CI Plus ECP Robustness Considerations, Attacks and Countermeasures

DISCLAIMER OF WARRANTY

Trusted Labs S.A.S., 6 rue de la Verrerie - CS 20001, 92197 Meudon Cedex, France
Tel: +33 1 30 97 26 20, Fax: +33 1 30 97 26 19, http://www.trusted-labs.com/

Table of Contents

Revision List

Ver.	Date	Status	Author(s)	Modifications
1.0	2017-10-31	Approved	Thanh-Ha Le Jean-Henri Granarolo Amira Barki	Publication
1.1	2018-03-05	Approved		Removed reference to audio buffers Document is marked as Confidential

1. Introduction

1.1. Purpose

The document “CI Plus ECP Robustness Considerations, Attacks and Countermeasures” assists the CI Plus Licensee in a more accurate understanding of the CI Plus ECP Robustness Rules. This document presents a non-exhaustive list of architectural design patterns compliant with the CI Plus ECP Robustness Rules and a set of relevant attacks and their associated countermeasures for these architectures.

1.2. References

[CI+RR]	CI Plus LLP, ILA Addendum for ECP – Exhibits ECP_B v1.1
[GP_TEE]	Global Platform Device Technology, TEE System Architecture, Version 1.1, January 2017 https://www.globalplatform.org/specificationsdevice.asp
[CC_Evaluation]	Common Methodology for Information Technology Security Evaluation – Evaluation methodology, Version 3.1, Revision 5, April 2017 https://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R5.pdf
[Movielabs_ECP]	MovieLabs Specification for Enhanced Content Protection – Version 1.1 http://movielabs.com/ngvideo/MovieLabs%20Specification%20for%20Enhanced%20Content%20Protection%20v1.1.pdf
[Trustzone]	https://www.arm.com/products/security-on-arm/trustzone

1.3. Acronyms

ASLR	Address Space Layout Randomization
CAS	Conditional Access System
CCP	Controlled Content Path
CI	Common Interface
CICAM	Common Interface Conditional Access Module
DFA	Differential Fault Attack
DMA	Direct Memory Access
DPA	Differential Power Analysis
DRAM	Dynamic Random-Access Memory

DTCP	Digital Transmission Content Protection
ECC	Error Correcting Codes
ECP	Enhanced Content Protection
EEPROM	Electrically Erasable Programmable Read-Only Memory
HDCP	High-Bandwidth Digital Content Protection
IOMMU	Input/Output Memory Management Unit
JTAG	Joint Test Action Group
MMU	Memory Management Unit
OS	Operating System
OTP	One-Time Programmable
PCB	Printed Circuit Board
PCIe	PCI Express
PIN	Personal Identification Number
PRNG	Pseudo Random Number Generator
RAM	Random Access Memory
RoT	Root of Trust
REE	Rich Execution Environment
ROM	Read Only Memory
RR	Robustness Rules
SoC	System-on-Chip
STB	Set-top Box
TA	Trusted Application
TEE	Trusted Execution Environment
UART	Universal Asynchronous Receiver Transmitter

2. Architectural design patterns

This section presents a list of widespread architectural design patterns. They are intended to support the Licensees to design the products that are compliant with [CI+RR].

2.1. Architecture blocks

This section defines the main architecture blocks that may be present in any CI Plus ECP compliant product. For each of these blocks, various implementations are possible. Concrete examples are detailed in the Design patterns section.

For each block, the Security Problem is defined as a list of critical assets associated with properties taken from the table below:

Property	Definition
Integrity	The assurance that the asset is uncorrupted and can only be modified by those authorized to do so. Integrity involves maintaining the consistency, accuracy and trustworthiness of the asset over its entire lifecycle.
Authenticity	The assurance that the asset originates from the source it claims to be from and has not been tampered with or altered. Authenticity involves proof of identity.
Confidentiality	The assurance that the asset cannot be read by an unauthorized component.
Non-modifiable	The assurance that no entity can change the asset.
Uniqueness	The assurance that the asset can be uniquely identified.
Device binding	The assurance that data can only be used on a unique device.
Unpredictability	The assurance that no one can guess the value of the asset.
Sufficient entropy[1]	The assurance of sufficient uncertainty about the generated random number.
Securely programmed	The assurance that the asset has not been altered when it is physically programmed (encoded in the circuit) during fabrication.
Irreversibly closed	The assurance that the deactivation action cannot be undone. This property is dedicated to debug ports.
Read only	The assurance that the asset can be accessed but cannot be modified. This property is dedicated to debug ports (e.g. UART).
Closed with password	The assurance that the asset is closed and a password is required to open it.
Access conditioned to a full memory wipe	The assurance that the asset cannot be accessed. If this happens, then the memory of the device is wiped.
Resistance to unsoldering	The assurance that the asset cannot be physically removed for analysis by an attacker.

[1]MovieLabs requires at least 128 bits of entropy for keys used to encrypt secrets in [Movielabs_ECP]

2.1.1. Secure Boot

The goal of the secure boot is to initialize the system in a secure way ensuring system integrity and authenticity. More precisely, a secure boot guarantees the following security properties:

- Integrity: any modification of the components loaded during the boot process should be detected and any modified component should not be loaded;
- Authenticity: any unauthenticated component should be detected and not be loaded.

An attacker able to compromise the integrity or authenticity of the system may be able to load a modified or forged image containing malicious features.

Boot sequences are typically implemented using several boot loaders. The first boot loader is loaded by a non-modifiable piece of code called the Boot ROM. The first boot loader should be authenticated by the Boot ROM using the hardware Root of Trust of the platform.

There are several ways to ensure the authenticity of that the first boot loader, including:

- Store the first boot loader in non-modifiable hardware (e.g. ROM);
- Store the first boot loader in Flash, but authenticate it and verity its integrity with a key stored in non-modifiable hardware (e.g. fuses or ROM).

Each boot loader after the first one should be correctly verified for integrity and authenticity by the previous stage.

Figure 1 illustrates an example of a secure boot process where the Root of Trust is stored in the ROM.

