riscure

Corstone-1000 PSA Level 2 Ready

Prepared for	**Arm**
Classification	**Public**
Project number	**20200272**
Document ID	**20200272-D1**
Document type	**Evaluation Technical Report**
Document version	**1.1**
EAN	**N/A**
Date	**16 March 2022**

TESTING
RvA L 605

This security evaluation report shall not be disclosed to security consultants, test laboratories or other security expertise service providers, without explicit prior approval from Riscure.

This report shall not be reproduced except in full, without written approval from the laboratory.

Arm Limited

Address	**110 Fulbourn Road** **CB1 9NJ Cambridge** **United Kingdom**

Riscure

Address	**Delftechpark 49** **2628 XJ Delft** **The Netherlands**
Phone	**+31 15 251 4090**
Email	**inforequest@riscure.com**
Internet	**www.riscure.com**

Revision history

Version	Date	Description
1.1	15 March 2022	Various fixes based on Arm feedback
1.0	2 March 2022	First version released to Arm

Executive summary

Arm requested Riscure to perform a PSA Level 2 Ready (PSA L2R) evaluation of their Corstone-1000. It is a System-on-Chip (SOC) Intellectual Property (IP) which uses a multicore approach to create a separation between the Secure Processing Environment (SPE) and the Non-secure Processing Environment (NSPE), thereby protecting the SPE from remote attackers. The Corstone-1000 is evaluated while implemented on an Field-Programmable Gate Array (FPGA), targeting PSA L2R certification, aiding further developers in achieving full PSA L2 certification.

The purpose of this evaluation was to verify if the Target of Evaluation (TOE) satisfies the PSA L2R security and assurance requirements. To achieve this goal, a conformity analysis was performed to establish the conformance of the Security Target [1] with the PSA Certified Level 2 Lightweight Protection Profile [12]. Furthermore, a vulnerability analysis was performed following the attacker profile outlined in the PSA Certified Attack Method [13]. The analyses were performed in a white-box manner, i.e. with access to source code and design documents of the TOE. The evaluation was conducted between 8 December 2021 and 15 March 2022, at Riscure's premises in Delft, the Netherlands.

During the conformity analysis, we have verified that the TOE supports 8 of the 9 PSA L2R security functions.

F.AUDIT is not implemented, as the TOE is a resource constrained device. Under these conditions, the F.AUDIT security function is waived.

During the vulnerability analysis, no attack paths were identified, based on the potential vulnerabilities, which could violate the security functions, given the PSA L2R attacker model.

Given the outcome of the conformity and vulnerability analysis, we conclude that the TOE meets all the requirements of PSA L2R.

Contents

List of tables

List of figures

1 Introduction

This Evaluation Technical Report (ETR) describes the results of the PSA L2R security evaluation of the Corstone-1000. Full identification of the TOE is provided in Section 1.6.

1.1 Methodology

During security evaluations, Riscure uses the following internal methodologies:

- Security Evaluation Methodology [22]
- Methodology for System Analysis and Implementation Reviews [21]
- Methodology for Fault Injection [19]
- Methodology for Side Channel Analysis [20]

Riscure also uses the following PSA L2 evaluation methodology:

- PSA Certified™ Level 2 Evaluation Methodology Version 1.1 [11]

This project was performed in line with the ISO17025:2017 quality standard:

- The target of evaluation is an IT system.
- The activities performed were software security testing and evaluation of IT security products.

The approach used for this evaluation is specified in PSA Certified Evaluation Methodology (PSA-EM) [11]. The security evaluation is performed using a white-box approach.

The evaluation consisted of two main phases:

- Conformity analysis phase, where we checked conformance of the Security Target (ST) with PSA Certified Level 2 Lightweight Protection Profile (PSA-PP) [12]. In particular the ST shall include all assets, threats and security functions of the PSA-PP.
- Vulnerability analysis and testing phase, where we performed vulnerability analysis on the TOE. For identification of the classes of attacks that can be performed on we referred to PSA Certified Attack Method (PSA-AM) [13]. The knowledge obtained in the identification is used to mount attacks with the objective of breaching the security requirements and compromising assets. Since effort for the evaluation is limited by time, not every vulnerability can be tested. Therefore, a selection of vulnerabilities is made which will be tested.

1.2 Assets

The assets for this evaluation, as defined in PSA-PP [12], are:

- The integrity of SPE updateable code.
- The integrity and confidentiality of root secrets:
  - The Initial Attestation Key (IAK)
  - The Hardware Unique Key (HUK)
  - The Boot encryption key (if supported)
- The integrity of root parameters:
  - Instance ID
  - Boot validation key (Root of Trust Public Key (ROTPK))
- The integrity of the lifecycle state.
- The integrity and confidentiality of Application Root of Trust (ARoT) or the NSPE data managed by the TOE through PSA Root of Trust (PSARoT) secure storage services.

1.3 Threats

The threats considered for this evaluation, as defined in PSA-PP [12], are:

T.ROGUE_CODE – An attacker succeeds in loading and executing rogue code on the device in the SPE, and compromises the TOE assets.

T.FIRMWARE_ABUSE – An attacker exploits a flawed version of the PSARoT (including hardware), for instance by sending malformed parameters, and compromises the TOE assets.

T.UPDATE_ABUSE – An attacker exploits a flaw in the firmware update mechanisms of the TOE, for instance by sending malformed parameters, by altering an authentic firmware update, by installing an old version of the firmware or by bypassing security checks, and installs a flawed version of the PSA updateable root of trust.

T.STORAGE – An attacker succeeds in illegally modifying or accessing assets stored on the TOE, for instance by bypassing checks related to TOE lifecycle.

T.DEBUG – An attacker succeeds in accessing TOE debug features and illegally modifies or accesses TOE assets.

T.WEAK_CRYPTO – An attacker exploits flaws in the use or implementation of cryptographic algorithms in the TOE and illegally modifies or accesses TOE assets.

T.IMPERSONATION – An attacker manages to make remote entities recognize a rogue device under

its control as a valid TOE.

1.4 Identification of the scope

The TOE of an PSA L2R evaluation is the combination of the hardware and firmware components supporting a device compliant with PSA specification. The PSA platform components that are in the scope of the security evaluation, as described in PSA-PP [12], are:

- PSA updateable root of trust, such as software isolation framework, generic services such as binding, initial attestation, generic crypto services, firmware update validation.
- PSA immutable root of trust, for example boot ROM, root secrets and IDs, isolation hardware, security lifecycle management and enforcement. This component cannot be updated.
- Trusted subsystems used by the PSARoT, such as security subsystems, trusted peripherals, SIM or SE, which include both hardware and software components were also in scope of the evaluation.

1.5 Evaluation identification

RTE reference:	20200272-D1
RTE version:	1.1
EAN:	N/A
Authors:	Refer to project team on page **??**
Approved by:	Refer to revision history on page 3
Last update:	March 16, 2022

Table 1.1: Evaluation identification

1.5.1 Notes

None.

1.6 TOE identification

Company name:	Arm
Product name:	Corstone-1000
HW reference:	SSE710
HW version:	AN550_v1
SPE name:	TF-M
SPE version:	`TF-Mv1.5.0-332-gf8c7e5361b`

Table 1.2: TOE identification

The identification of the TOE is provided in Table 1.2. This report contains the results of the evaluation of this specific TOE. These results apply only to the version and configuration of the TOE that was evaluated.

The TOE software is a combination of various open source projects, which are combined into the TOE software using a Yocto build configuration, which can be primarily identified by the Git branch `CORSTONE1000-2022.02.18`. More detailed TOE identification is provided in Section 3.

The materials received (document packages, source files and samples) for this evaluation are listed in Appendix B and Appendix C.

2 Product description

The TOE is a reference implementation of a Secure Element (SE) inside an SOC, together with the software stack it executes, and the integration documentation [4, 6] for creating a final product. The TOE is implemented in an FPGA reference board, and therefore targets PSA L2R.

Final developers can use the reference to tweak to their own needs, and then submit it for a full PSA Level 2 (PSA L2) evaluation.

