AADL Security Annex

DRAFT

Dave Gluch

October

2019

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-1020

1 Table of Contents

2 List of Figures

3 List of Tables

This is a draft of the SAE Architecture Analysis and Design Language (AADL) Security Annex (AADL-SA).

It is for review only.

RATIONALE

The AADL Security Annex (AADL-SA) supports the engineering of secure embedded (cyber-physical) systems by providing guidance for engineers in modeling and analyzing a system's security characteristics using the SAE Architecture Analysis and Design Language (AADL).

This Architecture Analysis & Design Language (AADL) Security Annex document was prepared by the SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division.

A.1 Scope

(1) This document describes the Architecture Analysis & Design Language Security Annex. The AADL Security Annex provides guidance and support for specifying, modeling, and analyzing a system's security characteristics within AADL architecture models. The approach presented in this document enables the engineering of secure cyber-physical systems using the AADL and supporting tool environments such as the Open Source AADL Tool Environment (OSATE).

A.2 Rationale

(2) Software-reliant, safety-critical embedded (cyber-physical) systems often operate in untrusted environments. In these environments, they are exposed to potential attacks trying to exploit vulnerabilities to cause damage, harm people, or compromise or steal important (e.g. classified) information.

(3) As a result, security is a critical consideration in the development and operation of these systems. Security protections and properties must be correctly and completely specified, implemented, and validated throughout the system development lifecycle. To do so, engineers need to capture security policies and requirements and ensure their correct implementation in a software system architecture and design. This security annex is intended to support these activities by adding the requisite elements to the AADL language and tools such as the Open Source AADL Tool Environment (OSATE) and providing guidance to specify, model, and analyze the security aspects of a system architecture.

(4) Traditionally, software security has focused on coding issues. For example, buffer overflows, use of unsafe functions (e.g. legacy functions from the C standard libraries - strcpy - instead of their secure versions - strncpy), types mismatches, improper memory handling, etc. have been the root causes of a large number of security issues.

(5) There is increased interest in going beyond software 'bugs' to identifying flaws in the software architecture and design [Arce 2014]. Examples of flaws resulting in security problems include misunderstanding architecturally significant requirements, poor architectural implementation, and violation of established design principles resulting in architecture flaws. As a result, there is an effort underway to define a common Architecture Weaknesses Enumeration that builds on the Common Software Weaknesses Enumeration [CWE] [Mirakhorli 2016].

(6) Capturing a software architecture with its security characteristics using a formal notation enables security analysis and vulnerability discovery early in the life cycle. This has motivated the development of AADL-specific modeling capabilities, which capture security requirements and policies within AADL models. By employing appropriate tools, these AADL models can be used to implement and validate the implementation of security policies and features.

A.3 Approach

(7) This annex has been developed to support the modeling and analysis of system architecture security using the AADL. It provides guidelines and defines AADL security properties and elements to support modeling and analysis of security within system architectures.

(8) The annex describes practices that utilize the core capabilities of the AADL, security-specific property sets and verification methods, and specialized AADL components for use within the OSATE environment. There is flexibility for a user to modify and add to the property sets and components provided by this annex.

(9) In this annex, the Architecture-Led Incremental System Assurance (ALISA) environment of OSATE is used as an exemplar of the recommended capabilities for capturing and verifying security policies and requirements. Other comparable systems may be used, provided they include the basic support required by this annex.

(10) A core of security verification within this annex is the use of formal statements (e.g. predicates) whose validity is assessed across all or a portion of a system software architecture. As part of this annex, a set of example verification method statements using Resolute [Resolute 2014] and JAVA are provided. These can be used directly for analysis or as exemplars for the development of additional methods. As appropriate, other analysis and constraint languages can be employed.

A.4 Terminology

(11) The security annex relies on a set of key concepts and associated terminology useful in modeling and analyzing secure embedded systems. This set is adapted from a variety of sources and does not conform in its entirety to any specific security perspective or standard. Table 1 summaries key terms used in this annex, including relevant terms from safety analysis.

(12) For the purposes of this document we distinguish security from safety based upon intentionality. That is, security is the freedom from loss due to intentional (malicious) actors and safety is the freedom from loss due to both intentional and unintentional causes. In traditional security engineering, the primary causal concern is unauthorized access by external actors and the focus is on approaches to prevent unauthorized access. Traditionally, in safety engineering, primary considerations are losses from any cause, sometimes without explicit consideration of malicious intent.

Table 1: Safety and Security Terminology

Term	Definition	Comments
loss	A loss is a condition that results from events such as accidents [Leveson 2012] or the realizations of hazards [Feiler 2016] or threats	Loss can also be considered an event, as in the definition “the action or state of not having or keeping something any more” [CAMBRIDGE].

	[NIST 2016]. Loss is used in a general sense of harm (i.e. deleterious condition).	
accident	An accident is an undesired or unplanned event that results in a loss, including loss of human life or human injury, property damage, environmental pollution, mission loss, etc. Adapted from [Leveson 2012]	This definition encompasses loss from intentional as well as unintentional events.
hazard	A hazard is a system state or set of conditions that, together with a particular set of environmental conditions, will lead to a loss. Adapted from [Leveson 2012]	
hazard contributor	A hazard contributor is a state or set of conditions of a subsystem or component that is part of or adds to a hazard.	This supports decomposition of a system hazard.
vulnerability (security hazard)	A vulnerability (security hazard) is a system state or set of conditions (including security procedures, internal controls, design, or implementation) that could be exploited by an attacker (i.e. with a particular set of environmental conditions, will lead to a loss).	This is similar to the [NIST 2012] definition Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source.
threat[1] (security threat)	A threat is a specified vulnerability plus a specification of an attacker, attacker access, and attacker capability to exploit the vulnerability (i.e. a *security hazard* with specified environmental conditions).	Success is ensured given an attacker with access and capability (i.e., addressing the 'will lead to' in the definition of hazard).
attack	An attack is an unauthorized attempt to access a system, usually with malicious intent.	
attacker	An attacker is an entity (or a coordinated set of entities) that engages in or attempts to engage in an attack.	

A.5 Overview

(13) The AADL Security Annex provides guidance and support for system architecture security specification, modeling, and analysis throughout the system lifecycle from concept to implementation through to operational system maintenance and upgrade.

(14) The security annex provides guidance and support to specify security policies and requirements, verify that security requirements satisfy security policies, and assure that there are mechanisms that enforce security policies and security requirements within a system architectural specification.

(15) The security annex provides guidance and support for architecture specification, modeling, and analysis of security protections, including information/data protection, access control and protection, and action/command Protection.

(16) The security annex provides guidance and support for architecture specification, modeling, and analysis of security architectures, including specialized security

[1] In general use, the term threat encompasses both intentional and unintentional environmental conditions. In the case of 'unintentional' safety considerations a threat would be the specification of the hazard (system conditions) plus a description of the requisite environmental conditions.

architectures such as multiple independent levels of security (MILS) and cross domain solutions (CDS).

(17) The security annex provides guidance and support for the modeling and analysis of vulnerabilities/threats/Attacks. – TBD

A.6 Security Policies and Requirements

(18) From [NIST 2016] a security policy is a set of rules that governs all aspects of security-relevant system and system element (technology, machine, and human, elements) behavior. The rules can be stated at very high levels (e.g., organizational policy that defines acceptable behavior of employees in performing their mission/business functions) or at very low levels (e.g., an operating system policy that defines acceptable behavior of executing processes and use of resources by those processes).

(19) We consider security policies to be general statements about security attributes of a system and security requirements as statements that define the functions and capabilities that must exist within a system to satisfy security policies.

(20) The implementation elements of a system must satisfy security requirements and their associated policies. However, for some the distinction is arbitrary. In order to provide flexibility, we present an approach which enables a user to distinguish between policies and requirements.

(21) The procedures and artifacts described in this annex assume a systematic policy and requirements documentation and analysis approach that includes tool support. This approach involves both natural language and verifiable (formalized) requirements statements and the use of assurance cases for verification of policies and requirements.

(22) As a representative systematic policy and requirements documentation and analysis approach for the procedures, artifacts, and examples presented in this document, we use the Architecture-Led Incremental System Assurance (ALISA) workbench, which is part of the OSATE tool suite.

(23) In using the ALISA workbench, security policies and requirements are documented, verification plans and verification methods for those policies and requirements are defined, and assurance cases for those policies and requirements are created. The assurance cases are the foundation for system security verification.

A.7 Documenting Security Policies and Requirements

(24) As an exemplar for the documentation of security policy and requirements, we use the ReqSpec capabilities that are part of the ALISA workbench. Both security policies and security requirements are captured as requirements declarations within the ALISA notation.

(25) Security policies are organized into system requirement sets that are explicitly identified as security policy statements by naming the file as a policies file (e.g. *TransportAircraftSecuirtyPolicies.reqspec*) and the system requirements set within that file (e.g. *TransportAircraftSystemSecurityPolicies*). An excerpt from security policies for a transport aircraft system is shown in Table 2, which shows a single requirement (policy) statement named *Security*.

Table 2: Example System Security Policy

```
system requirements TransportAircraftSystemSecurityPolicies : "System-Wide Security
Policies"
for TransportAircraft_Generic::transportAircraft.generic

[
       description "These are the high level (system) security policies for an air transport
aircraft."

       requirement Security:  "System Security must be provided"
       [
              description "Security protections that meets FAA aircraft security and
                           flight worthiness certification standards must be provided."

       ]
```

(26) Global requirements and global requirement sets can be used for security policies and requirements that apply to all the specified components of a system or security policies and requirements that can be applied to multiple systems.

(27) Once security policies are defined they can be decomposed (policies/requirements derived from that of an enclosing system-traceability across architecture layers) or refined (more detailed specification of a policy/requirement for the same system.). For example, a system security policy may be refined into policy statements about confidentiality, integrity, and availability. A policy statement about communications security can be decomposed in communications policy statements for subsystems.

(28) Eventually security policy statements should be decomposed and/or refined into security requirement statements for more detailed verification. Examples of refine and decompose are shown in Table 3.