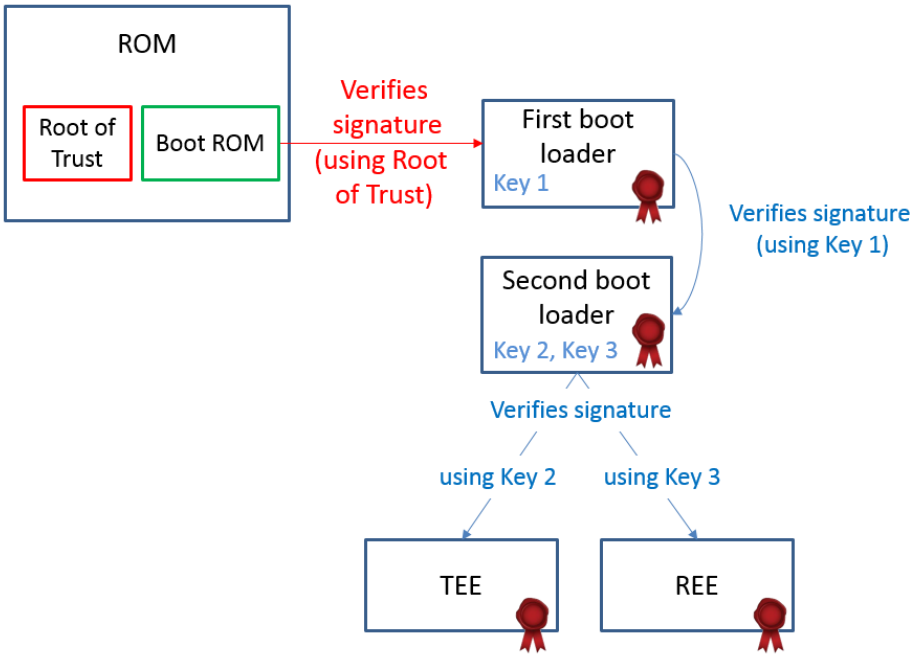

Figure 1: Example of secure boot

In case of failure during the secure boot, a safe state should be reached where no CI Plus functionality is available.

Table 1 below presents the assets of the Secure Boot and their properties as well as the potential attacks that may target these assets. A detailed description of these attacks is provided in Section 3.

Name	Asset	Definition	Security Properties	Potential Attacks
A_SB_RoT	Root of Trust	It is usually a public key (or its hash) that is stored in the on-SoC OTP or ROM and which is used to verify the integrity and authenticity of the first boot loader.	Non-modifiable Securely programmed	Secure Boot Root of Trust is modified during fabrication
A_SB_BL	Boot loader	The code and data of the boot loader.	Integrity Authenticity Confidentiality	Alter the normal execution of the boot loader (e.g. bypass the signature verification) to load an unauthorized software image. It can be done using Physical Fault Injection. Get boot loader image using Logical Memory Dump or Direct Memory Access. Re-flash the boot loader using Firmware Flashing (if the verification chain of the bootloader can be bypassed). "Confidentiality" is not mandatory. However, knowing the bootloader image will help the boot analysis easier.
A_SB_BLK	Boot loader keys	The keys used to verify the integrity and authenticity of the distinct stages of the boot process (aside from the first one). Each boot loader is signed and its code includes the key used to check the integrity and authenticity of the next boot loader.	Integrity	Modify the Boot loader keys to a known value to execute non-authorized code (instead of the legitimate Boot loader) via Physical Fault Injection or Firmware Flashing.

Table 1: Secure Boot Assets

2.1.2. Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure operating system running with higher privileges than the main – non-secure – operating system known as Rich Execution Environment (REE).

A Trusted Execution Environment guarantees the following security properties:

- Isolation: the TEE code and data should be protected from unauthorized access by the non-secure operating system. Moreover, the TEE code and data associated to a given Trusted Application (TA) (such as CI Plus Protection Functions) should not be accessible by another TA and vice versa;
- Integrity: the TEE should detect any alteration of loaded data;
- Confidentiality: the TEE confidential data should be protected from unauthorized access.

An architecture example of TEE is given in [GP_TEE].

There are several ways of implementing a Trusted Execution Environment, including:

- Software TEE;
- ARM TrustZone-based TEE;
- Dedicated secure processor TEE.

A system can contain several TEEs (e.g. a CI Plus TEE and other non-CI Plus TEEs). In this case, the CI Plus Trusted Boundary should clearly define which TEE the CI Plus features belong to, and the Licensee should consider the isolation between the different TEEs to ensure that non-CI Plus certified software cannot access CI Plus secrets.

Name	Asset	Definition	Security Properties	Potential Attacks
A_TEE_PF	Protection Functions (Except encryption/decryption performed in ECP Controlled Content Path)	The Protection Functions consist of Authentication, revocation, enforcement of Compliance Rules, maintaining the authenticity of Trust Values, maintaining the authenticity and Secrecy of Secret Values.	Authenticity Integrity Confidentiality	Get the code of the Protection Functions via External RAM Probing or Logical Memory Dump. Modify the code of the Protection Functions via Firmware Flashing. Alter the execution of Protection Functions (e.g. authentication) by using Physical Fault Injection or Code Injection. Retrieve secret data or key manipulated by Protection Functions using Physical Side-Channel Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack,

				Rowhammer Attack, Logical Memory Dump, Code Injection or Fuzzing. The property “confidentiality” is not mandatory. However, knowing the code of Protection Functions will help the analysis of these functions easier.
A_TEE_RD	TEE Runtime Data	TEE runtime data includes execution variables, runtime context, secure values of protection functions	Integrity Confidentiality	Get secret data manipulated by TEE using Physical Side-Channel Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack, Rowhammer Attack, Logical Memory Dump, Code Injection or Fuzzing Alter TEE runtime data using Physical Fault Injection, Rowhammer Attack, Code Injection.
A_TEE_PD	TEE persistent data	TEE persistent data includes TEE crypto keys (e.g. keys to authenticate TA) and trusted application properties.	Authenticity Integrity Confidentiality	Retrieve confidential data using Access to Debug Interfaces, Logical Memory Dump or Code Injection. Modify TEE persistent data via Firmware Flashing, Firmware Rollback, Access to Debug Interfaces.
A_TEE_FW	TEE Firmware	It contains TEE code and constant data.	Authenticity Integrity	Modify TEE Firmware via Firmware Flashing, Firmware Rollback to install a malicious or an old vulnerable firmware on the device.
A_TEE_ICD	TEE Initialization Code and Data	Code and data used until the complete activation of TEE security services.	Integrity	Alter the activation of TEE using Physical Fault Injection, Access to Debug Interfaces, Rowhammer Attack, Code Injection.
A_TEE_HC	Hardware Component	The hardware component used to host the firmware TEE	Resistance to unsoldering	Modify the firmware using Firmware Flashing attack.

