SystemReady Security Interface Extension

Version 1.1

User Guide

Non-Confidential

Issue 01

102872_0101_01_en

SystemReady Security Interface Extension
User Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	24 May 2022	Non-Confidential	First release
0101-01	25 January 2023	Non-Confidential	Updates for SystemReady IR 2.0 - Beta Release

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview......6
1.1 Glossary......6

2. SystemReady Security Interface Extension certification......8
2.1 How the certification process works......8

3. Test Security Interface Extension compliance......9
3.1 ACS test process......10
3.2 ACS prerequisites......11
3.3 Deploy the prebuilt ACS image......11
3.3.1 Deploy the prebuilt ACS image on ES or SR......11
3.3.2 Deploy the prebuilt ACS image on IR 2.0......12
3.4 Enroll the Secure Boot keys......12
3.4.1 Enrolling keys in EDK2......13
3.4.2 Enrolling keys in U-boot......14
3.5 Run SCT......15
3.5.1 Run SCT on ES or SR......16
3.5.2 Run SCT on IR 2.0......17
3.6 Run FWTS......18
3.7 Secure firmware update test......20
3.8 Review the ACS test result logs......22
3.8.1 Review the SCT logs......22
3.8.2 Review the FWTS logs......23
3.8.3 Review the firmware update logs......23
3.8.4 Review the TPM measured boot log......26

4. Related information......30

5. Next steps......31

1. Overview

This guide provides an overview of the certification and test process for the Arm SystemReady Security Interface Extension. Arm SystemReady is a set of standards and a compliance certification program that enables interoperability with generic, off-the-shelf operating systems and hypervisors. The Arm SystemReady certification program encompasses a broad set of devices from cloud to IoT edge.

Compliant systems that meet the Arm SystemReady terms and conditions are issued with a compliance certificate and can use the Arm SystemReady certified stamp logo.

The SystemReady Security Interface Extension is an additional certification that is an extension to SystemReady. It certifies that a device complies with industry standard security interfaces.

The Base Boot Security Requirements (BBSR) specification describes these security requirements and covers the following areas:

- UEFI authenticated variables
- UEFI secure boot
- UEFI capsule updates
- TPM 2.0 and measured boot

A Security Interface Extension certification provides assurance that the security interfaces covered by BBSR are implemented according to standards. However, interface compliance does not provide assurance that the underlying platform is secure. In the process of architecting a system, system-level threat modeling should be performed to evaluate threats, risks, and mitigations. The Platform Security Architecture (PSA) and the PSA Certified framework provides a comprehensive approach to platform security that is based on a defined set of security goals. PSA provides architecture and requirements specifications for building secure platforms. A Security Interface Extension certification complements PSA. PSA Certified provides a measure of the robustness of an implementation, through an assessment process that is performed by a security certification laboratory.

For more information about the Arm SystemReady certification program, see Arm SystemReady Certification Program.

This guide provides guidance for the Security Interface Extension only.

1.1 Glossary

This document uses the following terms and abbreviations.

Term	Meaning
ACS	Architecture Compliance Suite
DER	Distinguished Encoding Rules. A format that can be used for encoding keys.

Term	Meaning
FWTS	Firmware Test Suite, see https://wiki.ubuntu.com/FirmwareTestSuite/
PCR	Platform Configuration Register
SCT	UEFI Self-Certification Tests
SUT	System under test
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface

2. SystemReady Security Interface Extension certification

The SystemReady certification program ensures the highest possible standards for systems supporting off-the-shelf operating systems and hypervisors. To help you navigate the process, Arm provides the following support:

- This guide, which explains how to run the Architecture Compliance Suite (ACS) for the Security Interface Extension.
- The Arm SystemReady Compliance team helps evaluate your system for certification and provides feedback and guidance.
- Arm provides checklists, forms, and report templates to ensure that all necessary information is submitted with your certification request. For more details, see the Arm SystemReady Requirements Specification.
- The Arm SystemArchAC mailing list. Contact the Arm SystemReady Certification Program at `support-systemready-acs@arm.com` for further information.

2.1 How the certification process works

The Arm SystemReady Requirements Specification describes the SystemReady certification process. The certification process includes the following steps:

- You make a certification request.
- You provide information to allow Arm to evaluate certification readiness. Arm provides feedback.
- You run the Security Interface Extension ACS and make a certification submission including test logs. If issues are found, this step may need to be repeated multiple times. Use the SystemReady SIE Compliance Report template directory structure to collect all test results.
- Arm reviews your submission and issues the official certificate.

Note

The Security Interface Extension is an extension to the certification received for the SystemReady SR, ES, and IR bands. For a system to be considered for the Security Interface Extension certification, it must either already have certification or be in the certification process for one of the SystemReady bands.

---

3. Test Security Interface Extension compliance

The Security Interface Extension ACS is a compliance suite that tests for compliance with the requirements specified in the Base Boot Security Requirements specification. Passing the tests in this compliance suite is required before being granted the SystemReady Security Interface Extension certification.

The ACS is delivered as a bootable live OS image containing a collection of test suites. The ACS is also available in source form with a build environment.