Table 3: Refinement and Decomposition of Policies and Requirements

```
system requirements TransportAircraftSystemSecurityPolicies : "System-Wide Security
Policies" for TransportAircraft_Generic::transportAircraft.generic

-- This is an access control policy statement for the aircraft system.
requirement  AccessControlPolicy: "Security Controlled Access to all Aircraft Systems
and Resources must be provided."
       [
       description "Access to all Aircraft operational and maintenance Systems and
Resources shall be permitted only by authorized personnel."
       development stakeholder DevelopmentTeam.PrincipalEngineer
DevelopmentTeam.SecurityEngineer
```

```
                        ]
-- This is a refinement of the access control policy statement for the aircraft.
       requirement AccessControlPolicyAuthentication: "This is a refinement of the
AccessControlPolicy for the aircraft."
       [
       description "Access authorization to all Aircraft operational and maintenance
Systems and Resources shall employ multi-factor authentication."
         refines TransportAircraftSystemSecurityPolicies.AccessControlPolicy
              ]
-- This is a secure communication policy statement for the aircraft system.
requirement SecureCommunicationsPolicy: "Secure communications must be provided for
all flight- and safety-critical systems."
                        [
description "Communication systems must provide security measures to ensure only
authorized access and use."
                              development stakeholder
DevelopmentTeam.PrincipalEngineer DevelopmentTeam.EncryptionExpert
DevelopmentTeam.SecurityEngineer
                        ]

-- This is a requirement statement for the aircraft control system that decomposes
the Security and Secure Communication Policies for the aircraft.
       requirement  communicationProtectionReq2: "All aircraft external
communication for aircraft control and flight operations must employ the latest NIST
standard encryption algorithms""
       [
       description "All aircraft external communication for aircraft control and
flight operations must employ encryption algorithms that meet or exceed the standards
defined in NIST publication FIPS 140-2 or any superseding document that have been
released for use."
       decomposes TransportAircraftSystemSecurityPolicies.Security
TransportAircraftSystemSecurityPolicies.SecureCommunicationsPolicy
              ]
```

(29) If no distinction is made between security policy and security requirement statements and only security requirements are defined for a system, the security requirements are captured as system requirements and organized into system requirement sets. These can then be decomposed and refined into additional security requirements. An example for a global security requirement that all connections must be encrypted is shown in Table 4.

Table 4: Example Global Security Requirement

```
global requirements GlobalSecurity

[
       requirement encrypted_connections :
       "All connections in the system must be encrypted."  for connection
       [
              description "All connections have encryption."
              development stakeholder DevelopmentTeam.SecurityEngineer
       ]
]
```

A.8 Verification of Security Policies and Requirements

(30) The verification of security policies includes elements to validate an AADL architecture model against a specific security policy (e.g. checks of the communications and connections among components to ensure that there are no classified systems or entities communicating directly with unclassified entities).

(31) The verification and analysis described in this annex does not assure the effectiveness, self-consistency, or validity of a security policy(ies) for a specific application.

(32) As an exemplar for the documentation of security policy and requirements, we use the capabilities that are part of the ALISA workbench. This include the verify capabilities (i.e. the verify notation) of ALISA (OSATE). Security verification plans, procedures, and activities are organized using the assurance case and verification capabilities within ALISA (i.e. the Alisa and Assure notations). The results are captured using the Assure notation with ALISA (OSATE).

(33) A verification plan is captured in a .verify file as shown in Table 5, where a portion of the verification plan for the security policies of the Transport Aircraft System. The plan includes claims for the policies being verified. The Security claim in Table 5 refers to the policy requirement statement 'Security' shown in Table 2 for the Transport Aircraft. The activities declaration in Table 5 identifies the verification method(s), specifically the activity 'ConfirmCertifications' uses the 'ReviewSecurityCertification' method that is declared in the method registry 'AircraftSecurityVerificationMethods.'

Table 5: Example Security Verification Plan

```
verification plan TransportAircraftSecurityPolicyVerificationPlan : "Verification Plan for a
Transport Aircraft Security Policies"
for TransportAircraftSystemSecurityPolicies
[
       description "This is the top-level verification plan for the security policies for
the aircraft system."

       claim Security
       [
              activities
              ConfirmCertifications:
AircraftSecurityVerificationMethods.ReviewSecurityCertification ("certification
submissions")
       ]
```

(34) Verification methods are declared in method registry files. Portions of a method registry 'AircraftSecurityVerificationMethods' are shown in Table 6. The 'ReviewSecurityCertification' method is a manual review. The 'VerifyHasEncryption' and 'VerifyHasAuthenticatedEncryption' methods are Resolute claims and 'JavaCheckTrusted' is a Java method.

Table 6: Example Verification Methods

```
verification methods AircraftSecurityVerificationMethods :
"These are the security verification methods for Aircraft Systems."

[
       method ReviewSecurityCertification (document: string) : "A formal review to confirm
compliance with all required security certifications"
       [
              manual FormalPanelReview
              description "Formal review by a certification panel to confirm compliance"
       ]
...

        method VerifyEncryption (component): "verify that a component has encryption
    A Resolute claim that the encryption property is true."
       [
              resolute
Security_Exposure_Util.Security_Exposure_Util_public.Resolute.Resolute.is_encrypted
           description "This confirms encryption."
       ]


       method VerifyHasAuthenticatedEncryption (component): "Verify that the element has or
provides authenticated encryption -- Resolute"
       [
              resolute
Security_Enforcement_Resolute.Security_Enforcement_Resolute_public.Resolute.Resolute.compone
nt_has_authenticated_encryption
              description "This confirms the element has authenticated encryption."
       ]

...
method JavaCheckTrusted (component):
       "Use Java for isTrusted"
       [
       java
org.osate.securitylibrary.SecurityVerificationJava.bin.securityverification.SecurityClassifi
cationUtil.isTrusted()
       ]
```

(35) Both Resolute and Java methods are automated verification activities, which can be executed as part of the assurance case capabilities of ALISA. A portion of an assurance cases for the transport aircraft and the transport flight control system are shown in Table 7. These reference the verification plans for the transport aircraft security policies and requirements and verification plan for the aircraft control systems.

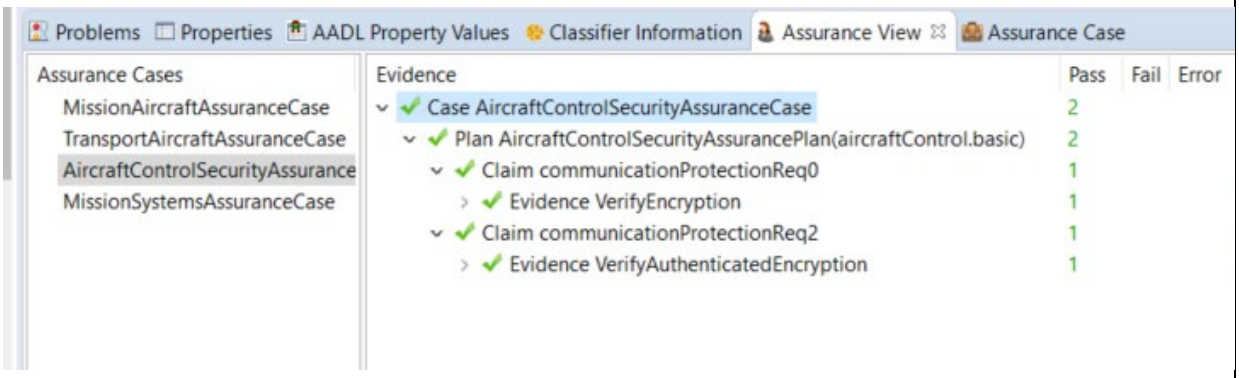
Table 7: Example Assurance Cases

```
assurance case TransportAircraftAssuranceCase for
TransportAircraft_Generic::transportAircraft
 [
assurance plan TransportAircraftAssurancePlanGeneric for
TransportAircraft_Generic::transportAircraft.generic
[
```

```
        description "Security Assurance Plan for Transport Aircraft"
        assure TransportAircraftSecurityPolicyVerificationPlan
TransportAircraftSecurityVerificationPlan

] ]

*****************************************************************************

assurance case AircraftControlSecurityAssuranceCase for AircraftControl_pkg::aircraftControl
 [

assurance plan AircraftControlSecurityAssurancePlan for
AircraftControl_pkg::aircraftControl.basic
[
        description "Security Assurance Plan for Transport Aircraft flight control system"
        assure AircraftControlSecurityVerificationPlan
] ]
```

(36) The partial output from running the assurance case for the aircraft control systems using the assurance case and verification plans shown in Table 7 are presented in Table 8. These are outputs of Resolute claim methods verifying encryption on communications.

Table 8: Partial Output of Aircraft Control Assurance Case



A.9 Information Security Levels

(37) There is a primary information security level property and an information security caveats property. Both property declarations are shown in Table 9.

(38) In instance models, information security levels and associated caveats are applicable only to data instances and to their associated classifiers in declarative models. However, to enable modeling at abstract levels (i.e. without having to define individual data components or runtime environments) it is useful to apply information security levels and caveats to systems, processes, devices, ports, and abstract components.

(39) Since the information security level and caveat properties are defined as inherit, a property association value within a component classifier is inherited by all subcomponents of instances of that classifier, which are instances of the component categories of the information security level or caveat property, unless overridden.

(40) With these modeling elements included, you can define a functional (conceptual) model where data is stored within and transferred among components, independent of the internal structure of the components or systems (i.e. complete implementations) using systems, processes and devices or creating models without defining a specific AADL category (i.e. using abstract). Thus, a complex cross domain system can be represented as discussed in the appendix: Cross Domain System Example.

Table 9: AADL Information Security Level Properties

```
-- Information Security Levels
       --
       Information_Security_Level: inherit enumeration (TopSecret, Secret, Confidential,
Unclassified)
       applies to (data, port, system, process, device, abstract);
       --
       Information_Security_Caveats: inherit aadlstring
       applies to (data, port, system, process, device, abstract);
```

(41) Information security properties are applied to AADL component categories that embody information (i.e. data) or those that can contain information either directly or indirectly as subcomponents. The abstract category is included, since it can be used in conceptual modeling and can be extended into one of the other categories listed.

(42) Caveats can be used to create multilateral-secure systems as in [Anderson 2008] Chapter 9. Combined with information security levels, this can be used to represent a system with a lattice-based permission architecture (see [Anderson 2008] Figure 9.6).

(43) The semantics of information security level properties are presented in Table 10.

Table 10: Semantics of Information Security Level Properties for AADL Component Categories

When declared for a data classifier, all data instances of the classifier have the specified security level and all subcomponents of those instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. data, abstract), unless the inherited value is overridden.

> All of the *provides data access* data classifiers for a data instance must have the same information security level as the data instance. This requirement is not enforced by the AADL property semantics and must be ensured by the user.

When declared for a data instance, that instance has the specified security level and all subcomponents inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. data, abstract), unless the inherited value is overridden.

> All of the *provides data access* data classifiers for a data instance must have the same information security level as the data instance. This requirement is not enforced by the AADL property semantics and must be ensured by the user.

When declared for a data or event data port, the information level and information security caveats apply to the data buffer or queue associated with the port (i.e. the data or event data port maps to a static variable in the source text that represents the data buffer or queue). The data passing through the port has the information security levels specified. The information level and information security caveats property values of the port and the data classifier for the port must be the same.

When declared for a system classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated system instances' external *data port*, *event data port* and *provides data access* features must have the same information security level as the system instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for a process classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated process instances' *data port*, *event data port* and *provides data access* features must have the same information security level as the process instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for a device classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated device instances' *data port* and *event data port* features must have the same information security level as the device instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for an abstract classifier, all extensions of that abstract classifier have the specified information security level consistent with the applicability and semantics of the information level security properties for the category of the extension.

Note that if a feature group is a feature of a component it is impacted, in that the features within the feature group, which have data classifiers, such as data ports, must comply with the semantics of information security levels presented in this section.

