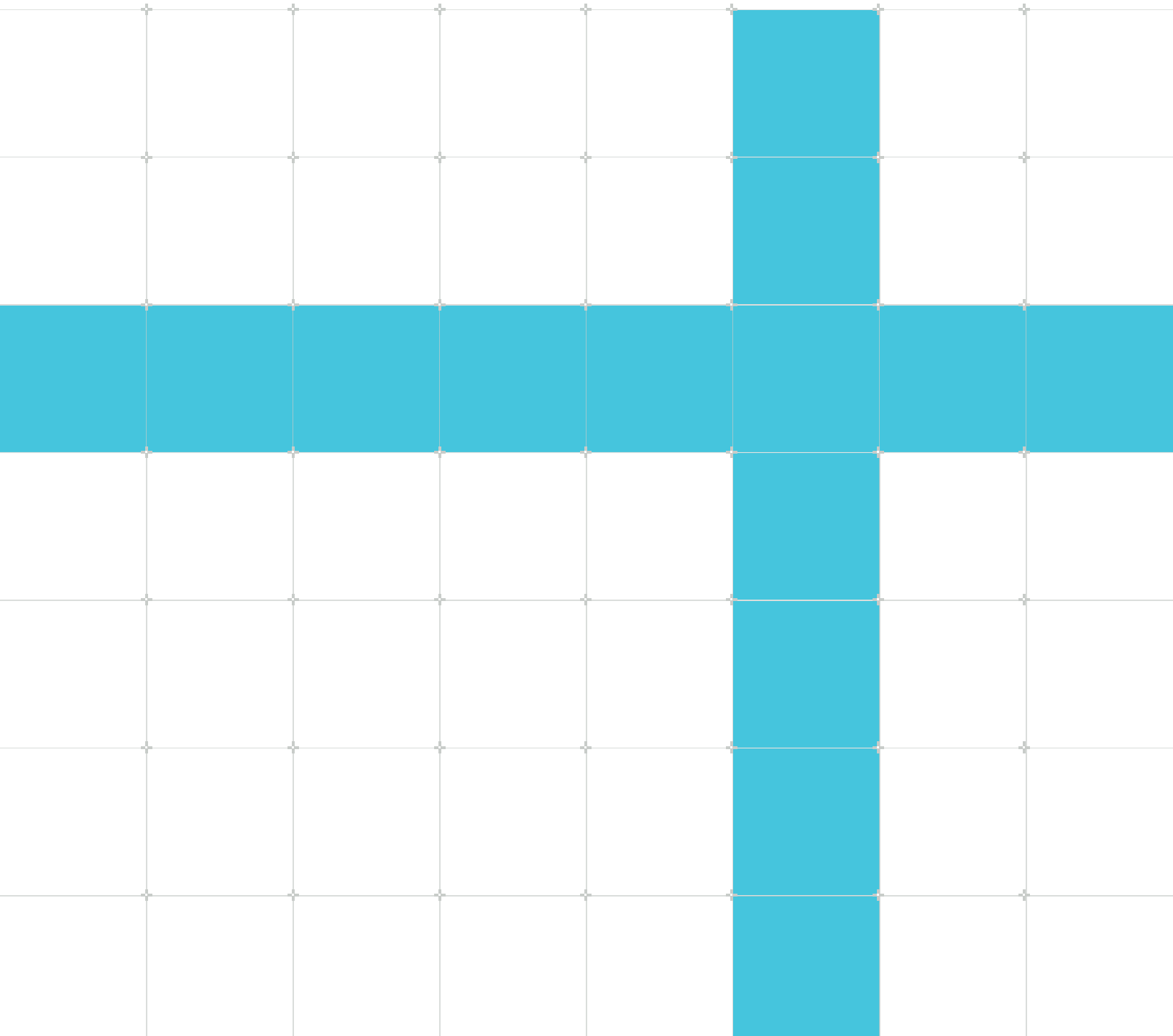
Arm® Authenticated Debug Access Control Test Suite

Version 0.8

User Guide

Issue 01
102545_0008_01_en



Arm® Authenticated Debug Access Control Test Suite

User Guide

Release Information

Document history

Issue	Date	Confidentiality	Change
0005-01	30 June 2021	Non-Confidential	Alpha release
0006-01	8 October 2021	Non-Confidential	Alpha release
0008-01	25 January 2022	Non-Confidential	Beta release

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, 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

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 a Beta product, that is a product under development.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1 Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Typographic conventions

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Signal names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace bold`	Language keywords when used outside example code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.
Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
Note	An important piece of information that needs your attention.

Convention	Use
	A useful tip that might make it easier, better or faster to perform a task.
	A reminder of something important that relates to the information you are reading.