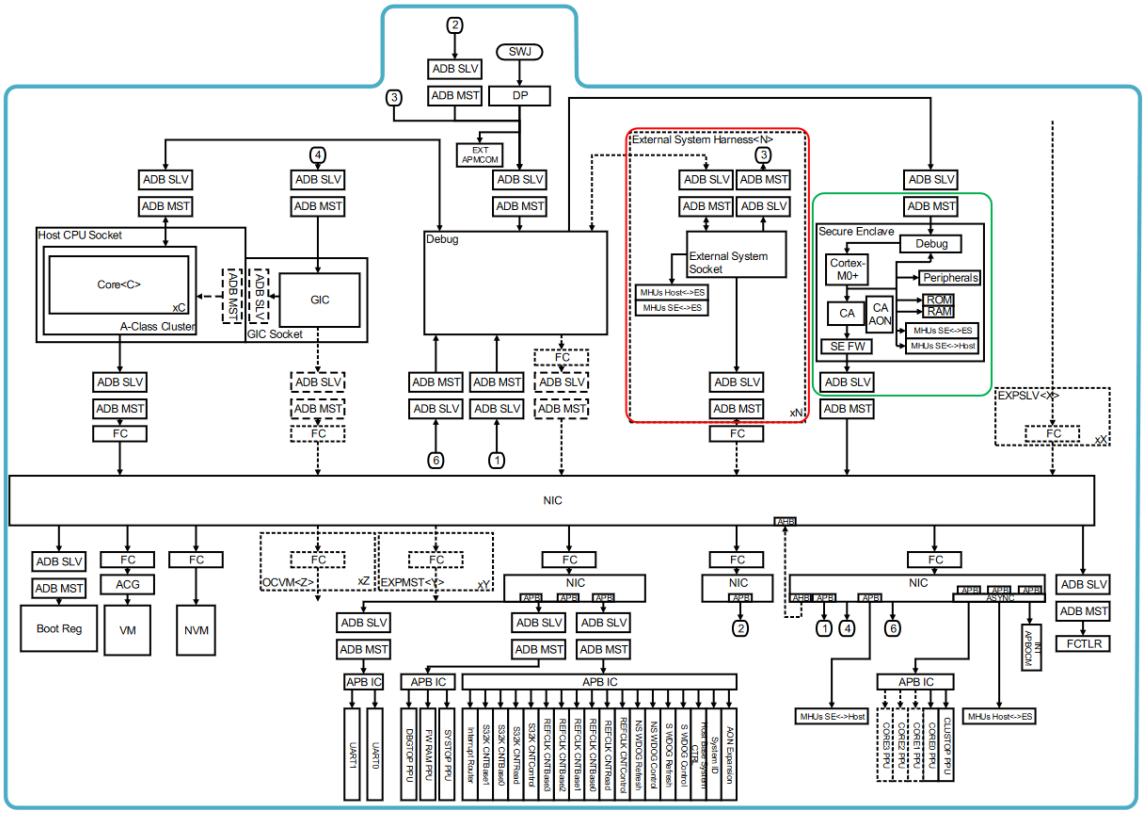

Figure 2.1: Hardware architecture overview of TOE [5]

Figure 2.1 shows the SE inside the full SOC, delimited by the green box, which is the hardware portion of the TOE. It is a combination of a Cortex-M0+ CPU with a private Read-Only Memory (ROM) and Static Random Access Memory (SRAM), Arm® CryptoCell™-312 (CC312) cryptographic accelerator, and various other peripherals. The main host CPU, used to implement the NSPE, is a Cortex-A35.

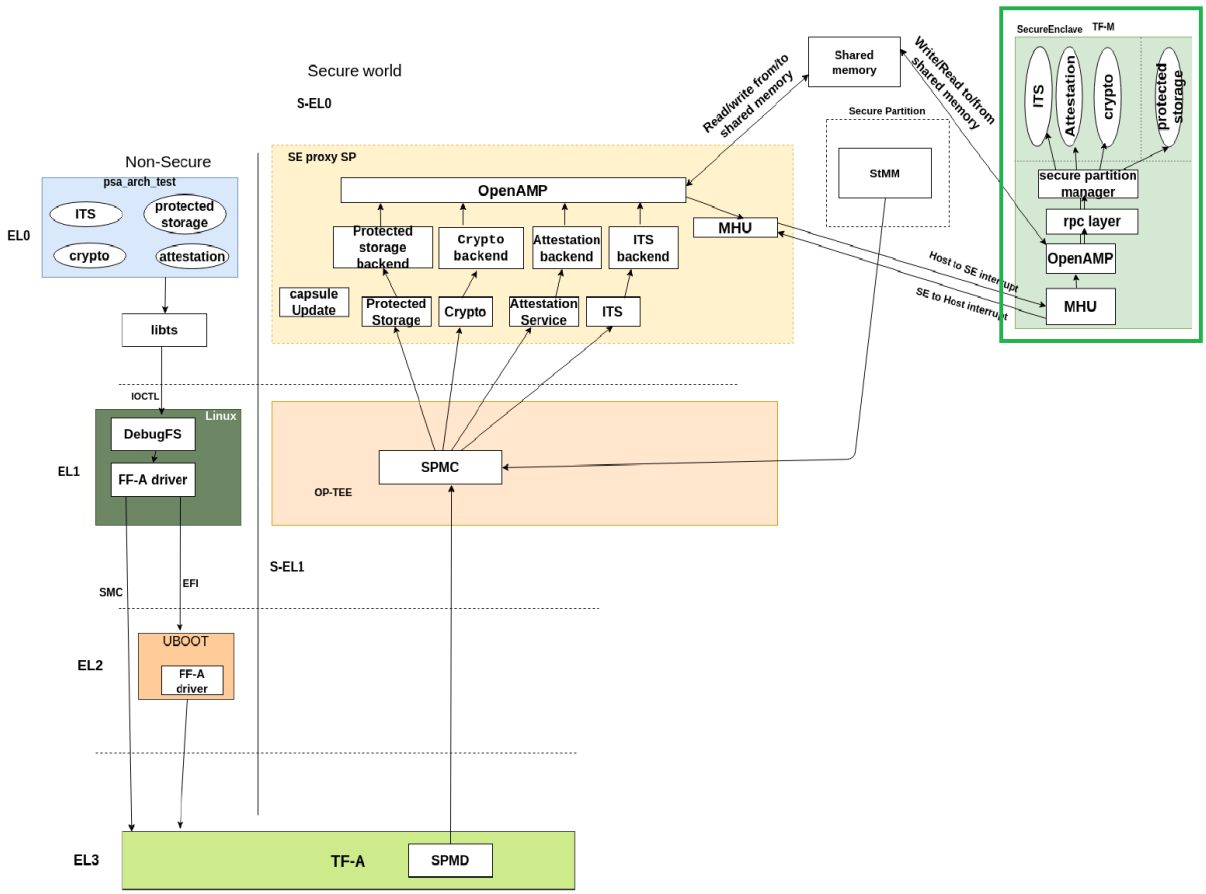

Figure 2.2: Software architecture overview of TOE [2]

Figure 2.2 shows an overview of the runtime software architecture of the SOC, where the green box on the top right delimits the software portion of the TOE. The regular interface into the TOE is a combination of the Message Handling Unit (MHU) and shared data residing in external Dynamic Random Access Memory (DRAM).

2.1 Intended usage

The Corstone-1000 MPS3 based board is a development kit built around the Corstone-SSE710 subsystem (the TOE), designed to enable developers to prototype designs and develop software. It is expected to be used within a development environment with controlled access to the physical location and the development environment. It is expected that once a developer has a working implementation, they would then design custom hardware with the specific extensions, ports and peripherals needed for their use case.

2.2 Operational environment

The TOE operational environment includes:

- Other Applications Root of Trust or other Applications executed in the SPE environment.
- Alternate OS and applications executed in the NSPE.
- Remote entities (for instance remote servers, administrators, users), in charge of personalizing the TOE, managing firmware update or interacting with the TOE.

The following assumptions hold on the operational environment of the TOE:

- Cryptographic keys and certificates outside of the TOE are subject to secure key management procedures.
- Actors in charge of TOE management, for instance for signature of firmware update, are trusted.
- Each TOE has a unique identifier, provisioned during manufacturing.
- Integrity of immutable code is ensured by hardware mechanism such as ROM, or EEPROM or FLASH memory that is locked before device delivery.
- The CC312 One Time Programmable (OTP) memory must be implemented as a true One Time Programmable memory module.
- The SE ROM must be replaced with an immutable ASIC ROM implementation.
- The NVM and Flash memories should be implemented as on-chip Flash memory. If this is not possible and an off-chip Flash device is used, SPE functionality must be used to encrypt any data written from the Secure Enclave to this memory, and to validate any data read from it. In that case the Protected Storage API shall be used, rather than the Internal Trusted Storage API.

2.3 Hardware and software interfaces

This section describes the interfaces that are present on the TOE which were in scope for this evaluation. These interfaces provide an attacker with methods for communicating with the TOE and have been identified by analysis of documentation.

Interface	Description	Notes
MHU interface	Hardware Interface between NSPE and SPE.	The software stack builds on top of this.

Interface	Description	Notes
Debug interface	JTAG via SWD connection.	This connection is used to debug the SPE during development lifecycle state. In the evaluated configuration, this interface is locked not accessible.
Host access region	Region between address `0x6000_0000` to `0xE000_0000` to access peripherals in the rest of the SOC	This includes the DRAM slot, the flash interface, and several other configuration registers. The SE can use the firewall peripheral to moderate access of other bus masters to this region.
OpenAMP library	Asynchronous communication library between the NSPE and the SPE.	First software interface layer running on top of the MHU hardware.
PSA functional APIs	Applications executing in the NSPE communicate with the SPE through APIs.	Second software interface layer running on top of the OpenAMP layer.

Table 2.1: Interfaces on the production TOE

2.4 Security functions

In order to mitigate the identified threats (see Section 1.3), Arm claims the following security functions in [1], as per PSA-PP:

- F.INITIALIZATION
- F.SOFTWARE_ISOLATION
- F.SECURE_STORAGE
- F.FIRMWARE_UPDATE
- F.SECURE_STATE
- F.CRYPTO
- F.ATTESTATION
- F.DEBUG

The following PSA L2R security functions are not claimed:

- F.AUDIT

In the remainder of this chapter we describe the implementation of the claimed security functions, based on delivered documentation, source code, and the ST [1].

F.INITIALIZATION

Goal

The TOE is started through a secure initialization process that ensures the authenticity and integrity of the firmware.

This security function mitigates T.ROGUE_CODE by preventing the installation firmware or piece of firmware code from unknown sources.

Implementation

The implementation of F.INITIALIZATION is described in the following documents: [2]. It is supported by online documentation of the `mcuboot` it is built on top of: [16].