A.10 Information/Data Protection

(44) Data security classification properties are declared in the property set *SecuirtyClassificationProperties* and related enforcement and analysis properties in the property set *SecurityEnforcementProperties.*

(45) User defined classifications are declared in the property set *UserdefinedSecurityConstants.* Additional AADL modeling elements are available in the package *SecurityAnnexCustomPkg*.

A.11 Security Clearances

(46) The security clearance properties include a principal security classification (e.g. Top Secret, Secret, Confidential) and a supplemental statement (e.g. specialized authorizations or restrictions).

(47) A secondary clearance and secondary supplemental statement property is included. No assumption is made about the relationship of the *Security_Clearance* property and the *Secondary_Security_Clearance* property.

(48) Security Clearance properties*, as* shown in Table 11, are contained in the property set *SecurityClassificationProperties*. This property set can be edited to allow modification of the enumerated values.

Table 11: AADL Security Clearance Properties

```
--
	Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance) applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
	--
	Security_Clearance_Supplement: inherit aadlstring applies to (system, device,
processor, virtual processor, thread, thread group, subprogram, subprogram group, process,
abstract);
	--
Secondary_Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance) applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
	--
	Secondary_Security_Clearance_Supplement: inherit aadlstring applies to (system,
device, processor, virtual processor, thread, thread group, subprogram, subprogram group,
process, abstract);
```

(49) Security Clearance properties are applied to elements that can actively access and process information. The abstract category is included, since it can be extended into one of the other categories listed.

(50) Table 12 presents the semantics of security clearances.

Table 12: Semantics of Security Clearance Properties for AADL Component Categories

When security clearances are declared for a system classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

> When security clearances are declared for a system instance, that instance has the specified clearance or clearances and all subcomponents have the same clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a device classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. abstract subcomponents have the specified security) unless the inherited property value is overridden.

> When security clearances are declared for a device instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a processor classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. virtual processor, abstract) unless the inherited property value is overridden.

> When security clearances are declared for a processor instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a virtual processor classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. virtual processor, abstract) unless the inherited property value is overridden.

> When security clearances are declared for a virtual processor instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a thread classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, abstract) unless the inherited property value is overridden.

> When security clearances are declared for a thread instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a thread group classifier, all thread subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for threads unless the inherited property value is overridden.

When security clearances are declared for a subprogram classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or

clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, abstract) unless the inherited property value is overridden.

When security clearances are declared for a subprogram instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for a subprogram group classifier, all subprogram subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for subprograms unless the inherited property value is overridden.

When security clearances are declared for a process classifier, all instances have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, thread, abstract) unless the inherited property value is overridden.

When security clearances are declared for a process instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited property value is overridden.

When security clearances are declared for an abstract classifier, all extensions of that abstract classifier have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the category of the extension.

A.12 Security Level Property

(51) The property *Security_Level* is used where no distinction is required between security subjects (i.e. *Security_Clearance properties*) and security objects (i.e. *Information_Secruity_Level* properties) in modeling a system. For flexibility, we declare the *Security_Level_Caveats* property as an AADL string property. The declaration of the basic security level properties is shown in Table 13.

(52) The *Security_Level* property has the enumerated values as shown. However, these can be replaced or extended by a user. As with the other security classification properties, the basic security level properties are declared as inherit, allowing inheritance through the architecture hierarchy.

(53) The semantics of the *Security_Level* properties are those described in Table 10 and Table 12 for the Information Security Level and Security Clearance properties.

Table 13: Security_Level Properties

```
-- Note: The SEI property set has a SecurityLevel property that is an integer.
--
       Security_Level: inherit enumeration (TopSecret, Secret, Confidential, Unclassified)
applies to (system, processor, virtual processor, thread, thread group, subprogram,
subprogram group, data, port, process, device, abstract);
--
       Security_Level_Caveats: inherit aadlstring applies to (system, processor, virtual
processor, thread, thread group, subprogram, subprogram group, data, port, process, device,
abstract);
```

A.13 Trusted Components

(54) A trusted component or system is one that is relied upon to a specified extent to enforce a specified security policy and one that has been verified to some defined level to warrant that trust. In essence, a trusted entity is one whose failure would break a security policy (i.e. the component or system is trusted to function as expected and to be unmodified from the expected [Wiki-Trusted]. For example, the trusted components of Multiple Independent Levels of Security/Safety (MILS) from [Rushby 2008] are such that "trusted components ...depend only on very simple environments that can be provided with strong assurance."

(55) Trusted is not an inherited property. It must be specified for a component classifier or instance. The property declaration for Trusted is shown in Table 14.

Table 14: Trusted Property

```
Trusted : aadlboolean applies to (system, process, thread, thread group, subprogram,
subprogram group, processor, virtual processor, bus, virtual bus, abstract);
```

A.14 Property Set Modification

(56) The *SecurityClassificationProperties* and *SecurityEnforcementProperties* property sets can be edited by a user.

(57) A principal objective of these editable property sets is to enable the modification of enumerated values. For example, a user might edit the *Security_Level* property to have High, Medium, and Low values rather than the values declared in the security classifications property set, as shown in Table 15.

Table 15: Example Property Modification

```
       Security_Level: inherit enumeration (High, Medium, Low) applies to (system,
processor, virtual processor, thread, thread group, subprogram, subprogram group, data,
port, process, device, abstract);
```

(58) With this capability, specific property analysis plugins can be developed that use the original property name in developing the analysis code but reads the user-specified values from the modified property set when executing an analysis.

A.15 Analyzing Security Levels

(59) Resolute or other constraint or formal expression language as well as JAVA methods can be used to analyze security classifications for a system model.

(60) The annex includes example *Resolute* security functions and claims in the library (AADL package) *Security_Classifications_Resolute*. Excerpts from this library are shown in Table 16. A complete listing is included in the appendix.

(61) The JAVA package *securityverification* has example JAVA classes and methods for analysis. A partial listing of this package is included in the Appendix.

Table 16: Example Resolute Security Level Claims

```
        has_security_clearance (cp:component) <=
        ** "component " cp " has a security clearance" **
        has_property(cp, SecurityClassificationProperties::Security_Clearance)
        --
        has_top_secret_security_clearance (cp: component) <=
        ** " component " cp " has Top Secret security clearance" **
        property (cp, SecurityClassificationProperties::Security_Clearance, "No_Value") =
"TopSecret"
        --

        has_information_security_level (cp:component) <=
        ** "component " cp " has an information security level" **
        has_property(cp, SecurityClassificationProperties::Information_Security_Level)
        --
        has_top_secret_information_security_level (cp: component) <=
        ** " component " cp " has Top Secret security information security Level" **
        property (cp, SecurityClassificationProperties::Information_Security_Level,
"No_Value") = "TopSecret"
        --

       has_security_level (cp:component) <=
        ** "component " cp " has an information security level" **
        has_property(cp, SecurityClassificationProperties::Security_Level)

        has_top_secret_security_level (cp: component) <=
        ** " component " cp " has Top Secret security information security Level" **
        property (cp, SecurityClassificationProperties::Security_Level, "No_Value")
= "TopSecret"
        --

        all_subcomponents_have_security_level_or_are_trusted (cp: component) <=
        ** "all subcomponents of component " cp " have a value for_security level" **
        exists(sbx: subcomponents(cp)).(has_property (sbx,
SecurityClassificationProperties::Security_Level))
                and
        (forall(sb: subcomponents(cp)).(has_property (sb,
SecurityClassificationProperties::Security_Level))
        or
        has_property(sb,SecurityClassificationProperties::Trusted))

all_subcomponents_have_security (cp: component) <=
        ** "all subcomponents of component " cp " have a value for information security level
or security clearance" **
        exists(sbx: subcomponents(cp)).(has_property (sbx,
SecurityClassificationProperties::Information_Security_Level))
                and
        forall(sb: subcomponents(cp)).(has_property (sb,
SecurityClassificationProperties::Information_Security_Level) or
    has_property (sb,SecurityClassificationProperties::Security_Clearance))
    --
```

A.16 Encryption

(62) Encryption is declared using the *Encryption* property. The *Encryption* property is an AADL record type and includes fields that declare encryption details.

(63) The declarations for the record type property *Encryption* are shown Table 17, which is an excerpt from the *SecurityEnforcementProperties* property set. The declaration of the enumerated property fields *encryption_form*, *encryption_mode*, and *padding* include the entry *to_be_specified*, which can be used to require that the element, to which the encryption property is assigned, must have encryption defined at some point to complete the security specification for the system architecture.

Table 17: The Encryption Property

```
Encryption: record (
       description: aadlstring;
       -- an informal description of the encryption
       encryption_form: enumeration (no_encryption, symmetric, asymmetric, hybrid,
authenticated_encryption, authentication_only, to_be_specified);
       -- if the encryption form is hybrid both symmetric and asymmetric are used.
       encryption_mode: list of enumeration (no_encryption, ECB, CBC, CFB, CTR, GCM,
CBC_MAC, to_be_specified);
       -- list is needed for hybrid encryption
       encryption_algorithm: list of enumeration (no_encryption, OTP, DES, TripleDES, AES,
RSA, ECC, to_be_specified);
       -- a list is needed for hybrid encryption
       -- the mode and algorithm listings must correlate
       padding: enumeration (no_padding, OAEP, to_be_specified);
       --
       authenticated_encryption_type: enumeration (no_authenticated_encryption, GCM,
CBC_MAC, Encrypt_then_MAC, MAC_then_Encrypt, Encrypt_and_MAC, AEAD, signcryption,
double_RSA);
       key_type: list of SecurityEnforcementProperties::key_classifier; -- references
classifiers
       -- The key_type can be used to declare key length (i.e. a key classifier is declared
       -- with a Key_Length property association). A list is needed for hybrid encryption.
The key type
       -- can also be declared in the classifier for key instances or as a property of a key
instance.
       private_key: SecurityEnforcementProperties::key_instance;  -- references an instance
       public_key: SecurityEnforcementProperties::key_instance; -- references an instance
       single_key: SecurityEnforcementProperties::key_instance;  -- references an instance
       ) applies to (data, port, abstract, system, bus, memory, device, processor,
       virtual processor, virtual bus, connection, process, thread, flow);
              --
```

(64) The *key_type* field is a list of encryption key classifiers. These are used to declare key related properties (e.g. key length) for the encryption and as classifiers for key instances. Key related properties and property types are shown in Table 23.

(65) A summary of the semantics of the encryption property is provided in Table 18.

