

Arm® Platform Security Architecture APIs Test Suite

Version 1.0

Validation Methodology



Arm® Platform Security Architecture APIs Test Suite

Validation Methodology

Copyright © 2018–2020 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
C	15 January 2019	Non-Confidential	Beta release. The document number has been changed.
D	04 June 2019	Non-Confidential	Beta quality with minor updates
E	30 September 2019	Non-Confidential	Beta quality with minor updates
F	28 February 2020	Non-Confidential	EAC quality with minor updates

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 © 2018–2020 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 Final, that is for a developed product.

Web Address

www.arm.com

Contents

Arm® Platform Security Architecture APIs Test Suite Validation Methodology

Preface

About this book	6
-----------------------	---

Chapter 1

Introduction

1.1	Scope of the document	1-9
1.2	Abbreviations	1-10
1.3	Platform Security Architecture APIs	1-11
1.4	Test suite	1-13
1.5	Test suite components	1-14
1.6	Directory structure	1-15
1.7	Feedback, contributions, and support	1-16

Chapter 2

Validation methodology

2.1	Test layering details	2-18
2.2	Test suite organization	2-20
2.3	Test execution flow	2-23
2.4	Integrating the test suite with the SUT	2-26
2.5	Test dispatcher	2-28
2.6	Analyzing test run results	2-29

Appendix A

Revisions

A.1	Revisions	Appx-A-32
-----	-----------------	-----------

Preface

This preface introduces the *Arm® Platform Security Architecture APIs Test Suite Validation Methodology*.

It contains the following:

- [About this book on page 6.](#)

About this book

This book describes the test suite for Platform Security Architecture APIs.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction

This chapter introduces the features and components of the test suite for Arm Platform Security Architecture APIs.

Chapter 2 Validation methodology

This chapter describes the validation methodology that is used for the test 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.

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 support-psa-arch-tests@arm.com. Give:

- The title *Arm Platform Security Architecture APIs Test Suite Validation Methodology*.
- The number 101447_0100_F_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.

Other information

- [Arm® Developer](#).
- [Arm® Information Center](#).
- [Arm® Technical Support Knowledge Articles](#).
- [Technical Support](#).
- [Arm® Glossary](#).

Chapter 1

Introduction

This chapter introduces the features and components of the test suite for Arm Platform Security Architecture APIs.

It contains the following sections:

- [*1.1 Scope of the document*](#) on page 1-9.
- [*1.2 Abbreviations*](#) on page 1-10.
- [*1.3 Platform Security Architecture APIs*](#) on page 1-11.
- [*1.4 Test suite*](#) on page 1-13.
- [*1.5 Test suite components*](#) on page 1-14.
- [*1.6 Directory structure*](#) on page 1-15.
- [*1.7 Feedback, contributions, and support*](#) on page 1-16.

1.1 Scope of the document

The goal of this document is to describe the validation methodology for Platform Security Architecture APIs test suites. It focuses on describing the framework and the methodology that is used to run the tests.

1.2 Abbreviations

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

Table 1-1 Abbreviations and expansions

Abbreviation	Expansion
API	Application Programming Interface
FF	Firmware Framework
ITS	Internal Trusted Storage
NSPE	Non-Secure Processing Element
PAL	Platform Abstraction Layer
PE	Processing Element
PS	Protected Storage
PSA	Platform Security Architecture
RoT	Root of Trust
SPE	Secure Processing Element
SPM	Secure Partition Manager
SUT	System Under Test
VAL	Validation Abstraction Layer

1.3 Platform Security Architecture APIs

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

PSA provides a recipe, based on industry best practice, that allows security to be consistently designed in, at both a hardware and firmware level. One of the goals of PSA is to make IoT security easier and quicker. This means having reliable, consistent APIs and useful built-in security functions for device manufacturers and the developer community. These PSA APIs provide a consistent developer experience, hiding the underlying complexity of the security system.

