SESIP Security Target for FreeRTOS

Amazon

Version: 1.9 Dated 2021/01/14
Based on [SESIP] methodology, version “Public Release v1.0”

CHANGES CONTROL

Version	Date	Author	Reason	Changes
1.0	2020/10/14	Amazon & jtsec	First Version	N/A
1.1	2020/10/19	Amazon & jtsec	Updated with more detail	Full document affected
1.2	2020/10/26	Amazon & jtsec	Ready for evaluation	Full document affected
1.3	2020/10/27	Amazon & jtsec	Alignment with [GUIDES]	Minor changes. ALC_FLR reworked.
1.4	2020/11/12	Amazon & jtsec	Modify logo and solve lab comments	Minor changes.
1.5	2020/11/24	Amazon & jtsec	Clarification of rollback capabilities.	Minor changes.
1.6	2020/12/21	Amazon & jtsec	Addressing Riscure comments	Version numbers updated Added reference to wolfssl API Removed Secure Sockets Library from the TOE scope Clarified wolfSSL FIPS mode and cryptographic hardware support Added LTS Code Quality checklist Other minor updates
1.7	2020/12/29	Amazon & jtsec	Addressing Riscure comments	Typographical error and other minor clarifications
1.8	2021/01/05	Amazon	Addressing review comments	Update library version numbers. Minor edits for clarity
1.9	2021/01/14	Amazon	Update	Add OE.RNG requirements

Table of Contents

1 INTRODUCTION

The Security Target describes the Platform (in this chapter) and the exact security properties of the Platform that are evaluated against [SESIP] (in chapter “Security requirements and implementation”) that a potential consumer can rely upon the product upholding if they fulfill the objectives for the environment (in chapter “Security Objectives for the operational environment”).

1.1 ST REFERENCE

See title page.

1.2 PLATFORM REFERENCE

TOE name	FreeRTOS
TOE version	202012.00-LTS
TOE identification	FreeRTOS version 202012.00-LTS
TOE Type	IoT operating system and libraries for microcontrollers. The TOE is delivered in source code form.

The TOE is composed of different parts that are identified in the following table:

TOE Parts	
FreeRTOS Kernel	V10.4.3
FreeRTOS+TCP	V2.3.2
corePKCS #11	V3.0.0
OTA Updates	V2.0.0
Mbed TLS	V2.24.0
wolfSSL	V4.5.0

1.3 INCLUDED GUIDANCE DOCUMENTS

The following documents are included with the platform:

Kernel base documentation		
[README_FIRST]	README FIRST.txt	202012.00-LTS
[KERNEL-API]	FreeRTOS Kernel API Reference	10.4.3

Libraries documentation		
[TCP-API]	FreeRTOS+TCP API Rerefence	2.3.2
[corePKCS#11-API]	corePKCS#11 Cryptoki Library API Reference	3.0.0
[OTA-API]	Over the Air (OTA) Update library API Reference	2.0.0
[Mbed TLS-API]	Mbed Transport Layer Security Library API Reference	2.24.0
[WOLFSSL-GUIDE]	wolfSSL User Manual Note that this version is also applicable to wolfSSL 4.5.0	4.1.0
Other documentation		
[WOLFSSL-MIGRATION]	FreeRTOS with mbedTLS to FreeRTOS with wolfSSL Migration Guide	1.0
[QUALIFICATION]	FreeRTOS Qualification Guide	2020
Adhoc SESIP guidance		
[GUIDES]	SESIP additional guidance for FreeRTOS	1.5

1.4 PLATFORM FUNCTIONAL OVERVIEW AND DESCRIPTION

1.4.1 OVERVIEW

FreeRTOS is an open source, real-time operating system for microcontrollers that makes small, low-power edge devices easy to program, deploy, secure, connect, and manage. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of software libraries suitable for use across industry sectors and applications. This includes securely connecting small, low-power devices to AWS cloud services through a secure connection. FreeRTOS is built with an emphasis on reliability and ease of use.