Table 18: Encryption Property Semantics

When encryption is declared for a data classifier or data instance, the data is encrypted as specified.
When declared for a data or event data port, the specified encryption applies to the data buffer or queue associated with the port (i.e. the data or event data port maps to a static variable in the source text that represents the data buffer or queue). The data passing through the port is encrypted as specified. The *Encryption* property value of the port and the data classifier for the port must be the same.
When encryption is declared for an abstract classifier, its extensions have the specified encryption consistent with the applicability and semantics of encryption properties for the category of the extension.
When encryption is declared for a system classifier or instance, all data within instances of the system classifier or within the system instance is encrypted as specified.
When encryption is declared for a bus classifier or a bus instance, encryption is provided as specified for all data transmitted through instances of the bus classifier or for the bus instance.
When encryption is declared for a virtual bus classifier or a virtual bus instance, encryption is provided as specified for all data transmitted through instances of the virtual bus classifier or for the virtual bus instance.
When encryption is declared for a memory classifier or a memory instance, encryption is provided as specified for all data contained in instances of the memory classifier or for the memory instance.
When encryption is declared for a device classifier or a device instance encryption is provided as specified for all data contained in instances of the device classifier or for the device instances and encryption is provided as specified for all data transmitted through instances of the device classifier or for the device instance.
When encryption is declared for a processor classifier or a processor instance, encryption is provided as specified for all data transmitted through instances of the processor classifier or through the processor instance.
When encryption is declared for a virtual processor classifier or a virtual processor instance, encryption is provided as specified for all data transmitted through instances of the virtual processor classifier or through the virtual processor instance.
When encryption is declared for a connection instance, the supporting transmission components must provide the specified encryption for all data transmitted through the connection. That is, the connection requires the encryption and decryption of data as specified, such that the input and output data types at each end of the connection are the same.
When encryption is declared for a flow, the components at each end of the flow must support the encryption as specified.
Some of the encryption declaration record fields can be specified as “to be specified” for an architecture model. For example, the *encryption_form* property field within the *encryption* property can be assigned the value *to_be_specified*, perhaps early in the development process and later a specific encryption property assignment can be made.

A.17 Data Authentication

(66) In general, data authentication encompasses both authenticating the origin and the integrity of data. The *Data_Authentication* property is a record property as shown in Table 19. In some cases, only one of these aspects (integrity or authenticity) is specified using the *Data_Authentication* property. For example, declaring a

message digest generated through a cryptographic hash function to ensure integrity.

Table 19: Data Authentication Property

```
Data_Authentication:  record
       (
       description: aadlstring;
       authentication_form: enumeration (no_authentication, MAC, MIC, signature,
signcryption, double_RSA, to_be_specified);
       authentication_algorithm: enumeration (no_authentication, RSA, ElGamal, DSA, CBC_MAC,
GCM, HMAC, UMAC, to_be_specified);
       padding : enumeration (no_padding, OAEP, to_be_specified);
       --
       -- key_Length is declared in the authentication key type classifier or
       -- in the classifier for the authentication key instance or for the key instance
       --
       hash_Length: Size; -- optional, if the message is hashed before authentication. Does
not apply to authenticated encryption.
       hash_algorithm: enumeration (no_hash, MD5, SHA1, SHA256, SHA512, SHA3,
to_be_specified);
       authentication_key_type: list of SecurityEnforcementProperties::key_classifier; --
references a classifier
       authentication_key: SecurityEnforcementProperties::key_Instance;  -- references an
instance
       )
applies to (data, port, abstract, system, bus, memory, device, processor, virtual processor,
virtual bus, connection, process, thread, flow);
```

(67) The data authentication property is used to declare that a data instance has the authentication as specified or that authentication is required by a connection (i.e. the data transmitted through the connection must be authenticated as specified) or that authentication is provided by a component (e.g. bus or virtual bus supporting a connection that requires data authentication) or when applied to a component that can contain data (e.g. system, device, abstract) it declares that all data contained in the component has authentication as specified.

(68) Key lengths for authentication are specified within the declarations for authentication key types or key instances.

(69) The semantics of the Data_Authentication property are described in Table 20.

Table 20: Data Authentication Property Semantics

When Data Authentication is declared for a data classifier or data instance, the data is authenticated as specified. When declared for a data or event data port, the specified data authentication applies to the data buffer or queue associated with the port (i.e. the data or event data port maps to a static variable in the source text that represents the data buffer or queue). The data passing through the port is authenticated as specified. The *Data_Authentication* property value of the port and the data classifier for the port must be the same.

When data authentication is declared for an abstract classifier, its extensions have the specified data authentication consistent with the applicability and semantics of data authentication properties for the category of the extension.

When data authentication is declared for a system classifier or instance, all data within instances of the system classifier or within the system instance is encrypted as specified.

When data authentication is declared for a bus classifier or a bus instance, data authentication is provided as specified for all data transmitted through instances of the bus classifier or for the bus instance.

When data authentication is declared for a virtual bus classifier or a virtual bus instance, data authentication is provided as specified for all data transmitted through instances of the virtual bus classifier or for the virtual bus instance.

When data authentication is declared for a memory classifier or a memory instance, data authentication is provided as specified for all data contained in instances of the memory classifier or for the memory instance.

When data authentication is declared for a device classifier or a device instance data authentication is provided as specified for all data contained in instances of the device classifier or for the device instances and data authentication is provided as specified for all data transmitted through instances of the device classifier or for the device instance.

When data authentication is declared for a processor classifier or a processor instance, data authentication is provided as specified for all data transmitted through instances of the processor classifier or through the processor instance.

When data authentication is declared for a virtual processor classifier or a virtual processor instance, data authentication is provided as specified for all data transmitted through instances of the virtual processor classifier or through the virtual processor instance.

When data authentication is declared for a connection instance, the supporting transmission components must provide the specified data authentication for all data transmitted through the connection. That is, the connection requires the data authentication and decryption of data as specified, such that the input and output data types at each end of the connection are the same.

When declared for a flow, the components at each end of the flow must support data authentication as specified.

A.18 Authenticated Encryption (AE)

(70) Authenticated Encryption provides message confidentiality, authenticity, and integrity by combining an encryption algorithm with an appropriate authentication mechanism.

(71) Both the *Encryption* property and the *Data_Authentication* property are required to completely specify authenticated encryption. An example specification of authenticated encryption using 2048-bit RSA encryption and an HMAC for authentication and integrity is shown in Table 21.

Table 21: Authenticated Encryption Example

```
SecurityEnforcementProperties::Encryption =>
[
       description => "This defines authenticated encryption using 2048-bit RSA encryption
and an HMAC function. The recipient's public key is used for encrypting the message and a
hash function with a symmetric key is used to create a MAC”;
       encryption_form => authenticated_encryption;
```

```
        authenticated_encryption_type=> Encrypt_then_MAC;
        encryption_algorithm => (RSA);
        key_length => (2048 bits);
        key_type => classifier (RSA2048);
        private_key => reference (RSA2048Private);
        public_key => reference (RSA2048Public);
];

SecurityEnforcementProperties::Data_Authentication =>
[
        description => "This defines the data authentication for the authenticated encryption
property for netBusB.";
        authentication_form => MAC;
        authentication_algorithm => HMAC-SHA256;
        hash_length => 256 bits; -- is this message digest length
        authentication_key=> reference (SymmeticKey128); -- the symmetric key
  ];
```

A.19 Encryption and Key Management

(72) An encryption or message authentication key can be declared as an abstract classifier or a data classifier. Either classifier can be extended. The examples in Table 22 extend the abstract key to create data classifiers.

Table 22: Abstract Component 'key'

```
-- A 'key' is defined as an abstract component.

abstract key
end key;

-- extend abstract key to data classifiers

        data symmetricKey extends key
        end symmetricKey;

        data publicKey extends key
        end publicKey;

        data privateKey extends key
        end privateKey;

        data AESKey256 extends symmetricKey
                properties
                        SecurityEnforcementProperties::keyLength => 256 bits;
        end AESKey256;
```

(73) Encryption and authentication key-related properties are shown in Table 23. The *CryptoPeriod* is the length of time a key or certificate is valid. The *Key_Classifier* is the type for the *key_type* field of the Encryption property shown in Table 17.

Table 23: Encryption and Authentication Key-Related Properties and Types

```
        Key_Length: Size applies to (abstract, data);
        --
```

```
Crypto_Period: Time applies to (abstract, data);
-- The Crypto Period is the time span that a cryptographic key is authorized for use
Text_Type: enumeration (plainText, cipherText) applies to (abstract, data);
-- type declarations for key classifiers and instances
Key_Classifier: type classifier (abstract, data);
Key_Instance: type reference (data);
```

A.20 Key and Certificate Management

(74) Key and certificate management modeling and analysis is done using key-related properties (Table 23) and key classifiers. Examples are shown in Table 22.

(75) Crypto key management encompasses the policies and procedures for creation, management, distribution, use, storage, and revocation of crypto keys and digital certificates.

(76) Key or certificate management can be modeled using an AADL annotated with requisite security properties. For example, a certificate can be modeled using a data classifier. As shown in Table 24.

Table 24: Certificate Modeled as an AADL Data Component

```
abstract CertificateAbs
end CertificateAbs;

data Certificate extends CertificateAbs
end Certificate;

data implementation Certificate.SSL_TLS
subcomponents
Subject: data subject.certificate;
Issuer: data issuer.certificate;
PeriodOfValidity: data periodOfValidity.certificate;
AdminInformation: data adminInformation.certificate;
ExtendedInformatio: data extendedInformation.certificate;
end Certificate.SSL_TLS;
```

A.21 Analyzing Encryption and Data Authentication

(77) The annex includes example *Resolute* functions and claims for analyzing encryption in the library (AADL package) *Secuirty_Enforcement_Resolute*. The library provides a basic set for use and exemplars for users to develop additional functions and claims. Some encryption analysis claims from the *Secuirty_Enforcement_Resolute* library are shown in Table 25.

Table 25: Example Resolute Encryption Functions and Claims

```
-- Encryption
has_encryption(el: aadl): bool =
has_property (el, SecurityEnforcementProperties::Encryption)

provides_encryption (cp:component): bool =
has_property (cp, SecurityEnforcementProperties::Encryption)
```

```
      connection_requires_encryption (conn: connection) : bool =
    has_property (conn, SecurityEnforcementProperties::Encryption)

    is_encrypted (dt: data) <=
    ** "The data element " dt " is encrypted" **
       has_property (dt, SecurityEnforcementProperties::Encryption)
     component_has_authenticated_encryption (cp: component) <=
      **"The aadl element " cp " has or provides authenticated encryption"**
    (property_member (property (cp, SecurityEnforcementProperties::Encryption),
    "encryption_form") = authenticated_encryption)
```

(78) The annex provides example *Resolute* functions and claims for analyzing data authentication in the library (AADL package) *Secuirty_Enforcement_Resolute*. The library provides a basic set for use and exemplars for users to develop additional functions and claims. Some encryption analysis claims from the *Secuirty_Enforcement_Resolute* library are shown in .

Table 26: Example Data Authentication Resolute Functions

```
    has_data_authentication (el: aadl): bool =
    has_property (el, SecurityEnforcementProperties::Data_Authentication)

    provides_data_authenticated_encryption (cp: component): bool =
    has_property (cp, SecurityEnforcementProperties::Data_Authentication)

    connection_requires_authenticated_encryption (conn: connection): bool =
    has_property (cp, SecurityEnforcementProperties::Data_Authentication)
```

A.22 Access Control and Protection

(79) The security annex provides the capability to model, assess, and assure security access control. Access control and protection encompasses the classification of subjects (users/subjects) and objects (information containers and computing resources)—authorization, authentication, and access control management (i.e. assigning and modifying classifications).