Arm PSA defines the following sets of API specifications:

- PSA Firmware Framework
- PSA Functional APIs

This section contains the following subsections:

- [1.3.1 PSA Firmware Framework on page 1-11.](#)
- [1.3.2 PSA Functional APIs on page 1-12.](#)

1.3.1 PSA Firmware Framework

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

PSA security model divides execution within the system into two domains:

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

NSPE contains application firmware, and OS kernel and libraries. It typically controls most I/O peripherals. SPE contains 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 described in the following table.

Table 1-2 PSA RoT components

Component	Description
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 accesses 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 RoT, refer to the specification documents mentioned in the *Additional reading* section of this document.

1.3.2 PSA Functional APIs

PSA Functional APIs are the top-level APIs used by application developers and RTOS vendors. These APIs have been designed for software developers who want to implement hardware security features without necessarily being security experts themselves.

These APIs provide the top-level essential services related to Crypto, Secure storage, and attestation tokens. For details, see the [Functional APIs specification](#).

1.4 Test suite

Architecture tests are a set of examples of the invariant behaviors that are specified by the PSA APIs specifications. Use these tests to check that these behaviors are interpreted correctly in your system.

These tests cover checks for the following categories of features, each covering a different area of architecture.

Table 1-3 Test categories and their descriptions

API type	Test category	Sub category	Description
PSA Firmware Framework	IPC	Level of isolation	Tests verifying the expected behavior of SPM involved in different levels of isolation, as defined by the specification.
		Client APIs	Tests verifying the correctness of client APIs.
		Secure partition APIs	Tests verifying the correctness of Secure partition APIs.
		Manifest input	Tests verifying manifest input parameters.
		PSA RoT lifecycle API	Tests verifying the correctness of the PSA RoT lifecycle API.
Functional APIs	Crypto	PSA Crypto APIs	Tests verifying the correctness of PSA Crypto APIs.
	<i>Internal Trusted Storage (ITS)</i>	PSA ITS APIs	Tests verifying the correctness of PSA ITS APIs.
	<i>Protected Storage (PS)</i>	PSA PS APIs	Tests verifying the correctness of PSA PS APIs.
	Initial Attestation	PSA Initial Attestation API	Tests verifying the correctness of the PSA Initial Attestation API.

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

1.5 Test suite components

The components of the test suite are described in the following table:

Table 1-4 Test suite components

Component	Description
Test suites	Contain self-checking tests that are written in C.
Substructure	Test supporting layers consist of a framework and libraries setup as: <ul style="list-style-type: none"> Tools to build the compliance tests <i>Validation Abstraction Layer</i> (VAL) library <i>Platform Abstraction Layer</i> (PAL) library
Documentation	Suite-specific documents such as test lists, porting guide, and API specification.

1.6 Directory structure

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

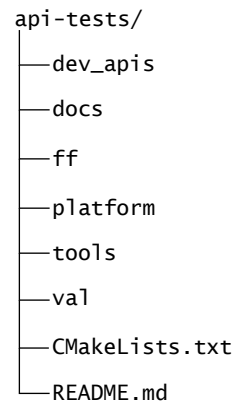


Figure 1-1 Test suite directory structure

dev_apis

has subsuites containing architecture tests for the Functional APIs specification. 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 Functional APIs specification. These tests are abstracted from the underlying hardware platform by the VAL.

docs

contains the test suite documentation.

ff

has subsuites containing architecture tests for PSA-FF specification. 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 specifications. These tests are abstracted from the underlying hardware platform by the VAL.

platform

contains files to form the 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.

tools

contains makefiles and scripts that are used to generate test binaries.

val

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.

CMakeLists.txt

contains information about CMake build support.

README.md

README file for PSA test suite.

1.7 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 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 the test suite.

It contains the following sections:

- [2.1 Test layering details](#) on page 2-18.
- [2.2 Test suite organization](#) on page 2-20.
- [2.3 Test execution flow](#) on page 2-23.
- [2.4 Integrating the test suite with the SUT](#) on page 2-26.
- [2.5 Test dispatcher](#) on page 2-28.
- [2.6 Analyzing test run results](#) on page 2-29.

