

# Arm® Platform Security Architecture Compliance Suite for Firmware Framework

Revision: r0p0

## Validation Methodology



# Arm® Platform Security Architecture Compliance Suite for Firmware Framework

## Validation Methodology

Copyright © 2018 Arm Limited or its affiliates. All rights reserved.

## Release Information

## Document History

Issue	Date	Confidentiality	Change
A	28 September 2018	Non-Confidential	Alpha release
B	30 October 2018	Non-Confidential	Minor edits

## Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

**Confidentiality Status**

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

**Product Status**

The information in this document is for an Alpha product, that is a product under development.

**Web Address**

<http://www.arm.com>

# Contents

## Arm® Platform Security Architecture Compliance Suite for Firmware Framework Validation Methodology

### **Preface**

About this book .....	7
Feedback .....	9

### **Chapter 1**

#### **Introduction**

1.1	Abbreviations .....	1-11
1.2	Platform Security Architecture Firmware Framework .....	1-12
1.3	Architecture Compliance Suite .....	1-13
1.4	Scope of the document .....	1-14
1.5	Components of Architecture Compliance Suite .....	1-15
1.6	Directory structure .....	1-16
1.7	Compliance sign-off process .....	1-17
1.8	Feedback, contributions, and support .....	1-18

### **Chapter 2**

#### **Validation methodology**

2.1	Compliance test layering details .....	2-20
2.2	Test suite organization .....	2-22
2.3	Test execution flow .....	2-24
2.4	Loading compliance test binaries .....	2-26
2.5	Analyzing test run results .....	2-28

A.1    *Revisions* ..... Appx-A-31

# Preface

This preface introduces the *Arm® Platform Security Architecture Compliance Suite for Firmware Framework Validation Methodology*.

It contains the following:

- [About this book on page 7.](#)
- [Feedback on page 9.](#)

## About this book

This book describes the compliance suite for Platform Security Architecture Firmware Framework.

### Product revision status

The *rmpr* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

*rm* Identifies the major revision of the product, for example, r1.

*pr* Identifies the minor revision or modification status of the product, for example, p2.

### Intended audience

This book is written for engineers who are specifying, designing, or verifying an implementation of the Arm® Platform Security Architecture Firmware Framework architecture.

### Using this book

This book is organized into the following chapters:

#### **Chapter 1 Introduction**

This chapter introduces the features and components of the compliance suite for Arm Platform Security Architecture Firmware Framework architecture.

#### **Chapter 2 Validation methodology**

This chapter describes the validation methodology that is used for Architecture Compliance Suite.

#### **Appendix A Revisions**

This appendix describes the technical changes between released issues of this book.

### Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the [Arm® Glossary](#) for more information.

### Typographic conventions

#### *italic*

Introduces special terminology, denotes cross-references, and citations.

#### **bold**

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

#### `monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

#### monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

#### `monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

#### `monospace bold`

Denotes language keywords when used outside example code.

## <and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments.  
For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

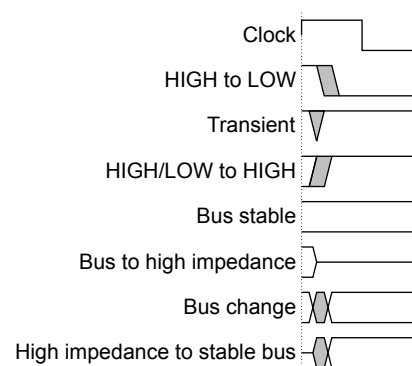
## SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

## Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



**Figure 1 Key to timing diagram conventions**

## Signals

The signal conventions are:

### Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW.  
Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

### Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

## Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

### Arm publications

- *Arm® Platform Security Architecture Firmware Framework specification* (ARM DEN 0063).
- *PSA Security model* (ARM DEN 0079).
- *Arm® Trusted Base System Architecture for Armv8-M* (ARM DEN 0062A).
- *PSA Trusted Boot and Firmware Update* (ARM DEN 0072A).
- *PSA Crypto API*
- *Armv8 Architecture Reference Manual, Armv8 for M-profile* (Arm DDI 00553A)

### Other publications

None.