The following sections describe the steps for running the ACS and collecting the test results. This process includes the following:

- Preparing the system under test and enrolling the Secure Boot keys.
- Booting the live OS image, which automatically runs a subset of the SCT test suite.
- Resetting and booting the live OS image into Linux, which automatically runs a subset of the FWTS test suite.
- Testing the firmware update mechanism by following a manual test procedure using a vendor-supplied update capsule.

After the test steps are completed, the test logs on the live OS storage device are reviewed and analyzed.

The following tests must pass to achieve certification:

- SCT authenticated variable tests
- SCT secure boot variable size test
- SCT secure boot image loading, variable update, and variable attribute tests
- FWTS authenticated variable tests
- Firmware update using an update capsule

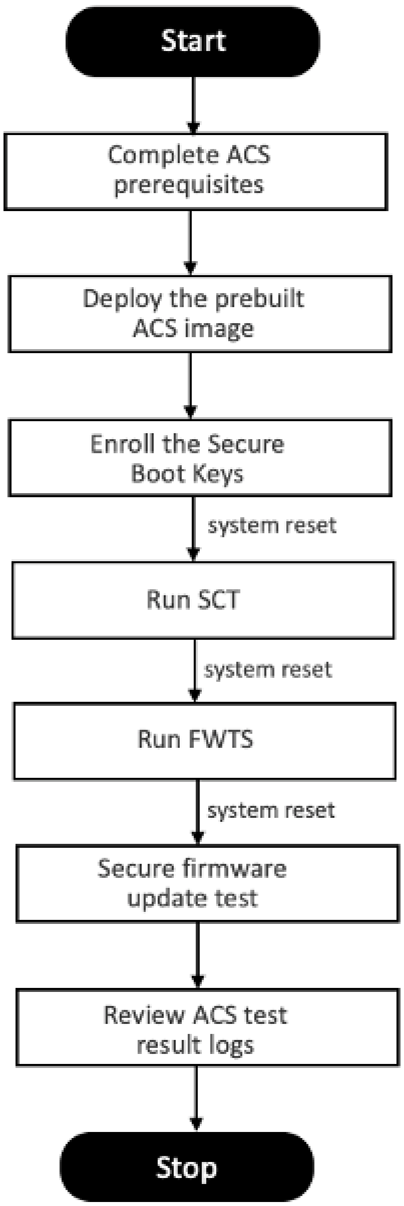
If a TPM is present on the system and the system supports TPM measured boot, the following additional tests must pass:

- SCT TCG2 protocol test
- FWTS tpm2 test
- Evaluation of the measured boot log

3.1 ACS test process

The following diagram shows the overall SIE ACS test process:

Figure 3-1: SIE Test process



3.2 ACS prerequisites

You must do the following before running the ACS:

1. The bootable ACS live image must be available on the system on a bootable storage device.
2. Prepare the SUT machine with up-to-date firmware and a host machine for SUT console access.
3. For Secure Boot the system firmware must be in Setup Mode, where the Secure Boot keys are cleared before starting the ACS. The mechanism to enroll Secure Boot keys is platform-specific and the procedure to enroll the keys must be available.
4. If the system supports in-band system firmware updates, the capsule update ACS test must be run. A vendor-provided firmware update capsule must be available on a storage device on the system.

3.3 Deploy the prebuilt ACS image

There are different ACS image deployment processes, one for IR 2.0 and another for the ES and SR bands of SystemReady.

3.3.1 Deploy the prebuilt ACS image on ES or SR

To set up the bootable storage device on the ES or SR bands of SystemReady, do the following:

1. Download the pre-built ACS image for SystemReady Security Interface Extension certification to a local directory on a Linux host machine.

   The prebuilt ACS image is available on GitHub at the following location:

   https://github.com/ARM-software/arm-systemready/tree/security-interface-extension-acs/security-interface-extension/prebuilt_images

   For information on the latest release and release tags, see the SystemReady Security Interface Extension ACS README.
2. Uncompress the prebuilt ACS image. Use a utility such as `xz` on Linux to uncompress the `sie_acs_live_image.img.xz` file.
3. On the Linux host machine, deploy the prebuilt ACS image to the storage device using the following commands:

   ```
   $ lsblk
   $ sudo dd if=/path/to/sie_acs_live_image.img of=/dev/sdX
   $ sync
   ```

   The `lsblk` command displays the storage devices in the system. Use the output of this command to identify the name of the target storage device.

The `dd` command copies the prebuilt ACS image to the storage device. Replace `/dev/sdX` with the name of the target storage device.

The `sync` command forces the copy operation to complete, so that you can unplug the target storage device.

3.3.2 Deploy the prebuilt ACS image on IR 2.0

For SystemReady IR 2.0, the ACS tests for SystemReady Security Interface Extension certification are part of the IR 2.0 live image.

To deploy the image, follow the IR 2.0 deployment steps in the SystemReady IR Integration, Test, and Certification Guide.

3.4 Enroll the Secure Boot keys

The ACS provides a set of keys for the UEFI Secure Boot variables PK, KEK, db, and dbx. Before running the test suite these test keys must be enrolled using the platform-specific procedure for the firmware of the platform under test.

The test keys are available on the `boot` partition of the ACS image.