2.1 Test layering details

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

The layered software stack approach is illustrated in the following figure.

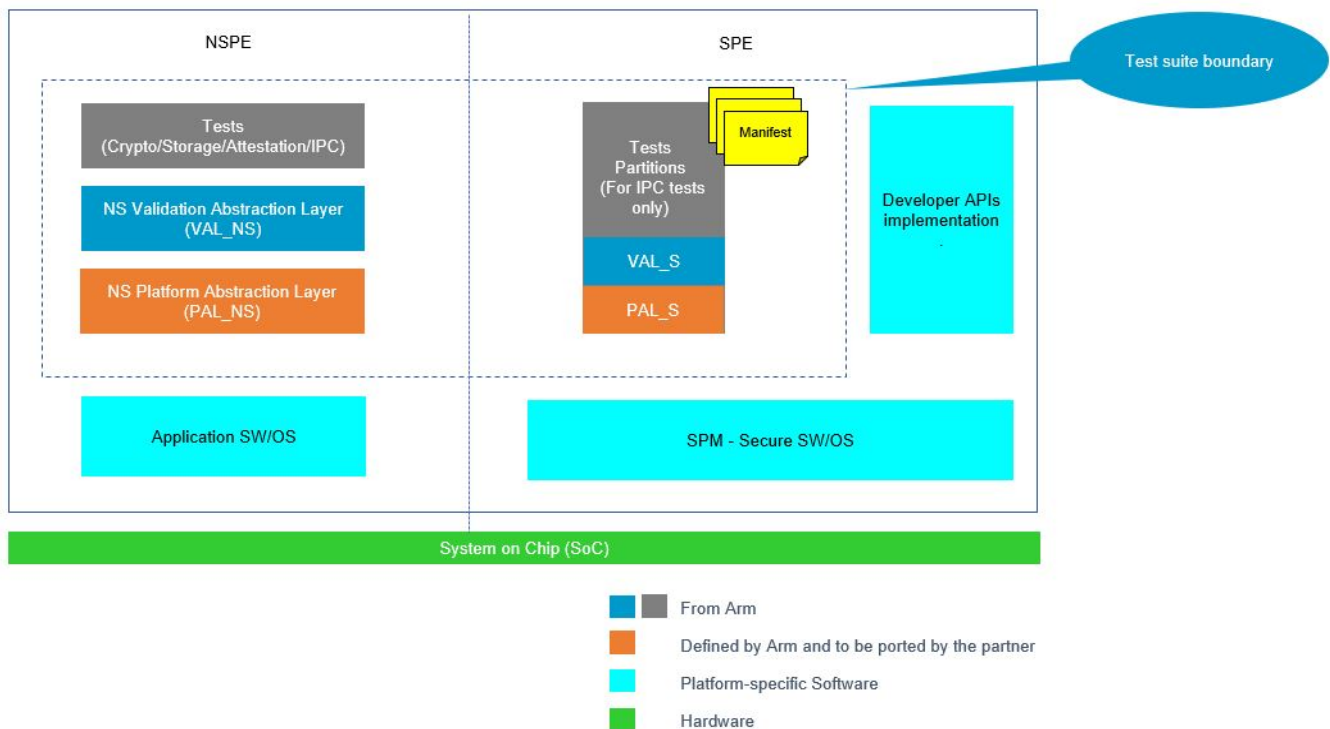


Figure 2-1 Layered software stack

The following table describes the constituents of the layered stack.