## Feedback

### Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

- The title *Arm Platform Security Architecture Compliance Suite for Firmware Framework Validation Methodology*.
- The number 101447\_0000\_B\_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

————— **Note** —————

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

---

# Chapter 1

## Introduction

This chapter introduces the features and components of the compliance suite for Arm Platform Security Architecture Firmware Framework architecture.

It contains the following sections:

- *1.1 Abbreviations* on page 1-11.
- *1.2 Platform Security Architecture Firmware Framework* on page 1-12.
- *1.3 Architecture Compliance Suite* on page 1-13.
- *1.4 Scope of the document* on page 1-14.
- *1.5 Components of Architecture Compliance Suite* on page 1-15.
- *1.6 Directory structure* on page 1-16.
- *1.7 Compliance sign-off process* on page 1-17.
- *1.8 Feedback, contributions, and support* on page 1-18.

## 1.1 Abbreviations

This section lists the abbreviations that are used in this document.

**Table 1-1 Abbreviations and their expansions**

Abbreviation	Expansion
FF	Firmware Framework
NSPE	Non-Secure Processing Element
PAL	Platform Abstraction Layer
PE	Processing Element
PSA	Platform Security Architecture
RoT	Root of Trust
SID	Secure function IDentifier
SPE	Secure Processing Element
SPM	Secure Partition Manager
SUT	System Under Test
VAL	Validation Abstraction Layer

## 1.2 Platform Security Architecture Firmware Framework

Arm *Platform Security Architecture* (PSA) is a holistic set of threat models, security analysis, hardware and firmware architecture specifications, and an open source reference implementation.

PSA provides a recipe based on industry best practice, that allows security to be consistently designed in, at both hardware and firmware level.

The PSA *Firmware Framework* (FF) defines a standard programming environment and firmware interfaces for implementing and accessing security services within a device's *Root of Trust* (RoT).

The PSA Security Model divides execution within the system into two domains:

- *Non-Secure Processing Environment* (NSPE)
- *Secure Processing Environment* (SPE)

NSPE contains the application firmware, and the OS kernel and libraries. It typically controls most I/O peripherals. SPE contains the security firmware and hardware resources that must be isolated from NSPE firmware and hardware resources. The security model requires that no NSPE firmware or hardware can inspect or modify any SPE hardware, code, or data.

Security functionality is exposed by PSA as a collection of RoT services. Each RoT service is a set of related security functionality. For example, there might be an RoT service for cryptography operations, and another for secure storage.

PSA subdivides the SPE into two subdomains:

- PSA RoT
- Application RoT

PSA RoT provides the fundamental RoT Services to the system and also manages the isolated execution environment for the Application RoT Services.

The main components of PSA RoT are:

- PSA Security Lifecycle: Identifies the production phase of the device and controls the availability of device secrets and sensitive capabilities such as Secure debug.
- PSA Immutable RoT: Hardware and non-modifiable firmware and data installed during manufacturing.
- Trusted Boot and Firmware Update: Ensures the integrity and authenticity of the device firmware.
- Secure Partition Manager: Manages isolation of the RoT services, the IPC mechanism that allows software in one domain to make requests of another, and scheduling logic to ensure that requests are eventually serviced.
- PSA RoT services: Provide essential cryptographic functionality and manage access to the immutable RoTs for Application RoT services.

The Firmware Framework specification:

- provides requirements for the SPM.
- defines a standard runtime environment for developing protected RoT Services, including the programming interfaces provided by the SPM for implementing and using RoT Services.
- defines the standard interfaces for the PSA RoT Services.

For details on SPM and PSA Root of Trust, see the following specification documents:

- *Arm® Platform Security Architecture Firmware Framework specification* (ARM DEN 0063).
- *PSA Security model* (ARM DEN 0079).
- *Arm® Trusted Base System Architecture for Armv8-M* (ARM DEN 0062A).
- *PSA Trusted Boot and Firmware Update* (ARM DEN 0072A).

## 1.3 Architecture Compliance Suite

Architecture compliance tests are a set of examples of the invariant behaviours that are specified by the PSA FF Specification. Use these tests to check that these behaviours are interpreted correctly in your system.