For the ES and SR bands of SystemReady, the keys are at the following path in the ACS image:

```
EFI\BOOT\bbr\security-interface-extension-keys
```

For IR band of SystemReady, the keys are at the following path in the ACS image:

```
security-interface-extension-keys
```

The test keys are available in the following formats:

- DER format, suitable for enrolling in EDK2
- Signed UEFI variable signature list blobs, suitable for u-boot

The following files are the DER formatted test keys:

- `TestDB1.der`
- `TestDBX1.der`
- `TestKEK1.der`
- `TestPK1.der`

The following files are the signed signature list formatted test keys:

- `TestDB1.auth`
- `TestDBX1.auth`

- `TestKEK1.auth`
- `TestPK1.auth`

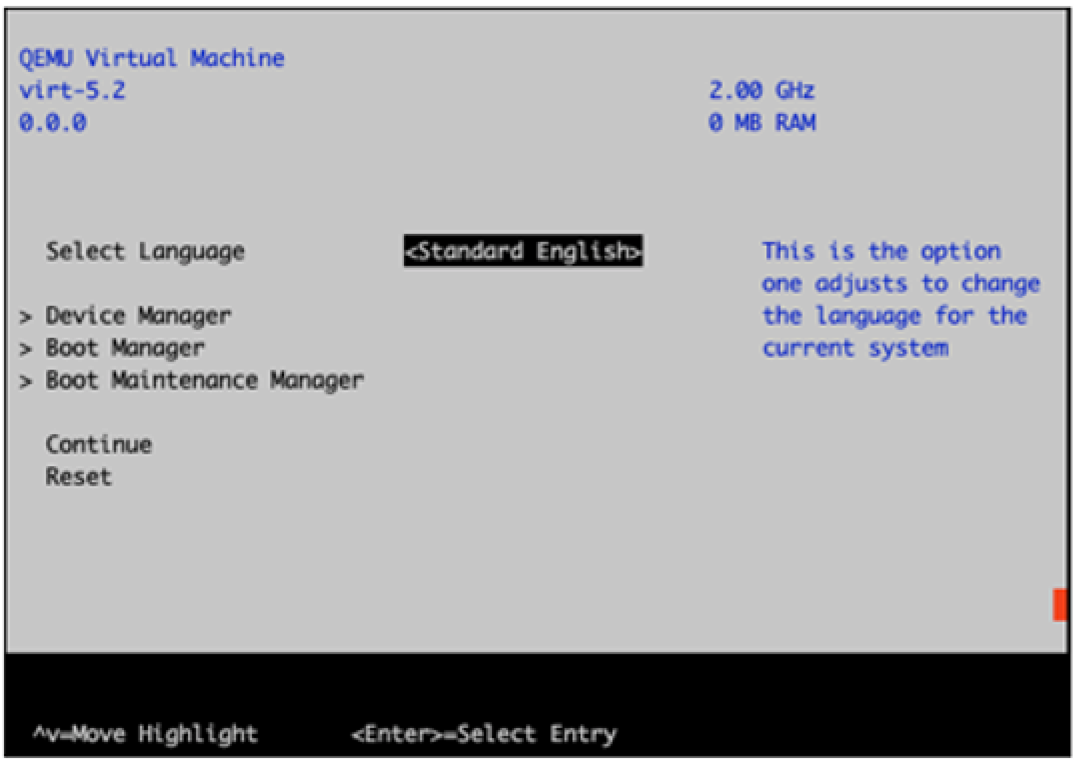
3.4.1 Enrolling keys in EDK2

The example in this section shows how to enroll the Secure Boot keys on QEMU with EDK2 based firmware.

Perform the following steps:

1. After starting the system, press `ESC` to enter the EDK2 menu:

   Figure 3-2: EDK2 menu example



2. Select Device Manager > Secure Boot Configuration.
3. Select Custom Mode for Secure Boot Mode.
4. Select Custom Secure Boot Options.
5. To enroll the Platform Key (PK), do the following:
   a. Select PK Options.
   b. Select Enroll PK > Enroll PK Using File.
   c. Select the ACS disk which has the `BOOT` label.
   d. The secure boot keys are located at the following path on the disk:
      - `EFI\BOOT\bbr\security-interface-extension-keys`