Table 2-1 Layered software stack components

Layer	Description
Tests	<p>A set of C-based directed tests, each of which verifies the implementation against a test scenario that is described by the PSA specification.</p> <p>These tests include checks related to PSA-FF and Functional APIs, and are expected to be run in Non-secure. PSA-FF tests may further use IPC calls to communicate test suite-defined Secure partition to cover the appropriate test scenario.</p> <p>These tests are abstracted from the underlying hardware platform by the VAL. This implies that porting a test for a specific target platform is not required.</p>
Secure partitions	<p>PSA-FF test suite defines three Secure partitions:</p> <ul style="list-style-type: none"> • Driver partition provides driver-related services such as print API to the PSA test suite Non-secure code and to the other partitions. • Client partition drives the Secure client test functions for the IPC tests. • Server partition drives the Secure server test functions for the IPC tests. <p>These Secure partitions must be integrated into your Secure software containing SPM. They are valid only for IPC tests. Functional APIs tests are not required to use these partitions.</p> <p>Secure partition-related manifest files are available in the <code>platform/manifests/</code> directory.</p>
VAL	<p>This layer provides a uniform and consistent view of the available test infrastructure to the tests in the test pool, by making appropriate calls to the PAL. It is designed such that it can be used both from Secure and Non-secure sides.</p> <p>This layer does not require porting 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

The directory structures of PSA-FF and Functional APIs test suites are described in this section.

PSA-FF test suite

The following figure shows the contents of the directories, subdirectories, and files in the PSA-FF test suite.

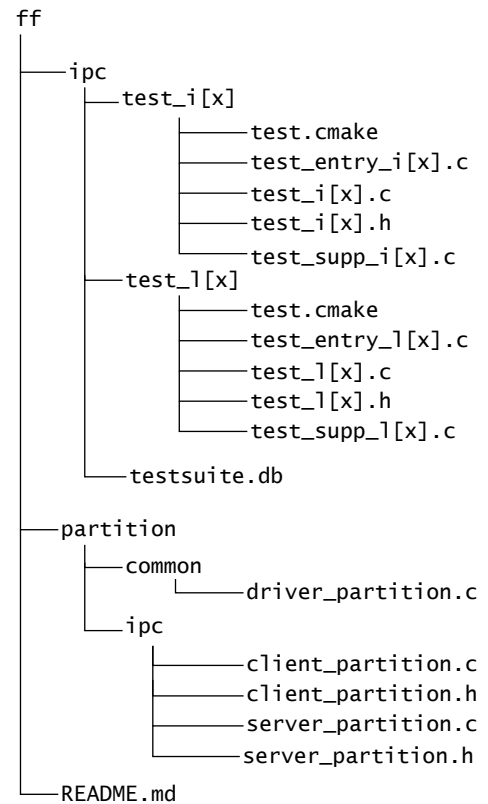


Figure 2-2 PSA-FF test suite directory structure

Table 2-2 Directory content

Directory	Content
ipc	Holds IPC tests.
test_ <i>[y]</i> [<i>x</i>]	Test directory containing IPC test related files. Here, <i>y</i> is: <i>i</i> for IPC tests. <i>l</i> for lifecycle tests.
test.cmake	Helps to identify the test files that must be compiled to generate the test binaries.
test_entry_ <i>i</i> [<i>x</i>].c	Holds the test entry point in NSPE and executes test functions from NSPE. For IPC tests, it can execute the same test functions from SPE, based on the test requirement.
test_ <i>[y]</i> [<i>x</i>].c and test_ <i>[y]</i> [<i>x</i>].h	Hold client test functions.
test_supp_ <i>[y]</i> [<i>x</i>].c	Holds server test functions.

Table 2-2 Directory content (continued)

Directory	Content
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 out the other tests.
partition	Contains partition files that provide different driver services to the tests and the dispatcher logic to dispatch specific client or server test functions.
README.md	This file contains information for building the PSA-FF test suite.

Functional APIs test suite

The following table shows the contents of the directories, subdirectories, and files in the Functional APIs test suite.