1.2 Additional reading

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

Table 1-2: Arm publications

Document name	Document ID	Licensee only
PSA Cryptography API	IHI 0086	No
Authenticated Debug Access Control Specification	DEN0101	No

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com

1.3 Other information

See the Arm® website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

2 ADAC overview

This chapter introduces ADAC and its test suite.

2.1 Abbreviations

This section lists the abbreviations used in this document.

Table 2-1: Abbreviations and expansions

Abbreviation	Expansion
ADAC	Authenticated Debug Access Control
PAL	Platform Abstraction Layer
PSA	Platform Security Architecture
RoT	Root of Trust
SPM	Secure Partition Manager
SUT	System Under Test
VAL	Validation Abstraction Layer

2.2 Introduction to ADAC

Authenticated Debug Access Control (ADAC) is a protocol that provides a way to use strong authentication to restrict device debug access to only the authorized entities.

The ADAC specification defines communication between the target debug device and the host debugger. It addresses functional layers that are above the physical debug link. Physical access to the target device is required for debug link operation.

The default mechanism for ADAC authentication relies on a challenge-response protocol where the target challenges the host and verifies the response validity. The response to the challenge is a signed authentication token, also called a debug token. The key used by the host to generate the signature must be trusted by the target through a chain of trusted entities already provisioned on the target. The ADAC specification defines a certificate format to build trust chains and offer the flexibility to deal with complex scenarios.

For more information on ADAC, see the Authenticated Debug Access Control Specification.

2.3 Scope of the ADAC test suite

The scope of the ADAC test suite is to:

- Verify if the tested target implements all ADAC commands.

- Check if the response received for a given ADAC command is one of the expected defined responses.
- Test commands, errors, and success criteria mentioned in the ADAC specification.

2.4 ADAC test suite

The ADAC test suite checks if a device-side implementation conforms to the behavior described in the ADAC specification. For host-side conformance to the ADAC specification, Arm is working directly with partner tool vendors to guarantee compatibility.

The ADAC tests are self-checking and portable C-based tests with directed stimulus. These tests are expected to run on the host platform only. The test suite requires a debug probe connected to the host to communicate between a debugger and a hardware target. The pyOCD project is used for this communication.

The tests send the ADAC commands from the host platform and verify the response obtained from the target. The tests are open source and available with an Apache v2.0 license, allowing for external contribution.

2.5 Tool requirements

The ADAC test suite is developed in C and compiled for the host execution environment.

The following are software prerequisites on the host platform:

- CMake v3.10 or greater
- GNU Arm Embedded Toolchain version 9-2019-q4-major.
- Python 3.7 or later version with pip and virtualenv.
- Access to PSA-ADAC source code repository, as a requirement applicable only to the beta release.

2.6 Prerequisites

The ADAC tests require the following setup for the device to execute the tests on a host platform.

- Credentials must be provisioned in the device and the public key of a root certificate must either be present on the target in hashed form or in its entirety and accessible to the ADAC target code.
- A set of tests for a runtime instance verifies a single cryptosystem algorithm. To test multiple algorithms, target device must be provisioned with required credentials and the tests must be run for each of the algorithm that must be tested.

3 Validation methodology

This chapter describes the validation methodology used for the ADAC test suite.

3.1 Test layering details

The ADAC tests use a layered software stack approach to enable porting across different test platforms.

The constituents of the layered stack are:

Test suite This layer is a collection of targeted tests. It verifies if the device behavior conforms to the ADAC specification. These tests map to one or more scenarios identified in the scenario document. They use interfaces that are provided by the VAL.

PAL This layer is the closest to hardware and is aware of underlying hardware details. It must be ported or tailored to specific host platform. It implements the debug link layer responsible for communication with the target.

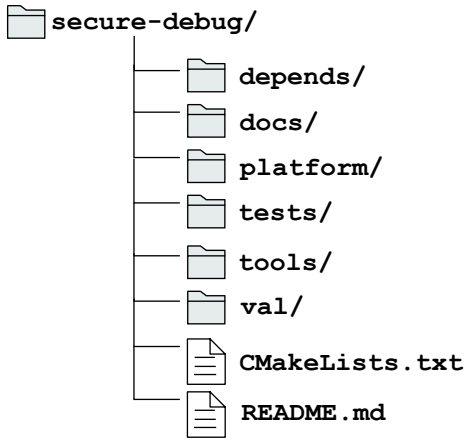
VAL This layer contains subdirectories for the VAL libraries. It provides a uniform and consistent view of the available test infrastructure to the tests in the test pool. This layer is not ported when the underlying hardware changes.

This release provides a reference implementation of the PAL layer using Unix sockets to communicate with the target. For more information on implementing the link layer, see the Porting Guide.