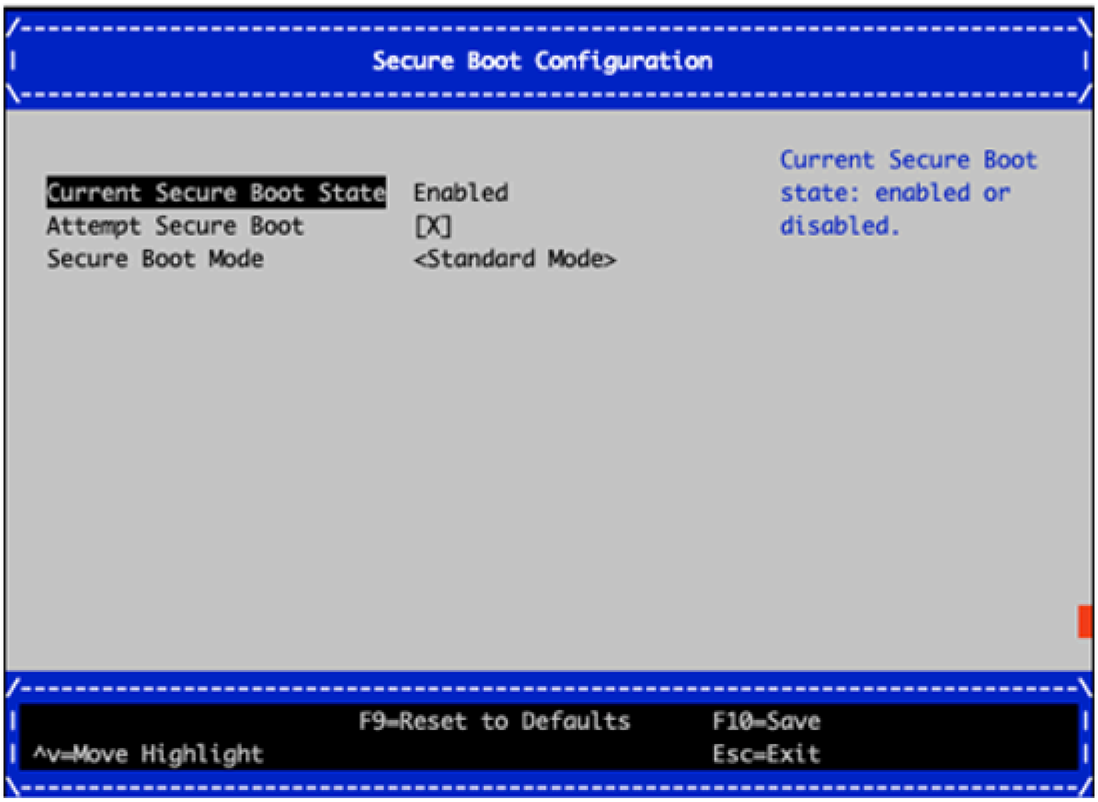
e. Select the `TestPK1.der` file for PK.

f. Repeat the above steps to enroll the keys for KEK (`TestKEK1.der`), db (`TestDB1.der`) and dbx (`TestDBX1.der`) selecting the following options:

- KEK Options
- DB Options
- DBX Option

6. After completing the above steps, secure boot is enabled as shown in the following diagram:

Figure 3-3: Secure Boot Configuration Example



7. Reset the system.

3.4.2 Enrolling keys in U-boot

For information about enrolling keys with U-boot firmware, see the U-boot document UEFI on U-Boot.

For U-boot the method to access and load the Secure Boot keys differs depending on the type of physical storage device where the SIE ACS is located.

For example, to initialize the USB subsystem:

```
==> usb start
```

To initialize MMC device 1:

```
==> mmc dev 1
```

For further details please refer to the U-boot documentation for the storage device type in use.

The following example shows how to enroll PK, KEK, db, and dbx with the SIE ACS provided keys loaded from USB device 0:

```
==> load usb 0 ${loadaddr} security-extension-acs-keys/TestPK1.auth && setenv -e -nv
 -bs -rt -at -i ${loadaddr}:$filesize PK
==> load usb 0 ${loadaddr} security-extension-acs-keys/TestKEK1.auth && setenv -e -
nv -bs -rt -at -i ${loadaddr}:$filesize KEK
==> load usb 0 ${loadaddr} security-extension-acs-keys/TestDB1.auth && setenv -e -nv
 -bs -rt -at -i ${loadaddr}:$filesize db
==> load usb 0 ${loadaddr} security-extension-acs-keys/TestDBX1.auth && setenv -e -
nv -bs -rt -at -i ${loadaddr}:$filesize dbx
```

3.5 Run SCT

The Security Interface Extension SCT is a subset of the SCT that focuses on security interfaces: authenticated variables, Secure Boot variables, Secure Boot image loading, and TCG2 protocol test for systems with TPMs.

There are different processes to start the Security Interface Extension SCT in the ACS live image, one for IR 2.0 and another for the ES and SR bands of SystemReady.

3.5.1 Run SCT on ES or SR

After resetting the system with the ACS Secure Boot keys enrolled, GRUB automatically starts the Security Interface Extension SCT as follows:

Figure 3-4: GRUB menu

After the tests complete, the system returns to the UEFI Shell. The test output is available in the `acs_results` directory of the ACS disk:

- `\acs_results\sct_results\Overall\Summary.log`

Note

If a TPM is present in the system and is supported by the firmware, the TCG2 protocol tests in SCT run and are included in the `Summary.log` file. If the TCG2 protocol is not present, these tests do not run.

3.5.2 Run SCT on IR 2.0

After resetting the system with the ACS Secure Boot keys enrolled, select the Security Interface Extension SCT option from the GRUB menu to start the Security Interface Extension SCT:

Figure 3-5: GRUB menu

After the tests complete, the system returns to the UEFI Shell. The test output is available in the `acs_results` directory of the ACS disk:

- `\acs_results\SIE\sct_results\Overall\Summary.log`

Note

If a TPM is present in the system and is supported by the firmware, the TCG2 protocol tests in SCT run and are included in the `summary.log` file. If the TCG2 protocol is not present, these tests do not run.

3.6 Run FWTS

FWTS is a set of Linux-based firmware tests. The Security Interface Extension ACS runs a subset of FWTS: the authenticated variable tests and `tpm2`, the TPM2 Trusted Platform Module 2 test.

In addition, if the system under test implements a TPM device, commands are executed to capture the results of TPM measured boot.