A.23 Subject Authentication

(80) The Subject Authentication property declares that a subject (component instance) can participate or participates in authentication as specified, including authentication negotiations employing the specified authentication protocol, or that the component (e.g. a bus or virtual bus) supports the authentication specified. The declaration for the *Subject_Authentication* property is shown in Table 27.

(81) All components engaged in authentications, including negotiations, must have the same subject authentication protocol. Each component bound to a component supporting authentication must have the same subject authentication protocol.

(82) When applied to an end-to-end flow, the components at each end of the flow must support subject authentication as specified.

(83) Any of the enumeration record fields of the Subject_Authentication property can be extended by users to include custom subject authentication approaches.

Table 27: The Subject Authentication Property

```
Subject_Authentication: record
    (
    description: aadlstring;
    authentication_access_type: enumeration (no_authentication, single_password, smart_card,
ip_addr, two_factor, multi_layered, bio_metric, to_be_specified);
    two_and_multi_layered_factors    : list of enumeration (no_multifactor, smart_card,
token, PIV, OTP, biometric, multi_layered, to_be_specified);
       -- the listing is such that the initial factor required for authentication is listed
first, the second factor is listed second, etc.
    authentication_protocol: enumeration (no_authentication, cert_services, EAP, PAP, SPAP,
CHAP, MS_CHAP, Radius, IAS, Kerberos, SSL, TLS, NTLM, to_be_specified) ;
    authentication_role: enumeration (no_authentication, authenticator, accessor, provider,
requirer, mutual);
    )
    applies to (abstract, system, process, thread,device, processor, virtual processor,
connection, bus, virtual bus, flow);
```

A.24 Secure flows

(84) An end-to-end flow can be used to specify that encrypted and/or authenticated data is transmitted and/or that subject authentication is employed between two components. The **aadlboolean** property *Secure_Flow* with the value true, is used to declare such a flow.

(85) The *Encryption*, *Data_Authentication*, and *Subject_Authentication* properties can be assigned to an end-end-to end flow, declaring the components at each end of the flow are interacting using the encryption and/or authentication specified.

A.25 Authorization

(86) Authorization establishes the access rights and actions of subjects and the details of authentication procedures and protocols. The security annex does not specifically address the modeling or analysis of authorization activities.

A.26 Access Control Modeling

(87) Core AADL elements and properties as well as security annex properties are employed in access control modeling and analysis. Specifically, the security properties are used to classify subjects and objects and specify authentication and the core elements define the interconnections and access paths.

A.27 Security Architectures

(88) In modeling and analyzing specialized security architectures, the security annex utilizes AADL core modeling capabilities, mappings of security-specific abstractions onto AADL elements, and specialized security properties and constructs.

(89) The annex can be used to model and analyze specialized security architectures such as Multiple Independent Levels of Security (MILS) and Cross Domain Solutions (CDS).

A.28 Analyzing Vulnerabilities/Threats/Attacks

(90) --TBD

Appendix B Informative Section

The section presents support material for the use of the security annex and examples of the application of the security annex.

B.1 Security Policies and Requirements Examples

(91) This example presents security policies and requirements for a generic E-enabled aircraft model. As an illustrative example, it is not comprehensive and is not intended to be used to define security policies or requirements for any operational aircraft.

(92) An AADL model of a generic E-enabled air transport aircraft is shown in Figure 1. Where the aircraft is represented as three major functional subsystems, aircraft control, airline information services, passenger Information and Entertainment Services as well as an aircraft airframe. The triad of major functional subsystems is a partitioning into three distinct security domains for the aircraft.

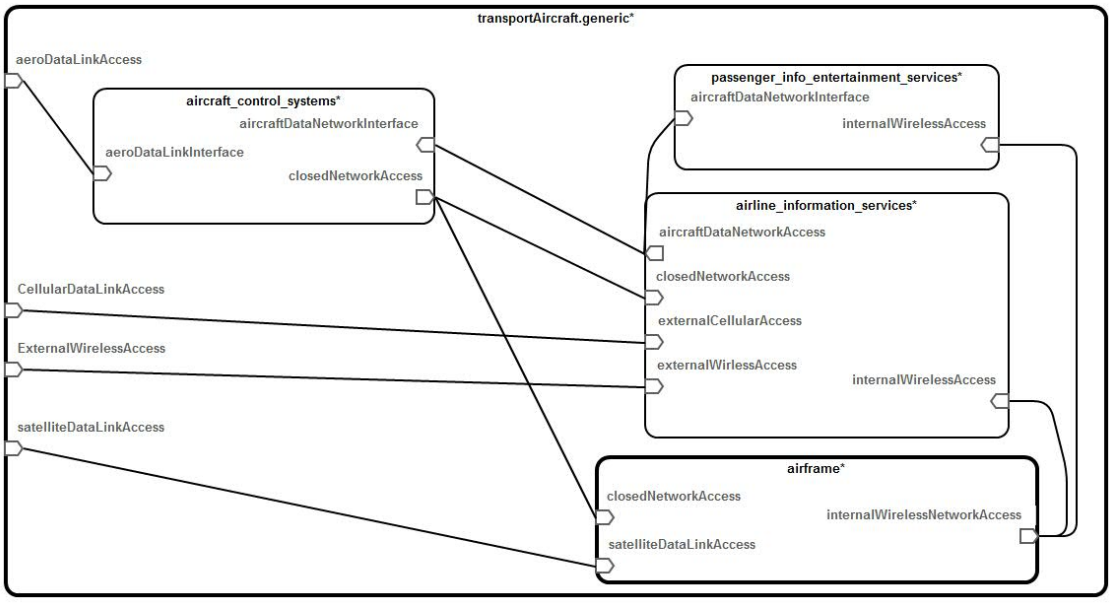


Figure 1: Generic E-Enabled Transport Aircraft

B.2 Security Policies

(93) The aircraft security policies are contained in a system requirements set *TransportAircraftSecuirtyPolicies.reqspec*. The developmental stakeholders for the system are from the *DevelopmentTeam* organization declared in the file *Stakeholders_Development.org*.

(94) An excerpt of security policies from the *TransportAircraftSecurityPolicies.reqspec* requirements set file is shown in Table 28.

Table 28: Example System Security Policies

```
system requirements TransportAircraftSystemSecurityPolicies: "System-Wide Security
Policies"
for TransportAircraftSystem_Generic::AirTransportOperationalSystem.multipassenger

[
   description "These are the high level (system) security policies for the
                Aircraft."

   requirement Security:  "System Security must be provided"
       [
        description "Security protections that meets FAA aircraft security and
                     flight worthiness certification standards must be provided ."

       ]

       requirement MasterSecurityPolicy: "A Master System Security Policy must be
developed and certified."
       [
       description "A master system security policy document must be developed
                    and certified by all of the agencies and organizations
                    involved in flight certification of the aircraft."
       ]

       requirement  AccessControlPolicy: "Security Controlled Access to all
                   Aircraft Systems and Resources must be provided."
                    [

         description "Access to all Aircraft operational and maintenance
                      Systems and Resources shall be permitted only by
                      authorized personnel."

         development stakeholder DevelopmentTeam.PrincipalEngineer
                                 DevelopmentTeam.SecurityEngineer
                    ]

       requirement SecureCommunicationsPolicy: "Secure communications must be
                   provided for all flight- and safety-critical systems."
                    [
          description "Communication systems must provide security measures to
                       ensure only authorized access and use."

             development stakeholder DevelopmentTeam.PrincipalEngineer
                                     DevelopmentTeam.EncryptionExpert
                    ]

]
```

B.3 Security Requirements

(95) Security requirements are captured as system requirements and organized into named system requirements sets. Each set applies to a specific architecture

system, subsystem, or element. An example for the aircraft control system ,*AircraftControl_pkg::aircraftControl.basic*, is shown in Table 29.

Table 29: Example Security Requirements Sets for Aircraft Control System

```
// Copyright 2019 Carnegie Mellon University. See Notice.txt
// Distribution Statement A: Approved for Public Release; Distribution is Unlimited.
//

system requirements securityReqs for AircraftControl_pkg::aircraftControl.basic
[
       requirement aircraftSystemsInformationSecurity: "Aircraft Systems Information
Security/Protection (ASISP) must be provided"
       [
              description "All aircraft control and flight information systems must have
security protection to ensure confidentiality, integrity, and availability."
       ]


       requirement securityAccessReq:  "Access to all aircraft data must be only by
authorized and authenticated entities"
       [
              description "All aircraft operational and performance data systems must have
security protection to ensure access only by authorized and authenticated entities."
       ]

       requirement communicationProtectionReq0: "All external and internal communications
relating to aircraft control and operation must be secure."
       [
              description "All aircraft communication systems must have security protection
to ensure access only by authorized and authenticated entities."
       ]

       requirement  communicationProtectionReq1: "All aircraft external and internal
communication for aircraft control and flight operations must employ encryption algorithms"
        [
         description "All aircraft external and internal communication for aircraft control
and flight operations must employ encryption algorithms that meet or exceed the standards
defined in NIST publication FIPS 140-2 or any superseding document that has been released.”
         ]
]
```

B.4 Security Policy Verification

(96) Verification of policies and security requirements are accomplished with the verify capabilities (i.e. the *verify* notation) of ALISA (OSATE). The procedures and activities are organized using the assurance case and verification capabilities within ALISA (i.e. the *Alisa* and *Assure* notations). The results are captured using the *Assure* notation with ALISA (OSATE).