The software boot chain consists of three components:

- BL1
- BL2
- TF-M

Figure 2.3 shows a diagram of the boot flow of the entire chip (including the NSPE).

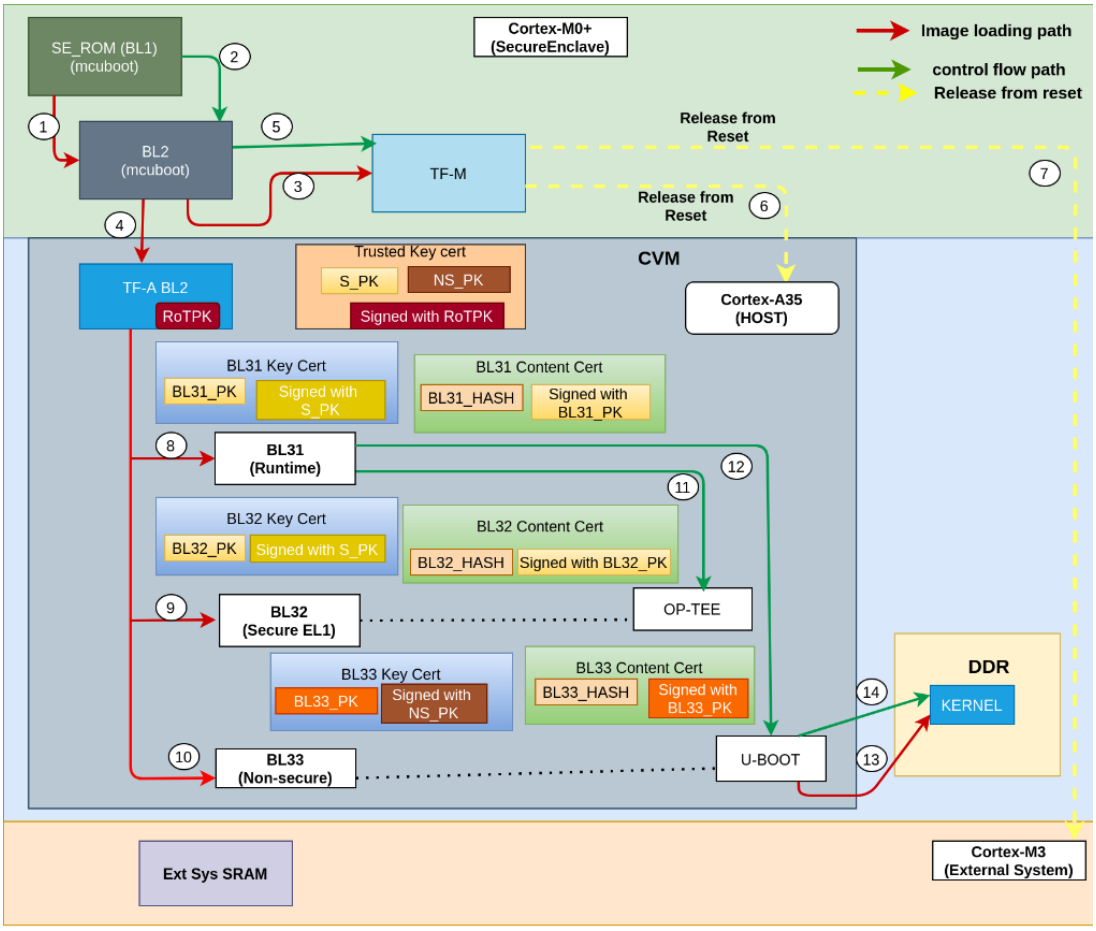

Figure 2.3: Boot flow overview of TOE [2]

The SE is the first component released from reset; the rest of the components will be kept in reset until the SE is done booting and releases them from reset.

BL1 is the first bootloader executed by the SE. It is stored in and executed from ROM which is only accessible to the SE. It moves the next bootloader, BL2, into its internal SRAM, also only accessible to the SE. It then first validates the ROTPK (RSA-2048 public key) through the SHA256 hash stored in the CC312 OTP, and then validates the next bootloader, BL2.

BL2, executed from internal SRAM, performs similar operations, but to set up the runtime SPE code, based TF-M. It also loads and verifies the first image for the NSPE, the “NSPE-BL2”. Furthermore, it supports signatures with three different ROTPKs, each of which again has the hash stored in OTP.

Once the SE is ready, it will release the NSPE from reset so that it can resume its own boot process, starting from a “NSPE-BL2”.

The bootloader is based on the open source project MCUBOOT. It works through a combination of Tag Length Value Records (TLVs) and corresponding data to perform the boot. Depending on compilation options, signatures and anti-rollback are checked for the TLV.

- `MCUBOOT_HW_ROLLBACK_PROT` controls whether a provided version field is valid against an OTP stored version. It is enabled for both BL1 loading BL2, and BL2 loading TF-M.
- `MCUBOOT_HW_KEY` controls whether the verification key is loaded from hardware backed storage. It is enabled for both BL1 loading BL2, and BL2 loading TF-M.
- `EXPECTED_SIG_TLV` enables the signature checks over the TLVs. This is enabled for both BL1 loading BL2, and BL2 loading TF-M.

Since this is a PSA L2R evaluation performed on an FPGA implementation, the TOE demonstrated that it has meet the PSA L2R implementation by emulation. The following components that are essential for the immutable PSARoT meets the requirements for the following reasons:

- ROM used for the first bootloader is emulated using Random Access Memory (RAM) set to be read only.
- OTP used to store information such as the hash of the ROTPK and anti-rollback counters is emulated using RAM as well. Additional logic is implemented such that the RAM section is only allowed to be written once and not cleared during TOE reset.

F.SOFTWARE_ISOLATION

Goal

The TOE provides isolation between the NSPE and the SPE and also between PSA Root of Trust and other executable code (such as Application Root of Trust) of the SPE.

This security function mitigates T.ROGUE_CODE by preventing software outside of the TOE from tampering with TOE assets.

Implementation

The implementation of F.SOFTWARE_ISOLATION is described in the following documents: [5, 2].

The main isolation between the SPE and NSPE is implemented by physically isolating the SPE from the rest of the SOC into a component referred to as the SE. It is positioned inside the rest of the SOC as shown in Figure 2.1.

The SE has its own bus, with its own private peripherals not exposed to the rest of the SOC. The primary interface to the SE is a mailbox peripheral, the MHU. Two instances are present; one for incoming and one for outgoing communication.

In addition to this peripheral, the SE and the rest of the SOC can communicate via a shared portion of the address space, ranging from `0×6000_0000` to `0×E000_0000`. The first 0.5GB of this memory range maps to the ”Host Master Expansion Region”, which contains peripherals such as external non-volatile memory control, and other SE to host (i.e. rest of the SOC) registers. The other 1.5GB of this space map to the external volatile memory (between `0×8000_0000` and `0×E000_0000`). In the TOE, this region is used to, among other things, allow the SE to access the flash peripheral directly.

To protect access within this shared address space from other bus masters in the SOC, the TOE configures access rights to allow only its own bus master ID access to these peripherals during BL1 using the “firewall” peripheral. The regions can be configured and locked. At the end of the setup, the region configuration itself can be locked, and they cannot be altered until reset. In the TOE all of the above is done during BL1.

The final interface is the debug interface, which is covered in F.DEBUG.

F.SECURE_STORAGE

Goal

The TOE protects the confidentiality and integrity of assets in a secure storage. The secure storage is bound to the platform. Only the TOE can retrieve and modify assets from this secure storage.

This security function mitigates T.STORAGE by preventing direct and unprotected access to assets.

Implementation

The implementation of F.SECURE_STORAGE is described in the following documents: [8, 2]. It is supported by online documentation of the `trusted-firmware-m` documentation which it utilizes: [15, 18].

The key assets of the TOE, i.e. the 256-bit HUK, (SHA256 hash of the) ROTPK, and IAK, are intended to be stored using the OTP memory of the cryptographic engine, the CC312.

- The hash of the ROTPK is read out and used during the boot process.
- The IAK is read out and stored in SPE local SRAM when attestation tokens are created, and cleared from memory when the request is fulfilled, or results in an error.
- The SPE contains code that allows reading out the HUK; however it is never used. Furthermore, according to [8] and [1], it cannot be read out directly; it can only be used for derivation operations.

The SPE runtime control is authenticated by using the hash-locked ROTPK in OTP, and the fact that the first boot code is stored in ROM. Further loaded code is verified and loaded by the aforementioned primitives, and stored in internal SRAM only accessible to the SPE.

Dynamic data assets are to be stored in internal flash, and managed via the protected storage functionality, implemented in the runtime SPE firmware.

The current TOE is implemented on an FPGA, without some of the storage primitives being available. The TOE meets the requirement of PSA L2R for the following reasons:

- OTP storage is emulated using RAM.
- ROM storage is emulated using RAM.
- Internal flash storage is emulated using external flash.

F.FIRMWARE_UPDATE

Goal

The TOE verifies the integrity and authenticity of the TOE update prior to performing the update. The TOE also rejects attempts of firmware downgrade.

This security function mitigates T.UPDATE_ABUSE by preventing installation of firmware from unknown sources or installation of obsolete firmware.

Implementation

The implementation of F.FIRMWARE_UPDATE is described in the following documents: [2]. It is supported by online documentation of the `mcuboot` it is built on top of: [16].