After completing the SCT tests, reset the system at the UEFI Shell using the `reset` command:

```
FS1:\acs_results\app_output\> reset
```

For the ES and SR bands of SystemReady, after the system is reset and the boot process reaches the GRUB menu, select the Linux BusyBox option to boot Linux:

Figure 3-6: GRUB menu for ES and SR - BusyBox

For the IR 2.0 band of SystemReady, after the system is reset and the boot process reaches the GRUB menu, select the Linux Boot for Security Interface Extension option to boot Linux:

Figure 3-7: GRUB menu for IR 2.0 - Yocto

```
                         GNU GRUB version 2.11
/----------------------------------------------------------------------------\
|*Linux                                                                      |
| Boot bbr/bsa                                                               |
| Security Interface Extension SCT (optional)                                |
| Linux Boot for Security Interface Extension (optional)                     |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
/----------------------------------------------------------------------------\
```

Linux boots, automatically runs a subset of FWTS, and outputs the TPM event log and PCRs.

If no TPM is present, the `tpm2` test and output of PCRs and the event log is skipped:

```
Test Executed are uefirtauthvar tpm2
Running 2 tests, results appended to /mnt/acs_results/fwts/FWTSResults.log
Test: Authenticated variable tests.
  Create authenticated variable test.                     1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                     1 passed
  Update authenticated variable test.                     1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                     1 passed
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
 Test skipped.
TPM event log not found at /sys/kernel/security/tpm0/binary_bios_measurements
```

If a TPM is present, the `tpm2` test is run and PCRs and event log are output to the `acs_results` directory:

```
Test Executed are uefirtauthvar tpm2
Running 2 tests, results appended to /mnt/acs_results/fwts/FWTSResults.log
Test: Authenticated variable tests.
  Create authenticated variable test.                     1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                     1 passed
  Update authenticated variable test.                     1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                     1 passed
```

```
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
  Validate TPM2 table.                                    1 passed
TPM2: dumping PCRs and event log
  Event log: /mnt/acs_results/tmp2/eventlog.log
  PCRs: /mnt/acs_results/tmp2/pcr.log
```

Logs are available in the `RESULTS` partition of the ACS storage device.

For the ES and SR bands:

- FWTS results: `acs_results/fwts/FWTSResults.log`
- Event log: `acs_results/tpm2/eventlog.log`
- PCRs: `acs_results/tpm2/pcr.log`

For the IR band:

- FWTS results: `acs_results/SIE/fwts/FWTSResults.log`
- Event log: `acs_results/SIE/tpm2/eventlog.log`
- PCRs: `acs_results/SIE/tpm2/pcr.log`

3.7 Secure firmware update test

The BBSR requires support for update capsules compliant with the UEFI specification for systems that perform in-band firmware updates. The Security Interface Extension ACS firmware update test is a manual test run from the firmware that requires a valid update capsule for the system's firmware.

The UEFI specification defines two methods of performing updates with capsules:

- The `UpdateCapsule()` runtime function (see UEFI section 8.5.3)
- Capsule on disk (see UEFI section 8.5.5)

Either method is acceptable for performing the capsule update test. The vendor of the system under test must provide the update procedure to use.

Some of the steps in the test procedure below use the `CapsuleApp.efi` program that is located on the ACS image at the following path: `EFI\BOOT\app\CapsuleApp.efi`.

Step #1. Preparation

Perform the following steps to prepare for the firmware update test:

1. Copy the vendor-provided update capsule image onto a storage device.
2. Prepare an invalid copy of the vendor provided-update capsule that has been tampered with, for use in a negative test. Using a copy of the vendor provided update capsule, use a binary editor such as the `xxd` command and an editor to modify the final byte of the image.

3. Copy the invalid update capsule onto the storage device.
4. Enable or install the storage device containing the capsule images on the system under test so that it is visible to firmware.

Step #2. Reset the system to get to the UEFI Shell

The firmware update tests are run manually from the UEFI Shell. Reset the system and allow the automated boot sequence to run SCT. Because SCT has run previously, as described in Run FWTS, to avoid overwriting the previous results the boot flow stop at the UEFI `Shell` prompt without running SCT, as shown in the following output:

```
Shell> echo -off
Press any key to stop the EFI SCT running
Load support files ... seconds
Load proxy files ...
Load test files ...
Continue test preparing...
Done!
- [ok]
- [ok]
- [ok]
Running CapsuleApp
FS1:\acs_results\app_output\>
```

Step #3. Dump the ESRT

Change to the `acs_results\fwupdate` directory and use the `CapsuleApp -E` command to dump the firmware's EFI System Resource Table (ESRT) into a log file named `esrt_dump.log`, as follows:

```
FS1:\acs_results\fwupdate\> CapsuleApp -E > esrt_dump.log
```

Step #4. Dump the FMP information advertised by the firmware

Change to the `acs_results\fwupdate` directory and use the `CapsuleApp -P` command to dump the FMP (Firmware Management Protocol) information into a log file named `fmp_dump.log`, as follows:

```
FS1:\acs_results\fwupdate\> CapsuleApp -P > fmp_dump.log
```

Step #5. Dump the update capsule header

CChange to the `acs_results\fwupdate` directory and use the `CapsuleApp -D` command to dump the header of the vendor provided update capsule into a log file named `capsule_header.log`, as follows:

```
FS1:\acs_results\fwupdate\> CapsuleApp -D [capsule-image] > capsule_header.log
```

Step #6. Test a firmware update with the tampered capsule image

This is a negative test that verifies whether the firmware update process correctly rejects a capsule that has been tampered with. The expected result is that the firmware update must not be processed.