The current version of specification contains the compliance suite for the following categories of features, each covering a different area of architecture. The specification and compliance suite will be updated to further cover architectural areas such as Attestation APIs and Secure storage APIs in future releases.

**Table 1-2 Test categories and their descriptions**

Test category	Sub category	Description
IPC	Level of isolation	Tests verifying the expected behavior of SPM involve in different level of isolation defined by the specification
	Client APIs	Tests verifying the correctness of each of client APIs.
	Secure partition APIs	Tests verifying the correctness of each of Secure Partition APIs.
	Manifest input	Tests verifying manifest input parameters
Crypto	PSA crypto APIs	Tests verifying the correctness of PSA Crypto APIs.

The compliance suite contains self-checking tests that are written in C. These tests have checks that are embedded within the test code. To view the list of compliance suites and how these different categories of features are checked for compliance, see test-list documents in the doc/ directory.

## 1.4 Scope of the document

The goal of this document is to describe the validation methodology for the Architecture Compliance Suite of the PSA FF specification. It focuses on describing the framework and the methodology within which the tests are run.

## 1.5 Components of Architecture Compliance Suite

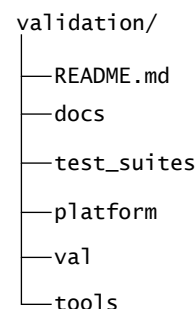
The components of the Architecture Compliance Suite are described in the following table:

**Table 1-3 Components of Architecture Compliance Suite**

Components	Description
Test suites	Contain self-checking compliance tests that are written in C.
Substructure	Test supporting layers consist of a framework and libraries setup as: <ul style="list-style-type: none"><li>• Tools to build the compliance tests</li><li>• VAL library</li><li>• PAL library</li></ul>
Documentation	Suite-specific documents such as testlists, porting guide, and API specification.

## 1.6 Directory structure

Components of the tests must be in a specific hierarchy for the compliance suite. When the release package is downloaded from GitHub, the top-level directory contains the files that are shown in the following figure.



**Figure 1-1 Test suite directory structure**

### **README.md**

README file for PSA FF compliance suite.

### **docs**

This directory contains the test suite documentation.

### **test\_suites**

This directory contains subsuites containing the Architecture Compliance Suite. This test suite is a set of C-based directed tests, each of which verifies the implementation against a test scenario that is described by the PSA FF specification. These tests are abstracted from the underlying hardware platform by the VAL.

### **platform**

This directory contains files to form PAL. PAL is the closest to hardware and is aware of the underlying hardware details. Since this layer interacts with hardware, it must be ported or tailored to specific hardware required for system components present in a platform. This layer is also responsible for presenting a consistent interface to the VAL required for the tests.

### **val**

This directory contains subdirectories for the VAL libraries. This layer provides a uniform and consistent view of the available test infrastructure to the tests in the test suite. The VAL makes appropriate calls to the PAL to achieve this functionality. This layer is not required to be ported when the underlying hardware changes.

### **tools**

This directory contains make files and scripts that are used to generate test binaries.



## 1.7 Compliance sign-off process

The future releases of this specification will be updated with compliance sign-off process and expectations from partner on the process for running PSA FF compliance suite.

## 1.8 Feedback, contributions, and support

For feedback, use the GitHub Issue Tracker that is associated with this repository.

For support, send an email to [support-psa-arch-tests@arm.com](mailto:support-psa-arch-tests@arm.com) with the details.

Arm licensees can contact Arm directly through their partner managers.

Arm welcomes code contributions through GitHub pull requests. See GitHub documentation on how to raise pull requests.

# Chapter 2

## Validation methodology

This chapter describes the validation methodology that is used for Architecture Compliance Suite.