The update binary is provided by the NSPE in the DDR. The NSPE will signal the TF-M to copy the update binary from the DDR to flash. TF-M also verified the version of the incoming binary against the one currently running to mitigate against rollback attack. Once the copy has been successful, the TOE will reset. Upon the next reset, the image will be authenticated by BL1 ROM. The hash of the 2048-bit RSA public key stored in the update binary is compared against the SHA256 stored in the OTP.

The TOE supports the concept of trial run. This is to ensure that the firmware update can properly boot up until the uboot of the NSPE is running. Once the uboot is running, TOE FW will receive a message stating that the TOE boot is complete. Only after this stage that the TOE indicates that the update is successful. This is done to make sure that there is always a firmware present to prevent device bricking.

The current TOE is implemented on an FPGA, without some of the storage primitives being available. The TOE meets the requirement of PSA L2R for the following reasons:

- OTP storage is emulated using RAM.
- ROM storage is emulated using RAM
- Internal flash storage is emulated using external flash.

F.SECURE_STATE

Goal

The TOE ensures the correct operation of its security functions. In particular, the TOE:

- Protects itself against abnormal situations caused by programmer errors or violation of good practices from code executed outside of the TOE, either from SPE or NSPE.
- Controls the access to its services by Applications and checks the validity of parameters of any operation requested from Applications.
- Enters a secure state upon platform initialization error or software failure detection, without exposure of any sensitive data.

This security function mitigates T.FIRMWARE_ABUSE by preventing exploitation of abnormal situations.

Implementation

The implementation of F.SECURE_STATE is not described in any TOE specific documentation. The general design upon which it is built is described in [9].

The secure state is ensured at a hardware level by physically separating the SE (used to implement the TOE) from the rest of the SOC. This means that if software fails to configure something correctly, internal assets can not be accessed, unless explicitly exported via one of the interfaces.

Furthermore, coding practices are employed to catch illegal states, and clear relevant data if such a state is encountered. Additionally, a system reset is implemented which gets triggered when certain illegal states are detected.

F.CRYPTO

Goal

The TOE implements state-of-the-art cryptographic algorithms and key sizes for protection of TOE assets. Weak cryptographic algorithms or key sizes may be available for specific usages and with specific guidance, but they shall not reduce security of provided state-of-the-art cryptography.

This security function mitigates T.WEAK_CRYPTO by preventing exploitation of cryptographic weaknesses to target TOE assets.

Implementation

The implementation of F.CRYPTO is described in the following documents: [8].

The cryptographic functionality is implemented using `mbedlts` as a wrapper around the CC312 hardware IP block.

The claimed cryptographic primitives all offer security equivalent to or higher than 100 bits of security:

- Symmetric algorithms (AES, ChaCha20) use key sizes of 256 bits. AES can also be used with 128-bit keys.
- RSA based cryptography uses key sizes of 2048 bits or higher.
- ECC based crypotography uses curve points of 224 bits or higher.

These sizes are sufficient to withstand state-of-the-art cryptographic attacks.

The CC312 implements an Random Number Generator (RNG) that passes the AIS-31 test suite and uses a NIST SP 800-90A compliant AES based DRGB, to generate 256 bits at a time.

F.ATTESTATION

Goal

The TOE provides an attestation service which reports on the device identity, firmware measurements and runtime state of the device. The attestation can be verified by remote entities.

This security function mitigates T.IMPERSONATION by providing a cryptographic proof of identity.

Implementation

The implementation of F.ATTESTATION is not described in any TOE specific documentation. The general design upon which it is built is described in [10].

The TOE uses the standard Platform Security Architecture (PSA) attestation setup, rooted in a 256-bit IAK that is stored in OTP, only accessible to the TOE. This token includes an instance ID (which is the SHA256 hash of a public attestation key), lifecycle state and further implementation details.

Since this is a PSA L2R evaluation performed on an FPGA implementation, emulation of the storage primitive (OTP) is sufficient to meet the PSA L2R requirement.

F.DEBUG

Goal

The TOE restricts access to debug features by deactivation or access control mechanism with the same level of security assurance as other security functions of this PSA-PP.

This security function mitigates T.DEBUG by preventing unauthorized access to debug features.

Implementation

The implementation of F.DEBUG is described in the following documents: [5].

In its current state, the TOE does not support debug access. Configuration bits are accessible to the TOE itself to enable debug access, but there is not functionality implemented that uses these bits. By default, these bits do not allow access to the debug interface.

An authentication scheme to enable access to the debug port is present in the source code of the SPE, but it is not compiled into the firmware via the `PLATFORM_PSA_ADAC_SECURE_DEBUG` flag.

3 TOE installation

TOE installation is described in the following guidance documents: [3, 7]. Specific instructions on how to obtain and build just the TOE software were provided via customer meetings.

The TOE is a combination of an SOC design provided as a bitstream, a software stack running on top of that SOC, and a set of integration guidance documentation. The bitstream is executed on an FPGA, which simulates the SOC, which is in turn implemented on a development board with various controllers to load the bitstream and software from an external storage device. Note that the board and FPGA itself are not part of the TOE.

The FPGA used to run the current TOE is the “Xilinx Kintex Ultrascale KU115” FPGA. The full board is commonly referred to as the “ARM MPS3” FPGA board, the specific version of the board used for the evaluation is identified as “HBI0309C”. The specific version is printed on the board for identification.

3.1 Hardware

The FPGA loads a bitstream using the board’s controller, which exposes a storage device over USB to upload both the bitstream to be used and the software it should run. The bitstream used to implemented the TOE is identified as “AN550_v1”. The SHA256 hash of this bitstream is `7178fcc698c9c40aab8e3b8ddc4967f27bd88feb904b931c9ada4d5fb08cb810`. The storage device exposed by the board can be used to verify that the right bitstream is present.

3.2 Software

Component	Version	Git identifier
Yocto configuration	`CORSTONE1000-2022.02.18`	`3.4-128-g43e5b03`
`mcuboot`	`v1.8.0`	`29099e1d17`
`mbedtls`	`v3.1.0`	`d65aeb3734`
`trusted-firmware-m`	`v1.5.0`	`TF-Mv1.5.0-332-gf8c7e5361b`
`open-amp`	`v2020.10.0`	`v2020.10.0-15-g347397d`

Component	Version	Git identifier

Table 3.1: Final TOE software versions

Table 3.1 summarizes the final versions of the software components that are part of the scope. These are the versions found as a Git tag, or Git branch, accompanied by the Git commit ID or the result of `git describe` between brackets. In this final version, a number of failing functional tests were resolved with regards to the cryptographic API.

The software of the TOE is a combination of various open source software projects. Collecting the right versions of each of those components, and their build tools, is done via a Yocto project. We obtained the TOE software with the following steps:

```
git clone https://git.yoctoproject.org/git/meta-arm -b CORSTONE1000-2022.02.18
```

The above command gets the configuration of the TOE versions as listed in Table 3.1. The matching version information of the Yocto configuration can be obtained with `git describe` in the downloaded folder. Then, using configuration file `meta-arm/kas/corstone1000-mps3.yml`, we obtained the TOE source code using the commands:

```
kas shell meta-arm/kas/corstone1000-mps3.yml
```

```
devtool modify trusted-firmware-m
```

This results in the software components in a local directory `build/workspace/sources/trusted-firmware-m`, with the software listed in Table 3.1. The specific Git commit IDs of the TOE software is also listed in `meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-m/trusted-firmware-m-corstone1000.inc`. The OpenAMP library is not included in this collection of sources; it is pulled in during the build step mentioned below, and can the be found in `build/tmp/work/corstone1000_mps3-poky-linux-musl/trusted-firmware-m/1.5+gitf8c7e5361b92b16108165601ea81c5d01feb3c22-r0/build/lib/ext/libopenamp-src`.

BL1 and BL2 are a combination of the repository of `mcuboot` and platform specific code part of the `trusted-firmware-m` repository. The runtime SPE code is a combination of `trusted-firmware-m`, `mbedtls` and the `open-amp` library.

The version of the `trusted-firmware-m` running on the TOE hardware can be verified during boot

via the boot messages produced by the M0 UART, which reports the versions present in the loaded image:

```
[INF] Primary   slot: version=1.5.0+0
[INF] Secondary slot: version=1.5.0+0
```

The version of other components cannot be verified.

Component	SHA256 hash
bl1.elf	2dd4e9655f8887ed9d0410a84abde2ba09fc6de4a237d4b9df98295ce08043f
bl2.elf	a43c69e582082338a2f380b5dfb680fcf5b60d5c7f7982854599aeb526c38df6
tfm_s.elf	d9be732b5f3016ff385871c4af0133becdadda68132a902ba81727ff4b22e63b

Table 3.2: TOE binary hashes

To build the SPE code (BL1, BL2 and TF-M), the following command is used:

```
bitbake trusted-firmware-m
```

The resulting binaries can be found in `build/tmp/work/corstone1000_mps3-poky-linux-musl/trusted-firmware-m/1.5+gitf8c7e5361b92b16108165601ea81c5d01feb3c22-r0/build`. The hash of each binary is listed in Table 3.2.

Component	SHA256 hash
bl1.bin	bbcd4a7e6788e3a1009c2340644813cfb6528f14c1977cee9f04a13f8c69613b
cs1000.bin	a480a5bc8d8271b2312968819dce134f38b0b32a7c625fef9f710423e2146e8c

Table 3.3: TOE binary hashes

In the evaluated TOE configuration, all binaries except BL1 get combined into a single binary. This includes the non-TOE binaries. The hashes of these components are listed in Table 3.3. They are provided to the TOE via the exposed storage device similarly to how the bitstream of is uploaded, and their presence can also be verified in this storage device.