A microcontroller contains a simple, resource-constrained processor that can be found in many devices, including appliances, sensors, fitness trackers, industrial automation, and automobiles. Many of these small devices can benefit from connecting to the cloud or locally to other devices, but have limited compute power and memory capacity and typically perform simple, functional tasks. Microcontrollers frequently run operating systems that may not have built-in functionality to connect to local networks or the cloud, making IoT applications a challenge. FreeRTOS helps solve this problem by providing the kernel to run low-power devices as well as software libraries that make it easier to securely connect to the cloud or other edge devices, so FreeRTOS can collect data from them for IoT applications and take action.

The TOE is provided to users in source code form and will be typically compiled to a specific hardware and flashed to devices as a single binary image, including the applications running on top of the TOE.

The main security features of the TOE are as follows:

- **Verification of Platform Identity and Instance Identity:** The TOE version is defined in the sources files and the preparative guidance [GUIDES] strictly forbids unauthorized modification of these numbers. Through the provisioning process in factory, it is possible to univocally identify the TOE instance identity.

- **Firmware update:** Over the Air (OTA) Updates OTA updates make it possible to update device firmware without an expensive recall or technician visit. This method allows to quickly respond to security vulnerabilities and software bugs that are discovered after the devices are deployed in field. Updates will be verified to be from a trusted source, along with other validations like version checking and compatibility. The FreeRTOS over-the-air (OTA) client library enables the app developer to manage the notification of a newly available update, download the update, and perform cryptographic verification of the firmware update.

  As specified by [GUIDES], code signature verification must be enabled using the "configOTA_ENABLE_CODE_SIGNATURE_VERIFICATION" macro that is located in the "aws_ota_agent_config.h" header file.

- **Secure Communications support:** The developer can use the TLS library to create embedded applications that communicate securely. The library is designed to make onboarding easy for software developers from various network programming backgrounds.

- **Isolation capabilities:** The TOE contains security features such as Task Isolation and Separation, which allows developers to securely isolate security critical code within the same linear memory space, using the processor's MPU/MMU. This feature enables microcontroller applications to be more robust and more secure by first enabling tasks to run in either privileged or unprivileged mode, and second restricting access to resources such as RAM, executable code, peripherals, and memory beyond the limit of a task's stack. Huge benefit is gained from, for example, allowing defining memory regions and assigning memory access permission and memory attributes to each of them as doing so will protect against many attack vectors such as buffer overflow exploits or the execution of malicious code loaded into RAM. This feature requires hardware MPU support.

- **Cryptographic Operations:** The TOE provides cryptographic operations capabilities through a common PKCS#11 interface. Two different cryptographic implementations can be configured, mbed and WolfSSL (FIPS 140-2 certified as can be seen in https://csrc.nist.gov/Projects/cryptographic-module-validation-program/Certificate/3389, to use the FIPS mode please follow the instructions provided in https://www.wolfssl.com/docs/fips-ready-user-guide/ ; version number can be obtained through wolfCrypt_GetVersion_fips()).

1.4.2 TOE SCOPE

The TOE consists of the FreeRTOS kernel as well as software libraries implementing FreeRTOS+TCP, corePKCS #11, over-the-air updates (OTA), TLS and cryptographic operations.

The TOE is intended to be used as an operating system as well as connectivity library component by IoT device manufactures in implementing firmware for microcontroller-based connected devices.

The TOE scope is depicted in Figure 1 TOE scope. The grayed parts are within the evaluation scope and the other parts are outside of the evaluated scope.

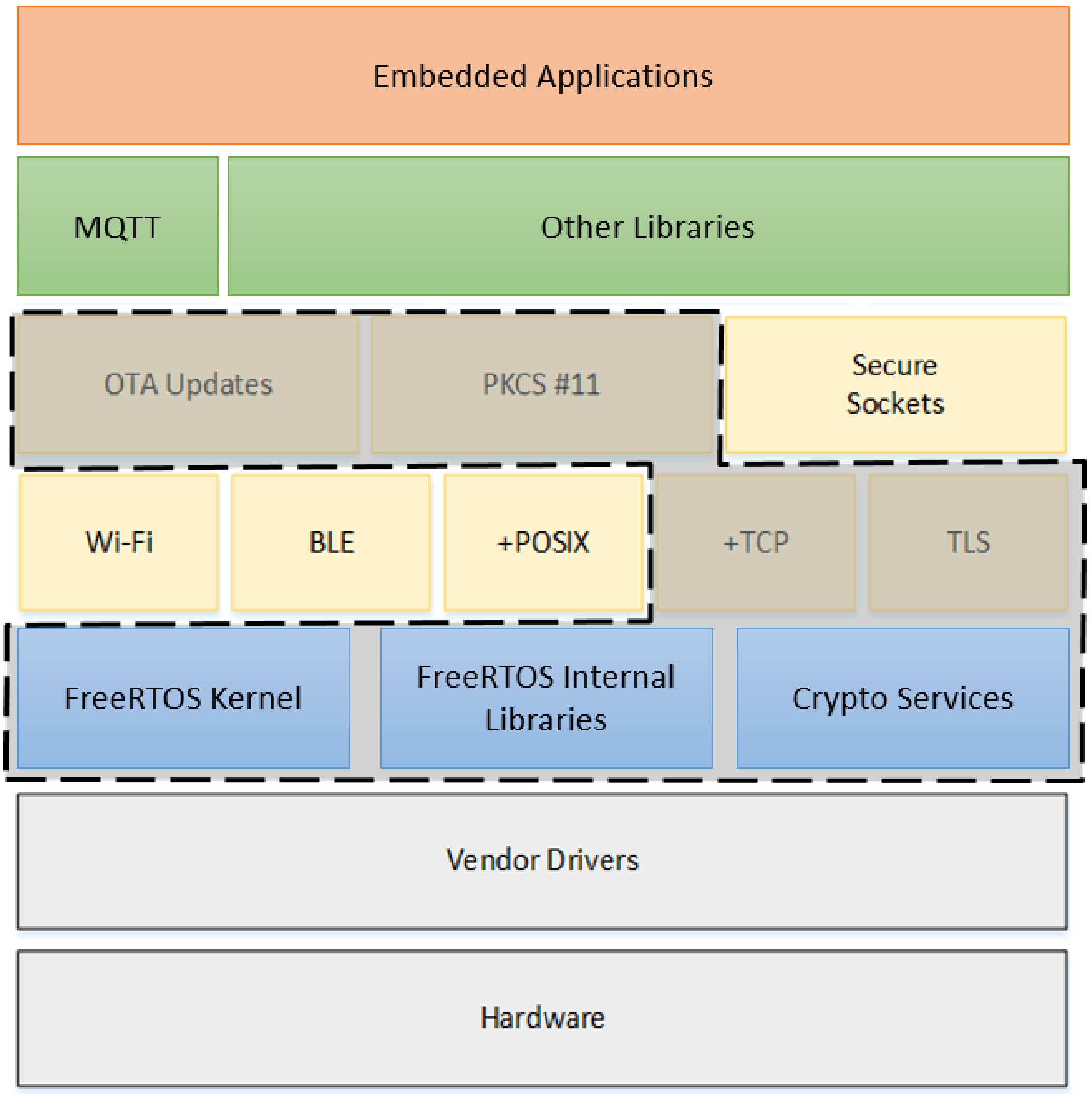

Figure 1 TOE scope

The physical scope includes only the applicable source code for the evaluated version FreeRTOS kernel and in scope libraries, and associated guidance documentation as listed in section 1.3. No hardware is included in the physical scope.

The **out of scope part (Non-TOE)** comprises:

- **Hardware:** MCU, peripherals and supporting MPU. Optionally, a secure element could be included implementing cryptographic algorithm engines or tamper-resistant storage. Please, note that to remain compliant to this ST, you shall use the evaluated cryptographic algorithms.

- **Vendor drivers:** In general, device drivers are independent of the underlying operating system and are specific to a given hardware configuration. A hardware abstraction layer (HAL) is a wrapper that provides common interfaces between drivers and higher-level application code. The HAL abstracts away the details of how a specific driver works and provides a uniform API to control similar devices. In this way, the app developer can use the same APIs to control various devices across multiple microcontroller (MCU) based reference boards.

- **Bootloader:** the bootloader implementation. It is recommended that the implementation of the bootloader enforces anti-rollback and firmware code-sign verification.

- **Other libraries:** Any companion library not explicitly included under the logical scope.

- **Embedded applications:** FreeRTOS is typically flashed to devices as a single compiled image with all of the components required for device applications. Independent of the individual microcontroller being used, embedded application developers can expect the same standardized interfaces to the FreeRTOS kernel and all FreeRTOS software libraries.

The logical scope includes:

- **FreeRTOS Kernel and FreeRTOS Internal Libraries:** The FreeRTOS kernel is a real-time operating system that supports numerous architectures. It is ideal for building embedded microcontroller applications. It provides:
  - A multitasking scheduler.
  - Multiple memory allocation options (including the ability to create completely statically-allocated systems).
  - Intertask coordination primitives, including task notifications, message queues, multiple types of semaphore, and stream and message buffers.

- **FreeRTOS+TCP:** FreeRTOS+TCP is a scalable, open source and thread safe TCP/IP stack for FreeRTOS. FreeRTOS+TCP provides a familiar and standards based Berkeley sockets interface, making it as simple to use and as quick to learn as possible. An alternative callback interface is also available for advanced users.

- **corePKCS #11 Module:** PKCS #11 is a standardised and widely used API for manipulating common cryptographic objects. It is important because the functions it specifies allow application software to use, create, modify, and delete cryptographic objects, without ever exposing those objects to the application's memory. The TOE supports two different underlying implementations: **mbed TLS** and **wolfSSL**.

- **OTA:** The Over-The-Air (OTA) Agent enables the app developer to manage the notification, download, and verification of firmware updates for FreeRTOS devices using HTTP or MQTT as the protocol. The OTA Agent can share a network connection with the application. By sharing a network connection, the app developer can potentially save a significant amount of RAM. In addition, the OTA Agent library lets the app developer define application-specific logic for testing, committing, or rolling back a firmware update.

- **TLS:** The FreeRTOS Transport Layer Security (TLS) interface is a thin, optional wrapper used to abstract cryptographic implementation details away from the Secure Sockets Layer (SSL) interface above it in the protocol stack. The purpose of the TLS interface is to invoke the software crypto library.

2 SECURITY OBJECTIVES FOR THE OPERATIONAL ENVIRONMENT

2.1 PLATFORM OBJECTIVES FOR THE OPERATIONAL ENVIRONMENT

For the platform to fulfill its security requirements, the operational environment (technical or procedural) shall fulfill the following objectives as described in [GUIDES] section “Environmental Requirements”:

- **OE.PHYSICAL:** The platform shall only be deployed in environments where physical attacks protections are not required.
- **OE.HOSTILE_CODE:** The application shall not allow execution of hostile code.
- **OE.MPU:** The hardware where the platform is deployed shall support MPU.
- **OE.PERSONNEL:** Only authorized and trustworthy personnel shall have access to the TOE development environment
- **OE.INTEGRITY:** No modification to the TOE by the application developer is allowed.
- **OE.UPDATES:** The application developer shall make the update mechanism available via AWS IoT OTA service and shall implement functionality to regularly check for updates. Application developers are required to only use newer versions of the TOE and do not go back to a potential vulnerable version.
- **OE.RNG**: The hardware where the platform is deployed shall provide a cryptographically secure random number generator as describe in the FreeRTOS Qualification Guide [QUALIFICATION].

3 SECURITY REQUIREMENTS AND IMPLEMENTATION

3.1 SECURITY ASSURANCE REQUIREMENTS

The claimed assurance requirements package is **SESIP2** as defined in [SESIP]. The assurance requirements are as follows:

Assurance Class	Assurance Families
ASE: Security Target evaluation	ASE_INT.1 – ST Introduction *ASE_OBJ.1 – Security requirements for the operational environment* **ASE_REQ.3 – Listed security requirements** *ASE_TSS.1 – TOE summary specification*
ADV: Development	ADV_FSP.4 – Complete functional specification
AGD: Guidance documents	AGD_OPE.1 – Operational user guidance AGD_PRE.1 – Preparative procedures
ALC: Life-cycle support	ALC_FLR.2 – Flaw reporting procedures
ATE: Tests	ATE_IND.1 Independent testing: conformance
AVA: Vulnerability Assessment	AVA_VAN.2 – Vulnerability analysis

3.1.1 FLAW REPORTING PROCEDURE (ALC_FLR.2)

The evaluated version meets the LTS Code Quality checklist, that includes static checking of the code with Coverity, and extensive unit testing amongst others:

#	Category	Checks
1	Complexity Score	All functions will have a GNU Complexity score of 8 or lower
2	Coding Standard	All functions will comply with the MISRA coding standard
3	Static Checking	All code will be statically checked with Coverity
4	Function Returns	All functions will have a single exit point
5	Code Testing	All code will have extensive unit tests. Gcov reports will be used to report the test coverage, and each library will have extended functional tests.
6	Requirements Documentation	All libraries will have documented requirements, which may include resource requirements, listing all dependencies, and porting requirements (as applicable)
7	Design Documentation	All libraries will have a design document, which may include application and cloud interface, state machines, and synchronization (as applicable).
8	Compiler Warning	Code will compile without generating any compiler warnings when the gcc -Wall -Wextra compiler options are used.

If however a user of the Platform finds a flaw, we ask that the user notifies AWS/Amazon Security via our vulnerability reporting page (https://aws.amazon.com/security/vulnerability-reporting/) or directly via email to aws-security@amazon.com.

Please do not create a public github issue.

When an email is received a ticket in the FreeRTOS internal system is automatically generated and a security expert is assigned to review this as critical priority tasks. Security tickets can only be viewed by people referenced from the ticket. Once the ticket is assessed and a vulnerability is confirmed the development team creates a disclosure and remediation plan, publish a public message disclosing the vulnerability and its remediation.

After verifying that the flaw is real, a fix is developed and merged into the current release candidate version of the TOE, where it forms part of the next release of the kernel or libraries that compose the TOE, and the internal issue is closed.

If needed, a new CVE number is requested to the competent authorities. Security updates with CVE's are listed here https://aws.amazon.com/freertos/security-updates/

The new version of the code will finally undergo the Code Quality checklist to guarantee the good form of the output.

For severe issues an updated version of the current release is made available as soon as the fix is available.

Users can stay aware of FreeRTOS updates and vulnerability subscribing to the mailing list in this address: https://www.freertos.org/RTOS-contact-and-support.html

When new releases of the TOE are made, customers can update their factory production lines and TOE devices that have already been provisioned with a previous version using the *"Secure Update of Platform"* Security Functionality.

3.2 SECURITY FUNCTIONAL REQUIREMENTS

The platform fulfills the following security functional requirements. All of them are tested for robustness as part of a SESIP Level 2 evaluation.

3.2.1 VERIFICATION OF PLATFORM IDENTITY

The platform provides a unique identification of the platform, including all its parts and their versions.

Conformance rationale:

The source code contains the version of the TOE and its libraries in the manifest file.

3.2.2 VERIFICATION OF PLATFORM INSTANCE IDENTITY

The platform provides a unique identification of that specific instantiation of the platform, including all its parts and their versions.