Table 2: TEE Assets

a. Software TEE

The CI Plus ECP Robustness Rules [CI+RR] state that TEE should be hardware-enforced. Thus, a pure software TEE will not be compliant with ECP robustness rules [CI+RR].

b. ARM TrustZone TEE

ARM TrustZone [Trustzone] is a technology allowing the implementation of a Trusted Execution Environment on general purpose ARM processors. It ensures isolation between the REE and the TEE by adding a specific bit called the “NS” (Non-Secure) bit on the AXI bus connecting the different hardware blocks of the SoC. Each block can be configured to accept or refuse requests based on their origin (TEE or REE). This allows, in particular, the separation of the memory into secure and non-secure regions.

c. Dedicated Secure Processor TEE

A dedicated Secure Core only implementing the Trusted Execution Environment guarantees the highest level of security for a TEE. In particular, it ensures that there is no shared cache between the REE and the TEE (as opposed to the ARM TrustZone architecture).

However, this architecture requires a dedicated mechanism to ensure that all the relevant peripherals are aware of the separation between the REE and the TEE. It also requires a way to configure the peripherals accordingly to react appropriately to requests coming from the general-purpose processor and the secure processor.

2.1.3. Secure Storage

A secure storage is a dedicated non-volatile memory area that guarantees the following security properties on the data it contains:

- Integrity: the secure storage should detect any modification (logical or physical) on any data stored inside of it;
- Confidentiality (for secret values): the data stored in the secure storage should be encrypted in order to prevent the physical memory dump and reverse;
- Anti-cloning: the secure storage should be device bounded;

Name	Asset	Definition	Security Properties	Potential Attacks
A_SS_RoT	Storage Root of Trust	The storage RoT binds stored data and keys to the device. It should have at least 128 bits of entropy.	Integrity Confidentiality Device binding	Get the storage RoT via Physical Side-Channel Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack, Rowhammer Attack, Logical Memory Dump, Code Injection
A_SS_SV	Secret Values	Secret values minimally include	Authenticity Integrity	Get secret values via Physical Side-Channel

		Device Keys, Content Keys, Intermediate Keys and Protocol Secrets.	Confidentiality Device binding	Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack, Rowhammer Attack, Logical Memory Dump, Code Injection, during execution when the secret value is in clear. Secret values are considered as encrypted (by the storage RoT) in storage. Modify secret values during execution using Physical Fault Injection, Rowhammer Attack, Code Injection Retrieve secret values and use them on a different, less secure device.
A_SS_TV	Trust Values	Trust Values include Certificates, Revocation Information, Critical Security Update Version and SRM.	Authenticity Integrity	Modify trusted values using Physical Fault Injection, Rowhammer Attack, Code Injection

Table 3: Secure Storage Assets

2.1.4. Cryptographic engine

A cryptographic engine is used to perform cryptographic computations (e.g. PRNG, encryption/decryption or signature) faster and more securely than by using the main or the secure processor. It can also have its own internal ROM and fuses to store immutable keys securely.

There are multiple ways to implement a cryptographic engine, including:

- A dedicated special purpose CPU on the SoC;
- A coprocessor on the PCB.

Name	Asset	Definition	Security Properties	Potential Attacks
A_CE_PRNG	PRNG	Pseudo Random number generation operation. It may rely on hardware and/or software mechanisms.	Unpredictability Sufficient entropy	Alter the pseudo random number generation operation or outputs using Physical Fault Injection, Code Injection.

A_CE_OPE	Cryptographic operations	Crypto operations are associated to Secret Values, code and execution.	Integrity Confidentiality	Retrieve Secret Values using Physical Side-Channel Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack, Rowhammer Attack, Code Injection, Logical Memory Dump. Alter the execution of crypto operations using Physical Fault Injection**Error! Reference source not found.**. The goal is to get Secret Values or to skip cryptographic operations.
A_CE_KEY	Secret keys	The secret keys manipulated by the crypto engine.	Authenticity Integrity Confidentiality	Retrieve Secret Keys using Physical Side-Channel Analysis, Physical Fault Injection, External RAM Probing, Access to Debug Interfaces, Direct Memory Access, Cache Attack, Rowhammer Attack, Code Injection, Logical Memory Dump.

Table 4: Cryptographic Engine Assets

2.1.5. ECP Controlled Content Path

The ECP Controlled Content Path (CCP) is the path followed by the ECP Controlled Content from reception by the Host until storage, export to network or display on the user screen. This path should implement end-to-end security in the sense that non-encrypted ECP Controlled Content should never be accessible from outside the CI Plus Trusted Boundary.

The content itself is mainly video content. This content usually undergoes several transformations before being finally displayed to the user, including:

- Decryption;
- Re-protection;
- Decoding/Transcoding;
- Composition;
- Rendering.

There are several ways of implementing a secure ECP Controlled Content Path:

- Each operation can be implemented by a separate hardware block;
- All operations can be performed in isolated regions of the RAM.

The ECP Controlled Content Path should at least guarantee that:

- The outside of the CI Plus Trusted Boundary (which includes the REE) cannot access unencrypted data in any of the video buffers;

- The inside of the CI Plus Trusted Boundary (which includes the TEE) should have the minimal required permissions, i.e. write access but no read access to unencrypted data in the video buffers when not required. For instance, if the transformations on video content are implemented by several hardware blocks, then the TEE may need write access to these blocks but the TEE should have no read access to the final video content (which may be unencrypted).

When leaving the CI Plus Trusted Boundary to be displayed on the screen or exported to network, the output should be protected using DTCP-IP, HDCP 1.4 or HDCP 2.2.