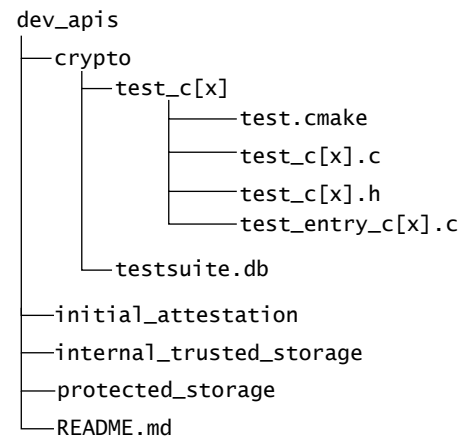


Figure 2-3 Functional APIs test suite directory structure

Table 2-3 Functional APIs directory contents

Directory or file	Content
crypto	Holds Crypto tests.
test_[x][y]	Test directory containing test-related files. [x] can be: <ul style="list-style-type: none"> c for Crypto tests a for Initial Attestation p for Protected Storage s for Internal Trusted Storage [y] is the test number.
test.cmake	Helps to identify the test files that must be compiled to generate the test binaries.
test_[x][y].c and test_[x][y].h	Hold the actual test functions.
test_entry_c[x].c	Holds the test entry point in NSPE and executes test functions from NSPE.
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 out the other tests.
initial_attestation	Holds Initial Attestation tests.
internal_trusted_storage	Holds Internal Trusted Storage tests.

Table 2-3 Functional APIs directory contents (continued)

Directory or file	Content
protected_storage	Holds Protected Storage tests.
README.md	This file contains information for building the Functional APIs test suite.

2.3 Test execution flow

This section provides details of the test execution flows for PSA-FF tests and Functional APIs tests.

PSA-FF tests

The test compilation tool generates the NPSE and SPE archives for IPC tests. For details about IPC test archives, see [2.4 Integrating the test suite with the SUT on page 2-26](#). You must integrate test suite SPE archives with your Secure software stack containing the SPM, such that it gets access to PSA-defined client APIs and Secure partition APIs. The NSPE libraries generated by the test suite must be integrated with the NSPE OS such that test suite NSPE code gets access to the PSA-defined client APIs.

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 tests entry point (`val_entry` symbol) as an application entry point in Non-secure privileged mode.

The PSA tests query the VAL layer to get the necessary information to run the tests. This information can include 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_ix.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.

All the tests are executed sequentially. The dispatcher in the VAL queries the next test on the completion of the present test. The dispatcher also makes VAL (and in turn PAL) calls to save and reports each of the test results. For details about the dispatcher, see [2.5 Test dispatcher on page 2-28](#).