4 Conformity analysis

In this chapter we present our analysis of the conformity of the ST [1] provided by Arm with the PSA-PP. The analysis is based on provided documentation (Section 4.1), source code (Section 4.2) and manual verification testing (Section 4.3).

The conformity analysis is summarized in Table 4.1. It shows the summary of the activities performed for each of the security functions claimed and our pass judgment on the conformity of the security function.

Security function	Document review	Code review	Covered by functional testing	Pass
F.INITIALIZATION	Yes	Yes	No	Yes
F.SOFTWARE_ISOLATION	Yes	Yes	Yes	Yes
F.SECURE_STORAGE	Yes	Yes	Yes	Yes
F.FIRMWARE_UPDATE	Yes	Yes	No	Yes
F.SECURE_STATE	Yes	Yes	Yes	Yes
F.CRYPTO	Yes	Yes	Yes	Yes
F.ATTESTATION	Yes	Yes	Yes	Yes
F.AUDIT	No	No	No	N/A
F.DEBUG	Yes	Yes	Yes	Yes

Table 4.1: Conformity analysis summary

The following security functions rely on storage primitives which are not available in the TOE. However, since the evaluation is a PSA L2R, this does not hinder conformity of those security functions:

- **F.INITIALIZATION** – We found that the software implementation and hardware design conform to the security function. However, since the TOE is a reference implementation in an FPGA, silicon

vendor shall support the immutable storage primitives (ROM, OTP, internal flash), which are needed to ensure authenticity and integrity of the firmware from the very beginning.

- **F.SECURE_STORAGE** – Similarly to F.INITIALIZATION, we found that the the software implementation and hardware design conform to the security function. However, this security function also heavily relies on the immutable and internal storage primitives which are not yet available in the TOE. Silicon vendor shall integrate the immutable and internal storage to protect the SPE key assets integrity, as well as the confidentiality of the HUK and IAK in particular.
- **F.ATTESTATION** – Similarly to F.INITIALIZATION and F.SECURE_STORAGE, we found that the the software implementation and hardware design conform to the security function. However, without the protection of the IAK, this security function cannot yet be met. This shall be provided by the silicon vendor.
- **F.FIRMWARE_UPDATE** – Similar to the previously discussed security functions, we found that the the software implementation and hardware design conform to the security function. However, without the underlying immutable storage primitives, it is not possible to fully conform to PSA L2. Silicon vendor shall provide the immutable storage as well as the internal storage to claim full conformance.

For a full PSA L2, building on top of the current TOE, these conformity issues must be addressed.

Furthermore, the F.AUDIT security function is not claimed. PSA-PP states that this security function is optional for resource constrained devices. In [1], Arm states that the TOE is resource constrained, and therefore does not implement this security function. This is in line with the PSA-PP, and does not hinder conformity of the TOE.

4.1 Documentation

We analyzed the following design documentation, to verify the claims of the ST: [5, 2, 3, 8].

We found that the documentation is consistent with the claims made in [1]. It provides sufficient evidence for all 8 claimed security functions, to determine that 8 security functions conform to the protection profile, and 0 security functions do not yet conform to the protection profile. A summary of the implementation of these security functions, aided by the documentation, is provided in Section 2.4.

4.2 Source code

We reviewed critical parts of the `mcuboot` and `trusted-firmware-m` implementation (versions as listed in Table 3.1), with a focus on the parts specifically added for the TOE, for conformance with the claims in the ST.

The TOE is isolated from the rest of the SOC in the form of a SE, with an isolated processing unit and isolated storage components, which inherently protects the TOE from many attacks, as the rest of the SOC cannot access these components. Data and parameters which are provided externally through the mailbox interface and shared memory interface are copied to local memory, checked and sanitized. Critical checks, such as verification of firmware during boot, are implemented properly.

During the review we found that the shared external memory, used to implement the communication between the NSPE and SPE, uses the `open-amp` library to communicate the low level logical part on both sides. This library in isolation cannot be used to properly protect data from being modified after it is being checked. This increases the attack surface, and in particular opens up the possibility for Time Of Check Time Of Use (TOCTOU) type attacks, for example (as described in V-016). However, further analysis into the details of the implementation did not show that an attack path is present which would compromise the TOE assets, and therefore does not affect the conformity claim of F.SOFTWARE_ISOLATION and F.SECURE_STATE.

4.3 Manual verification

As manual verification activities, we performed the TOE software build steps, hardware and software installation steps, verified that the boot process works as intended, and that the PSA functional tests can be run.

Based on the boot logs, we could confirm that the SPE is verified and brought up correctly.

Based on running the PSA functional test suite from the NSPE, we could confirm the following:

- The 1 attestation test passes successfully.
- The 10 internal storage tests passed successfully.
- 17 of the protected storage tests passed successfully.
    - 11 of the protected storage tests were skipped, as they are optional, and do not impact conformance of F.SECURE_STORAGE.
- 59 of the cryptography tests passed.
    - 4 of the cryptography tests did not pass in the originally provided SPE. After a second iteration (see also **??**), 4 of the cryptographic tests did not pass.
        - The test suite includes a check for hash operation suspend and resume, accounting for 2 of the failed test cases. This functionality is not implemented, and not claimed, and hence does not impact conformance of F.CRYPTO.
        - The other 2 failing cases are related to the fact that there is a mismatch between error codes that are returned by `mbedtls`, and expected by `trusted-firmware-m`. This is

acknowledged and explained in [1].

These tests also support the claim to F.SECURE_STATE.

Furthermore, Arm demonstrated that in the TOE configuration of the firmware, the debug interface is not exposed. They also demonstrated how using a build of the SPE software with the `PLATFORM_PSA_ADAC_SECURE_DEBUG` flag enabled, the debug interface can be used.

5 Vulnerability analysis and testing

In this chapter we present the known (Section 5.2) and potential (Section 5.3) vulnerabilities considered while analyzing the TOE. The analysis is based on documentation and source code provided by Arm (see also Section 4.1 and Section 4.2), and on publicly known information.

For each potential vulnerability we provide a description, the effect the vulnerability can have on the security functions and the impact on the TOE. Vulnerabilities that were considered applicable and practical include a reference to one or more attack paths, which are separately listed.

Attack paths that were not confirmed by Arm and not resolved, required testing and scoring.

5.1 Overview

In total, 16 potential vulnerabilities were considered during the vulnerability analysis. Of these potential vulnerabilities 0 were found applicable and practical.

The vulnerability analysis is summarized in Table 5.1. It lists for each of the vulnerabilities whether it was found applicable, and practical if applicable. If a vulnerability was found practical and applicable, one or more attack paths were formulated and listed in the table. Furthermore, for various vulnerabilities, tests were performed and listed in the final column. These can be verification tests, to verify whether the conclusions on applicability and practicality were accurate, or vulnerability tests, to determine the attacker potential required to successfully exploit the attack path.

Ref.	Title	Appli-cable	Prac-tical	Attack paths	Tests
V-001	Data injection during secure boot	No	-	-	-
V-002	Brute force of the firmware authentication key	Yes	No	-	-
V-003	Data injection during firmware update	No	-	-	-
V-004	Data injection during runtime mailbox handling	No	-	-	-
V-005	Incorrectly configured debug port security	No	-	-	-
V-006	Brute force of the debug access authentication key	No	-	-	-

Ref.	Title	Appli-cable	Prac-tical	Attack paths	Tests
V-007	Replay of captured attestation token	No	-	-	-
V-008	Data exfiltration through unprotected peripherals	No	-	-	-
V-009	Incorrect configuration of the RNG	No	-	-	-
V-010	Physical probing to extract SPE	No	-	-	-
V-011	Side channel analysis of cryptographic operations	No	-	-	-
V-012	Fault injection to bypass secure boot checks	No	-	-	-
V-013	Fault injection to bypass firmware update checks	No	-	-	-
V-014	Fault injection to bypass secure debug access checks	No	-	-	-
V-015	Fault injection on cryptographic operations	No	-	-	-
V-016	TOCTOU in shared memory	No	-	-	-

Table 5.1: Vulnerability analysis summary

The attack paths are individually listed in Table 5.2. It lists for each attack path if it was resolved or not during the course of the security evaluation of the TOE. For tested attack paths a JIL rating is given, based on the evidence obtained from those tests.

Ref.	Title	Resolved	JIL rating
No TOE specific attack paths were identified			

Table 5.2: Attack path summary

Table 5.3 lists each of the performed tests individually.

Ref.	Title

No TOE specific vulnerability tests were performed

Table 5.3: Test summary

5.2 Known vulnerabilities

We consulted public vulnerability databases [14, 17], Google and GitHub on 2 February 2022, for public vulnerabilities related to the TOE or its family. Table 5.4 lists per component the results. Table 3.1 lists the versions of each component. In summary, no publicly known vulnerabilities were found.

TOE component	Public vulnerability
`mbedtls`	No publicly available vulnerability. This is the latest `mbedtls` version.
`mcuboot`	`mcuboot v1.9.0` is coming soon. Based on the release notes of mcuboot 1.9.0, there are no security fixes compared to the current version.
`trusted-firmware-m`	No publicly available vulnerability. Note that the version used in the TOE is not the latest TF-M version, since RC2 and the full v1.5.0 is available.
`open-amp`	No publicly available vulnerability. This is the latest OpenAMP version.