It contains the following sections:

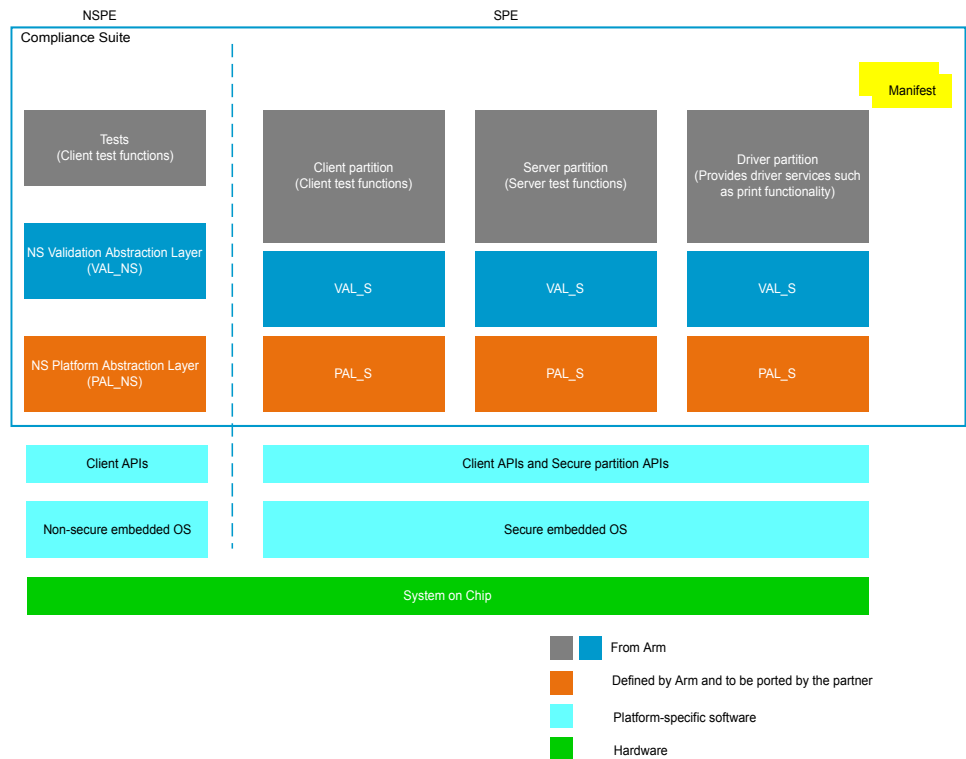
- *2.1 Compliance test layering details* on page 2-20.
- *2.2 Test suite organization* on page 2-22.
- *2.3 Test execution flow* on page 2-24.
- *2.4 Loading compliance test binaries* on page 2-26.
- *2.5 Analyzing test run results* on page 2-28.

## 2.1 Compliance test layering details

PSA FF compliance tests are self-checking and portable C-based tests with directed stimulus.

Compliance tests use the layered software stack approach to enable porting across different test platforms. The constituents of the layered stack are:

- Tests
- Secure partitions
- VAL
- PAL



**Figure 2-1 Layered software stack**

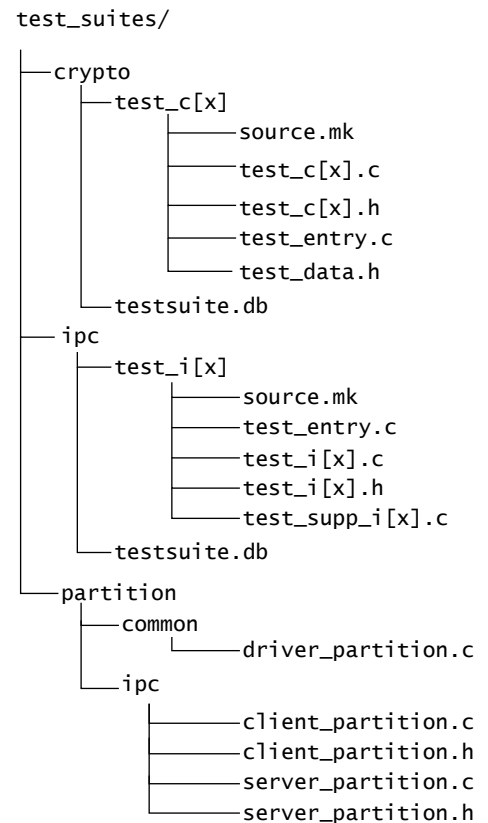
The following table describes the different layers of a compliance test.