Conformance rationale:

During the TOE provisioning, the ThingName of the device is set obtaining an instance identity. The procedure to carry out the provision is described in the [GUIDES]. The ThingName is stored by the user in the chosen location (e.g. Flash, SE, etc…) and is accessible during the TOE execution.

There is no way to change this instance identity during the whole life of the deployed TOE.

3.2.3 SECURE UPDATE OF PLATFORM

The platform can be updated to a newer version in the field such that the integrity, authenticity and confidentiality of the platform is maintained.

Conformance rationale:

Upon being initialized OTA Agent will check for new update by sending a message to the configured server. The OTA Update Manager service relies on existing security mechanisms, such as Transport Layer Security (TLS) mutual authentication, used by AWS IoT.

When server has a new update to serve it will send a JSON formatted message containing all the update meta-data to the OTA Agent which will parse it. Depending on the configuration the agent will either store the new image in a file, or write it to a specified flash address which has been reserved as storage for updates. If the update is accepted, the OTA Agent will start receiving packets from the server to be stored on the device, and will validate if the file is correct by checking its signature.

After validation the OTA Agent will notify the application about the result of update and outsourcing task of scheduling software restart, using board specific restart functionality.

After the device restarts it will commence running the new (updated) TOE software image.

*The application developer is required to make the update mechanism available via its infrastructure, as described in section "*Security Objectives for the operational environment*".*

3.2.4 SECURE UPDATE OF APPLICATION

The application can be updated to a newer version in the field such that the integrity, authenticity and confidentiality of the application is maintained.

Conformance rationale:

The over-the-air (OTA) Agent is designed to simplify the amount of code the app developer must write to add OTA update functionality to the product. That integration burden consists primarily of initialization of the OTA Agent and, optionally, creating a custom callback function for responding to the OTA completion event messages.

For a client to accept an OTA update, the version number of the update it's receiving needs to be higher than the version of the firmware that it's currently running.

The application version of the device software is set in the "ota_application_version.h" header file with the "APP_VERSION_MAJOR", "APP_VERSION_MINOR", and the "APP_VERSION_BUILD" macros.

3.2.5 SECURE COMMUNICATION SUPPORT

The platform provides the application with one or more secure communication channel(s).

The secure communication channel authenticates <*configured endpoints*> and protects against <*disclosure, modification and replay*> of messages between the endpoints, using <*TLSv1.2 and TLSv1.3*>.

Conformance rationale:

The app developer can use the TLS library to create embedded applications that communicate securely. The library is designed to make onboarding easier for software developers from various network programming backgrounds.

According to [GUIDES], the developer is required to force the use of TLS and to set the root of trust server certificate for the socket.

Note that the library will use the 3rd party configured TLS library (mbed or WolfSSL).

3.2.6 SOFTWARE ATTACKER RESISTANCE: ISOLATION OF PLATFORM

The platform provides isolation between the application and itself, such that an attacker able to run code as an application on the platform cannot compromise the other functional requirements.

Conformance rationale:

FreeRTOS supports privileged and unprivileged tasks using the support of a MPU when available.

The data maintained by the RTOS kernel (all non stack data that is private to the FreeRTOS source files) is located in a region of RAM that can only be accessed while the microcontroller is in Privileged mode.

FreeRTOS APIs are located in a region of Flash that can only be accessed while the microcontroller is in Privileged mode. These APIs are made available to unprivileged tasks via System Calls which causes a temporary switch to Privilege mode. The privilege escalations are only allowed from inside the System Calls. This prevents privilege escalations originating from an unprivileged task (other than escalations performed by the hardware itself when an interrupt is entered).

System peripherals can only be accessed while the microcontroller is in Privileged mode. Standard peripherals (UARTs, etc.) are accessible by any code but can be explicitly protected using a user definable memory region.

3.2.7 SOFTWARE ATTACKER RESISTANCE: ISOLATION OF PLATFORM PARTS

The platform provides isolation between platform parts, such that an attacker able to run code in <*the data memory*> can compromise neither the integrity and confidentiality of <*the executable memory*> nor the provision of any other security functional requirements.