Table 5.4: Public vulnerability

5.3 Potential vulnerabilities

V-001 Data injection during secure boot

Secure boot relies on data stored external to the TOE to be loaded and executed. If this data is not checked and sanitized properly, this can transiently or permanently compromise the TOE assets.

Effect of attack

Data injection during the secure boot process can lead to T.ROGUE_CODE. This could violate F.INITIALIZATION or F.SOFTWARE_ISOLATION. This could be the result of, for example, writing to a secure section of volatile memory due to a wrong controller configuration, or by altering external volatile

memory contents offline.

Impact on TOE

The TOE performs authentication checks to ensure that only signed code can be loaded. We sampled critical parts of the boot implementation, and found that these checks are implemented correctly. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-002 Brute force of the firmware authentication key

The SPE relies on proper authentication of the mutable code detect malicious modifications, to protect TOE assets. Weak authentication exposes these assets to the NSPE.

Effect of attack

A weak key used to authenticate access to SPE debugging capabilities could be guessed within reasonable time and can lead to T.ROGUE_CODE. This could violate F.INITIALIZATION or F.SOFTWARE_ISOLATION. For example, a 32-bit value could be guessed within reasonable time. Furthermore, if this fixed value is re-used across all devices, it only needs to be extracted once to compromise assets of all TOEs in the field.

Impact on TOE

The TOE relies on SHA256 and RSA-2048 to verify its externally loaded code. Breaking this scheme through brute force is not practical with current state of the art brute force capabilities. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-003 Data injection during firmware update

The firmware update mechanism relies on data provided externally through the mailbox interface. If this data is not checked and sanitized properly, this can permanently compromise the TOE assets.

Effect of attack

Data injection during the firmware update process can lead to T.UPDATE_ABUSE or T.ROGUE_CODE. This could violate F.INITIALIZATION, F.SOFTWARE_ISOLATION or F.FIRMWARE_UPDATE. This could be the result of, for example, loading firmware that is not verified beforehand, or not implementing a version checking mechanism that allows downgrading to a previous version.

Impact on TOE

The TOE performs authentication checks to ensure that only signed code can be loaded. We sampled critical parts of the update and boot implementation, and found that these checks are implemented correctly. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-004 Data injection during runtime mailbox handling

The primary interface between the NSPE and SPE is implemented as a mailbox interface. Illegal or unexpected parameters could trigger unexpected behavior that compromises the TOE assets.

Effect of attack

Data injection on the runtime communication mechanism between the NSPE and SPE or ARoT and SPE can lead to a variety of threats:

- Unchecked cryptographic input parameters could lead to using keys with a weaker cryptographic algorithm. This can lead to T.WEAK_CRYPTO.
- Unchecked destination address pointers can lead to writing data to unintended locations. This can lead to T.ROGUE_CODE and T.FIRMWARE_ABUSE.
- Unchecked input parameters can lead to processing unexpected data sizes which do not fit in local buffers. This can lead to T.ROGUE_CODE and T.FIRMWARE_ABUSE.
- Unchecked source address pointers can lead to reading data from unintended locations. This can lead to T.STORAGE.
- Access to unintended commands could lead to opening up previously locked debug interfaces. This can lead to T.DEBUG.

This in turn could violate F.INITIALIZATION, F.SOFTWARE_ISOLATION, F.SECURE_STORAGE, F.SECURE_STATE, F.CRYPTO or F.DEBUG.

Impact on TOE

We sampled critical parts of the SPE and NSPE interface, and found that these checks are implemented correctly to ensure correct input is provided. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-005 Incorrectly configured debug port security

The SPE relies on proper protection of its debug interface to protect TOE assets. Weak protection exposes these assets to the NSPE.

Effect of attack

Insufficient protection of SPE debugging capabilities can lead to T.DEBUG. This could violate F.DEBUG. For example, the locking mechanism might be present, but not enabled in the field be implemented through a register accessible to the NSPE. Simply writing the unlock value to this register could open the debugging capabilities.

Impact on TOE

Based on the documentation and source code review, we concluded that the debug port is configured correctly. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-006 Brute force of the debug access authentication key

The SPE relies on proper authentication of the debugging party before allowing debug access to protect TOE assets. Weak authentication exposes these assets to the NSPE.

Effect of attack

A weak access key used to authenticate access to SPE debugging capabilities could be guessed within reasonable time and can lead to T.DEBUG. This could violate F.DEBUG. For example, a 32-bit value could be guessed within reasonable time. Furthermore, if this fixed value is re-used across all devices, it only needs to be extracted once to compromise assets of all TOEs in the field.

Impact on TOE

Re-enabling debug access to the TOE via authentication is not (yet) implemented, and the debug interface is fully closed. Therefore this vulnerability is not applicable.

V-007 Replay of captured attestation token

Attestation tokens could be captured and replayed to maliciously attest as a particular TOE instance.

Effect of attack

Captured attestation tokens could be replayed to an attestation service. This can lead to T.IMPERSONATION, which could violate F.ATTESTATION.

Impact on TOE

The TOE uses the standard PSA attestation design, which uses a two-way challenge to prevent replay attacks with previously signed tokens. Therefore, we conclude that exploitation of this potential vulnerability is not applicable.

V-008 Data exfiltration through unprotected peripherals

While IPs that execute firmware are properly separated in terms of SPE and NSPE, peripherals that can be controlled by the NSPE might not be, and could be used to access TOE assets.

Effect of attack

SOCs implement various peripherals that operate on data. These peripherals are often shared between different processing units present in the SOC and, if not configured properly, could be used by, for example, the NSPE to access data indirectly. This can lead to T.STORAGE and violate F.SECURE_STORAGE or F.SOFTWARE_ISOLATION. For example, a Direct Memory Access (DMA) engine might be used to process large chunks of data. If input and output descriptors that are prepared by the NSPE are used by the SPE without verification or further separation, this peripheral could be used to access TOE assets.

Impact on TOE

The TOE strictly uses peripherals internal to the TOE for dealing with its assets. Therefore, this potential vulnerability is not applicable.

V-009 Incorrect configuration of the RNG

Randomness is an important primitive for a secure TOE. Random numbers are used to generate single use random challenges, derive cryptographic keys and building blocks of cryptographic schemes such as ECDSA. An RNG that does not have enough entropy can weaken or defeat these features, compromising TOE assets.

Effect of attack

An RNG typically needs to be enabled and configured. Furthermore, health checks need to be enabled, configured, performed and sanctions need to be attached once such checks indicate that the RNG is compromised. A low entropy RNG can lead to T.WEAK_CRYPTO, which could violate F.CRYPTO. For example, the registers that configure the RNG could be open to the NSPE. If the NSPE can read out a

randomness seed, it could predict the outcome of the RNG, or it could configure the RNG to no longer generate new random numbers, which could leave it returning the same number for every random number request, or returning a predictable value such as all zeros. SECP256R1 ECDSA signing scheme relies on a randomly generated number for each signature to protect the confidentiality of the private key and uniqueness of each signature. Furthermore, various security functions such as F.ATTESTATION, F.DEBUG and F.SECURE_STORAGE rely on randomly generated data for a particular session.

Impact on TOE

The TOE uses the CC312 to generate random numbers. It cannot be reconfigured at runtime. The TRNG and DRGB are compliant with the BSI AIS-31 and NIST SP 800-90A respectively. Therefore, this potential vulnerability is not applicable.

V-010 Physical probing to extract SPE

Physical attacks on the TOE could lead to extracting TOE assets. If these assets are stored in external memory, these memories could be removed and read out.

Effect of attack

The TOE is responsible for protecting various internal assets (HUK, IAK, ROTPK). By physically removing and reading out the memory chips that store these assets, an attacker with physical access to the TOE can obtain its contents and alter them. This can lead to T.STORAGE, which could violate F.SECURE_STORAGE.

Impact on TOE

Although in the presented TOE, critical assets are stored in external memory which simulates internal memory, critical assets shall be stored in interal memory in a final TOE, as described in the integration guidance. Therefore, this potential vulnerability is not applicable.

V-011 Side channel analysis of cryptographic operations

Execution of cryptographic operations can leak information through unintended channels (such as timing, power or electromagnetic emanations), which be exploited through side channel analysis, if this leakage has a correlation with the processing of key material. This can leak keys, compromising TOE assets.

Effect of attack

By measuring a particular side channel during the execution of a cryptographic operation, bits of the private key can be retrieved if correlations exists between the key and the side channel.

Extraction of the private key material used for attestation, the IAK, can lead to T.IMPERSONATION, which could violate F.ATTESTATION. Extraction of the symmetric key HUK can lead to T.STORAGE, which could violate F.SECURE_STORAGE.

Impact on TOE

Side channel analysis attacks typically require specialized or standard equipment. Even if the attack can be performed with standard equipment and in relatively quick fashion (i.e. < one month for identification, < one week for exploitation), the rating of a successful attack would still exceed the limit of 16.

Furthmore, compromising the instance unique HUK would require local attacks, so it would not lead to a remote and scalable attack, and therefore be out of scope of a PSA L2R evaluation. Note that this restriction does not apply to the IAK.

Therefore, we conclude that this potential vulnerability is not applicable to the TOE.

V-012 Fault injection to bypass secure boot checks

While the secure boot mechanism might not have logical flaws, physical perturbation or fault injection might be feasible to bypass critical code paths. A successful fault injection campaign on the secure boot mechanism can lead to runtime control over the TOE, compromising its assets.