To perform this test, follow the vendor provided firmware update procedure to perform the update using a capsule that has been tampered with, as described in Step #1. Preparation.

```
FS1:\acs_results\fwupdate\> CapsuleApp FS2:\DeveloperBox.tampered.cap >
 fwupdate_tampered.log
```

Step #7. Test a firmware update using the CapsuleApp with the vendor provided capsule

This test step verifies that the vendor-provided update capsule is applied correctly. To perform this test, follow these steps:

1. Use the vendor-provided procedure to perform the update using the valid vendor-provided update capsule. The expected result is that the capsule is processed successfully, and the firmware is updated.

   ```
   FS1:\acs_results\fwupdate\> CapsuleApp FS2:\DeveloperBox.cap > fwupdate.log
   ```

2. Reset the system.
3. Repeat the test step to dump the FMP into a log file named `fmp_post_update_dump.log` to verify that the FMP advertises the new firmware version:

   ```
   FS1:\acs_results\fwupdate\> CapsuleApp -P > fmp_post_update_dump.log
   ```

3.8 Review the ACS test result logs

The log files generated by running the ACS tests are located in a separate partition within the live image called `acs_results`.

This results partition can be viewed on the directory `/mnt` when booted into Linux BusyBox on the system under test.

The storage device holding the results partition can also be viewed on a host machine.

For the ES and SR bands of SystemReady, the logs are located at the path: `acs_results/`.

For the IR band of SystemReady, the logs are located at the path: `acs_results/SIE/`.

To submit the test logs to Arm for certification, use the directory structure in the SystemReady SIE Compliance Report template.

3.8.1 Review the SCT logs

Review the SCT summary log created during the SCT phase of test execution at the following location:

```
sct_results/Overall/Summary.log
```

The expected result is that the following tests pass:

- `RuntimeServicesTest\VariableServicesTest\AuthVar_Conf` (34 tests)
- `RuntimeServicesTest\VariableServicesTest\AuthVar_Func` (16 tests)
- `RuntimeServicesTest\VariableServicesTest\BBSRVariableSizeTest\BBSRVariableSizeTest_func` (2 tests)
- `RuntimeServicesTest\SecureBootTest\ImageLoading` (17 tests)
- `RuntimeServicesTest\SecureBootTest\VariableAttributes` (8 tests)
- `RuntimeServicesTest\SecureBootTest\VariableUpdates` (10 tests)

If a TPM is present in the system and is supported by the firmware, the TCG2 protocol tests in SCT run and are included in the `Summary.log` file. If the TCG2 protocol is not present, these tests do not run. The following TCG2 protocol tests must pass:

- `TCG2ProtocolTest\GetActivePcrBanks_Conf` (2 tests)
- `TCG2ProtocolTest\GetCapability_Conf` (3 tests)
- `TCG2ProtocolTest\HashLogExtendEvent_Conf` (10 tests)
- `TCG2ProtocolTest\SubmitCommand_Conf` (2 tests)

3.8.2 Review the FWTS logs

Review the FWTS results log created during the FWTS phase of test execution at the following location:

```
fwts/FWTSResults.log
```

The expected result is that all tests pass. The `tpm2` test is only applicable to ACPI-based systems that support TPM measured boot. An example of the expected results is shown below:

```
Test: Authenticated variable tests.
  Create authenticated variable test.                      1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                      1 passed
  Update authenticated variable test.                      1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                      1 passed
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
  Validate TPM2 table.                                     1 passed
```

3.8.3 Review the firmware update logs

There are six logs created during the firmware update test procedure that must be reviewed:

- ESRT dump: `fwupdate/esrt_dump.log`

- FMP dump: `fwupdate/fmp_dump.log`
- Capsule header dump: `fwupdate/capsule_header.log`
- Tampered firmware update log: `fwupdate/fwupdate_tampered.log`
- Firmware update log: `fwupdate/fwupdate.log`
- Post-firmware update FMP dump: `fwupdate/fmp_post_update_dump.log`

3.8.3.1 ESRT log review

The expected result for the ESRT log file `esrt_dump.log` is that it shows a table entry for all updateable system firmware components.

The following is an example of an ESRT log file:

```
FS1:\acs_results\fwupdate\> type esrt_dump.log

 ##############
 # ESRT TABLE #
 ##############
 EFI_SYSTEM_RESOURCE_TABLE:
 FwResourceCount    - 0x1
 FwResourceCountMax - 0x40
 FwResourceVersion  - 0x1
 EFI_SYSTEM_RESOURCE_ENTRY (0):
   FwClass                  - 50B94CE5-8B63-4849-8AF4-EA479356F0E3
   FwType                   - 0x1 (SystemFirmware)
   FwVersion                - 0x26
   LowestSupportedFwVersion - 0x1
   CapsuleFlags             - 0x1
   LastAttemptVersion       - 0x26
   LastAttemptStatus        - 0x0 (Success)
```

3.8.3.2 FMP log review

The expected result for the FMP log file `fmp_dump.log` is:

- The `ImageTypeId` fields match the `FwClass` advertised by the ESRT.
- The `AUTHENTICATION_REQUIRED` attribute is set, indicating that image authentication is required.