B.5 Cross Domain System Example

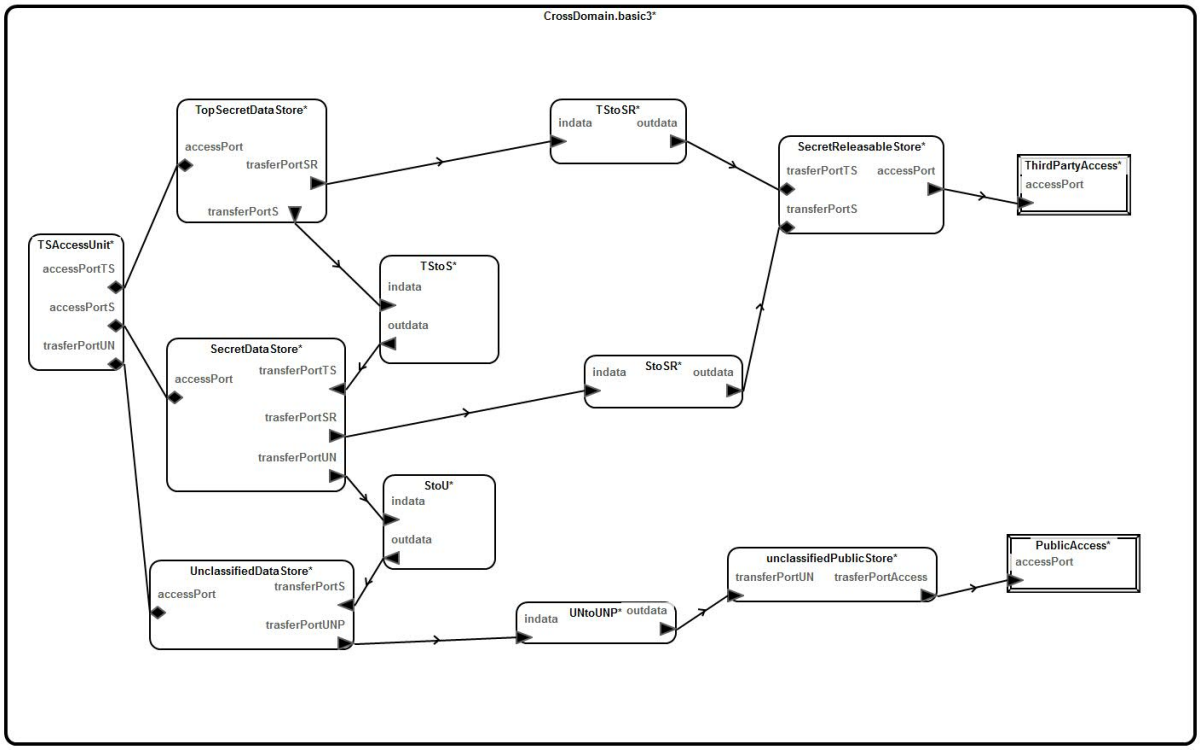
(97) This is a generalized cross domain solution (CDS) example that is a blend from the references [Gehani 2012] and [Smith 2015]. It is not intended to represent any operational system.

(98) The model consists of three primary data stores (top secret, secret, and unclassified) and two data stores for data that can be released (secret releasable and unclassified for public release). There is a trusted controller (subject) who can access and modify all three data stores (e.g. can transfer data from a lower to a higher classification or can filter data from a higher to a lower classification). There are downgrading filters that downgrade top secret to secret, secret to unclassified, top secret to secret releasable, secret to secret releasable, and unclassified to unclassified public release.

(99) To simplify the example, we use a single classification property *Security_Level* rather than the security clearance and information security levels properties, which distinguish between subjects working with data and the objects (data). The property sets used in this work are shown in the Appendix B.6.

(100) The AADL graphical representation of the cross domain solution architecture used in this work is shown in Figure 2.

Figure 2: AADL Model of the Cross Domain Solution Architecture



(101) The three principal data stores are TopSecretStore, SecretStore, and UnclassifiedDataStore. The unclassified data store contains confidential and other data that is not for public access or use and as needed, is filtered through the filter UNtoUNP, which results in unclassified data for public release. The other filters filter TopSecret to Secret (i.e. TStoS), TopSecret to Secret with restrictions (TStoSR), Secret to Unclassified (StoU), and Secret to Secret with restrictions (StoSR). The data stores SecretReleaseableStore and unclassifiedPublicStore are accessible

from external entities (subjects). The TSAccessUnit is the system that can access and modify all three principal data stores (e.g. can add data and "pump" data from a lower to a higher classification or can filter data from a higher to a lower classification). The TSAccessUnit and all of the filters are 'Trusted' systems.

(102) Using AADL property associations, appropriate security levels are assigned to components. Examples are shown in Table 30.

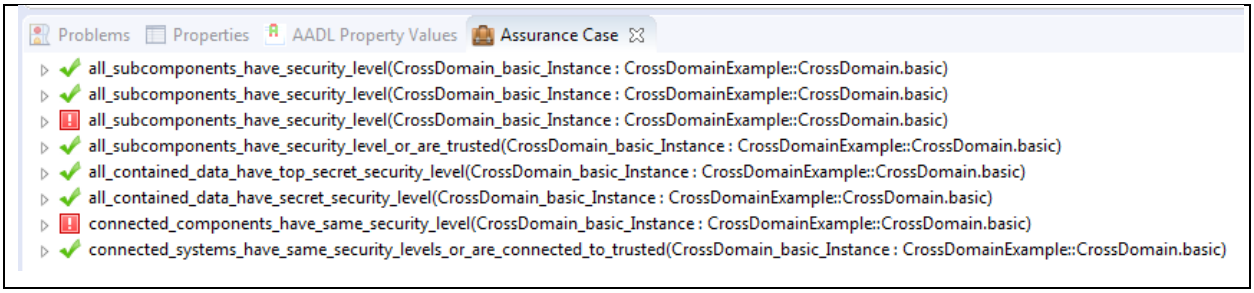
Table 30: Security Level Property Associations

```
                    -- security level properties --------
SecurityClassificationProperties::Security_Level => TopSecret applies to TopSecretDataStore;
SecurityClassificationProperties::Security_Level => TopSecret applies to
TopSecretDataStore.topsecretdata;
SecurityClassificationProperties::Security_Level => Secret  applies to SecretDataStore;
SecurityClassificationProperties::Security_Level => Unclassified  applies to
UnclassifiedDataStore;
SecurityClassificationProperties::Security_Level => Unclassified applies to
unclassifiedPublicStore;
SecurityClassificationProperties::Security_Level => Secret applies to SecretReleasableStore;
SecurityClassificationProperties::Security_Level => Unclassified  applies to PublicAccess;
SecurityClassificationProperties::Security_Level => Secret  applies to ThirdPartyAccess;
                    -- trusted components (filters)
SecurityClassificationProperties::Trusted => true applies to TStoS;
SecurityClassificationProperties::Trusted => true applies to StoU;
SecurityClassificationProperties::Trusted => true applies to UNtoUNP;
SecurityClassificationProperties::Trusted => true applies to StoSR;
SecurityClassificationProperties::Trusted => true applies to TStoSR;
SecurityClassificationProperties::Trusted => true applies to TSAccessUnit;
```

(103) Example Resolute prove statements assessing security classifications within the cross domain system and their results are shown in Table 31. These ensure that all components have an assigned security level or are trusted and that all connected components are connected to components of the same security level or are connected to trusted components.

Table 31: Resolute Claims and Analysis Results

```
                            -- security level checks
prove all_subcomponents_have_security_level(this.TopSecretDataStore) -- should be true
prove all_subcomponents_have_security_level(this.SecretDataStore) -- should be true
prove all_subcomponents_have_security_level (this)  -- not true because some are trusted
prove all_subcomponents_have_security_level_or_are_trusted (this)  -- should be true
prove all_contained_data_have_top_secret_security_level(this.TopSecretDataStore) -- should
be true
prove all_contained_data_have_secret_security_level(this.SecretDataStore) -- should be true
                            -- security connection checks
prove connected_components_have_same_security_level (this) -- should be false (some trusted)
prove connected_systems_have_same_security_levels_or_are_connected_to_trusted(this) --
should be true
```



(104) In a modified version of the model, a bus is included that represents the network connection between the Secret Releasable Store System and the Third Party Access Device. The specification of the authenticated encryption that is required by the connection and is provided by the bus is shown in Table 32. Note that both the encryption property and the Data Authentication property must be specified for authenticated encryption.

Table 32: Example Authenticated Encryption Property Associations

```
SecurityEnforcementProperties::Encryption =>
  [
       description => "Defines authenticated encryption using 2048 bit RSA encryption on the
message and an HMAC function of the encrypted message for authentication and integrity. The
recipient's RSA public key is used for encrypting the message and a hash function with
symmetric (secret) key is used to create a MAC";
       encryption_form => authenticated_encryption;
       authenticated_encryption_type=> Encrypt_then_MAC;
       padding => OAEP;
       encryption_algorithm => (RSA);
       key_length => (2048 bits);
       key_type => classifier (RSA2048);
       public_key => reference (RSA2048RecipientPublic); -- recipient's public key
   ] applies to netBus;

SecurityEnforcementProperties::Data_Authentication =>
   [
       description => "Defines the data authentication for the authenticated encryption
property for netbus using a symmetric key HMAC.";
       authentication_form => MAC;
       authentication_algorithm => "HMAC-SHA256";
       hash_length => 256 bits;  -- the message digest length
       key_length => 128 bits;  -- symmetric key
   ] applies to netBus;
```

(105) In a further extension of the model, we use a MILS inspired architecture for the TSAccessUnit implementation. The structure of the TSAccessUnit architecture implementation is shown in Figure 3.

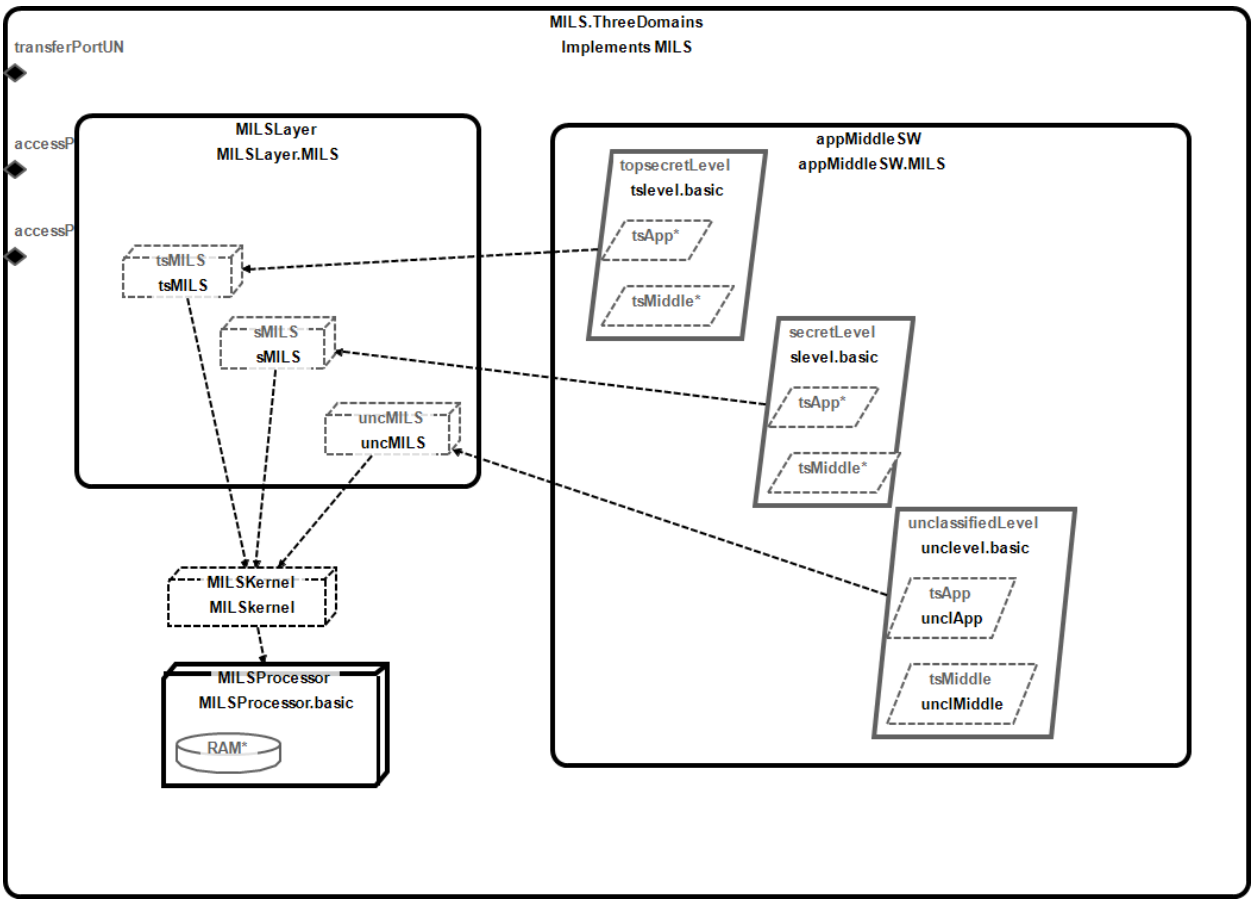


Figure 3: MILS Architecture of the TSAccessUnit

(106) The architecture is layered where the application software and middleware are modeled as threads within separate security level processes. There are top secret, secret, and unclassified security level processes. Each process is bound to an individual virtual processor, resulting in one virtual machine partition for each of the security levels. These partitions are bound to a MILS separation kernel (a virtual processor) that is bound to a hardware processor.