**Table 2-1 Layers of compliance test**

Layer	Description
Tests	<p>A set of C-based directed tests, each of which verifies the implementation against a test scenario described by the PSA FF specification.</p> <p>These tests are expected to be run in Non-secure and may further use IPC calls to secure functions in the partition to cover the appropriate test scenario.</p> <p>These tests are abstracted from the underlying hardware platform by the VAL. This implies that it is not required to port a test for a specific target platform.</p> <p>The current release of tests contains:</p> <ul style="list-style-type: none"> <li>• IPC tests: <code>test_suites/ipc</code> directory</li> <li>• Crypto tests: <code>test_suites/crypto</code> directory</li> </ul>
Secure partitions	<p>The compliance suite presents manifest files for the partitions that are required to the run tests. These partitions further provide secure functions with APIs that are specific to scenario validation.</p> <p>You must parse these manifests, compile and integrate these partitions to your Secure software stack containing the SPM using your build tool.</p> <p>The secure partition related manifest files are available at:</p> <ul style="list-style-type: none"> <li>• <code>platform/targets/&lt;targetName&gt;/manifests/common/</code> directory.</li> </ul> <p>Partitions formed using this manifest provide driver related services such as UART print which are exercised by all tests.</p> <ul style="list-style-type: none"> <li>• <code>platform/targets/&lt;targetName&gt;/manifests/ipc/</code> directory.</li> </ul> <p>This directory contains multiple manifest files. Partitions formed using these manifests are used by IPC tests only. Therefore, generating binaries for tests other than IPC is not required to use these partitions code base.</p>
VAL	<p>This layer provides a uniform and consistent view of the available test infrastructure to the tests in the test pool. The VAL makes appropriate calls to the PAL to achieve this. The VAL is designed such that it can be used both from Secure and Non-secure sides.</p> <p>This layer is not required to be ported when the underlying hardware changes.</p>
PAL	<p>This layer is the closest to the hardware and is aware of the platform details. It is responsible for presenting the hardware through a consistent interface to VAL. This layer must be ported to the specific hardware present in the platform. The PAL is designed such that it can be used from both Secure and Non-secure sides.</p>

## 2.2 Test suite organization

Test suite directory has subdirectories containing tests and partitions. The directory structure of the test suite is shown in the following figure.



**Figure 2-2 Test suite directory structure**

The following table shows the contents of the directories, subdirectories, and files shown in the directory structure.

**Table 2-2 Directory content**

Directory	Content
crypto	Holds crypto tests.
ipc	Holds ipc tests.
testsuite.db	A database file representing tests to be compiled and run as part of specific suite. This provides flexibility to run specific tests individually by commenting the other tests out.
partitions	Contains partition files that provide different driver services to the tests and the dispatcher logic to dispatch specific client or server test functions
test_i[x] or test_c[x]	Test directory containing test related files where [x] is the test number..
test_entry.c	Holds the test entry point in NSPE and executes client tests from NSPE and SPE sequentially based on the test requirement.
test_data.h	Contains the test stimuli for the crypto test case. The file is absent in test_c001 test.
test_x001.c	Holds client test functions. x is i for IPC tests and c for Crypto tests.

**Table 2-2 Directory content (continued)**

Directory	Content
test_supp_i[x].c	Holds server test functions. This is available only for IPC tests.
source.mk	Helps to identify the test files that must be compiled to generate the test binaries.

## 2.3 Test execution flow

You must integrate test suite tool generated partition libraries (SPE archives) into your secure software stack containing SPM and test suite NSPE libraries with PSA defined client APIs and RoT APIs interfaces. These interfaces can also be provided as part of NSPE OS.

Then the *System Under Test* (SUT) boots to an environment that enables the test functionality. This implies that the SPM is initialized, and PSA FF partitions are ready to accept requests.

On the Non-secure side, the SUT boot software gives control to the compliance tests entry point (`val_entry` symbol) as an application entry point in Non-secure privileged mode.