Effect of attack

By briefly disturbing the normal operation conditions of the TOE, for example by bringing its input voltage outside of the specified range, faulty behavior might occur in the Central Processing Unit (CPU) or its peripherals. If brief and precise enough, this can cause the TOE to perform unexpected behavior. If this occurs during the signature verification check of the secure boot sequence, this can lead to T.ROGUE_CODE, which could violate F.INITIALIZATION.

Impact on TOE

Fault injection attacks typically require specialized or standard equipment. Even if the attack can be performed with standard equipment and in relatively quick fashion (i.e. < one month for identification, < one week for exploitation), the rating of a successful attack would still exceed the limit of 16.

Furthmore, compromising the runtime control of the SPE would require local attacks, so it would not lead to a remote and scalable attack, and therefore be out of scope of a PSA L2R evaluation.

Therefore, we conclude that this potential vulnerability is not applicable to the TOE.

V-013 Fault injection to bypass firmware update checks

While the firmware update mechanism might not have logical flaws, physical perturbation or fault injection might be feasible to bypass critical code paths. A successful fault injection campaign on the firmware update mechanism can lead to runtime control over the TOE, compromising its assets.

Effect of attack

By briefly disturbing the normal operation conditions of the TOE, for example by bringing its input voltage outside of the specified range, faulty behavior might occur in the CPU or its peripherals. If brief and precise enough, this can cause the TOE to perform unexpected behavior. If this occurs during the signature verification check of the firmware update sequence, this can lead to T.UPDATE_ABUSE or T.ROGUE_CODE, which could violate F.FIRMWARE_UPDATE or F.INITIALIZATION.

Impact on TOE

Fault injection attacks typically require specialized or standard equipment. Even if the attack can be performed with standard equipment and in relatively quick fashion (i.e. < one month for identification, < one week for exploitation), the rating of a successful attack would still exceed the limit of 16.

Furthmore, compromising the runtime control of the SPE would require local attacks, so it would not lead to a remote and scalable attack, and therefore be out of scope of a PSA L2R evaluation.

Therefore, we conclude that this potential vulnerability is not applicable to the TOE.

V-014 Fault injection to bypass secure debug access checks

While the secure debug mechanism might not have logical flaws, physical perturbation or fault injection might be feasible to bypass critical code paths. A successful fault injection campaign on the secure debug protection mechanism can lead to runtime control over the TOE, compromising its assets.

Effect of attack

By briefly disturbing the normal operation conditions of the TOE, for example by bringing its input voltage outside of the specified range, faulty behavior might occur in the CPU or its peripherals. If brief and precise enough, this can cause the TOE to perform unexpected behavior. If this occurs during the signature verification check of the secure debug unlock process, this can lead to T.DEBUG, which could violate F.DEBUG.

Impact on TOE

Re-enabling debug access to the TOE via authentication is not (yet) implemented, and the debug interface is fully closed. Therefore this vulnerability is not applicable.

V-015 Fault injection on cryptographic operations

Exposing the execution of a cryptographic operation to faults which are timed and, depending on the technique used, located accurately, could result in faulty outputs which can be leveraged by an attacker to recover the private key (or parts thereof) used during this operation. This would compromise TOE assets.

Effect of attack

By briefly disturbing the normal operation conditions of the TOE, for example by bringing its input voltage outside of the specified range, faulty behavior might occur in the CPU or its peripherals. If brief and precise enough, this can cause the TOE to perform unexpected behavior while executing cryptographic operations. If the result is a faulty ciphertext, this can be leveraged by an attacker to recover private key material.

Extraction of the private key material used for attestation, the IAK, can lead to T.IMPERSONATION, which could violate F.ATTESTATION. Extraction of the symmetric key HUK can lead to T.STORAGE, which could violate F.SECURE_STORAGE.

Fault based attacks on cryptographic operations are well-studied and documented for a wide range of cryptographic algorithms.

Impact on TOE

Fault injection attacks typically require specialized or standard equipment. Even if the attack can be performed with standard equipment and in relatively quick fashion (i.e. < one month for identification, < one week for exploitation), the rating of a successful attack would still exceed the limit of 16.

Furthmore, compromising the instance unique HUK would require local attacks, so it would not lead to a remote and scalable attack, and therefore be out of scope of a PSA L2R evaluation.

Therefore, we conclude that this potential vulnerability is not applicable to the TOE.

V-016 TOCTOU in shared memory

The first software layer on top of the mailbox interface implementing the communication between the NSPE and the SPE, OpenAMP, uses shared buffers in DRAM. These buffers can be accessed by both the SPE and NSPE. The hardware does not provide any primitives which prevent NSPE access once the SPE is operating on them. This opens the possibility for TOCTOU style attacks, where the NSPE modifies the data in these buffers after the SPE has validated their validity, but before the SPE actually uses the data in its execution flow.

Effect of attack

Replacing a previously validated length could result in copying more data than expected, triggering a buffer overflow, which could result in T.ROGUE_CODE, compromising SPE runtime control, for example. Additionally, previously authenticated data could be replaced by malicious data, after authentication. These are examples that would violate security functions such as F.SOFTWARE_ISOLATION and F.SECURE_STATE.

Impact on TOE

We reviewed samples of the TF-M code which operate on the data in the DRAM. In all reviewed samples, values were copied to local variables, not accessible to the NSPE, before being validated and used. Therefore, we conclude that this potential vulnerability is not applicable to the TOE.

---

6 Summary

Arm requested Riscure to perform a PSA L2R evaluation of their Corstone-1000. The purpose of the evaluation was to verify if the TOE statisfies the PSA L2R security and assurance requirements. To achieve this goal, a conformity analysis was performed to establish the conformance of ST [1] with PSA-PP. Furthermore, a vulnerability analysis was performed following the attacker profile outlined in [13]. The analyses were performed in a white-box manner, i.e. with access to source code and design documents of the TOE. The evaluation was conducted between the 8 December 2021 and 15 March 2022, at Riscure's premises in Delft, the Netherlands.

During the conformity analysis, we have verified that the TOE supports the following PSA L2R security functions:

- F.INITIALIZATION
- F.SOFTWARE_ISOLATION
- F.SECURE_STORAGE
- F.FIRMWARE_UPDATE
- F.SECURE_STATE
- F.CRYPTO
- F.ATTESTATION
- F.DEBUG

In total, 16 potential vulnerabilities were considered during the vulnerability analysis. Through source code review in which we sampled critical parts of the TOE software, and found that none of these potential vulnerabilities could lead to an attack path that could be exploited by an attacker within the PSA L2 scope. The vulnerability analysis is summarized in more detail in Table 5.1.

Given the outcome of the conformity and vulnerability analyses, we conclude that the TOE meets all the requirements of PSA L2R.

A References

[1] Arm. *Corstone-1000 JSADEN002-PSA_Certified_Level_2_PP-1.1*. Version 1.0. Feb. 2022.

[2] Arm. *Corstone-1000 software architecture*. Version 1.0. Dec. 2021.

[3] Arm. *Corstone1000 User Guide*. `https://gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-docs/-/blob/9eeaf0105d78400dc3f3f36e3cda5ec14c286425/docs/embedded-a/corstone1000/user-guide.rst`. Online; Accessed: 2022-01-27; Git commit ID: 9eeaf010. 2022.

[4] Arm. *Arm CorstoneTM SSE-710 Subsystem, Configuration and Integration Manual*. Version r0p0. Apr. 2021.

[5] Arm. *Arm CorstoneTM SSE-710 Subsystem, Technical Reference Manual*. Version r0p0. Apr. 2021.

[6] Arm. *Arm CorstoneTM-1000 Cryptographic Extension, Technical Reference Manual Addendum*. Version r0p0. July 2021.

[7] Arm. *Arm CorstoneTM-1000 for MPS3, Application Note AN550*. Version B. Dec. 2021.

[8] Arm. *Arm CryptoCell-312, Technical Reference Manual*. Version r1p1. June 2020.

[9] Arm. *Arm Platform Security Architecture Firmware Framework (DEN0063)*. Version 1.0.0. June 2019.

[10] Arm. *Arm PSA Attestation API 1.0 (IHI0085)*. Version 1.0.2. Feb. 2020.

[11] ARM. *PSA Certified Level 2 Evaluation Methodology*. Version 1.1. Feb. 2020.

[12] ARM. *PSA Certified Level 2 Lightweight Protection Profile*. Version 1.1. Feb. 2020.

[13] ARM. *PSA Certified Level 3 Attack Methodology*. Version 1.1. Feb. 2020.

[14] *Common Vulnerabilities and Exposures*. May 2020. URL: `https://cve.mitre.org/cve/search_cve_list`.

[15] *Internal Trusted Storage (ITS) Service*. `https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git/tree/docs/technical_references/design_docs/tfm_its_service.rst?id=49a28600f9dd640638f667273ef15acb6d1a8e1c`. Online; Accessed: 2022-01-27; Git commit ID: 49a2860. 2022.

[16] *mcuboot design*. `https://github.com/mcu-tools/mcuboot/blob/41f123c24ad11fcc75c7deb452214b55f2c57ac4/docs/design.md`. Online; Accessed: 2022-01-27; Git commit ID: 41f123c. 2021.

[17] *National Vulnerability Database*. May 2020. URL: https://nvd.nist.gov/vuln/search.

[18] *Protected Storage Service Integration Guide*. https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git/tree/docs/integration_guide/services/tfm_ps_integration_guide.rst?id=49a28600f9dd640638f667273ef15acb6d1a8e1c. Online; Accessed: 2022-01-27; Git commit ID: 49a2860. 2022.

[19] Riscure. *Methodology for Fault Injection (Internal Confidential Document)*. Version 2.0. Aug. 2020.

[20] Riscure. *Methodology for Side Channel Analysis (Internal Confidential Document)*. Version 2.3. Feb. 2022.

[21] Riscure. *Methodology for System Analysis and Implementation Reviews (Internal Confidential Document)*. Version 2.1. July 2021. 28 pp.

[22] Riscure. *Security Evaluation Methodology (Internal Confidential Document)*. Version 3.0. Dec. 2019. 35 pp.

B Received materials

The list of deliverables provided by Arm is presented in Table B.1.

Date	Filename
20 January 2021	Corstone-710 Draft JSADEN002-PSA_Certified_Level_2_PP-1.1_v0.3.pdf SHA256: *b012ef7e0a2a27acfc41b27dd0f97dfbc39b733e037b9d863f03537e64299782*
13 January 2021	DAI0550A_02_arm_corstone_710_for_mps3.pdf SHA256: *44f451c8b62d00f3be6a43c9bb320ec71f74998607a4fdd3fd05f8308da52a19*
13 January 2021	Corstone-710 software architecture.pdf SHA256: *c94833ec73dc279c5bbf8722f194cf21d568631aa67e8990980caaadc3f253d1*
26 May 2021	cc312_cryptocell_technical_reference_manual_100774_0101_02_en.pdf SHA256: *6bf8b9a7f48660c3a47972bda29964ef1eaef44fe5da1bf1619ddfbcdc56877b*
26 May 2021	Corstone-710 JSADEN002-PSA_Certified_Level_2_PP-1.1.pdf SHA256: *89e342d511d6b355daf5d4eef7856d789f9af1b7caa8c5e44518e463ef1b9043*
26 May 2021	Corstone-710 software architecture.pdf SHA256: *a365bb282b9be507e9e9696745fdc22cdb996d20ad2d4ffb1642ed12a7a14d5e*
26 May 2021	corstone_sse710_cryptographic_extensions_trm_addendum_102276_0000_01_en.pdf SHA256: *22a61f00502635f53d3445e60af8b5a725d9c188a60f72095935d508aa48eced*
26 May 2021	corstone_sse710_subsystem_CIM_102343_0000_01_en.pdf SHA256: *e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855*
26 May 2021	corstone_sse710_subsystem_technical_reference_manual_102342_0000_01_en.pdf SHA256: *85c2a1e6177386eefd4125c6489fd3b36b422b641719a335889571e6e0423640*
26 May 2021	DAI0550A_arm_corstone_710_for_mps3.pdf SHA256: *0a87b10080e5dd5e3f8f94bc5c32b89615025dc148b95e44b3c033f3dc5ca5af*

Date	Filename
27 May 2021	DAI0550A_02_arm_corstone_710_for_mps3.pdf SHA256: *44f451c8b62d00f3be6a43c9bb320ec71f74998607a4fdd3fd05f8308da52a19*
27 May 2021	Corstone-710 software architecture.pdf SHA256: *c94833ec73dc279c5bbf8722f194cf21d568631aa67e8990980caaadc3f253d1*
27 May 2021	Corstone-710 Draft JSADEN002-PSA_Certified_Level_2_PP-1.1_v0.3.pdf SHA256: *b012ef7e0a2a27acfc41b27dd0f97dfbc39b733e037b9d863f03537e64299782*
09 December 2021	cc312_cryptocell_technical_reference_manual_100774_0101_02_en.pdf SHA256: *6bf8b9a7f48660c3a47972bda29964ef1eaef44fe5da1bf1619ddfbcdc56877b*
09 December 2021	Corstone_1000_Cryptographic_Extensions_TRM_Addendum_102276_0000_02_en.pdf SHA256: *f11ebe82f1e04ec8688c876da2a061d8a16b898de3e162d3580b057752446f46*
09 December 2021	corstone_sse710_subsystem_configuration_and_integration_manual_102343_0000_01_en.pdf SHA256: *34282801aa6073057ea0acba132bf2d0992da6b3cfed0966c3a248dbdde4ef63*
09 December 2021	corstone_sse710_subsystem_technical_reference_manual_102342_0000_01_en.pdf SHA256: *85c2a1e6177386eefd4125c6489fd3b36b422b641719a335889571e6e0423640*
09 December 2021	DAI0550A_arm_corstone_1000_for_mps3.pdf SHA256: *5c5bf67b9ea676c212532ff604581dbb254188cee7fe07460b6358844982f57c*
11 January 2022	DAI0550B_arm_corstone_1000_for_mps3.pdf SHA256: *714838b6f887adc563d02806e3ee51f93e11b3a0296641a3a9819d57c8d31e72*
10 January 2022	Corstone-1000 JSADEN002-PSA_Certified_Level_2_PP-1.1.pdf SHA256: *b488d8907ed06d2378510fbc8aab0eccd8ea742dfed4f9f3bfdd0875cc701344*
10 January 2022	Corstone-1000 Software Architecture v1.0.pdf SHA256: *714e172025975394124e4147cf52c40c78cc33e680c0ccd921d5eab031dfbc82*
10 January 2022	DAI0550B_arm_corstone_1000_for_mps3.pdf SHA256: *714838b6f887adc563d02806e3ee51f93e11b3a0296641a3a9819d57c8d31e72*

Date	Filename
31 January 2022	Corstone-1000 JSADEN002-PSA_Certified_Level_2_PP-1.1(1).pdf SHA256: `b6ae4dc8072d5c1cc7cf73c200587969f195f07c1868292cb252cc4176bc1eac`
25 February 2022	Corstone-1000 JSADEN002-PSA_Certified_Level_2_PP-1.1.pdf SHA256: `0c05ac4146c128e82735319b703ec71e137eb27fa8d4d89aeb69d9c22bcb987d`
1 March 2022	Corstone-1000 JSADEN002-PSA_Certified_Level_2_PP-1.1.pdf SHA256: `b5d3185e74680a48840723a3416dadc653c1708f33c2f3aad5e9c3cdffb069e3`

Table B.1: Received materials

C Received samples

The list of samples provided by Arm is presented in Table C.1.

#	Date	State	Used for test	Description
20200272/1	4 March 2021	Functional	N/A	Arm MPS3 FPGA board to run the Corstone-1000 bitstream
20200272/2	4 March 2021	Functional	N/A	Arm MPS3 FPGA board to run the Corstone-1000 bitstream
20200272/3	4 March 2021	Functional	Running the functional tests	Arm MPS3 FPGA board to run the Corstone-1000 bitstream
20200272/4	4 March 2021	Functional	N/A	4GB DDR memory
20200272/5	4 March 2021	Functional	N/A	4GB DDR memory
20200272/6	4 March 2021	Functional	N/A	4GB DDR memory

Table C.1: Received samples

D Acronyms

ARoT Application Root of Trust. 9, 35

CC312 Arm® CryptoCell™-312. 12, 14, 17, 19, 22, 38

CPU Central Processing Unit. 39–41

DMA Direct Memory Access. 37

DRAM Dynamic Random Access Memory. 13, 15, 42

ETR Evaluation Technical Report. 8

FPGA Field-Programmable Gate Array. 4, 12, 18, 20, 22, 24, 27

HUK Hardware Unique Key. 9, 19, 28, 38, 39, 41, 42

IAK Initial Attestation Key. 9, 19, 22, 28, 38, 39, 41

IP Intellectual Property. 4

JTAG Joint Test Action Group. 15

MHU Message Handling Unit. 13–15, 18

NSPE Non-secure Processing Environment. 4, 9, 12, 14–18, 21, 29, 34–37, 42

OTP One Time Programmable. 14, 17–20, 22, 28

PSA Platform Security Architecture. 22, 29, 37

PSA L2 PSA Level 2. 12, 43

PSA L2R PSA Level 2 Ready. 4, 8, 10, 12, 16, 18, 20, 22, 27, 39, 40, 42, 43

PSA-AM PSA Certified Attack Method. 4, 8

PSA-EM PSA Certified Evaluation Methodology. 8

PSA-PP PSA Certified Level 2 Lightweight Protection Profile. 4, 8–10, 15, 22, 27, 28, 43

PSARoT PSA Root of Trust. 9, 10, 18

RAM Random Access Memory. 18, 20

RNG Random Number Generator. 22, 37, 38

ROM Read-Only Memory. 12, 14, 17–20, 28

ROTPK Root of Trust Public Key. 9, 17–19, 38

SE Secure Element. 12, 14, 15, 17–19, 21, 29

SOC System-on-Chip. 4, 12, 13, 15, 18, 19, 21, 24, 29, 37

SPE Secure Processing Environment. 4, 9, 14, 15, 17–21, 23, 28–30, 34–37, 40, 42

SRAM Static Random Access Memory. 12

ST Security Target. 4, 8, 16, 27, 28, 43

SWD Single Wire Debug. 15

TLV Tag Length Value Record. 18

TOCTOU Time Of Check Time Of Use. 29, 42

TOE Target of Evaluation. 4, 7–43, 51

E Historical TOE versions

During the execution of the evaluation, we received some software components which were later updated. The versions of these older components are listed in Table E.1.

Component	Version	Git identifier
Yocto configuration	CORSTONE1000-2022.01.18	3.4-121-gccf342a
trusted-firmware-m	v1.5.0	TF-Mv1.5.0-RC1-142-g49a28600f9

Table E.1: Historical TOE software versions that were provided during the evaluation