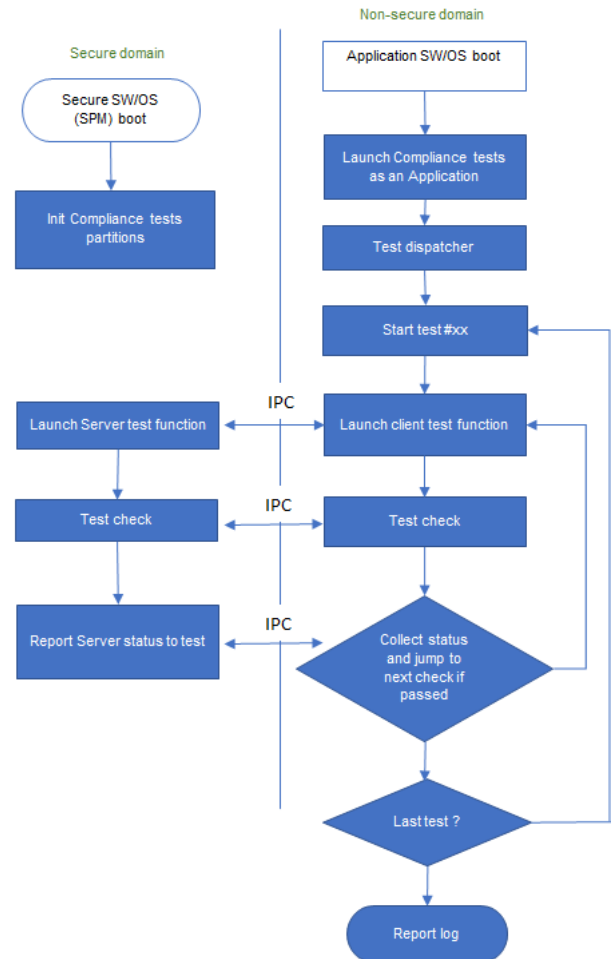


Figure 2-4 Test execution flow for PSA-FF IPC tests

Functional APIs tests

The test compilation tool generates the NPSE archives for Functional tests as described in the [2.4 Integrating the test suite with the SUT on page 2-26](#) section.

You must integrate the test suite NSPE archives with your Non-secure software stack such that it gets access to PSA defined Functional APIs. The SUT then boots to an environment that enables the test functionality. The SUT boot software gives control to the test entry point (`val_entry` symbol) as an application entry point in the Non-secure privileged mode.

The tests query the VAL to get necessary information to run the tests. This information can include memory maps, interrupt maps, and hardware controller maps. Based on the test scenario, the test calls Functional APIs and reports the test results using the VAL print API (in turn PAL API ported to the specific platform).

All the tests are executed sequentially. The dispatcher in the VAL queries the next test on the completion of the present test. For information about the dispatcher, see [2.5 Test dispatcher on page 2-28](#).

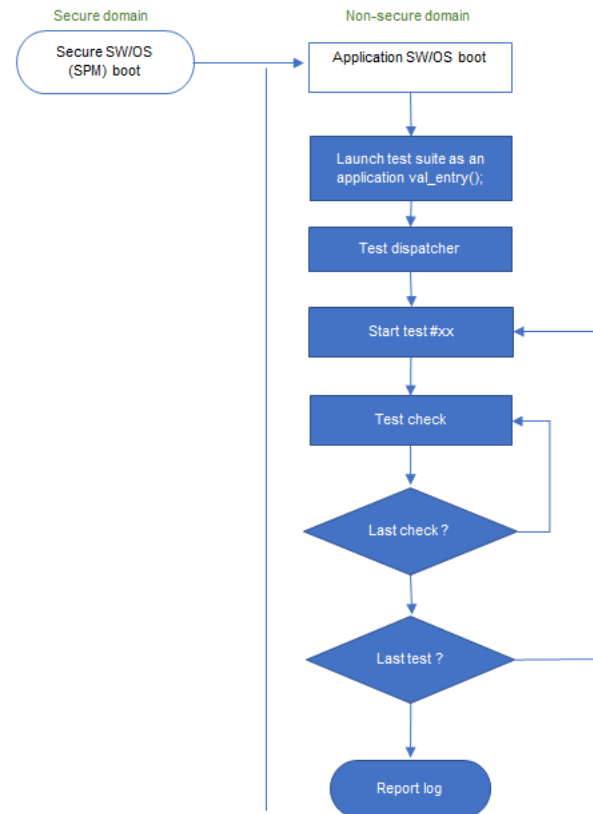


Figure 2-5 Test execution flow for Functional APIs tests

2.4 Integrating the test suite with the SUT

The test compilation flow creates the following libraries that you must integrate with your SUT software.

- **Test framework**

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

— <BUILD_PATH>/BUILD/val/val_nspe.a
— <BUILD_PATH>/BUILD/platform/pal_nspe.a

- **Combined tests archive**

The test compilation flow generates a combined test archive by combining all the Non-secure test objects for Non-secure tests. The generated archive is placed at <BUILD_PATH>/<top_level_suite>/<suite>/test_combine.a. Integrate this archive library with the test framework libraries and NS OS library to generate an NSPE binary. The dispatcher function within the VAL calls each test entry function one after another, to run the Non-secure tests.