The following is an example of an FMP log file:

```
FS1:\acs_results\fwupdate\> type fmp_dump.log

 ############
 # FMP DATA #
 ############
 FMP (0) ImageInfo:
   DescriptorVersion  - 0x3
   DescriptorCount    - 0x1
   DescriptorSize     - 0x70
   PackageVersion     - 0xFFFFFFFF
   PackageVersionName - "Unknown"
   ImageDescriptor (0)
     ImageIndex                  - 0x1
```

```
    ImageTypeId                 - 50B94CE5-8B63-4849-8AF4-EA479356F0E3
    ImageId                     - 0x584F425645444E53
    ImageIdName                 - "Socionext Developer Box"
    Version                     - 0x26
    VersionName                 - "build #38U"
    Size                        - 0x280000
    AttributesSupported         - 0xF
      IMAGE_UPDATABLE           - 0x1
      RESET_REQUIRED            - 0x2
      AUTHENTICATION_REQUIRED   - 0x4
      IN_USE                    - 0x8
      UEFI_IMAGE                - 0x0
    AttributesSetting           - 0xF
      IMAGE_UPDATABLE           - 0x1
      RESET_REQUIRED            - 0x2
      AUTHENTICATION_REQUIRED   - 0x4
      IN_USE                    - 0x8
      UEFI_IMAGE                - 0x0
    Compatibilities             - 0x0
      COMPATIB_CHECK_SUPPORTED  - 0x0
    LowestSupportedImageVersion - 0x1
    LastAttemptVersion          - 0x26
    LastAttemptStatus           - 0x0 (Success)
    HardwareInstance            - 0x0
FMP (0) PackageInfo - Unsupported
```

3.8.3.3 Capsule header log review

The expected result of the capsule header log file `capsule_header.log` is that the log shows a valid `CapsuleHeader`, `FmpHeader`, and `FmpPayload`.

The following is an example of a valid capsule header log file:

```
FS1:\acs_results\fwupdate\> type capsule_header.log

[FmpCapsule]
 CapsuleHeader:
   CapsuleGuid      - 6DCBD5ED-E82D-4C44-BDA1-7194199AD92A
   HeaderSize       - 0x20
   Flags            - 0x0
   CapsuleImageSize - 0x2DC035
 FmpHeader:
   Version             - 0x1
   EmbeddedDriverCount - 0x0
   PayloadItemCount    - 0x1
   Offset[0]           - 0x10
 FmpPayload[0] ImageHeader:
   Version                - 0x2
   UpdateImageTypeId      - 50B94CE5-8B63-4849-8AF4-EA479356F0E3
   UpdateImageIndex       - 0x1
   UpdateImageSize        - 0x2DBFDD
   UpdateVendorCodeSize   - 0x0
   UpdateHardwareInstance - 0x0
```

3.8.3.4 Firmware update with tampered capsule

For the firmware update test with the tampered with capsule, the expected result is that no firmware update occurs.

The following is an example of a valid `fwupdate_tampered.log` file:

```
FS1:\acs_results\fwupdate\> type fwupdate_tampered.log
CapsuleApp: creating capsule descriptors at 0xFF7BC898
CapsuleApp: capsule data starts          at 0xF491F018 with size 0x2DC035
CapsuleApp: block/size                      0xF491F018/0x2DC035
```

3.8.3.5 Firmware update with vendor capsule

For the firmware update test with the vendor provide with capsule, the expected result is that a firmware update occurs.

The expected output in `fwupdate.log` should reflect that a firmware update occurred.

The following is an example of a valid `fwupdate.log` file:

```
FS1:\acs_results\fwupdate\> type fwupdate.log
CapsuleApp: creating capsule descriptors at 0xFF7BC018
CapsuleApp: capsule data starts          at 0xF491F018 with size 0x2DC035
CapsuleApp: block/size                      0xF491F018/0x2DC035
Updating firmware - please wait....................................
```

Following the firmware update the FMP is dumped a second time producing the log file `fmp_post_update_dump.log`. This log file must reflect the newly updated firmware.

3.8.4 Review the TPM measured boot log

There are two logs created during the FWTS phase of test execution that must be reviewed for TPM measured boot:

- Event log: `tpm2/eventlog.log`
- PCRs: `tpm2/pcr.log`

The steps below explain how to verify key requirements defined in the TCG Firmware Profile specification. The measurements for a particular system are highly platform-specific. The TCG Firmware Profile specification dictates the specific requirements.

1. Verify that the cumulative SHA256 measurements from the event log match the TPM PCRs 0-7.

   The events logged in the TPM event log must match the actual measurements extended in the TPM PCRs. It is trivial to perform a visual comparison of this by viewing the SHA256 PCR values in the `pcr.log` file and the computed values at the end of `eventlog.log`.