It is important that the RAM of the ECP Controlled Content Path is protected against physical data extraction, such as by using encryption or scrambling. The RAM may also be protected against software attacks that may target the REE using memory filtering and memory partitioning. The configuration of the memory partition and memory filters should be securely controlled.

The encryption/scrambling should take place before the data leaves the main SoC to prevent bus probing, i.e. prevent an attacker from acquiring unprotected content during transit from the SoC to the RAM.

Name	Asset	Definition	Security Properties	Potential Attacks
A_CCP_MC	Media Content	Unencrypted media content. It can be compressed or uncompressed.	Confidentiality	Recover media content via External RAM Probing, Direct Memory Access, Logical Memory Dump, Code Injection.
A_CCP_FW	CCP Firmware	The code and static data of the CCP Firmware.	Integrity Authenticity	Install a malicious or vulnerable firmware by Firmware Flashing, Firmware Rollback.
A_CCP_RD	CCP Runtime Data	The runtime data generated and used during the execution of CCP firmware. CCP runtime data can be parameters to configure or control memory filtering, memory partitioning, content encryption/scrambling during transformations. The integrity of CCP runtime data is necessary to detect modifications that might lead to leakage of content buffer data.	Integrity	Alter CCP runtime data via Physical Fault Injection, Rowhammer Attack, Code Injection.

Table 5: ECP Controlled Content Assets

2.1.6. Debug ports

Debug ports are present on most modern PCBs and include JTAG and UART connectors. The UART debug port can help the attacker gather information about the boot sequence and in some cases, gain access to a command line interface on the device. The JTAG port can in theory provide unlimited access to the SoC, and by extension to any component (Flash, RAM, …) manipulated by the SoC.

The debug ports should not be open on CI Plus devices deployed in the field.

Common status of the debug ports includes:

- UART:
  - Irreversibly closed;
  - Read only access.
- JTAG:
  - Irreversibly closed (e.g. fuses burned);
  - Closed with password (e.g. requires a specific signed activation command or a secret password);
  - Access conditioned to a full memory wipe.

Name	Asset	Definition	Security Properties	Potential attacks
A_Debug_ UART	UART debug port	A port providing access to debug functions.	Irreversibly closed or Read only	Access directly to the debug interface Disable the protection mechanisms of this debug interface (e.g. using Physical Fault Injection to bypass the authentication) and access to the debug functions
A_Debug_J TAG	JTAG port	A port providing access to debug functions.	Irreversibly closed or Closed with password or Access conditioned to a full memory wipe	Access directly to the debug interface Disable the protection mechanisms of this debug interface (e.g. bypass the authentication using Physical Fault Injection) and access to the debug functions

Table 6: Debug Ports Assets

2.2. Design patterns

This section details the different architectural design patterns for the ECP CICAM as well as the ECP Host (i.e. Set-top box or TV).

In the following sections, the term CI Plus Trusted Boundary refers to the set of hardware and software blocks that implements the Protection Functions. That is mainly the ECP Controlled Content Path, the cryptographic engine and the TEE (if there is any). It is worth mentioning that the *Secure Storage* and the *Secure Boot* blocks are not included in the CI Plus Trusted Boundary, however these two blocks should be present in any CI Plus ECP product so as to be compliant with ECP Robustness Rules [CI+RR].

2.2.1. CICAM design patterns

Two possible architectural design patterns for the CICAM are described below:

- The CICAM makes use of a TEE (i.e. some CI Plus Protection Functions are implemented in software)
- There is no TEE implemented in the CICAM (i.e. pure hardware implementation of all CI Plus Protection Functions)

As shown in Figure 2, the ECP CICAM receives CAS Controlled Content that it first decrypts and then re-encrypts (using CI Plus Content Control Key established between the CICAM and the Host). The obtained encrypted content is then forwarded to the ECP Host.

In both cases (i.e. with and without TEE), the CI Plus Encryption (which is part of the CI Plus Protection Functions) is implemented in hardware.

a. CICAM with a TEE

As illustrated in Figure 2, the CI Plus Trusted Boundary consists of:

- The "CI Plus Encryption" Protection Function implemented in hardware and the Protected RAM which is secure by hardware security mechanisms. It is typically the termination of the CAS Controlled Content Path.
- A TEE where the CI Plus Protection Functions implemented in software are executed. The TEE may also be used to execute the Trusted Application of a CA System providing the key materials for the decryption of the CAS Controlled Content. Typically, the TA of the CA System also provides instructions to the CI Plus Protection Functions such as the usage rules and the applicable version of the revocation list. The Trusted RAM of the TEE is protected by the security mechanisms provided by the TEE.
- A crypto engine that is used by the TEE, the secure boot and the secure storage.

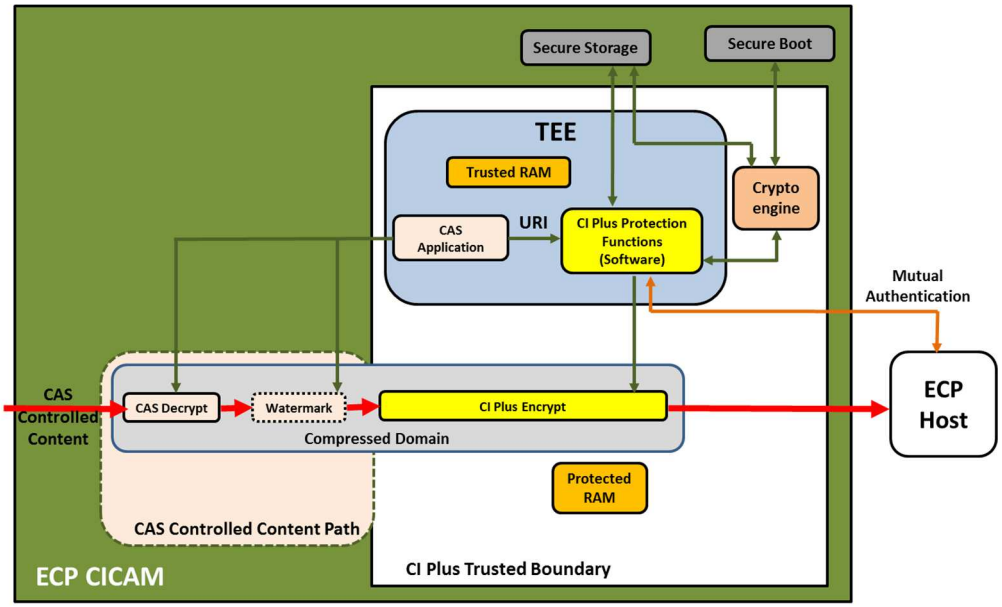

Figure 2: CICAM with TEE

b. CICAM without TEE

When there is no TEE in the CICAM, the CI Plus Trusted Boundary consists of:

