavp.

[CCpart1v2.3]: Common Criteria for Information Technology Security Evaluation Part 1: Introduction and general model, v.2.3, August 2005.

[CCpart1]: Common Criteria for Information Technology Security Evaluation Part 1: Introduction and general model, v.3.1 revision 3, July 2009.

[CCpart2]: Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, v.3.1 revision 3, July 2009.

[CCpart3]: Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components, v.3.1 revision 3, July 2009.

[CCDB-2007-09-01]: CCRA Supporting Document 2007-09-01 Composite product evaluation for Smart Cards and similar devices, v.1.0, revision 1, September 2007.

[CCMVS]: The CCM Validation System (CCMVS), available at:
http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

[CEM]: Common Methodology for Information Technology Security Evaluation, v.3.1 revision 3, July 2009.

[CIM-PP]: Consistency Instruction Manual for development of US Governments Protection Profiles, release 3, February 2005.

[CMACVS]: The CMAC Validation System (CMACVS), available at: http://csrc.nist.gov/groups/STM/cavp/documents/mac/CMACVS.pdf.

[CRP]: J. Willemson, Certificate Revocation Paradigms, available at http://arxiv.org/pdf/cs/9909012.pdf.

[CWA 14169]: CEN Workshop Agreement 14169, Secure signature-creation devices EAL 4+, March 2004.

[CWP]: Cisco White Paper – Public Key Infrastructure Certificate Revocation List versus Online Certificate Status Protocol, available at: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6638/ps6664/product_data_sheet0900aecd80313df4.pdf.

[D1.2]: ASSERT4SOA Deliverable 1.2 – ASSERT Language v.2. ASSERT4SOA Consortium, September 2012.

[D1.3]: ASSERT4SOA Deliverable 1.3 – ASSERT Profiles. ASSERT4SOA Consortium, March 2012.

[D1.4]: ASSERT4SOA Deliverable 1.4 – ASSERT Model and Language v.3. ASSERT4SOA Consortium, September 2013.

[D4.1]: ASSERT4SOA Deliverable 4.1 – Design and description of evidence-based certificates artifacts for services. ASSERT4SOA Consortium, September 2011.

[D4.2]: ASSERT4SOA Deliverable 4.2 – Matching algoritm for evidence-based certification v2, and proof-of-concept. ASSERT4SOA Consortium, September 2012.

[D4.3]: ASSERT4SOA Deliverable 4.3 – Architectural solutions for evidence-based certification. ASSERT4SOA Consortium, September 2013.

[D5.1]: ASSERT4SOA Deliverable 5.1 – Model Composition. ASSERT4SOA Consortium, September 2013.

[D7.1]: ASSERT4SOA Deliverable 7.1 – Framework Requirements v2. ASSERT4SOA Consortium, October 2012.

[DoW]: ASSERT4SOA Annex I – “Description of Work”, ASSERT4SOA Consortium, June 2010

[ECSS-Q-ST-80C]: ECSS-Q-ST-80C Space product assurance – Software product assurance, March 2009.

[FIPS 140]: FIPS PUB 140-2 (Security Requirements for Cryptographic Modules, Annex A: Approved Security Functions (Draft May 30, 2012).

[FIPS 197]: Federal Information Processing Standards Publication 197 – Specification for the Advanced Encryption Standard (AES) November 26, 2001.

[FM2008]: B. Chetali and Q. Nguyen, Industrial Use of Formal Methods for a High-Level Security Evaluation, in FM 2008: Formal Methods, Lecture Notes in Computer Science, Volume 5014/2008.

[GCMVS]: The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS), available at:
http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

[GD-PP]: US Government Directory Protection Profile for Medium Robustness Environments, version 1.1., July 2007.

[ICCC2010]: V. Bagini, F. Guida, C. Majorani, R. Menicocci, M. Orazi, CC approaches to the certification of the components of a system when the system certification is not possible. Presentation given at 11th ICCC (International Common Criteria Conference), 21-23 September 2010, Antalya, Turkey.

[ICCC2012]: S.P. Kaluvuri, M. Bezzi, A. Sabetta, Y. Roudier, R. Menicocci, V. Bagini, A. Riccardi, M. Orazi, Applying Common Criteria to Service Oriented Architectures, presentation given at ICCC 2012 (13th International Common Criteria Conference), 18-20 September, 2012, Paris (FR).

[ICCC2013] S.P. Kaluvuri, M. Bezzi, R. Menicocci, V. Bagini, A. Riccardi, M. Orazi, Applying Common Criteria to SOA: Dynamic Certification Lifecycle, presentation submitted at ICCC 2013 (14th International Common Criteria Conference), 10-12 September, 2013, Orlando (US).

[INFN CA]: INFN Certification Authority statistics (issued certificates, revoked certificates), available at: https://security.fi.infn.it/CA/en/stats.

[ISO27001]: ISO/IEC 27001:2005 Information technology -- Security techniques -- Information security management systems – Requirements.

[ITSEC]: Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria, Version 1.2, June 1991.

[MMT]: The Multi-block Message Test (MMT) for DES and TDES, available at:
http://csrc.nist.gov/groups/STM/cavp/documents/des/DESMMT.pdf.

[MOVS]: NIST Special Publication 800-17 – Modes of Operation Validation System (MOVS): Requirements and Procedures, available at: http://csrc.nist.gov/publications/nistpubs/800-17/800-17.pdf.

[NASA-STD-8739.8]: NASA-STD-8739.8 Software Assurance Standard, July 2004.

[NIST SP 800-38A]: NIST Special Publication 800-38A – Recommendation for Block Cipher Modes of Operation - Methods and Techniques, December 2001.

[NIST SP 800-38B]: NIST Special Publication 800-38B – Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005.

[NIST SP 800-38C]: NIST Special Publication 800-38C – Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, May 2004.

[NIST SP 800-38D]: NIST Special Publication 800-38D – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007.

[NIST SP 800-38E]: NIST Special Publication 800-38E – Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, January 2010.

[NIST SP 800-38F]: NIST Special Publication 800-38F – Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, December 2012.

[NIST SP 800-67]: NIST Special Publication 800-67 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Rev.1, January 2012.

[NIST SP 800-131A]: NIST Special Publication 800-131A – Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, January 2011, available at: http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf.

[NVD] National Vulnerability Database home page, http://nvd.nist.gov/

[OSSTMM]: OSSTMM 3 – The Open Source Security Testing Methodology Manual, December 2010.

[RFC 2560]: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP.

[RFC 5280]: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

[SAM]: SAM (Safety Assessment Methodology) v2.1 Electronic, November 2006.

[Skip]: SKIPJACK and KEA Algorithm Specifications, available at: http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf.

[SQL]: J.R. Groff, P.N. Weinberg, and A.J. Oppel, SQL: The Complete Reference, 3rd Ed., McGraw-Hill, 2009.

[TAMOS ST]: Tivoli Access Manager for Operating Systems 5.1 Security Target, version 1.6.5, February 2006.

[TMOVS]: NIST Special Publication 800-20 – Modes of Operation Validation System for the Triple Data Encryption Algorithm (TMOVS): Requirements and Procedures, available at: http://csrc.nist.gov/publications/nistpubs/800-20/800-20.pdf.

[X509]: ITU-T Recommendation X.509: Information Technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks.

[XTSVS]: The XTS-AES Validation System (XTSVS), available at: http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf.