TO-Protect 2.x Security Target

Release TODO001_E

15-Nov-2022

Authored by *Trusted Objects*

Table of contents

1 Introduction **1**
- 1.1 ST reference . . . 1
- 1.2 Platform reference . . . 1
- 1.3 Included guidance documents . . . 2
  - 1.3.1 Followed security guidances . . . 2
  - 1.3.2 Roles . . . 3
- 1.4 Platform functional overview and description . . . 3
  - 1.4.1 The TOE boundary . . . 4
  - 1.4.2 TLS dialog example . . . 5
  - 1.4.3 The TOE Structure . . . 5
    - 1.4.3.1 TO-Protect . . . 5
    - 1.4.3.2 The secure storage . . . 6
    - 1.4.3.3 The perso API . . . 7
    - 1.4.3.4 The secure Upgrade . . . 7
    - 1.4.3.5 LibTO and the helpers . . . 7
    - 1.4.3.6 The application . . . 7
    - 1.4.3.7 The hardware platform . . . 7

2 Security Objectives for the Operational environment **8**
- 2.1 Platform Objectives for the Operational Environment . . . 8
  - 2.1.1 Securing the presence of a debug or Bootloader functionality . . . 8
  - 2.1.2 Secure provisioning and preparative procedures (AGD_PRE.1) . . . 9

3 Security requirements and implementation **10**
- 3.1 Security Assurance Requirements . . . 10
  - 3.1.1 Flaw Reporting Procedures (ALC_FLR.2) . . . 10
  - 3.1.2 Vulnerability Survey (AVA_VAN.1) . . . 10
  - 3.1.3 Survey of potential vulnerabilities . . . 11
    - 3.1.3.1 Out-of-scope vulnerabilities . . . 11
    - 3.1.3.2 Considered vulnerabilities . . . 12
- 3.2 Security Functional Requirements . . . 12
  - 3.2.1 Attestation of platform genuineness . . . 12
  - 3.2.2 Verification of Platform Identity . . . 13
  - 3.2.3 Secure Update of Platform . . . 13
  - 3.2.4 Secure Initialization of Platform . . . 13
  - 3.2.5 Attestation of Platform State . . . 13
  - 3.2.6 Secure Communication Support (ECDHE_ECDSA case) . . . 14
  - 3.2.7 Secure Communication Support (PSK case) . . . 14
  - 3.2.8 Physical Attacker Resistance . . . 15

4 Mapping and sufficiency rationales **16**
- 4.1 SESIP1 sufficiency . . . 16

5 Potential vulnerabilities **17**
- 5.1 Defence against fault injection . . . 17
- 5.2 Defence against side-channel attacks . . . 17
- 5.3 During the Handshake (using ECDHE-ECDSA) . . . 17
  - 5.3.1 Fault attack on the clients random generation . . . 18
  - 5.3.2 Fault attack on the ephemeral random private key generation . . . 19

5.3.3 Side-channel attack on the ephemeral public key generation . . . 19
5.3.4 Fault attack on the ephemeral public key generation . . . 19
5.3.5 Side-channel attack on the signing of all previous messages . . . 19
5.3.6 Fault attack on the servers certificate verification . . . 19
5.3.7 Side-channel attack on the ECDHE (pre_master_secret calculation) . . . 20
5.3.8 Fault attack on the ECDHE (pre_master_secret calculation) . . . 20
5.3.9 Side-channel attack on the master_secret calculation . . . 20
5.3.10 Side-channel attack on the finished calculation and checking . . . 20
5.3.11 Fault attack on the finished . . . 21
5.3.12 Side-channel attack on the computation of the key_block . . . 21
5.3.13 Fault attack on the computation of the key_block . . . 21
5.4 During the Handshake (using PSK) . . . 21
5.5 Abbreviated handshake, using either ECDHE_ECDSA or PSK . . . 22
5.6 Exchange of encrypted payloads (client to the server case) . . . 23
5.6.1 Fault attack on the generation of the random IV . . . 24
5.6.2 Side-channel attack on the MAC computation . . . 24
5.6.3 Fault attack on the MAC computation . . . 24
5.6.4 Side-channel attack on the CBC encryption . . . 24
5.6.5 Fault attacks on the CBC encryption (key recovery) . . . 24
5.6.6 Fault attacks on the CBC encryption (data recovery) . . . 25
5.7 Exchange of encrypted payloads (server to the client case) . . . 25
5.7.1 Side-channel attack on the CBC decryption . . . 25
5.7.2 Fault injection attack on the MAC verification . . . 25

6 Security Impact Analysis 26

7 Configuration Management 27

8 Document History 28

Bibliography 29

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

TO-Protect 2.x Security Target, Release TODO001_E, Trusted-Objects, 15 Nov 2022.

1.2 Platform reference

Table 1.1: References for the target

Element name	Value
TOE name	TO-Protect TLS
TOE version	2.1.6
TOE identification	TOPR-TP-00-2.1.6 and TOPR-TP-01-2.1.6
TOE Type	Software library

The TOE is identified under 2 references, please refer to *Security Impact Analysis* for a security impact analysis of this point.

1.3 Included guidance documents

The following documents are included with the platform:

Table 1.2: Documents associated to the target

Reference	Name	Version
[Manual]	TO-Protect 2.x User Manual	TODO002_E
[Admin]	Administration commands	TODO003_B
[ALC_FLR]	Flaw remediation	TODO004_B
[Upgrade]	Secure upgrade Manual	TODO005_B
[Errata]	TO-Protect 2.x Errata sheet	TODO006_B
[Attacks]	TO-Protect 2.x Attacks on TLS	TODO007_C
[TLS12]	TLS 1.2 specification	Final
[DTLS12]	DTLS 1.2 specification	Final
[PSK-TLS]	Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)	Final
[PSK-TLS-SHA256]	Pre-Shared Key Ciphersuites for TLS with SHA256/384 and AES Galois Counter mode	Final
[TLS-ECC-SUITES]	TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)	Final
[AES]	ADVANCED ENCRYPTION STANDARD (AES)	Final
[HMAC]	HMAC: Keyed-Hashing for Message Authentication	Final
[Extensions]	Transport Layer Security (TLS) Extensions (list)	2021-10-15
[TLS-Extensions]	Transport Layer Security (TLS) Extensions	Final
[Hash-DRBG]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators	SP 800-90A Revision 1

1.3.1 Followed security guidances

On the top of all specifications, and in order to reach the security objectives, the following guidance have been taken into accound and respected during the development.

Table 1.3: Security recommendations

Reference	Name	Version
[TLS-ANSSI]	Recommandations de sécurité relatives à TLS.	Final
[TLS-weaknesses]	Summarizing Known Attacks on Transport . Layer Security (TLS) and Datagram TLS (DTLS)	Final

1.3.2 Roles

With **TO-Protect**, there are 3 roles :

- A User role, able to run any functionality in **TO-Protect**, including establishing a TLS session, etc. This role can be played by the application, developed and operated by the end-customer.
- An Upgrader role, played by the part of software responsible of realizing an upgrade from a version N of the TOE to a superior version.
- An Admin role, which has the responsibility to run the admin commands. These commands are only accessible to someone having access to the admin keys.

1.4 Platform functional overview and description

This document is the security target for **TO-Protect**, and is based on [SESIP] methodology, version Public Release 1.1.

The TOE consists of a binary library to be included along with the end-user application, once integrated into an application by a developer, it contributes to adding security to an IoT Product, connected to an IoT management platform. It implements [TLS12] and permits to securely setup a TLS link between an IoT device and a host server.

It is providing a full-featured TLS implementation, responsible for :

- Managing all TLS-related messages
  - Establishing/restoring a session
  - Encrypt/Decrypt/sign/verify payloads
- Secure the storage of security-critical assets
  - Securely store assets (eg. session keys, private keys, PSK etc)
- Contribute to the IoT security by ensuring basic services :
  - Generating secure random, robust against replay attacks, using state-of-the-art algorithms

From a SESIP standpoint, the TOE is a component of the IoT platform. The TOE scope is depicted in the Fig. 1.1. Only the parts within the red rectangle, named TOE Boundary is to be evaluated. The other parts, outside this boundary are out-of-scope of this evaluation. For instance, we do not include the secure upgrade.

1.4.1 The TOE boundary

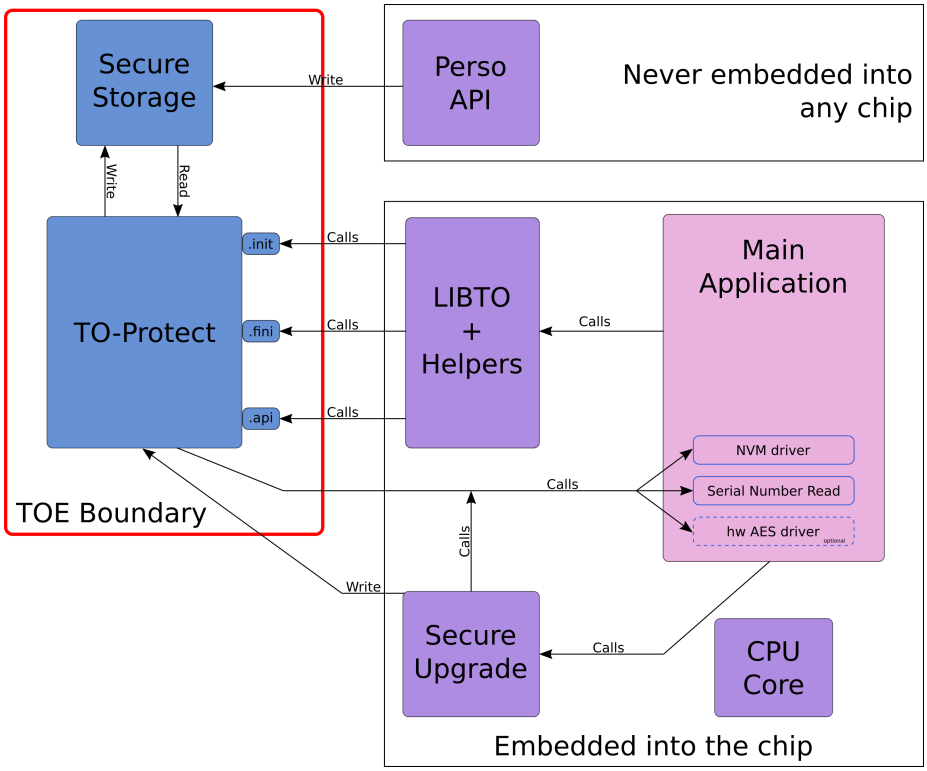

Fig. 1.1: The TOE boundary

In Fig. 1.1, we can see an application, using **TO-Protect** through **libTO**. **LibTO** is a software library, delivered as a source which is used to abstract the access to the secure element. We do not figure the physical protocol used to transport the TLS frames, this is under the applications responsibility to do this. **LibTO** will take the messages and perform the call to the right **TO-Protect** entry point.

1.4.2 TLS dialog example

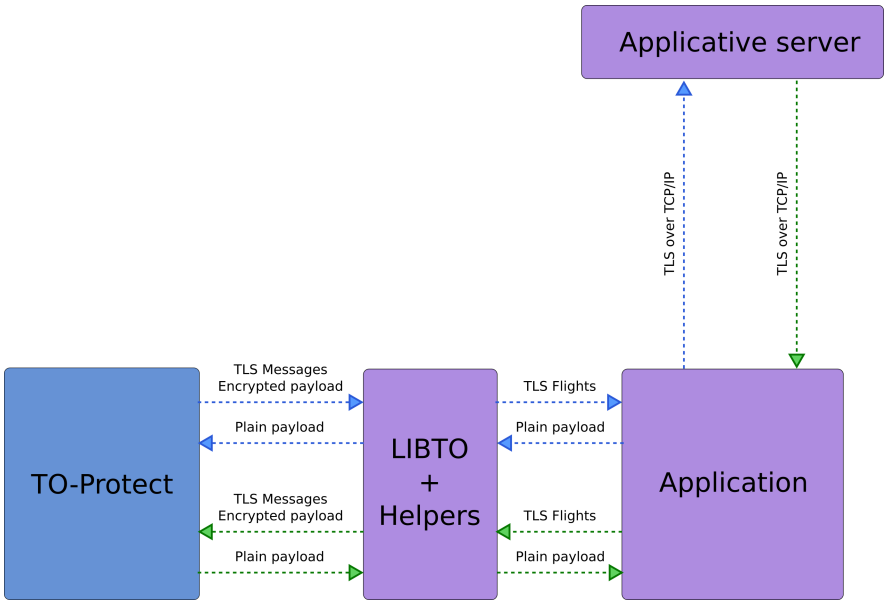

Fig. 1.2: Synthesis of a TLS dialog between a server and the applicative through **TO-Protect**

Fig. 1.2 shows how **TO-Protect** and the embedded application interact each-other in this case. This is only an example of what could happen if the application receives (green) or wants to send (blue) an encrypted payload from/to the server. Obviously, in the case of a complete session establishment, the dialog is more complex and would need much more interaction, but whats important here is the fact all the secrets (keys essentially) lie into **TO-Protect** secure storage and are manipulated by it.

1.4.3 The TOE Structure

1.4.3.1 TO-Protect

TO-Protect effectively consists in a binary library made of two main elements :

- The Vector table
  - Located at the very start of the binary and contains few information like the version, and some capability description (what are the options that were used to build the binary, roughly).
  - Contains the different *always present* entry points, like *init*, *fini*, as well as an entry point for each *api* included into the product.
  - This tables size is predictable, no change into an api has any visible impact on the table. Adding or removing an api entry-point will NOT result in any change in the vector tables structure, simply the api dispatcher will route the call either to an error message (in the case entry-point has been removed) or to the newly-introduced feature.
- **TO-Protect** code itself
  - Consisting in the different api dispatchers and the related functionality, performing the necessary actions.

  - Including the secure-storage management
  - Performing all cryptographic computations

- **TO-Protect** security benefits
  - Confidentiality : The cryptographic algorithms that **TO-Protect** implements are used to encrypt/decrypt all your dialog with the servers, thus ensuring your data are not observable.
  - Authenticity : Each **TO-Protect** instance can be personalized individually, providing your device a unique personality which can be used to authenticate it.
  - Integrity : **TO-Protect** ensures that every message sent through the TLS channel has been transmitted as requested.

1.4.3.2 The secure storage

The **secure storage** is the place where all security-relevant assets are placed. It is a piece of memory, located into NVM, which will contain the following data elements, stored in a non-readable manner and entirely shuffled differently each time one of these data elements is changed :

- The main DRBG implements an Hash_DRBG, as specified in [Hash-DRBG], which is used to generate a different random seed each time **TO-Protect** is restarted (for instance, at each power-up of the device). This DRBG, which is one of the first things that **TO-Protect** uses when performing its initialization is not observable, and is only used to generate the different seeds that will be used later on during the session (eg. generate randoms, keys, of simply *good enough* values used to wipe or randomize buffers. The original Seed is set in personalization mode.
- The **public** and **private key** elements
  - Those keys are stored into the secure storage, but have their own masking and integrity mechanism, in order to ensure that not only they are safe in NVM, but also when transfered into RAM for conducing cryptographic computations
  - If selecting a PSK-only security, those public and private keys may not be present in the product
- The **PSK**, which security is similar to what is made with the public-keys elements.
- The TLS keys and assets for every available communication slot
  - The **master-secret**, resulting from a successfully opened session, and which is used to resume it
  - The **session keys**, used to encrypt/decrypt and sign/verify the exchanged payloads
- The servers **public key**
  - The X509-certificate issued for this server
  - Its **public key** (extracted from the X509 certificate)
  - Its common name
- The main **CA**
  - The **public key** which will be used to authentify any X-509 certificate
  - The related key identifier
- The **admin keys**
  - The encryption/decryption/mac keys

  - The associated capabilities associated to each key

- The **upgrade keys**
  - These keys are device-specific and used to perform the upgrade to a newer **TO-Protect** version

1.4.3.3 The perso API

This API is not part of **TO-Protect**, and is used to create the secure-storage first content, during the personalization phase. It is not available to the end-customer either. It is only listed here for explaining how the secure-storage is first created. This feature is not deactivated, it is simply not present on the final product as the secure-storage is loaded already provisioned.

1.4.3.4 The secure Upgrade

The secure upgrade is not part of **TO-Protect**, and is used solely in case of an upgrade of the **TO-Protect** binary to be made on the field, for working-around a security vulnerability or introducing a new functionality. It performs the update in a secure way, ensuring :

- The new version is newer than the older one, it is not possible to go back to an older release.
- Confidentiality : The new content is encrypted
- Integrity : The upgrade package is digitally signed
- Authenticity : The signing key is solely known to Trusted-Objects

1.4.3.5 LibTO and the helpers

LibTO and the helpers are used to extract the different TLS messages, identify them, extract the information that is really relevant to **TO-Protect**, and finally call the right entry-point.

1.4.3.6 The application

This is the application that will benefit from **TO-Protect**, it is implementing the protocol used to transmit the TLS messages (eg. TCP, TCP over USB, wifi etc.).

1.4.3.7 The hardware platform

All these software elements run on a hardware platform, not figured here, but supposingly implementing an ARM Cortex CPU (Cortex-M0/3/4/7/23/33/35P/55). No specific security features are expected from it, simply the presence of embedded NVM is required.

2. Security Objectives for the Operational environment

In order for **TO-Protect** to fulfil its security requirements, we demand that the operational environment achieves the following.

2.1 Platform Objectives for the Operational Environment

UNIQUE_INSTANCE

Each instance of **TO-Protect** must be associated with a unique content of its secure-storage. During production, secure-storage content is not re-used. This is explained in the *Secure provisioning and preparative procedures (AGD_PRE.1)* section.

DEBUG_DISABLED

At the last stages of production, any debug/bootloader feature must be either deactivated or configured in a way it disallows the direct reading of the TOE. This is explained in the *Securing the presence of a debug or Bootloader functionality* section.

TRUSTED_PRODUCTION

The TOE is able to protect the assets after their loading, under the strict condition of a trusted production site. This is explained in the *Secure provisioning and preparative procedures (AGD_PRE.1)* section.

SECURE_UPGRADE

With each new upgrade of **TO-Protect** comes some recommendations and source code to be integrated in order to perform a correct platform upgrade. These procedures should be followed cautiously and strictly in order to maintain the security of the assets. They are described in the [Upgrade] document.

2.1.1 Securing the presence of a debug or Bootloader functionality

In order for **TO-Protect** to fulfil its security requirements, you **MUST** comply with the following :

- The JTAG, or any debug capability must be configured in such a way that either it is not possible to enter into such a mode, or that when entering it, the whole **TO-Protect** code and secure storage content is erased.
- Any Bootloader, or hardware mechanism that may be used to download from the chip the NVM content, or to reprogram it partially has to be deactivated.

2.1.2 Secure provisioning and preparative procedures (AGD_PRE.1)

Part of the life-cycle of the product is the provisioning, at which moment the personalization is performed. This specific and critical part of the products production is entirely under Trusted-Objects responsibility, the end-user can only provide his requirements (for instance, the Private/Public key pairs for the device, the CA public key etc). Doing this way, we consider that part of the preparative procedures, which will avoid any security malfunction of the product are :

- Generation of a DRBG seed using a certified hardware (eg. a HSM)
- Ensure that a secure-storage content cannot be used more than once, thus ensuring of the unicity of each **TO-Protect** instance

As those two points are taken into account outside of the TOE, and therefore outside of the SESIP certification, the only case which may happen is the case where the secure storage is defective or has not been programmed in the chips NVM. In this case, as stated in the *Included guidance documents* in the **TOP_init()** entry-point description, the initialization of the TOE will fail, making it inoperant.

When the TOE has been provisioned, it is possible to check its identification and software version as stated in *Verification of Platform Identity*.

3. Security requirements and implementation

3.1 Security Assurance Requirements

The claimed assurance requirements package as follows, as described in [SESIP].

Table 3.1: SESIP1 sufficiency table

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
AGD: Guidance documents	**AGD_OPE.1** operational user guidance **AGD_PRE.1** Preparative procedures
ALC: Life-cycle support	**ALC_FLR.2** Flaw reporting procedures
AVA: Vulnerability Assessment	**AVA_VAN.1** Vulnerability survey

3.1.1 Flaw Reporting Procedures (ALC_FLR.2)

In the document [ALC_FLR], the procedure used to report, evaluate and communicate about a potential defect or vulnerability. Associated to this an Errata [Errata] exists, which lists all known defects about the TOE.

3.1.2 Vulnerability Survey (AVA_VAN.1)

In accordance with the requirement for a vulnerability analysis survey (AVA_VAN.1) the developer has performed a vulnerability survey and submits the following test results to demonstrate the consideration of publicized potential vulnerabilities relating to the TOE:

Table 3.2: Vulnerability survey

Element	Responsible	Information source	Monitoring method
C Cross-compiler	GNU	https://www.gnu.org/software/gcc/	Monitor the discussion forum. Check for new releases and change logs.
Security attacks	None	https://iacr.org/	Follow scientific activity around the new security weaknesses of cryptographic algorithms, other software implementations, hardware chips.
Security standards	FIPS	https://csrc.nist.gov/publications/detail/fips/140/3/final	Follow standard evolutions, only use secure algorithms in a way they ensure security.
Security standards	ANSSI	https://www.ssi.gouv.fr/administration/reglementation/confiance-numerique/le-referentiel-general-de-securite-rg	Follow good practices in order to avoid known traps. Make good and robust usage of cryptography.
Security knowledge	None	https://www.keylength.com/	Ensure key lengths and algorithms will be sufficient to allow data protection in the desired time scope.
TLS recommendations	ANSSI	https://www.ssi.gouv.fr/guide/recommandations-de-securite-relati	Make sure that a secure usage of TLS is made
RFC 7457	IETF	https://datatracker.ietf.org/doc/html/rfc7457	Ensure that the known attacks over TLS/DTLS have been taken into account correctly.

3.1.3 Survey of potential vulnerabilities

In this section, we will list the possible vulnerabilities as they appear to us, trying to build a hierarchy of possible attacks, and finally explain why those do not apply to our TOE.

3.1.3.1 Out-of-scope vulnerabilities

Due to the software-only nature of the TOE, and the fact we do not want to impose a list of chips able to run it, we turn this around by specifying constraints, and therefore limitations on this platform. The security being the result of both the TOE and the selection of the chip, as well as its correct configuration, we will consider out-of-scope all the following vulnerabilities :

- Exploit an embedding application vulnerability (for instance a buffer tampering, overflow, stack smashing etc). Guidance will be given to perform code reviews, implement counter-measures, etc. We cannot be taken responsible for something on which we have no control.
- Exploit a hardware vulnerability, which could for instance make possible the reading of the Embedded memory using physical methods, like FIB[1], EFM[2], or any micro-probing technique[3].
- Exploit a mis-configuration of the hardware device. For instance, if the developer forgets to deactivate the JTAG[4] bus on the product, a hacker could plug a probe and manage to get the full Non-volatile memory content, putting the customers assets in danger. Guidance is given not to forget about this, but we cannot proceed any further.
- Exploit a mis-configured or weak bootloader[5] which would allow the non-volatile memory content to be extracted, guidance is given not to forget about this, but we cannot proceed any further.
- Extract the TOE code or the secure-storage content from the non-volatile memory of the micro-controller

[1] Focused Ion Beam. This is a common technique used to debug/modify a silicon circuit by modifying physical structures to bring a new functionality or create an easy probing point. At the early ages of the silicon industry, it was even used to create complete circuits for sampling/engineering.

[2] Electrostatic Force Microscope . This hardware could be used to detected charges by scanning the surface of a micro-circuit, and therefore give a clue on the content of a non-volatile memory cell.

[3] Micro-probing techniques (involving either tiny metal probes or lasers) can be used to probe signals (measure or force a potential on a metal line). This can be used to read data live and therefore directly get a bit value even on buried signals.

[4] Bus originally designed for testing purposes, which has evolved to allow also debug techniques (placing breakpoints, reset the chip, dump/set cpu registers etc). Very convenient for the developer, it is the main backdoor that has to be closed at the end of the production in order to seal the product. Different sealing options may exists depending on the Silicon vendor, which open to different options being available. We cannot have any influence on those choices, we simply require to use a security level making the extraction of non-volatile memory impossible (for instance on some chips, the JTAG may remain available, but NVM is wiped before the access is authorized).

[5] Piece of code, generally residing into ROM, which can be used to download/upload/verify the non-volatile memory content. It is mainly used for maintenance or production purposes. In most cases, it can be locked in a way it is not usable when getting out of the production site. Guidance will require the direct download of the firmware is not possible with the Bootloaders setting in the field.

3.1.3.2 Considered vulnerabilities

- Extracting assets from the secure storage.
  - The secure storage content is out-of-reach of an attacker (see *Out-of-scope vulnerabilities*), but still we consider the case where an attacker would get access to it.
- Removing the power supply when non-volatile memory write occur to jeopardize the secure-storage content.
  - The TOE is equipped with an anti-tearing mechanism, ensuring that the secure-storage content is always in a consistent state. It therefore disallows this kind of attacks.
- Physical attacks on the TLS protocols implementation (ECDHE-ECDSA case)
  - see section *During the Handshake (using ECDHE-ECDSA)*
- Physical attacks on the TLS protocols implementation (PSK case)
  - see section *During the Handshake (using PSK)*
- Physical attacks on the TLS protocols implementation (Abbreviated handshake either ECDHE-ECDSA or PSK)
  - see section *Abbreviated handshake, using either ECDHE_ECDSA or PSK*
- Physical attacks on the TLS protocols implementation (Payload encryption to the server)
  - see section *Exchange of encrypted payloads (client to the server case)*
- Physical attacks on the TLS protocols implementation (Payload decryption by the client)
  - see section *Exchange of encrypted payloads (server to the client case)*

3.2 Security Functional Requirements

3.2.1 Attestation of platform genuineness

The platform provides an attestation of the application, in a way that ensures that the application cannot be cloned or changed without detection.

Refering to [Manual], and more precisely **TOP_authenticate()**. It lets the host application to challenge the authenticity and integrity of **TO-Protect**. The method employed need the hosting application to generate a random buffer and challenge the capability of TO-Protect to correctly reply the expected response. This challenge involves knowing a secret key, and being able to compute the expected integrity value (computed and checked in **TOP_init()**). Not passing this test mandates the host application to stop using this TO-Protect instance.

3.2.2 Verification of Platform Identity

The platform provides a way to identify it, including all its parts and their versions.

Refering to [Manual], and more precisely **TOP_get_product_number()** and **TOP_get_software_version()**. Using both functionalities will give you the full identification of the platform.

For making sure the correct product, corresponding to this certification is used :

- **TOP_get_product_number() MUST** give you back **TOPR-TP-01-2.1.6**.
- **TOP_get_software_version() MUST** give back Major= **2**, Minor= **1** and Revision= **6**

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.

TO-Protect is not able to update itself, but the application using it can simply replace it as a whole. The embedding platform has to perform the upgrade as described in the [Upgrade] document, which include decrypting and checking the integrity of **TO-Protect**, and finally check the return value of the **TOP_init()** entry-point each time it is started to be used. When starting to use **TO-Protect**, it is possible to get the full identification of **TO-Protect** through :

- **TOP_get_product_number()**, **TOP_get_software_version()** will give back the full identification (including software capability and version).
- **TOP_get_software_integrity()** will check and give back the integrity information, after making sure that the software integrity is correct.

It is also possible to make sure it is authentic, by using **TOP_authenticate()**.

3.2.4 Secure Initialization of Platform

The platform ensures its authenticity and integrity during platform initialization. If the platform authenticity or integrity cannot be ensured, the platform will go to **Internal Error**.

During the execution of the **TOP_init()** entry-point, the integrity of the binary of **TO-Protect** is verified, as well as the integrity of the **secure storage**. **TOP_init()** if the first entry-point to be called, no other can be invoked before it. The integrity of the **TO-Protect** code includes both an integrity check, as well as a check using a signature mechanism.

3.2.5 Attestation of Platform State

The platform provides an attestation of the state of the platform, such that it can be determined that the platform is in a known state.

This is made by the **TOP_init()** entry-point, which gives back a status, interpreted as a verdict about the correct state of **TO-Protect**.

3.2.6 Secure Communication Support (ECDHE_ECDSA case)

The secure communication channel authenticates **the platform and any external server** *and protects against* **disclosure, modification, replay, erasure, Man-in-the-middle** *of messages between the endpoints, using* **TLS and DTLS 1.2, with the cipher-suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256**.

TO-Protect effectively establishes a secure communication channel with an external TLS server. During the establishment of the secured communication, a mutual authentication of both the server and the device is performed. This authentication is done at two moments during the handshake :

It is performed during the *Server Key Exchange* and *Client Key Exchange* respective message processing. When doing so, the server and client exchange respectively an ephemeral ECDH key, which signature is verified, and which is also used to perform a challenge (in this case the *Client Hello* and *Server Hello* random buffers are used).

Table 3.3: Key sizes in the ECDSA/ECDHE case

Algorithm	Key length(s)
AES Encryption Keys	128 bits
HMAC Signature Keys	128 bits
ECC Ephemeral Key (for ECDHE)	256 bits, using secp256r1
ECC Key pair (for ECDSA)	256 bits, using secp256r1
HMAC Key	256 bits
Hash DRBG using SHA-256 (random number generator)	256 bits

3.2.7 Secure Communication Support (PSK case)

The secure communication channel authenticates **the platform and any external server** *and protects against* **disclosure, modification, replay, erasure, Man-in-the-middle** *of messages between the endpoints, using* **TLS and DTLS 1.2, with the cipher-suite TLS_PSK_WITH_AES_128_CBC_SHA256**.

TO-Protect effectively establishes a secure communication channel with an external TLS server. During the establishment of the secured communication, a mutual authentication of both the server and the device is performed. This authentication is done at two moments during the handshake :

It is performed when processing the respective client/server finished messages. In these messages, the master_secret is used as a secret key, used to sign (using HMAC) the different messages exchanged, including the two *Client Hello* and *Server Hello* random buffers.

Table 3.4: Key sizes in the PSK case

Algorithm	Key length(s)
AES Encryption Keys	128 bits
HMAC Signature Keys	128 bits
HMAC Key	256 bits
Hash DRBG using SHA-256 (random number generator)	256 bits

3.2.8 Physical Attacker Resistance

The platform detects or prevents attacks by an attacker with physical access before the attacker compromises any of the other functional requirements.

Refer to *Sca-def* anf *Fault-def* for more information about this.

4. Mapping and sufficiency rationales

4.1 SESIP1 sufficiency

Table 4.1: SESIP1 suffiency table

Assurance Class	Assurance Families	Covered by	Rationale
ASE: Security Target evaluation	**ASE_INT.1** ST Introduction	Section *Introduction*	The ST reference is in Section 1.1, the platform reference in Section 1.2, its overview and description is in Section 1.4
	ASE_OBJ.1 Security requirements for the operational environment	Section *Security Objectives for the Operational environment*	The requirements for the operational environment are described in Section 2
	ASE_REQ.3 Listed security requirements	Section *Security Assurance Requirements*	The security requirements are described in Section 3.1
	ASE_TSS.1 TOE Summary Specification	Section *Security Assurance Requirements*	All SFRs are listed per definition , and for each SFR the implementation and verification is defined in the SFR.
AGD: Guidance documents	**AGD_OPE.1** operational user guidance	Section *Included guidance documents*	The platform evaluator will determine whether the provided evidence is suitable to meet the requirement.
	AGD_PRE.1 Preparative procedures	Section *Secure provisioning and preparative procedures (AGD_PRE.1)*	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 Procedures (ALC_FLR.2)*	The flaw reporting and remediation procedure is described in Section 3.1.1
AVA: Vulnerability Assessment	**AVA_VAN.1** Vulnerability survey	Section *Vulnerability Survey (AVA_VAN.1)*	The vulnerability survey is described in Section 3.1.2

5. Potential vulnerabilities

In this section, we will go through the potential vulnerabilities we can meet in the scope of a TLS 1.2 implementation. We will, for each case describe the applicability of each attack scenario and the associated counter-measures that are used to counter the attack.

5.1 Defence against fault injection

Concerning the resistance against the Fault injection, protections have been implemented, which consist in (but are not limited to) :

- Randomizing buffers before/after use to disallow key zeroing, and wipe any trace of important data in memory.
- Program control flow constant surveillance
- Checking of all cryptographic computation results (avoid DFA-like attacks)
- Integrity checking of critical assets
- etc.

5.2 Defence against side-channel attacks

About the resistance against side-channel, we have developed a complete set of cryptographic software implementations, which embed counter-measures against side-channel attacks employing computation randomization and testing. These counter-measures are specific to each algorithm.

5.3 During the Handshake (using ECDHE-ECDSA)

During this handshake several operations are conduced, which can be summarized as follows (the expected order of the operations is respected). The Client and the Server will exchange data, starting by random numbers (the hello messages), which will be used later on to generate the **master_secret**. The next operation is the mutual authentication, during which each party will try to make sure that the other is really the one it pretends to be (the ECDHE + ECDSA part). Then the calculation of the p**re_master_secret** and **master_secret** to pursue with the calculation/checking of the finished messages, these messages being the hash of all previously exchanged messages. Finally, each party can compute the **key_block**, which will be used to establish the encryption/mac keys. Once this last hurdle is passed, both parties can exchange encrypted payloads.

In the following scenarios, we make the hypothesis that the attacker has a device in hands, which he wants to attack in a way he can break the security and recover the keys. Obviously, his ultimate goal being to grab the device private key, but, depending on the scenarios, some other intermediate key knowledge would also be interesting, therefore we try to sweep over as many scenarios as we can.

Table 5.1: Potential attacks on ECDHE_ECDSA handshake

Client side	Server side	Attack Type	Section	Rating	Rationale
Generation of random		Fault	Section 5.3.1	Low	Section 5.2
	Generation of random	N/A			
	Generation of Ephemeral key pair	N/A			
	Signing of the key pair	N/A			
Generation of a random private key		Fault	Section 5.3.2	High	Section 5.1
Generation of Ephemeral key pair		SCA	Section 5.3.3	High	Section 5.2
Generation of Ephemeral key pair		Fault	Section 5.3.4	None	Section 5.1
Signing of the handshake message		SCA	Section 5.3.5	High	
Verification of the servers certificate		Fault	Section 5.3.6	High	Section 5.1
	Verification of the clients certificate	N/A			
Computation of the ECDHE	pre_master_secret	SCA	Section 5.3.7	High	N/A
Computation of the ECDHE		Fault	Section 5.3.8	None	N/A
Computation of the master_secret		SCA	Section 5.3.9	High	N/A
Computation of the clients finished		SCA	Section 5.3.10	High	N/A
	Computation of the servers finished	N/A			
Checking of the servers finished		Fault	Section 5.3.11	None	N/A
	Checking of the clients finished	N/A			
Computation of the secret keys		SCA	Section 5.3.12	High	Section 5.2
Computation of the secret keys		Fault	Section 5.3.13	None	N/A

5.3.1 Fault attack on the clients random generation

This attack would target at forcing the client to generate a random value with a lesser entropy than expected. The impact of this attack is not so critical, as this random value is always used in conjunction with the servers random, minimizing the impact of this altered entropy regarding the key quality.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.3.2 Fault attack on the ephemeral random private key generation

This attack aims at corrupting the private ephemeral key generation, on the client side.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.3.3 Side-channel attack on the ephemeral public key generation

This attack aims at recovering the private ephemeral key, on the client side.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.4 Fault attack on the ephemeral public key generation

This attack aims at corrupting the public ephemeral key generation, on the client side. As far as we understand, this attack is non-sense, as both parties wont be able to agree on the same pre_master_secret, and therefore wont be in a position to communicate later on (in this case, the socket will be closed by at least one side).

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.5 Side-channel attack on the signing of all previous messages

This attack aims at recovering the static private key of the client.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.6 Fault attack on the servers certificate verification

This attack aims at forcing a client to either refuse a legitimate certificate (although this can be done by different ways, starting with corrupting the communication), or accept a non-valid servers certificate, being the base of a more sophisticated attack where either the server or the connection has been compromised (the Client thinks he talks to the right server, while it is not the case). Only this second scenario will be considered seriously.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.3.7 Side-channel attack on the ECDHE (pre_master_secret calculation)

This attack would target at recovering the clients private key by performing a side-channel attack on the scalar multiplication. Effectively, the client will take the servers public key (from the certificate), and multiply it with its own private key. Succeeding leads to the discovery of the clients private key and the recovery of all exchanges between the client and the server.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.8 Fault attack on the ECDHE (pre_master_secret calculation)

This attack would result in corrupting the pre_master_secret on the client side only. It has no interest, as the communication between the two parties will not be possible (they will not share the same pre_master_secret), and both the client and the server will close the connection after the first payload is received.

Impact on our product:

- This attack is rated as not feasible

5.3.9 Side-channel attack on the master_secret calculation

This attack would target at recovering the master_secret key by performing a side-channel attack on the PRF function. This attack involves to have a compromised server, which would always send the same ephemeral key, in order to keep the pre_master_secret identical from a computation to the other.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.10 Side-channel attack on the finished calculation and checking

For this scenario, we make the hypothesis that the client is requesting, under some specific circumstances an abbreviated handshake to the server. It is supposed that the hacker has an easy way of forcing the client to enter this negotiation. Under this hypothesis, this scenario is possible. The goal for the hacker being to gather as many power curves he needs to be able to implement its side-channel attack on master_secret, when involved in the finished calculations.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.11 Fault attack on the finished

This attack aims at corrupting either the finished message calculation (the one sent to the server), or the decision regarding the finished message received from the server (in the case of a compromised server).

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.3.12 Side-channel attack on the computation of the key_block

For this scenario, we make the hypothesis that the client is requesting, under some specific circumstances an abbreviated handshake to the server. It is supposed that the hacker has an easy way of forcing the client to enter this negotiation. Under this hypothesis, this scenario is possible. The goal for the hacker being to gather as many power curves he needs to be able to implement its side-channel attack on master_secret, when involved in the computation of the key_block.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.3.13 Fault attack on the computation of the key_block

For this scenario, we make the hypothesis of a fault being injected during the key_block computation. Effectively, this scenario is very unlikely to happen, as, as soon as the keys are used, if either the server or the client detects an error during the MAC computation, the socket will be closed.

This attack scenario is rated as **None**.

5.4 During the Handshake (using PSK)

Differently from what is made with ECDHE_ECDSA, the PSK handshake does not require any private or public keys to be made, simply each party has in hands a **pre_shared_secret** which will be used as a pre_master_secret. The dialog starts the same way, with the exchange of the hello messages, which random value will be used the same way than before to compute the master_secret. The remaining messages are identically computed to end-up with each party having the encryption/mac keys in hand, ready to exchange encrypted payloads.

Effectively, we do not see any things being really different from the ECDHE_ECDSA scenario, at least on an attacker standpoint, therefore we re-use all of the formelly explained attacks.

Table 5.2: Potential attacks on ECDHE_ECDSA handshake

Client side	Server side	Attack	Section	Rating	Impact
Generation of random		Fault	Section 5.3.1	Low	Section 5.1
	Generation of random	N/A			
Using the PSK	=> pre_master_secret	N/A			
Computation of the master_secret		SCA	Section 5.3.9	High	Section 5.2
Computation of the clients finished		SCA	Section 5.3.10	High	Section 5.2
	Computation of the servers finished	N/A			
Checking of the servers finished		Fault	Section 5.3.11	None	N/A
	Checking of the clients finished	N/A			
Computation of the secret keys		SCA	Section 5.3.12	High	Section 5.2
Computation of the secret keys		Fault	Section 5.3.13	None	N/A

5.5 Abbreviated handshake, using either ECDHE_ECDSA or PSK

The abbreviated handshake is a protocol capability that enables the establishment of new secret keys, while re-using the already established master_secret and newly generated random values. Based on the former master_secret, and using the newly exchanged hello messages, we can compute fresh encryption/mac keys.

Once again, the scenarios are not so different than before, and we simply re-se them.

Table 5.3: Potential attacks on abbreviated handshake

Client side	Server side	Attack	Section	Rating	Impact
Generation of random		Fault	Section 5.3.1	Low	Section 5.1
	Generation of random	N/A			
Computation of the clients finished		SCA	Section 5.3.10	High	Section 5.2
	Computation of the servers finished	N/A			
Checking of the servers finished		Fault	Section 5.3.11	None	N/A
	Checking of the clients finished	N/A			
Computation of the secret keys		SCA	Section 5.3.12	High	Section 5.2
Computation of the secret keys		Fault	Section 5.3.13	None	N/A

5.6 Exchange of encrypted payloads (client to the server case)

In this scenario, and the following one (server to client), we do not consider a compromised server:

- Due to the protocol robustness, it seems unrealistic or even useless, as a compromised server able to fake, without being detected, would necessarily need all keys to be broken. What would be the need to try to attack the data exchange in this case ?
- A partially compromised server has not been envisioned either. Indeed it could consist in a server having its credentials safe, but for which the protocol implementation has been biaised. For instance, it could forget to close the connection in case a MAC is detected to be wrong, thus allowing to run fault injection on the client side during the MAC computation. This type of scenario was bringing us too far in the analysis and was rejected although it could be more realistically setup under some circumstances.
- Any attempt to fake something, which results in a de-synchronisation between parties (in TLS) will result in the connection being closed. DTLS is more flexible, but the fact he disallows the second message with the same sequence to be taken into account closes a lot of possibilities and opens very few.

When the keys have been generated, some data exchange can start. For this part, we only support AES_CBC_128. Some protocol aspects have to be taken into account:

- The IV is sent first on the message (in plain)
- The MAC is computed on the plain text (including the IV) and not on the padding
- A padding is added to the plaintext message to fit with the block ciphers size
- The encryption covers the message, MAC and padding (the IV remains in plain)
- The padding consistency is checked a part, as it is not part of the MAC.

Just for memory, the structure of an exchanged message is the following:

Table 5.4: Payload encrypted/maced parts

Data Item	Header	IV	Data blocks (plenty)	MAC	Padding
Encrypted/Plain	Plain	Plain	**Encrypted**	**Encrypted**	**Encrypted**
MACed ?	**MACed**	**MACed**	**MACed**	Not MACed	Not MACed

Table 5.5: Potential attacks on message encryption

Client side	Server side	Attack	Section	Rating	Impact
Generation of random IV		Fault	Section 5.6.1	High	Section 5.1
Computation of the MAC		SCA	Section 5.6.2	Low	Section 5.2
Computation of the MAC		DFA	Section 5.6.3	Low	Section 5.1
Computation of the Encryption		SCA	Section 5.6.4	High	Section 5.2
Computation of the Encryption		DFA	Section 5.6.5	Low	Section 5.1
Computation of the Encryption		Fault	Section 5.6.6	High	Section 5.1
	Decryption of the message	N/A			
	Checking of the MAC	N/A			

5.6.1 Fault attack on the generation of the random IV

For this scenario, we try to influence the quality of the random IV value thats included into the message. Effectively the attackers goal is to manage to have several messages sent-over with the same IV, in order to identify identical messages. TLS should disallow any source of leakage on the transported payload, and the specification is clear (section 6.2.3.2 of [TLS12]) that this data must remain UNPREDICTABLE.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.6.2 Side-channel attack on the MAC computation

For this scenario, the attacker will try to attack the MAC computation using side-channel techniques. Unfortunately, it has no access to any data that can be MACed, and the resulting MAC either. Therefore this attack is not feasible unless the encryption key is broken first.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.6.3 Fault attack on the MAC computation

For this scenario, the attacker will try to attack the MAC computation using a fault injection, using various techniques. This will only be feasible once the encryption key has been obtained.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.6.4 Side-channel attack on the CBC encryption

For this scenario, the attacker will try to attack the CBC encryption using side-channel techniques. This attack is the first step to be made before being able to break the MAC key.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.6.5 Fault attacks on the CBC encryption (key recovery)

In this scenario, the attacker will try to attack the CBC encryption using a fault injection. The attack could seem undetectable, because the MAC is computed on the plaintext, not the ciphertext. By the way, a verification of the padding part is mandatory and, thanks to the chaining, will be most probably be altered when doing the fault injection. Another blocking aspect of the protocol is the fact that it will be impossible to get twice the same plaintext ciphered, as the IV is generated randomly each time. Unless the clients implementation is biaised, this attack will therefore not be possible to be setup.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.6.6 Fault attacks on the CBC encryption (data recovery)

In this scenario, the attacker will try to attack the CBC encryption using a fault injection in order to try to avoid it and therefore get the plaintext back, with or without the IV being xored with it. The attack could be undetectable because the MAC is computed on the plaintext, and not the ciphertext.

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

5.7 Exchange of encrypted payloads (server to the client case)

Table 5.6: Potential attacks on message decryption

Client side	Server side	Attack	Section	Rating	Impact
	Generation of random IV				
	Computation of the MAC				
	Computation of the Encryption				
Decryption of the message		SCA	Section 5.7.1	High	Section 5.2
Decryption of the message		Fault	Section 5.7.2	Low	Section 5.1
Checking of the MAC		SCA	Section 5.6.2	Low	Section 5.2
Checking of the MAC		Fault	Section 5.7.2	Low	Section 5.2

5.7.1 Side-channel attack on the CBC decryption

For this scenario, the attacker will try to attack the CBC computation using side-channel techniques.

Impact on our product:

- **TO-Protect** uses a side-channel resistant crypto library, and is therefore **immune** to this attack.

5.7.2 Fault injection attack on the MAC verification

For this scenario, the attacker will try to fault the MAC verification in order to try to have the client accept forged messages while the hacker does not knows the MAC key. It can be used to replay messages (in TLS with one fault during the MAC verification, in DTLS with two faults one to let him accept a message whose sequence is wrong and one to have him accept the message while the MAC is wrong).

Impact on our product:

- **TO-Protect** embeds protections against fault injection, and is therefore **immune** to this attack.

6. Security Impact Analysis

TO-Protect exists under 2 references, each of it targetting at a specific familly of ARM products :

- TOPR-TP-00-2.1.6 - Targets at any core in the ARM-v6m, ARM-v7m or ARM-v8m architectures (Namelly Cortex-M0+, M3, M4, M7, M23, M33, M35P and M55)
- TOPR-TP-01-2.1.6 - Targets at some cores in the ARM-v7m and ARM-v8m architectures (Namelly Cortex-M3, M4, M7, M33, M35P and M55)

The differences between the two implementations are essentially on the code size and some specific optimisations introduced to benefit from a performance boost when doing heavy cryptographic operations. The security level remains the same and the counter-measures have been validated in both cases with the same level of security and testing. As a consequence, we consider that the security impact of such changes are not introducing any security issue.

7. Configuration Management

Refer to the bibliography for a complete and exhaustive list of documents referenced in this security target.

8. Document History

Table 8.1: History

Author	Version	Date	Comment
Vincent Dupaquis	TODO001-A	21 dec 2021	First release
Vincent Dupaquis	TODO001-B	23 may 2022	Second release after lab comments
Vincent Dupaquis	TODO001-C	5 sep 2022	Changed the date format.
			Updated the list of referenced documents
			Added a specific section for the debug and bootloader in §2
			Added links to specific sections/documents in §2.1
			Added the product and revisions numbers for identification in §3.2.2
Vincent Dupaquis	TODO001-D	30 sep 2022	Changed the authentication.
Vincent Dupaquis	TODO001-E	14 oct 2022	Added the attacks document reference.
Vincent Dupaquis	TODO001-E	15 nov 2022	Minor typos fixed.

Bibliography

[SESIP] Security Evaluation for IoT Platforms. See https://www.trustcb.com/iot/sesip/

[TLS12] Transport Layer Security Protocol, Version 1.2. See https://datatracker.ietf.org/doc/html/rfc5246

[DTLS12] Datagram Transport Layer Security Protocol, Version 1.2. See https://datatracker.ietf.org/doc/html/rfc6347

[PSK-TLS] Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). See https://tools.ietf.org/html/rfc4279

[PSK-TLS-SHA256] Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode. See https://tools.ietf.org/html/rfc5487

[TLS-ECC] Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier. See https://tools.ietf.org/html/rfc8422

[TLS-ECC-SUITES] TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM). See https://tools.ietf.org/html/rfc5289

[AES] ADVANCED ENCRYPTION STANDARD (AES). See https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

[Hash-DRBG] Recommendation for Random Number Generation Using Deterministic Random Bit Generators. See https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[HMAC] HMAC: Keyed-Hashing for Message Authentication. See https://tools.ietf.org/pdf/rfc2104.pdf

[Extensions] Transport Layer Security (TLS) Extensions (list). See http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml

[TLS-Extensions] Transport Layer Security (TLS) Extensions. See https://datatracker.ietf.org/doc/html/rfc4366

[Manual] TODO002_E - TO-Protect 2.x User Manual. This document presents in detail the TO-Protect API.

[Upgrade] TODO005_B - Secure upgrade manual. This document explains how to upgrade TO-Protect when already deployed in the field.

[Admin] TODO003_A - Administration manual. This document presents the administration commands API, and the way to send secure administration commands to TO-Protect.

[ALC_FLR] TODO004_B - The Flaw reporting procedure. This document explains how to declare an issue, and how it is processed to end-up finding a solution.

[Errata] TODO006_B - TO-Protect 2.x Errata Sheet. This document lists all known issues in TO-Protect 2.x series.

[Attacks] TODO007_C - TO-Protect 2.x Attacks on TLS. This documents all the attacks taken into account.

[TLS-ANSSI] Recommandations de sécurité relatives à TLS. See https://www.ssi.gouv.fr/guide/recommandations-de-securite-relatives-a-tls

[TLS-weaknesses] Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). See https://datatracker.ietf.org/doc/html/rfc7457