Conformance rationale:

The platform provides isolation between the executable and the data memory. By not allowing any program execution from data memory, any arbitrary code execution attack based on a code injection vulnerability can be prevented as program memory will be read-only and data memory will be non-executable. This is done by using a special linker script which locates the data and code memory in different regions and then calling an MPU_Init function which sets the isolation using the MPU.

The application developer is required to use a MPU enabled hardware platform as described in section "Security Objectives for the operational environment".

3.2.8 SOFTWARE ATTACKER RESISTANCE: ISOLATION OF APPLICATION PARTS

The platform provides isolation between parts of the application, such that an attacker able to run code as one of the <*unprivileged tasks*> cannot compromise the integrity and confidentiality of the other application parts.

Conformance rationale:

Tasks can be created to run in either Privileged mode or Unprivileged mode. Unprivileged tasks can only access their own stack and a limited number of user definable memory regions. This limit is determined by the capability of the MPU and can be adjusted by configuration. User definable memory regions are assigned to tasks when the task is created, and can be reconfigured at run time if required (A Privileged mode task can set itself into unprivileged mode, but once in unprivileged mode it cannot set itself back to privileged mode):

- *Privileged Tasks: A privileged task has access to the entire memory map. Privileged tasks can be created using either the xTaskCreate() or xTaskCreateRestricted() API function (and their static versions xTaskCreateStatic() and xTaskCreateRestrictedStatic()).*

- *Unprivileged Tasks: An unprivileged task only has access to its own stack. In addition, it can be granted access to a limited number of user definable memory regions (limited by MPU capability). Unprivileged tasks can only be created using the xTaskCreateRestricted() API. Note that xTaskCreate() API must not be used to create an unprivileged task.*

No data memory is shared between Unprivileged tasks, but Unprivileged tasks can pass messages to each other using the standard queue and semaphore mechanisms. Shared memory regions can be explicitly created by using a user definable memory region but this is discouraged.

A Privileged mode task can set itself into Unprivileged mode, but once in Unprivileged mode it cannot set itself back to Privileged mode.

The application developer is required to use a MPU enabled hardware platform as described in section "Security Objectives for the operational environment".

3.2.9 CRYPTOGRAPHIC OPERATION

The platform provides the application with <*table below operations*> functionality with <*table below algorithms*> as specified in <*table below specification*> for key lengths <*table below list of key lengths*> and modes <*table below list of modes*>.

The following table provides iterations of this SFR:

Operations	Algorithm	Specification	Key lenghts	Modes
SigGen, SigVer	ECDSA	FIPS 186-4	256 bits	Curve: P-256
SigGen, SigVer	RSA	FIPS 186-4	2048	N/A
Hash	SHS	FIPS 180-4	N/A	SHA2-256

Conformance rationale:

Cryptographic functionality is provided through the PKCS#11 interface, that abstracts two possible configurations, which need to be defined in compilation time:

- *WolfSSL version 4.5.0: The wolfSSL embedded SSL library is a lightweight, portable, C-language-based SSL/TLS library targeted at IoT, embedded, and RTOS environments primarily because of its size, speed, and feature set.* WolfSSL is FIPS 140-2 certified as can be seen in https://csrc.nist.gov/Projects/cryptographic-module-validation-program/Certificate/3389
- *mbed TLS version 2.24.0: mbed TLS offers an SSL library with an intuitive API and readable source code, so the app developer can actually understand what the code does. In addition, the mbed TLS modules are as loosely coupled as possible and written in the portable C language.*