- **Test suite Secure partitions**

Along with test framework and combined tests libraries, the IPC tests require the SPE binaries. The test suite compilation flow generates the following Secure partition archives for IPC tests. You must integrate these test suite partition archives with your SPE code such that it follows the level of isolation rules defined in the PSA-FF specification. Load the resultant SPE binary into the Secure main memory.

Table 2-4 Libraries and protection domains

Test suite partition libraries	Protection domain
<build_dir>/BUILD/partition/driver_partition.a	PSA-RoT
<build_dir>/BUILD/partition/client_partition.a	Application-RoT
<build_dir>/BUILD/partition/server_partition.a	Application-RoT

————— **Note** —————

- The client and server test functions of all the tests are compiled as part of `client_partition` and `server_partition` respectively. All these functions are loaded into the Secure main memory and are available at same time.
- If an SUT has main memory size constraints, you can compile and run the tests in a bulk of test sets, for example, 10 tests at time. To do this, remove the test references other than the ones required from the respective suite specific `testsuite.db` file. Repeat this process for all the test sets.

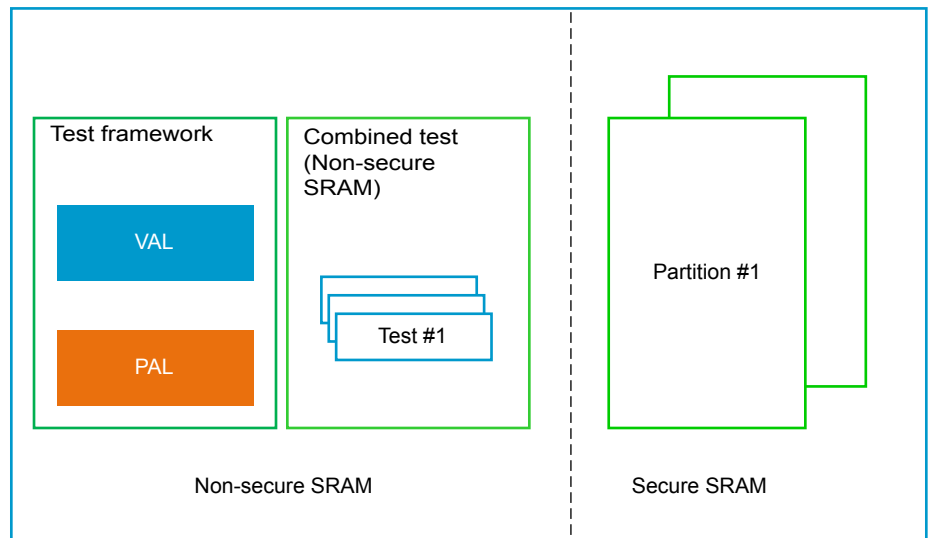


Figure 2-6 Loading test binaries

2.5 Test dispatcher

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. Query the `test_entry` function address.
2. Call the `test_entry` function of the test and execute the tests.
3. Wait for completion of the test.
4. Print and save the result of the test.
5. Repeat steps 1-4 until the end of the last test.
6. Report the test suite result summary.

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.

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.

2.6 Analyzing test run results

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

1. Performing any test initializations.
2. Dispatching the 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 Architecture Test Suite - Version 0.7 *****

Running.. Crypto Suite
*****

TEST: 201 | DESCRIPTION: Testing psa_crypto_init API: Basic
TEST RESULT: PASSED

*****

TEST: 202 | DESCRIPTION: Testing crypto key management APIs
      Failed at Checkpoint: 3
      Actual: 1
      Expected: 0
TEST RESULT: FAILED (Error Code=0x1)

*****

***** Crypto Suite Report *****
TOTAL TESTS      : 2
TOTAL PASSED     : 1
TOTAL SIM ERROR  : 0
TOTAL FAILED     : 1
TOTAL SKIPPED    : 0
*****