(107) The MILS kernel schedules each security level virtual processor on a static timeline. The binding of the MILSKernel to the hardware processor provides the memory for each of the partitions and these are managed at runtime as separate space partitions, that is the Runtime_Protection property is true for each of the processes.

(108) The AADL textual specification for the MILS implementation is shown in Table 33.

Table 33: The Three Domain MILS System Implementation

```
system MILS
       features
              accessPortTS: in out data port;
              accessPortS:  in out data port;
              transferPortUN: in out data port;
       end MILS;
```

```
system implementation MILS.ThreeDomains
	subcomponents
		appMiddleSW: system appMiddleSW.MILS;
		MILSLayer: system MILSLayer.MILS;
		MILSKernel: virtual processor MILSKernel;
		MILSProcessor: processor MILSProcessor.basic;
	--
	properties                    --
	-- Schedule the partitions on a fixed timeline
	Scheduling_Protocol => (FixedTimeline) applies to MILSKernel;
--
--	Bind the applications to the virtual processors
	Actual_Processor_Binding => (reference (MILSLayer.tsMILS)) applies to
appMiddleSW.topsecretLevel;
	Actual_Processor_Binding => (reference (MILSLayer.sMILS)) applies to
appMiddleSW.secretLevel;
	Actual_Processor_Binding => (reference (MILSLayer.uncMILS)) applies to
appMiddleSW.unclassifiedLevel;
--
-- Bind the virtual processors to the separation kernel
	Actual_Processor_Binding => (reference (MILSKernel)) applies to MILSLayer.tsMILS;
	Actual_Processor_Binding => (reference (MILSKernel)) applies to MILSLayer.sMILS;
	Actual_Processor_Binding => (reference (MILSKernel)) applies to MILSLayer.uncMILS;
-- Bind MILS separation kernel to the hardware processor
	Actual_Processor_Binding => (reference (MILSProcessor)) applies to MILSKernel;

	end MILS.ThreeDomains;
```

B.6 AADL Property Files

(109) The security property file SecurityClassificationProperties is shown in Table 34.

Table 34: Security Classification Properties

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is Unlimited.

property set SecurityClassificationProperties is

-------------------------------------------------------------------
-------------------------------------------------------------------
       -- This property set is editable.
       -- A principal objective of these editable property sets is
       -- to enable the modification of enumerated values.
       -- For example, a user might edit the Security_Level property
       -- to have High, Medium, and Low values rather than the values
       -- assigned in this property set.
-------------------------------------------------------------------
-------------------------------------------------------------------
--
-- Security Classifications
--
       -- Security Clearances
       --
       -- The security clearance properties include a primary
       -- security classification (e.g. Top Secret, Secret, Confidential)
       -- and an optional supplemental security statement.
       --
       Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance)
       applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);
       --
       Security_Clearance_Supplement: inherit aadlstring
       applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);
       --
       -- The secondary security clearance is provided in the event of multiple clearances
       -- (e.g. a clearance from two different agencies.) The secondary clearance also has a
       -- primary security classification and an optional supplemental security statement.
       --
       -- No assumption is made about the relationship between the Security_Clearance
property
       -- and the Secondary_Security_Clearance property.
       --
       Secondary_Security_Clearance: inherit  enumeration (TopSecret, Secret, Confidential,
No_Clearance)
       applies to (system, device, processor, virtual processor, thread, thread group,
subprogram, subprogram group, process, abstract);
       --
       Secondary_Security_Clearance_Supplement: inherit aadlstring
       applies to (system, device, processor, virtual processor, thread,  thread group,
subprogram, subprogram group, process, abstract);
-----------------------------------------
-- Information Security Levels
       --
       -- The information security level properties include a primary
       -- data security classification (e.g. Top Secret, Secret, Confidential)
```

```
        -- and a caveat data security statement (e.g. control markings).
        --
        Information_Security_Level: inherit enumeration (TopSecret, Secret, Confidential,
Unclassified)
        applies to (data, port, system, process, device, abstract);
        --
    -- The Information security caveats property is an enumeration.
        -- The values included in the declaration of the caveats
        -- represent a partial listing.
        -- The values can be changed and added to meet specific project or organizational
requirements.
        -- When the  security level and caveat classification are "Top Secret" and
"//SI/TK//RELIDO"
        -- this specifies the concatenated classification "TOP SECRET//SI/TK//RELIDO"
        --
        Information_Security_Caveats: inherit list of enumeration (FOUO, NOFORN,
NOCONTRACTOR, PROPIN, IMCON, ORCON)
        applies to (data, port, system, process, device, abstract);

----------------------------------------
        --  Security Levels
        --
        -- For models which do not need to differentiate between subject security clearances
and
        -- object (information) security levels, the SecurityLevel and SecurityLevelCaveats
properties can be used.
        -- These are defined with conventional TopSecret, Secret, Confidential, and
Unclassified.
        -- However, these can be changed to more generic values (e.g. High, Medium, Low) as
desired.
        --
        Security_Level: inherit enumeration (TopSecret, Secret, Confidential, Unclassified)
        applies to ( system, processor, virtual processor, thread, thread group, subprogram,
subprogram group,  data, port, process, device, abstract);
        --
        Security_Level_Caveats: inherit list of enumeration (FOUO, NOFORN, NOCONTRACTOR,
PROPIN, IMCON, ORCON)
        applies to (system, processor, virtual processor, thread, thread group, subprogram,
subprogram group,  data, port, process, device, abstract);

----------------------------------------
        -- Trusted component
        --
        -- We use trusted component/system to mean a component/system that is relied upon to
a specified extent
        -- to enforce a specified security policy and has been verified to some level to
warrant that trust.
        -- For example the trusted components of MILS [Rushby] "trusted components ...
        --  depend only on very simple environments that can be provided with strong
assurance."
        --
        -- [Rushby] Rushby, J. Separation and Integration in MILS (The MILS Constitution),
        -- Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA.
        -- https://pdfs.semanticscholar.org/0398/5ca22524e10f6fab9dd966c61c4ab3de7f74.pdf
        --
        Trusted : aadlboolean applies to (system, process, thread, thread group, subprogram,
subprogram group,
                processor, virtual processor, bus, virtual bus, abstract);