- The "CI Plus Encryption" Protection Function which is implemented in hardware. It is typically the termination of the CAS Controlled Content Path;
- A set of secure hardware components that provide all the CI Plus Protection Functions aside from the CI Plus Encryption Protection Function. The Protected RAM is secure by hardware security mechanisms;
- A crypto engine that is used by the CI Plus Protection Functions, the secure boot and the secure storage.

However, considering the complexity of the Protection Functions, it is considered as practically challenging to implement this architecture.

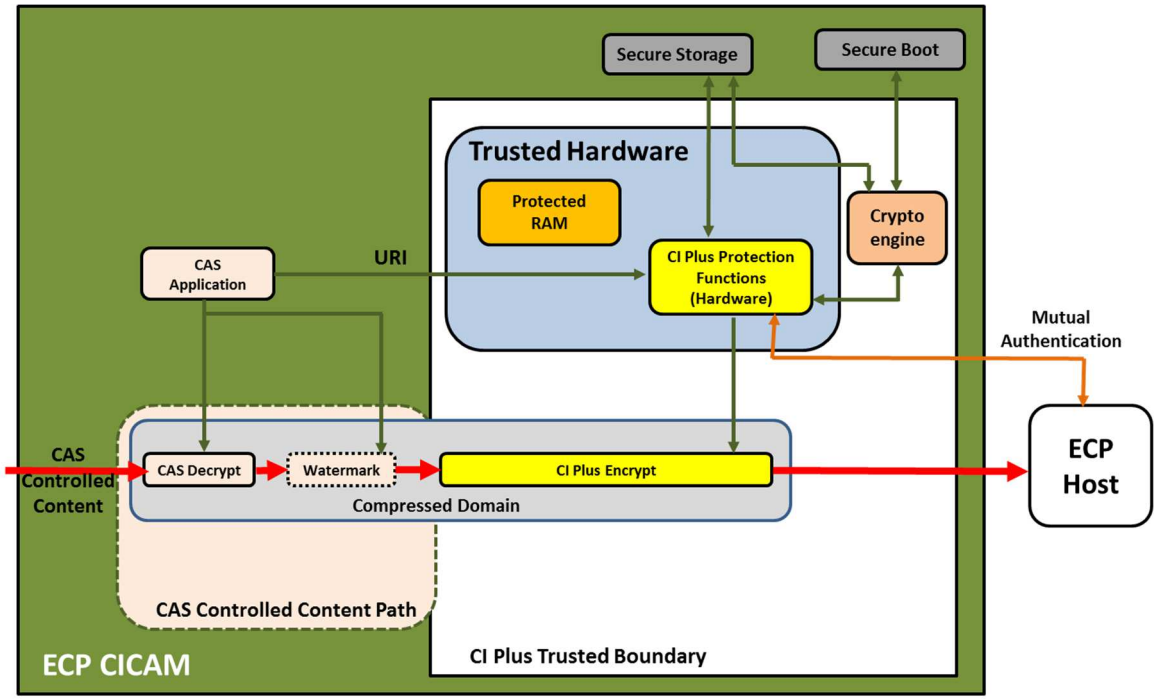

Figure 3: CICAM without TEE

2.2.2. Host design patterns

The Host design patterns can be classified into the following four main categories depending on whether the ECP content is exported out of the CI Plus Trusted Boundary (or not) and the way it is exported:

- Local consumption (i.e. the Host is the TV and the ECP content does not leave the TV: it is neither exported to a content storage nor to a HDMI nor to network);
- Export to Content storage;
- Export to HDMI output (i.e. the Host is a STB and provides the ECP content to the TV via HDMI);
- Export to network.

Regardless of the considered design pattern, the ECP Host always includes a TEE that performs most of the CI Plus Protection Functions.

Note: Protection Functions not implemented as software running in the TEE should be implemented in Hardware, as defined in the [CI+RR].

a. Local Consumption

As shown in Figure 4, the ECP Host receives from the ECP CICAM the CI Plus ECP Controlled Content that it first decrypts before decoding and displaying it on the screen.

When the Host is the TV and the ECP content is never stored, the CI Plus Trusted Boundary consists of:

- The CI Plus Controlled Content Path, that performs the “CI Plus Decryption” operation and content decoding;
- A TEE running all the CI Plus Protection Functions (to load the key necessary for decryption for instance) aside from the CI Plus Decryption Protection Function;
- A crypto engine that is used by the TEE, the secure boot and the secure storage.

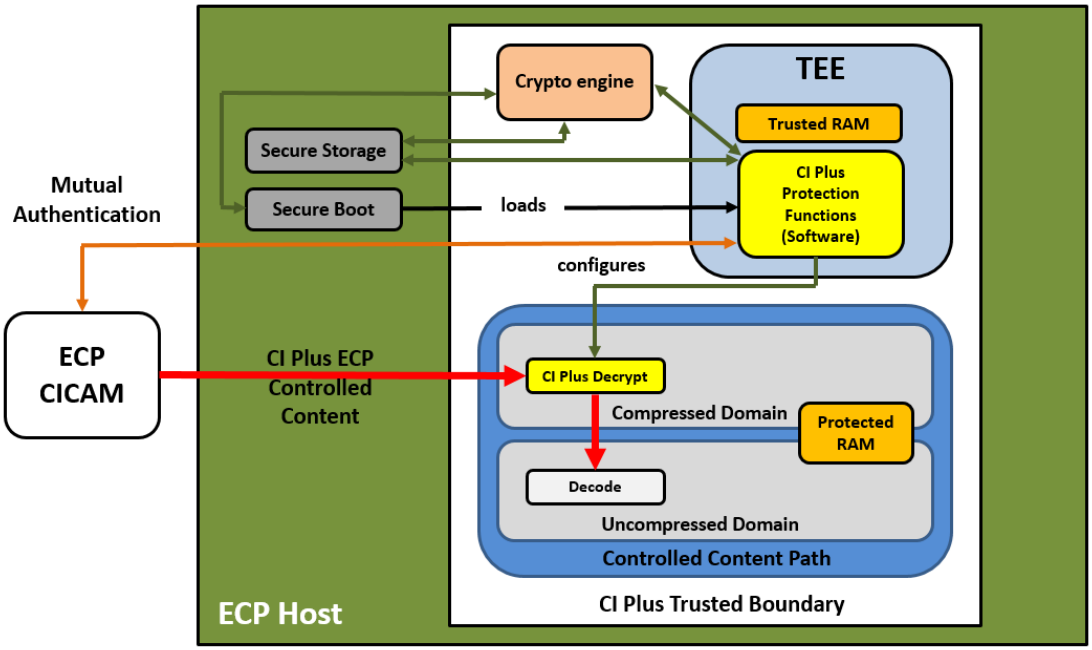

Figure 4: Host - Local Consumption

b. Export to Content Storage

As shown in Figure 5, the ECP Host receives from the ECP CICAM the CI Plus ECP Controlled Content that it first decrypts before either decoding and displaying it on the screen or re-encrypting it for storage on content storage.

When the Host is the TV and the ECP content may be stored on a content storage, the CI Plus trusted boundary consists of:

- The CI Plus Controlled Content Path, that performs the “CI Plus Decryption” operation, content decoding, displaying and/or re-encrypting for storage;
- A TEE running all the CI Plus Protection Functions (to load the key necessary for decryption and re-protection for instance) aside from the CI Plus Decryption Protection Function;
- A crypto engine that is used by the TEE, the secure boot and the secure storage.

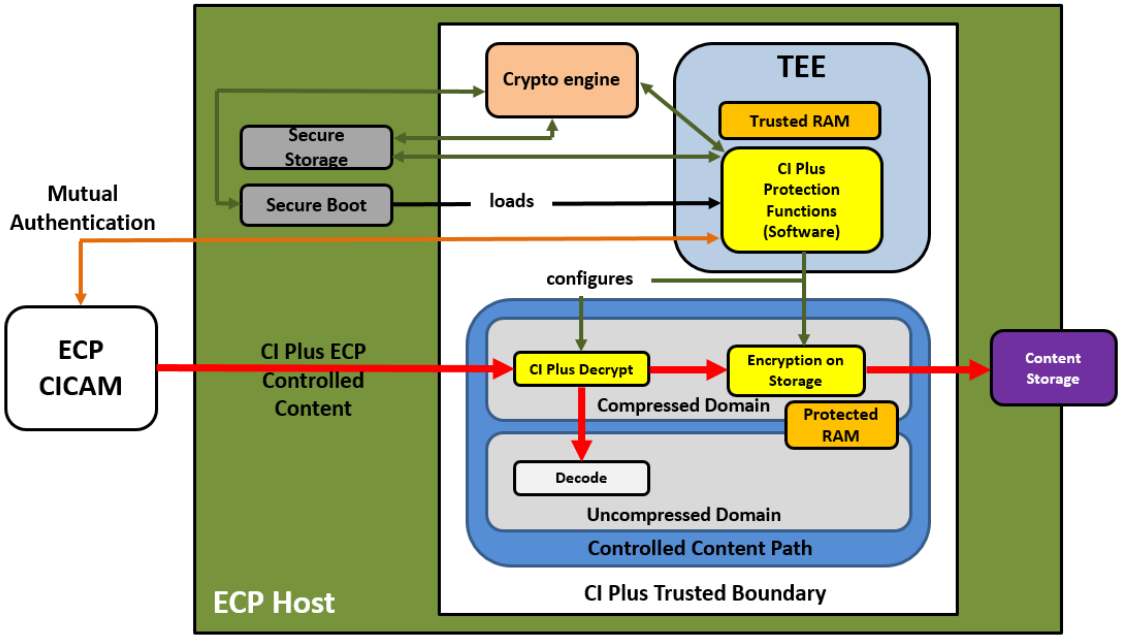

Figure 5: Host - Export to Content Storage

c. Export to HDMI output

As shown in Figure 6, the ECP Host receives from the ECP CICAM the CI Plus ECP Controlled Content that it decrypts, decodes, transforms and HDCP encrypts it before exporting it to HDMI output.

When the Host is a Set-top Box, the CI Plus trusted boundary consists of:

- The CI Plus Controlled Content Path, that performs the "CI Plus Decryption", content decoding, transformation and HDCP encrypting;
- A TEE running all the CI Plus Protection Functions (to load the key necessary for decryption and for controlling the activation of the HDCP protection for instance) aside from the CI Plus Decryption Protection Function;
- A crypto engine that is used by the TEE, the secure boot and the secure storage.

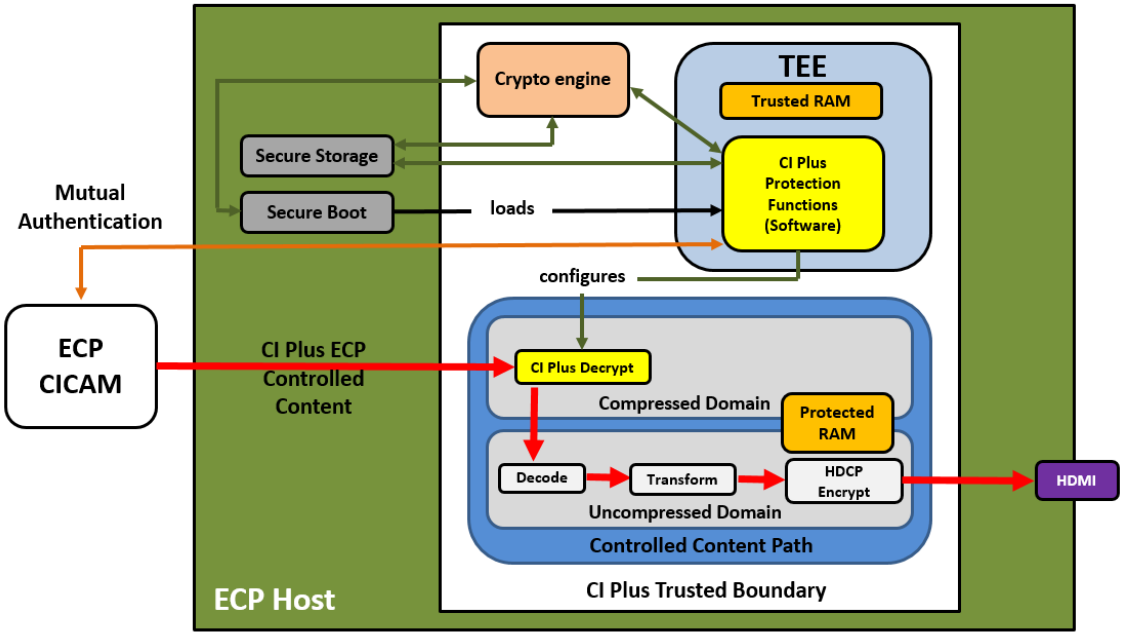

Figure 6: Host - Export to HDMI

d. Export to network output

As shown in Figure 7, the ECP Host receives from the ECP CICAM the CI Plus ECP Controlled Content that it first decrypts, then decodes and DTCP/HDCP encrypts it before exporting it to network output.

When the ECP Controlled Content is exported to network output, the CI Plus Trusted Boundary consists of:

- The CI Plus Controlled Content Path, that performs the "CI Plus Decryption", content decoding, transformation and HDCP/DTCP encrypting;
- A TEE running all the CI Plus Protection functions (to load the key necessary for decryption for instance) aside from the CI Plus Decryption Protection Function;
- A crypto engine that is used by the TEE, the secure boot and the secure storage.

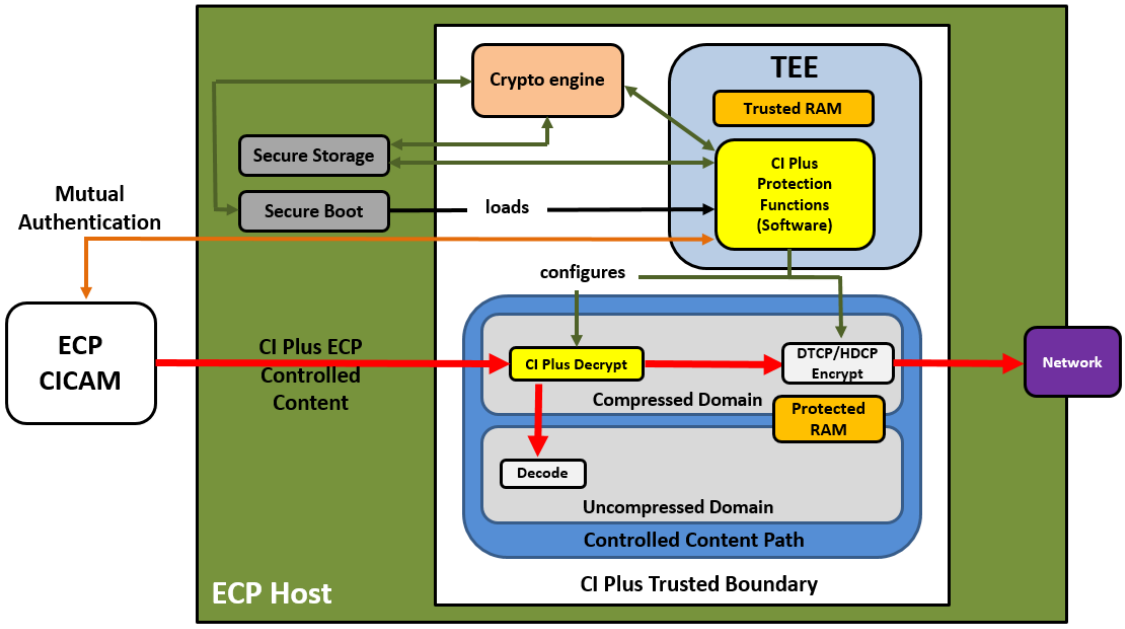

Figure 7: Host - Export to Network

3. Attacks

This section defines the attack environment (attacker profile, attacker expertise) and lists the different hardware and software attack techniques that may target the CI Plus architecture blocks defined in Section 2. For each attack, the attacker profile, the attacker expertise and the needed tools are specified.

3.1. Attack environment

3.1.1. Attacker profiles

The attacker profiles considered in this document are the following:

- **Local attacker**: such an attacker has a physical access to the device, required to perform all attacks described in the Hardware attack techniques chapter. An example for this attacker profile is a malicious consumer with hardware skills who wants to extract premium content from his own device to distribute it on the internet;

- **TEE-user attacker**: this attacker may or may not have a physical access to the device. He has the capability to provision a Trusted Application in the TEE. An example for this attacker profile is an attacker taking advantage of a vulnerability in the TA loading mechanism to provision malicious Trusted Applications in the TEE;

- **REE-root attacker:** this attacker may or may not have a physical access to the device. He has the capability to execute code in the kernel side of the non-secure operating system. This attacker profile is the most common in software attacks on TEEs, as it is considered that the non-secure operating system is not trusted. An example for this attacker profile is a malicious consumer with software skills who wants to extract premium content from his own device to distribute it on the internet;

- **REE-user attacker:** this attacker may or may not have a physical access to the device. He has the capability to execute code in the user side of the non-secure operating system. An example for this attacker profile is a remote hacker who is able to connect to a CI Plus LLP device (e.g. on an administration interface) and wants to extract premium content from it. Any attack successfully conducted by this attacker profile could be damageable for CI Plus LLP, as it is highly scalable and requires the lowest possible level of access to the device.

Note: the *TEE-root* attacker is excluded on purpose. Indeed, it is considered that such attacker profile is too strong and there would not be any relevant countermeasure to prevent an attacker in control of the TEE kernel to access CI Plus content.