The following example shows where the PCR values and event log values match.

```
SHA256 values for PCRs 0-7 from pcr.log
sha256:
 0 : 0x4A17B720C5E37DCD65533EB47CDE5B5E1E93E9A5953B42E913F2C83D88576685
 1 : 0x8EFBC5102BEB859074EC99DB20009BD213726B57777DA560B7BC7AA567C22425
 2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
 3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
 4 : 0xFE3C30CA8D4CACCAAAE635D60DC3132D1B5C93E0F2BB092BF0D83287D76B1210
 5 : 0x768A45048228ECE6EF442FA88AF60DFB19D8ABCB1869E1DBDBEAEA1244353037
 6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
 7 : 0x7D852DB48CA55F36243903877E416D4AF77AA8755010C064884799E70F51664D
SHA256 values for PCRs 0-7 from eventlog.log
pcrs:
  sha256:
    0
  : 0x4a17b720c5e37dcd65533eb47cde5b5e1e93e9a5953b42e913f2c83d88576685
    1
  : 0x8efbc5102beb859074ec99db20009bd213726b57777da560b7bc7aa567c22425
    2
  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    3
  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    4
  : 0xfe3c30ca8d4caccaaae635d60dc3132d1b5c93e0f2bb092bf0d83287d76b1210
    5
  : 0x768a45048228ece6ef442fa88af60dfb19d8abcb1869e1dbdbeaea1244353037
    6
  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    7
  : 0x7d852db48ca55f36243903877e416d4af77aa8755010c064884799e70f51664d
```

2. Verify the `EV_NO_ACTION` event for Specification ID version.

   The first event in the event log must be the Specification ID version. This is an `EV_NO_ACTION` event and is not extended into a PCR.

   For example:

```
- EventNum: 0
  PCRIndex: 0
  EventType: EV_NO_ACTION
  Digest: "0000000000000000000000000000000000000000"
  EventSize: 45
  SpecID:
  - Signature: Spec ID Event03
    platformClass: 0
    specVersionMinor: 0
    specVersionMajor: 2
    specErrata: 0
    uintnSize: 2
    numberOfAlgorithms: 4
```

3. Verify `EV_POST_CODE` events for measurements of firmware to PCR[0].

   All mutable Secure and Non-secure firmware components must be measured into PCR[0] using the `EV_POST_CODE` event type. The suggested event data values are provided in BBSR.
4. Verify `EV_POST_CODE` events for measurements of signed critical to data PCR[0].

   All signed critical data must be measured into PCR[0] using the `EV_POST_CODE` event type, with platform-specific event data.
5. Verify Secure Boot policy measurements.

   The contents of the `SecureBoot`, `PK`, `KEK`, `db`, and `dbx` variables must be measured into PCR[7] using the `EV_EFI_VARIABLE_DRIVER_CONFIG` event type.

   The following example shows the measurement of the SecureBoot variable:

```
- EventNum: 3
  PCRIndex: 7
  EventType: EV_EFI_VARIABLE_DRIVER_CONFIG
  DigestCount: 4
  Digests:
  - AlgorithmId: sha1
    Digest: "d4fdd1f14d4041494deb8fc990c45343d2277d08"
  - AlgorithmId: sha256
    Digest: "ccfc4bb32888a345bc8aeadaba552b627d99348c767681ab3141f5b01e40a40e"
  - AlgorithmId: sha384
    Digest:
 "2cded0c6f453d4c6f59c5e14ec61abc6b018314540a2367cba326a52aa2b315ccc08ce68a816ce09c6ef2ac7e51
  - AlgorithmId: sha512
    Digest:
 "94a377e9002be6e1d8399bf7674d9eb4e931df34f48709fddd5e1493bfb96c19ee695387109a5a5b42f4871cbee
  EventSize: 53
  Event:
    VariableName: 61dfe48b-ca93-d211-aa0d-00e098032b8c
    UnicodeNameLength: 10
    VariableDataLength: 1
    UnicodeName: SecureBoot
    VariableData: "01"
```

6. UEFI BootOrder and Boot#### variables.

   If the UEFI `BootOrder` and `Boot####` variables are used by the firmware, they must be measured into PCR[1] with event types `EV_EFI_VARIABLE_BOOT` or `EV_EFI_VARIABLE_BOOT2`.
7. Verify boot attempt measurements

   Platform firmware must record each boot attempt into PCR[4] using the event type `EV_ACTION` with the action string `"Calling EFI Application from Boot Option"`.
8. Verify PCR[1] measurements.

   Measurements of security relevant configuration data go into PCR[1].

   This should include configuration data such as the security lifecycle state of a system.

   Security relevant SMBIOS structures must be measured into PCR[1] using event type `EV_EFI_HANDOFF_TABLES`. This should include structures that identify the platform hardware for example manufacturer, model number, version, and so on.

9. Verify `EV_SEPARATOR` measurements

   The `EV_SEPARATOR` event delineates the point in platform boot where the platform firmware relinquishes control of making measurements into the TPM. There must be an `EV_SEPARATOR` measurement for each PCR[0] through PCR[7].

4. Related information

Here are some resources related to material in this guide:

- Base Boot Security Requirements
- Arm Community
- Arm SystemReady Certification Program
- Arm SystemReady Requirements Specification
- Introduction to SystemReady

The following GitHub repositories are related to this guide:

- security-interface-extension branch in the arm-systemready repository

5. Next steps

In this guide you learned about the SystemReady Security Interface Extension certification and how to run the ACS tests to produce the test logs needed for a certification submission.

For further details about the certification process see the Arm SystemReady Requirements Specification.

For support, please send an email to `support-systemready-acs@arm.com`.