Entering standby..
■
```

Figure 2-7 Test suite summary

Debugging of a failing test

Since each test is organized with a logical set of self-checking code, if a failure occurs, 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.

- If the default prints do not give enough information, you can recompile and rerun the test binaries with high print verbosity level. See the PSA test suite build README to understand how test verbosity can be changed.
- In the above example, test 2 is failing. This test is located at `dev_apis/crypto/test_c002/`
- Since the failure message is shown as checkpoint 3, go to this print point in the test source code and debug the failing cause. The checkpoints are reserved in the test suite as shown below:
 - Checkpoints 1-100 are reserved for Functional APIs tests. Checkpoints print messages with numbers which can come from `test_[x][y].c` file. Here, `[x]` is reserved for Functional API tests and `[y]` is the test number.
 - Checkpoints 101-200 are reserved for client test functions of IPC tests and prints related to these numbers can come from `test_i[y].c`
 - Checkpoints 201-300 are reserved for server test functions of IPC tests and prints related to these numbers can come from `test_supp_i[y].c`
- Status of the failure code (0x1 in this example) is mapped with a structure `val_status_t` that is available at `val/common/val.h`. Look for enum that is dedicated to this number to see the status in verbatim form.

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-32.](#)

A.1 Revisions

Table A-1 Issue A

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

Table A-2 Differences between Issue A and Issue B

Change	Location	Affects
Updated the path to secure manifest files.	See 2.1 Test layering details on page 2-18	All revisions
Updated the test execution flow and SPE binary information.	See the following sections: <ul style="list-style-type: none"> 2.3 Test execution flow on page 2-23 2.4 Integrating the test suite with the SUT on page 2-26 	All revisions

Table A-3 Differences between Issue B and Issue C

Change	Location	Affects
Added information about Functional APIs.	See the following sections: <ul style="list-style-type: none"> 1.3 Platform Security Architecture APIs on page 1-11 1.4 Test suite on page 1-13 1.6 Directory structure on page 1-15 2.2 Test suite organization on page 2-20 2.3 Test execution flow on page 2-23 	All revisions
Added ITS and PS information.	See the following sections: <ul style="list-style-type: none"> 1.2 Abbreviations on page 1-10 1.4 Test suite on page 1-13 	All revisions
Moved information about the test dispatcher to a new section.	See 2.5 Test dispatcher on page 2-28	All revisions
Updated the test suite summary and debugging details.	See 2.6 Analyzing test run results on page 2-29	All revisions

Table A-4 Differences between Issue C and Issue D

Change	Location	Affects
Added PSA RoT sub category.	See 1.4 Test suite on page 1-13 .	All revisions
Updated details about the compliance sign-off process.	See Compliance sign-off process.	All revisions
Added lifecycle test directory in the PSA-FF directory structure.	See 2.2 Test suite organization on page 2-20 .	All revisions
Updated the section with details about integrating the test suite with the SUT.	See 2.4 Integrating the test suite with the SUT on page 2-26 .	All revisions

Table A-5 Differences between Issue D and Issue E

Change	Location	Affects
Added CMakeLists.txt to the directory structure.	See 1.6 Directory structure on page 1-15 .	All revisions
Updated source.mk and test_entry.c to test.cmake and test_entry_i[x].c respectively.	See 2.2 Test suite organization on page 2-20 .	All revisions

Table A-5 Differences between Issue D and Issue E (continued)

Change	Location	Affects
Updated the information about PSA-FF and Functional APIs test execution.	See 2.3 Test execution flow on page 2-23 .	All revisions
<ul style="list-style-type: none"> Updated the combined test archive section. Updated the image for loading test binaries. 	See 2.4 Integrating the test suite with the SUT on page 2-26 .	All revisions
Updated the dispatcher flow.	See 2.5 Test dispatcher on page 2-28 .	All revisions

Table A-6 Differences between Issue E and Issue F

Change	Location	Affects
Removed the compliance sign-off process section from Introduction.	See Chapter 1 Introduction on page 1-8 .	All revisions
Updated the description for Secure partitions.	See 2.1 Test layering details on page 2-18 .	All revisions