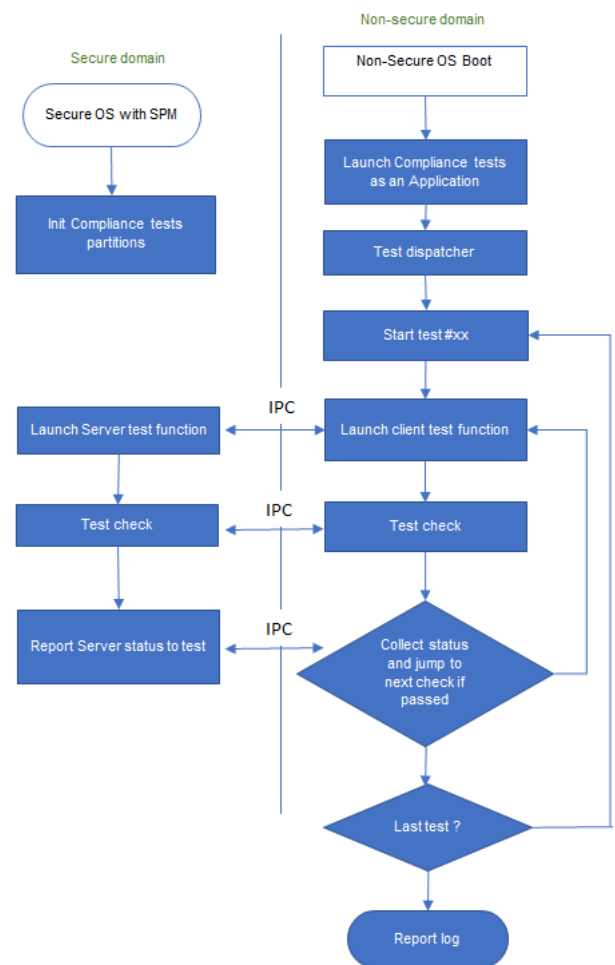
The PSA compliance tests query the VAL layer to get the necessary information to run the tests. This information includes memory maps, interrupt maps, and hardware controller maps.

Based on the test scenario, the test and partition communicate with each other using IPC APIs that are defined in the specification, and report the test results using VAL print API (in turn PAL API ported to the specific platform). Each IPC test scenario is driven using dedicated client-server tests functions. The client functions are available in `test_i00x.c` and are suffixed with `client_test_` label. Based on test needs, client functions are executed either in NSPE or SPE or both. Server functions are available in `test_supp_ix.c` and are suffixed with `server_test` label. They are always executed in SPE. Crypto tests contain only the client tests and do not have server test functions. These client functions in Crypto tests interact with PSA Crypto library that is loaded into SPE to get crypto services using PSA defined Crypto APIs. These client functions are suffixed with `crypto_test_` in the tests.

Due to RAM and flash size constraints, all the tests may not be available at the same time. The dispatcher in the VAL queries the PAL to load the next test on the completion of the present test. The PAL may optionally communicate with the external world to load the next test. The dispatcher also makes VAL (and in turn PAL) calls to save and reports each of the test results.

Information about the environment in which a host test harness is running, is beyond the scope of this document. However, it is presumed that the SUT is communicating with the host using Serial port, JTAG, Wi-Fi, USB or any other means that allow for access to the external world.





**Figure 2-3 Test execution flow**

## 2.4 Loading compliance test binaries

An SUT can have RAM and flash size constraints. It is possible that all the tests do not fit into these types of memories and may not be available at the same time. If an SUT does not have this limitation, compliance test binaries can be stored in main memory directly. Otherwise, they can be stored in the secondary memory.

The following binaries are required to be loaded into memory:

- SPE binary

You must integrate test suite partition code with SPE and load it as SPE binary. Note that client test functions that are required to run in SPE and server test functions of all tests are compiled as part of `client_partition` and `server_partition` respectively. All of these functions are loaded into Secure SRAM and are available at same time. If an SUT has Secure SRAM size constraints, you can compile and run these test functions in a bulk of test sets. For example, 10 tests at time. To do this, remove test references other than the required test bulk test set from the respective suite specific `testsuite.db` file. Repeat this process for all the test sets.