3.2 Directory structure

The following figure shows the top-level directory structure of Secure debug within the `psa-arch-tests` git repository.

Figure 3-1: Secure debug



depends	contains the pre-built shared libraries that implement standard link layer communication protocols. You can add your custom debug interface libraries and reference it for your platform layer API. For more information on the platform layer, see the Porting Guide.
docs	contains the User Guide, Porting Guide and scenarios to be tested.
platform	contains files to form the PAL. It must be ported to specific host platform.
tests	contains the ADAC subsuite tests that verify if the device behavior conforms with the ADAC specification.
tools	contains makefiles and scripts that are used to generate test binaries.
val	contains subdirectories and files to build the VAL libraries. This layer provides a uniform and consistent view of the available test infrastructure to the tests.
CMakeLists.txt	contains information on CMake build support.
README.md	README file for PSA ADAC test suite.

3.3 Build and execution steps

This section provides steps to build and execute the ADAC test suite for a given host platform.

Build steps

The steps and the dependencies to build the host-side executable are described in detail in the README file. The following steps compile and generate the test image for the ADAC suite:

1. `cd psa-arch-tests/secure-debug`

2. `mkdir <host_build_dir>`
3. `cd <host_build_dir>`
4. `cmake ../ -G"Unix Makefiles" - DTARGET=<target_name> -DSUITE=<suite_name>`

For example,

```
cmake ../ -G"Unix Makefiles" -DTARGET=emulation -DSUITE=ADAC
cmake --build .
```

This generates the `psa_adac_test` executable in the `<host_build_dir>` directory.

For more information on building and executing the test suite, see the platform-specific README.

Execution steps

The executable test must be provided with the host-side credentials (key and certificate chain) as command-line arguments. Depending on the debug link layer used, additional command-line parameters must be provided.

```
./psa_adac_test <path_to_key_file> <path_to_certificate> / <optional list to link-layer details>
```


Note

The README file describes the details for executing the test on a Unix-host platform. The details for executing the test suite on a reference hardware platform (`musca-b1`) is described in the README file within the platform directory for the host `musca_b1`.

3.4 Test status report

When the test suite is run on a host platform, each successful test must report either PASS or SKIP.

The following is an example code of a successful test pass.

```
***** PSA ADAC Architecture Test Suite - Version 0.8 *****

Running.. Secure Debug Suite
*****************************

TEST: 801 | DESCRIPTION: Testing ADAC Protocol Host API| UT: psa_challenge
psa_adac_issue_command       :  169 : info  : host  : Sending challenge request
psa_adac_parse_response      :  215 : debug : host  : status = 0x0000, data_count =
 9
psa_adac_parse_response      :  229 : info  : host  : Receiving challenge..
psa_adac_issue_command       :  169 : info  : host  : Sending challenge request
psa_adac_parse_response      :  215 : debug : host  : status = 0x0000, data_count =
 9
psa_adac_parse_response      :  229 : info  : host  : Receiving challenge..
Challenge response obtained is unique

TEST RESULT: PASSED
```

The following code displays the status of all the tests scheduled in a single execution run.

```
************ Secure Debug Suite Report **********
TOTAL TESTS     : 11
TOTAL PASSED    : 10
TOTAL SIM ERROR : 0
TOTAL FAILED    : 1
TOTAL SKIPPED   : 0
****************************************
```

3.5 Additional notes

The following are a few additional points to consider:

- The data structures and the packet format described in the specification are defined in the `psa_adac.h` header file within the `psa-adac` repository.
- The `val_adac.h` defines the API for the following tasks:
  - Managing the host credentials (uses mbedtls services).
  - Building and constructing ADAC commands.
  - Parsing the response obtained from the debug target.
- The PAL API which is responsible for implementing the debug link layer is defined in `pal_interfaces.h`. These APIs must be ported for your host platform. For more information on API porting, see the Porting Guide.
- The `psa-adac` repository provides a tool for generating keys and certificates. The same is used to generate credentials for authenticating the host.

Appendix A Revisions

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

A.1 Revisions

This section consists of all the technical changes between different versions of this document.

Table A-1: Issue 0005-01

Change	Location
First release.	-

Table A-2: Differences between Issue 0005-01 and 0006-01

Change	Location
No technical updates.	-

Table A-3: Differences between Issue 0006-01 and 0008-01

Change	Location
Removed RDDI abbreviation.	See 2.1 Abbreviations on page 8.
Updated ADAC test suite section.	See 2.4 ADAC test suite on page 9.
Removed RDDI references in tool requirements section.	See 2.5 Tool requirements on page 9.
Updated the Build and execution steps section.	See 3.3 Build and execution steps on page 11.
Updated the test codes in Test status report section.	See 3.4 Test status report on page 12.