end SecurityClassificationProperties;
```

(110) The security property file SecurityEnforcementProperties is shown in Table 35.

Table 35: Security Enforcement Properties

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is Unlimited.

property set SecurityEnforcementProperties is

-----------------------------------------------------
-- Data Encryption
-----------------------------------------------------

       -- The encryption property is used to declare that a data instance is encrypted as
specified or that encryption is required by a connection
       -- or that encryption is provided by a component (e.g. bus or virtual bus supporting
a connection that requires encryption) or when applied
       -- to a component that can contain data (e.g. system, device, abstract) it declares
that all data contained in the component is encrypted as specified
       -- or that the component encrypts data received as specified and outputs the
encrypted data.

       Encryption: record (
              description                  : aadlstring;
              -- an informal description of the encryption
              encryption_form              : enumeration (no_encryption, symmetric,
asymmetric, hybrid, authenticated_encryption, authentication_only, to_be_specified);
              -- if the encryption form is hybrid both symmetric and asymmetric are used.
              encryption_mode              : list of enumeration (no_encryption, ECB, CBC,
CFB, CTR, GCM, CBC_MAC, to_be_specified);
              -- list is needed for hybrid encryption
              encryption_algorithm: list of enumeration (no_encryption, OTP, DES,
TripleDES, AES, RSA, ECC, to_be_specified);
              -- a list is needed for hybrid encryption
              -- the mode and algorithm listings must correlate
              padding: enumeration (no_padding, OAEP, to_be_specified);
              --
              authenticated_encryption_type: enumeration (no_authenticated_encryption, GCM,
CBC_MAC, Encrypt_then_MAC, MAC_then_Encrypt, Encrypt_and_MAC, AEAD, signcryption,
double_RSA);
              key_type                     : list of
SecurityEnforcementProperties::key_classifier; -- references classifiers
              -- The key_type can be used to declare key length (i.e. a key classifier is
declared
              -- with a Key_Length property association). A list is needed for hybrid
encryption. The key type
              -- can also be declared in the classifier for key instances or as a property
of a key instance.
              private_key           : SecurityEnforcementProperties::key_instance;  --
references an instance
              public_key                   : SecurityEnforcementProperties::key_instance; -
- references an instance
              single_key          : SecurityEnforcementProperties::key_instance;  --
references an instance
              ) applies to (data, port, abstract, system, bus, memory, device, processor,
                                   virtual processor, virtual bus, connection, process,
thread, flow);
       --
       --------- encryption modes-----------
       -- Electronic Code Book (ECB)
       -- Cipher block chaining (CBC)
```

```
        -- Ciphertext feedback (CFB)
        -- CTR mode (CM) or integer counter mode (ICM) or segmented integer counter (SIC)
mode
        -- Galois/Counter Mode (GCM) is a mode of operation for symmetric-key cryptographic
block ciphers
        -- Cipher block chaining (CBC)
        -- The Advanced Encryption Standard (AES) with five modes Electronic Code Book (ECB),
        -- Cipher Block Chaining (CBC), Cipher FeedBack (CFB), Output FeedBack (OFB), and
Counter (CTR)
        --
        -------  Authenticated Encryption --------------
        --
        -- The authenticated encryption field is used to declare that a data instance has the
authenticated encryption
        -- as specified or that authenticated encryption is required by a connection or that
authenticated encryption
        -- is provided by a component (e.g. bus or virtual bus supporting a connection that
requires authenticated encryption)
        -- or when applied to a component that can contain data (e.g. system, device,
abstract) it declares that all data
        -- contained in the component has authenticated encryption as specified.
        --
        -- The data authentication property must be specified with the
        -- authenticated encryption field declaration to define
        -- the specifics of the authentication and encryption.
        --
-----------------------------------------------------------------
-- Properties and types for  Encryption and Authentication Keys
-----------------------------------------------------------------

        Key_Length: Size applies to (abstract, data);
        --
        Crypto_Period: Time applies to (abstract, data);
        -- The Crypto Period is the time span that a cryptographic key is authorized for use
        Text_Type: enumeration (plainText, cipherText) applies to (abstract, data);
        -- type declarations for key classifiers and instances
        Key_Classifier: type classifier (abstract, data);
        Key_Instance: type reference (data);


------------------------------
-- Authentication Properties
------------------------------

        --     Data Authentication -------
        --
        -- The data authentication property is used to specify, any one or all of, integrity
(data has not been modified) and, as appropriate
        -- authenticity (source authentication) and non-repudiation.
        --
        -- The data authentication property is used to declare that a data instance has
authentication as specified or that authentication is required by a connection
        -- or that authentication is provided by a component (e.g. bus or virtual bus
supporting a connection that requires data authentication) or when applied
        -- to a component that can contain data (e.g. system, device, abstract) it declares
that all data contained in the component has authentication as specified.
        --
        Data_Authentication:  record
        (
                description                          : aadlstring;
```

```
            authentication_form   : enumeration (no_authentication, MAC, MIC, signature,
signcryption, double_RSA, to_be_specified);
            authentication_algorithm: enumeration (no_authentication, RSA, ElGamal, DSA,
CBC_MAC, GCM, HMAC, UMAC, to_be_specified);
            padding                              : enumeration (no_padding, OAEP,
to_be_specified);
            --
            -- key_Length is declared in the authentication key type classifier or
            -- in the classifier for the authentication key instance or for the key
instance.
            --
            hash_Length                          : Size; -- optional, if the message is
hashed before authentication. Does not apply to authenticated encryption.
            --hash_algorithm                     : aadlstring;
            hash_algorithm                       : enumeration (no_hash, MD5, SHA1,
SHA256, SHA512, SHA3, to_be_specified);
            authentication_key_type      : list of
SecurityEnforcementProperties::key_classifier; -- references a classifier
            authentication_key           : SecurityEnforcementProperties::key_Instance;
-- references an instance
       )
       applies to (data, port, abstract, system, bus, memory, device, processor,
                           virtual processor, virtual bus, connection, process, thread,
flow);

       --     Subject Authentication -------
       --
       --The Subject Authentication property declares that a subject (component instance)
can participate or participates in authentication as specified,
       -- including authentication negotiations employing the specified authentication
protocol, or that the component (e.g. a bus or virtual bus) supports the
       -- authentication specified.
       --

    Subject_Authentication: record
    (
       description    : aadlstring;
       authentication_access_type: enumeration (no_authentication, single_password,
smart_card,  ip_addr, two_factor, multi_layered, bio_metric, to_be_specified);
       two_and_multi_layered_factors: list of enumeration (no_multifactor, smart_card,
token, PIV, OTP, biometric, multi_layered, to_be_specified);
       -- the listing is such that the initial factor required for authentication is listed
first, the second factor is listed second, etc.
       authentication_protocol: enumeration (no_authentication, cert_services, EAP, PAP,
SPAP, CHAP, MS_CHAP, Radius, IAS, Kerberos, SSL, TLS, NTLM, to_be_specified) ;
       authentication_role: enumeration (no_authentication, authenticator, accessor,
provider, requirer, mutual);
    )
    applies to (abstract, system, process, thread,device, processor, virtual processor,
connection, bus, virtual bus, flow);

       --  two_factor is a subset of multi_layered but is included since it is a prevalent
multi-layered type.
       --
       --       Acronyms and values ----
       --       The Password Authentication Protocol (PAP)
       --    The Shiva PAP (SPAP)
       --    Challenge Handshake Authentication Protocol (CHAP)
       --    Microsoft CHAP (MS-CHAP)
       --    The Extensible Authentication Protocol (EAP)
       --       Remote Authentication Dial-In User Service (RADIUS)
```

```
        --        Internet Authentication Service (IAS)
        --        Secure Socket Layer (SSL)
        --        Transport Layer Security (TLS)
        --        NT (New Technology) LAN Manager (NTLM) is a suite of Microsoft security
protocols
        --        intended to provide authentication, integrity, and confidentiality to
users.
        --    Kerberos
        --    Certificate services (cert_services)

-----------------------------------------
        --  secure end-to-end flows
-----------------------------------------

        Secure_Flow: aadlboolean applies to (flow);
        --
        -- The Secure_Flow property specifies that the data in an end-to-end flow
        -- is not altered by any element along the flow.

-----------------------------------------
        -- certificate and encryption key management properties
-----------------------------------------

        Key_Distribition_Method: enumeration (public_broadcast_channel,
public_one_to_one_channel, encrypted_channel, QKD, direct_physical_exchange, courier)
applies to (all);
        --authenticated_channel? encrypted_and_authenticated_channel?

-------------------------------------------
-- Properties related to the platform and its support for security enforcement
-------------------------------------------
        -- name of the operating system used
        -- associated with the application component, the virtual processor, or
processor/(hardware) system
        OS: aadlstring applies to (all);

        -- name of the programming language used; associated with an application component
        Language: aadlstring applies to (all);

        -- external exposure of component, physical or some other form
        Exposed : aadlboolean applies to (bus, virtual bus, processor, device, system,
memory);


end SecurityEnforcementProperties;
```

Table 36: Custom Security Package

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is
Unlimited.

-- This package contains specific component and other AADL declarations as examples
for Security modeling and analysis.
```

```
package SecurityAnnexCustomPkg

public

with SecurityEnforcementProperties;
-------------------------------------------
-------------------------------------------
-- A security 'key' is defined as an abstract component.

abstract key
end key;

-- extend abstract key to data classifiers

       data symmetricKey extends key
       end symmetricKey;

       data publicKey extends key
       end publicKey;

       data privateKey extends key
       end privateKey;

       data AESKey256 extends symmetricKey
              properties
                     SecurityEnforcementProperties::key_Length => 256 bits;
       end AESKey256;

-----------------------------------------
       -- certificates
       abstract CertificateAbs
       end CertificateAbs;

       data Certificate extends CertificateAbs
       end Certificate;

       data subject
       end subject;

       data implementation subject.certificate
       end subject.certificate;

       data issuer
       end issuer;

       data implementation issuer.certificate
       end issuer.certificate;

       data periodOfValidity
       end periodOfValidity;

       data implementation periodOfValidity.certificate
       end periodOfValidity.certificate;
```

```
        data AdminInformation
        end AdminInformation;

        data implementation AdminInformation.certificate
        end AdminInformation.certificate;

        data extendedInformation
        end extendedInformation;

        data implementation extendedInformation.certificate
        end extendedInformation.certificate;

        data implementation Certificate.TLS_SSL
              subcomponents
                    Subject: data subject.certificate;
                    Issuer: data issuer.certificate;
                    PeriodOfValidity: data periodOfValidity.certificate;
                    AdminInformation: data adminInformation.certificate;
                    ExtendedInformatio: data extendedInformation.certificate;
        end Certificate.TLS_SSL;


end SecurityAnnexCustomPkg;
```

4 Normative References

[DO-356]
Standard*: RTCA DO-356, Airworthiness Security Methods and Consideration*, 2014-09-23, http://standards.globalspec.com/std/9870299/rtca-do-356.

[DO-178B/C]
DO-178B/C Software Considerations in Airborne Systems and Equipment Certification, December 1992. Revised, December 2011.

[DO-254]
DO-254 Design Assurance Guidance for Airborne Electronic Hardware, April 2000.

5 Informative References

[Anderson 2008]
Anderson, R. *Security Engineering. 2nd Edition*, Wiley, 2008.

[Arce 2014]
Arce, I. et al. *AVOIDING THE TOP 10 SOFTWARE SECURITY DESIGN FL AWS*, IEEE Center for Secure Design, IEEE Computer Society, https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf.

[CNSS]
Committee on National Security Systems (CNSS) Glossary, CNSSI No. 4009, April 6, 2015

[CWE]
Common Weakness Enumeration, http://cwe.mitre.org/index.html.

[Feiler 2016]
Feiler, P. H.; Delange, J.; Gluch, D. P.; & McGregor, J.D. *Architecture-Led Safety Process*. CMU/SEI-2016-TR-012. Software Engineering institute, Carnegie Mellon University. 2016. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484826.

[Gehani 2012]
Gehani, A & Ciocarlie, G. F. Composing Cross-Domain Solutions, 2nd Layered Assurance Workshop (LAW), affiliated with the 28th Annual Computer Security Applications Conference (ACSAC), 2012, www.csl.sri.com/users/gehani/papers/LAW-2012.Streaming.pdf.

[Hansson 2008]
Hansson, J. & Feiler, P. H. Building Secure Systems using Model-Based Engineering and Architectural Models, Software Engineering Institute White Paper, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=29187.

[LAW]
The Law Dictionary, http://thelawdictionary.org/article/what-is-public-trust-security-clearance/

[Leveson 2012]
Leveson, Nancy, G. *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press. 2012.

[Mirakhorli 2016]
Mirakhorli, M. Common Architecture Weakness Enumeration (CAWE), IEEE Software Blog, April 25, 2016, http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html.

[NIST 2016]
Ross, R; McEvilley, M; and Oren, J. C. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*, NIST Special Publication 800-160, November 2016.

[NIST 2012]
NIST SP 800-30 Rev 1, *Guide for Conducting Risk Assessments*, National Institute of Standards and Technology, September 2012.

[OCL 2006]
Object Management Group (OMG); Object Constraint Language OMG Available Specification Version 2.0, May 2006.

[Resolute 2014]
Andrew Gacek, John Backes, Darren Cofer, Konrad Slind, & Mike Whalen, "Resolute: an assurance case language for architecture models," *Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology*, Portland, Oregon, USA, October 18 - 21, 2014, pages 19-28.

[BKCASE 2016]
BKCASE Editorial Board. 2016. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 1.7. R.D. Adcock (EIC). Hoboken, NJ:

[Rushby 2008]
Rushby, J. *Separation and Integration in MILS (The MILS Constitution)*, Technical Report SRI-CSL-08-XX, February 2008, Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA. https://pdfs.semanticscholar.org/0398/5ca22524e10f6fab9dd966c61c4ab3de7f74.pdf

[Smith 2015]
Smith, Scott D. *Shedding Light on Cross Domain Solutions*, SANS Institute InfoSec Reading Room, November 6, 2015, https://www.sans.org/reading-room/.../shedding-light-cross-domain-solutions-36492.

[Shawn Fitzgerald]
Fitzgerald, Shawn, "An Introduction to Authenticated Encryption," March, 2013. https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2013/april/an-introduction-to-authenticated-encryption/ .

[Wiki-Trusted]
https://en.wikipedia.org/wiki/Trusted_system

[DND] Department of Defence Canada