4 SESIP2 MAPPING AND SUFFICIENCY RATIONALES

Assurance Class	Assurance Families	Covered by	Rationale
ASE: Security Target evaluation	ASE_INT.1 ST Introduction	Section "Introduction" and "Title"	The ST reference is in the Title, the TOE reference in the "Platform reference", the TOE overview and description in "Platform functional overview and description".
	ASE_OBJ.1 Security requirements for the operational environment	Section " Security Objectives for the operational environment"	The objectives for the operational environment in " Security Objectives for the operational environment" refers to the guidance documents.
	ASE_REQ.3 Listed Security requirements	Section "Security Functional Requirements"	All SFRs in this ST are taken from [SESIP]. "Verification of Platform Identity" is included. "Secure Update of Platform" is included.
	ASE_TSS.1 TOE Summary Specification	Section " Security requirements and implementation"	All SFRs are listed per definition, and for each SFR the implementation and verification are defined in Security Functional Requirements.
ADV: Development	ADV_FSP.4 Complete functional specification	Section "Included guidance documents". API documents.	The guidance documents include the API needed to interact with the TOE and invoke the implemented security functionality.
AGD: Guidance documents	AGD_OPE.1 Operational user guidance	Section "Included guidance documents". [GUIDES].	The platform evaluator will determine whether the provided evidence is suitable to meet the requirement.

	AGD_PRE.1 Preparative procedures	Section "Included guidance documents". [GUIDES].	The platform evaluator will determine whether the provided evidence is suitable to meet the requirement.
ALC: Life-cycle support	ALC_FLR.2 Flaw reporting procedures	Section "Flaw Reporting Procedure (ALC_FLR.2)"	The flaw reporting and remediation procedure is described.
ATE: Tests	ATE_IND.1 Independent testing: conformance	N.A. An independent testing is performed by the platform evaluator to ascertain the presence of conformance problems.	The platform evaluator performs independent testing, to confirm that there are no conformance problems.
AVA_VAN.2	AVA_VAN.2 Vulnerability analysis	N.A. A vulnerability analysis is performed by the platform evaluator to ascertain the presence of potential vulnerabilities.	The platform evaluator performs penetration testing, to confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE. Penetration testing is performed by the platform evaluator assuming an attack potential of Basic.

5 REFERENCES

Code	Reference
[SESIP]	GlobalPlatform Technology, Security Evaluation Standard for IoT Platforms (SESIP), GP_FST_070, Public Release v1.0, March 2020
[README_FIRST]	README FIRST.txt 202012.00-LTS
[KERNEL-API]	FreeRTOS Kernel API Reference 10.4.3
[TCP-API]	FreeRTOS+TCP API Rerefence 2.3.2
[corePKCS#11-API]	PKCS#11 Cryptoki Library API Reference 3.0.0
[OTA-API]	Over the Air (OTA) Update library API Reference 2.0.0
[mbed TLS-API]	Transport Layer Security Library API Reference 2.24.0
[WOLFSSL-GUIDE]	wolfSSL User Manual 4.1.0
[WOLFSSL-MIGRATION]	FreeRTOS with mbedTLS to FreeRTOS with wolfSSL Migration Guide 1.0
[QUALIFICATION]	FreeRTOS Qualification Guide 2020
[GUIDES]	SESIP additional guidance for FreeRTOS 1.5

6 ACRONYMS

Acronym	Description
AES	Advanced Encryption Standard
AWS	Amazon Web Services
API	Application Programming Interface
BLE	Bluetooth Low Energy
CC	Common Criteria
CVE	Common Vulnerability Exposure
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GNU	GNU’s Not Unix
GP	Global Platform
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LTS	Long Term Support
MISRA	Motor Industry Software Reliability Association
MCU	Micro Controller Unit
MPU	Memory Protection Unit
MQTT	Message Queuing Telemetry Transport
OTA	Over the Air
PKCS	Public-Key Cryptography Standards
RSA	Rivest, Shamir, Adleman
RTOS	Real Time Operating System
SESIP	Security Evaluation Standard for IoT Platforms
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSL	Secure Sockets Layer

ST	Security Target
TCP	Transport Control Protocol
TLS	Transport Layer Security
TOE	Target of Evaluation