Test suite partition libraries:

— `<build_dir>/BUILD/partition/driver_partition.a`

Additional SPE libraries explicitly for IPC tests:

— `<build_dir>/BUILD/partition/client_partition.a`

— `<build_dir>/BUILD/partition/server_partition.a`

- NSPE binary

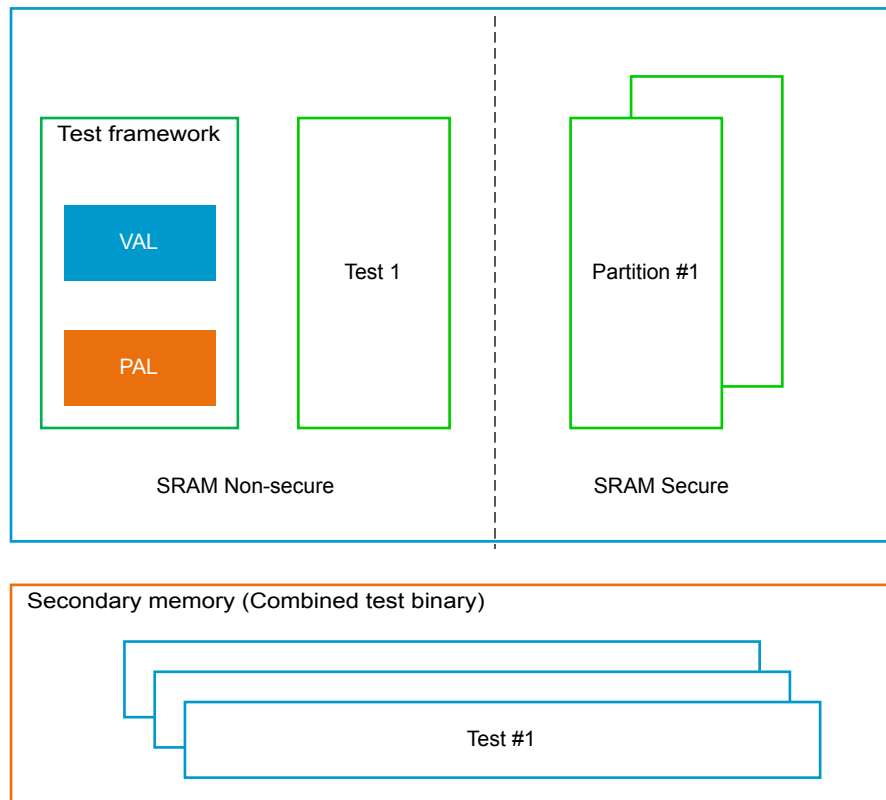
Compliance test compilation flow creates two archive files that contain code for the test framework - VAL and PAL API, and test dispatcher logic that must be available in the main memory and executed as an application in NSPE. Link these archives with OS library to be able to generate an NSPE binary.

— `<BUILD_PATH>/BUILD/val/val_nspe.a`

— `<BUILD_PATH>/BUILD/platform/pal_nspe.a`

- Combined test binary

Compliance test compilation flow also creates a combined test binary containing all Non-secure test binaries together at `<BUILD_PATH>/BUILD/<suite>/test_elf_combine.bin`. You can load this binary into Non-secure SRAM if the SUT has enough space. Otherwise, this can be loaded into secondary storage. The dispatcher function within the VAL reads this binary and loads each of test sections using PAL API to Non-secure memory one after another. The addresses of the various sections must be provided using `target.cfg`.



**Figure 2-4 Loading test binaries**

The dispatcher has the following responsibilities:

Each test must present the `test_entry` function address to the dispatcher. To this function, the dispatcher passes a pointer to a structure containing the function pointers to all the available VAL functions. These functions make the appropriate VAL function call.

The flow of the dispatcher is as follows:

1. Request VAL to load the metadata of the next test into the main memory.
2. Parse the metadata that is associated with the test.
3. Verify that the test is compatible with the SUT.
4. Load the test code and data sections into the appropriate locations in the main memory.
5. Call the `test_entry` function of the test and execute tests.
6. Wait for test completion.
7. Print and save the result of the test.
8. Repeat steps 1-6 till the end of the last test.

To facilitate test reporting and management of observing aspects, the PSA FF system contains UART for printing the status of tests. If a display console is not available, the PAL can be updated to make the test results available to the external world through other means.

## 2.5 Analyzing test run results

Each test follows a uniform test structure that is defined by VAL.

1. Performing any test initializations.
2. Dispatching the client-server test functions.
3. Waiting for test completion.
4. Performing the test exit.

The test may pass, fail, skip or be in an error state. For example, if test times out or the system hangs, it means that something went wrong and the test framework was unable to determine what happened. In this case, you may have to check the logs. If a test fails or skips, you may see extra print messages to determine the cause.

The test suite summary is displayed at the end. An example snapshot of the test suite summary is shown in the following figure.

```

***** PSA Compliance Suite - Version 1.0 *****

Running... IPC Suite
*****

TEST: 1 | DESCRIPTION: Testing psa_framework_version and psa_version APIs
[Info] Executing tests form non-secure
[Check1] psa_framework_version
[Check2] psa_version
[Info] Executing tests form secure
[Check1] psa_framework_version
[Check2] psa_version
TEST RESULT: PASSED

*****

TEST: 2 | DESCRIPTION: Testing RoT connect and disconnect cases
[Info] Executing tests form non-secure
[Check1] Accept and close connection
[Check2] Test psa_connect with allowed minor version policy
[Check3] Test psa_call with allowed status code
[Check4] Test client_id
Checkpoint 211 : Error Code=0x11
[Check4] FAILED
TEST RESULT: FAILED (Error Code=0x11)

*****

No more valid tests found. Exiting.

***** REGRESSION SUMMARY *****
TOTAL TESTS      : 2
TOTAL PASSED     : 1
TOTAL SIM ERROR  : 0
TOTAL FAILED     : 1
TOTAL SKIPPED    : 0

*****

Entering standby

```

**Figure 2-5 Test suite summary**

## Debugging of a failing test

Since each test is organized with a logical set of self-checking code, in the event of a failure, searching for the relevant self-checking point is a useful point to start debugging.

Consider the above snippet of a failing test on the display console.

Here are some debugging points to consider.

- Test 2 is failing for Non-secure client tests run. The test is `test_suites/ipc/test_i002` directory.
- Each IPC test has a client test file `test_ix.c` and server test file. The `[Check x]` prints are available in client tests file. Hence it is required to see `test_suites/ipc/test_i002/test_i002.c` file and go to client test function which prints `Test client_id` statement.

————— **Note** —————

For crypto tests, you must see only the `test_cx.c` file since crypto tests will not have server test files.

- Since the failure is shown as Checkpoint 211, you must see the server test function dedicated to `Test client_id` and debug the failing point. Checkpoints are reserved in the compliance suite as shown below:
  - Checkpoints 1-100 are reserved for VAL.
  - Checkpoints 101-200 are reserved for client test functions.
  - Checkpoints 201-300 are reserved for server test functions.
- Status of the failure (11 in this example) is mapped with a structure `val_status_t` that is available at `val/common/val.h`.
- You can recompile and rerun the tests binaries with high verbosity level if the default prints do not give enough information. Refer to `--verbose` switch of `setup.sh` script to see how a test verbosity can be changed.

# Appendix A

## Revisions

This appendix describes the technical changes between released issues of this book.

It contains the following section:

- [A.1 Revisions on page Appx-A-31.](#)

## A.1 Revisions

**Table A-1 Issue 101447-01**

Change	Location	Affects
This is the first revision of the document.	-	All revisions

**Table A-2 Differences between Issue 101447-01 and Issue 101447-02**

Change	Location	Affects
Updated path to secure manifest files.	See <a href="#">2.1 Compliance test layering details on page 2-20</a> .	All revisions
Updated test execution flow and SPE binary information.	<ul style="list-style-type: none"> <li>See <a href="#">2.3 Test execution flow on page 2-24</a>.</li> <li>See <a href="#">2.4 Loading compliance test binaries on page 2-26</a></li> </ul>	All revisions