3.1.2. Attacker expertise

Specialist expertise refers to the level of generic knowledge of products and attack methods. Three expertise levels are defined as follows (according to the reference [CC_Evaluation]):

- **Laymen** are unknowledgeable compared to experts or proficient attackers, with no particular expertise;

- **Proficient** attackers are knowledgeable in that they are familiar with the security behavior of the product or system type;

- **Experts** are familiar with the underlying algorithms, protocols, hardware, structures, security behavior, principles and concepts of security employed, techniques and tools for the definition of new attacks.

3.2. Hardware attack techniques

3.2.1. Physical Side-Channel Analysis

Attack description

Physical side-channel attacks[2] exploit the dependency between manipulated data/operations and physical signals of the device called leakage signals (power consumption, electromagnetic emanation, photonic emission, duration of operations...). The physical side-channel attack can use a low number of physical signals (simple analysis) or a high number of physical signals (statistical analysis) to find out the secret information. Differential Power Analysis (DPA) and Correlation Power Analysis (CPA) are in the second case.

There exists also another kind of side-channel attack, called Profiling Attack (e.g. Template Attack) which contains two steps. In the first step, the attacker uses an under-control device, performs a high number of leakage signals to learn about the leakage model (the profiling step). In the second step, the attacker uses this model to find out the secret key with only some leakage signals (the attack step).

Physical side-channel analysis is frequently used to reveal the secret key of a cryptography implementation but can also be used to learn about the different steps of an operation. Depending on the leakage level of the device and the implementation, the useful information obtained from the analysis can be much or less. Physical side-channel analysis is usually the preliminary step of a physical fault injection attack (3.2.2), in which the attacker looks for where to inject the perturbation.

Attack environment

- Attacker profiles: To perform this attack, the attacker needs to be a **Local attacker**.

- Attacker expertise: The attack usually requires an **Expert** to identify the attack path, specific parameters and the way to pre-process the traces (noise filtering, resynchronization) for a new target. A person with **Proficient** level is usually sufficient to repeat the attack.

- Tools: The attacker needs tools to acquire and analyze side-channel signals.

[2] https://arxiv.org/abs/1611.03748

- For the acquisition of traces, the attacker can use for example an electromagnetic probe to capture the electromagnetic radiation of the device during the execution of the targeted operation. Power measurements can be done by adding a small resistor in line with the device under testing. The traces are usually acquired by a digital oscilloscope;
- The trace analysis can be done off-line (i.e. after the trace acquisition during execution). Some signal processing tools can be used to pre-process the traces (noise filtering, resynchronization). Then an analysis tool corresponding to the analyzed cryptographic algorithm will be used to exploit the dependence between the traces and the manipulated data/operations in order to find out Secret Values.

Countermeasures

The countermeasures of physical side-channel analysis can be divided in two groups. The first group aims at reducing the leakage by noise adding, operation desynchronization or hardware designs (e.g. current balancing, current smoothing). The second group aims at breaking the dependency between the physical signals and manipulated values by masking techniques.

3.2.2. Physical Fault Injection

Attack description

Physical fault injection attacks temporarily change the normal behavior of the targeted device in order to observe leaking information via abnormal behavior or to directly bypass security mechanisms. The perturbation can be done via:

- the variation of clock signal or of power[3] supply;
- the injection of an external source such as high-energy electromagnetic pulses or focused laser beams;
- the temperature variation.

The main consequences of fault injections are:

- Modifying a value read from memory during the read operation: this may concern data or address information;
- Modifying a value that is stored in volatile memory;
- Changing the characteristics of random numbers generated;
- Modifying the program flow (skipping an instruction, replacing an instruction with another one, inverting a test, generating a jump, generating calculation errors).

Differential Fault Attack (DFA) aims at exploiting the difference between a normal output and erroneous one of a cryptographic implementation to retrieve the secret key.

Faults can also be injected into a conditional control (for example during an authentication step) to skip the control or to change the result (from false to correct) of the authentication. The attacker first uses the physical side-channel signal to identify the conditional control operation and then injects a perturbation into this operation to change or skip it.

The fault injection is frequently used during the security evaluation of smartcard. Some recent studies show that it is also applicable in other embedded devices of which CI Plus devices. The

[3] https://eprint.iacr.org/2016/810

secure boot, the access control (for example the JTAG port control) can be the target of physical fault injection attacks. The attacker can also target the anti-rollback mechanism of the TEE and attempt to load an old and vulnerable TEE image. The attacker needs to perform a fault injection to bypass the firmware signature verification and the version check.

Permanent modifications of EEPROM or Flash memory devices can be achieved in some cases by injecting energy sources. However, this kind of fault is more difficult to obtain than transient faults during execution.

The efficiency of a physical fault injection attack depends on different factors: the sensibility of the hardware device to perturbations, the leakage level of side-channel signals (which are used to trigger the faults) and hardware and/or software countermeasures.

Attack environment

- Attacker profiles: to perform this attack, the attacker needs to be a **Local attacker**.

- Attacker expertise: Depending on the complexity of the implementation and the attack path, the attack requires an **Expert** or a **Proficient** profile

- Tools:
    - The attacker needs energy sources (e.g. laser, electromagnetic) or variation sources (clock, temperature) to generate faults;
    - The attacker needs also a trigger equipment to inject faults into the device at right moments with correct durations. The attacker can use the same tools for side-channel analysis to acquire traces which are used to trigger injections;
    - If a precise injection location is needed, a mechanic platform enabling 3-dimentional movements (X, Y and Z axes) is required;
    - For DFA, a software tool is used to analyze the faulty outputs and find out the Secret Values.

Countermeasures

Hardware countermeasures consist in detecting the presence of suspect variations (clock control, voltage/glitch/light control, temperature control) or prevent the penetration of external sources. Randomization of internal execution (by varying the code, the clock) and reduction of the side-channel leakage are also solutions to make the injection trigger more difficult. Redundancy/integrity checks at software or hardware levels enable the fault detection.

3.2.3. External RAM Probing

Attack description

The goal of this attack is to retrieve confidential data or keys by probing data buses between the SoC and an external RAM. If the data buses can be located, tiny probe needles are positioned on wires to tap data exchanges between the processor and the memory. In the combination of a logic analyzer, it is possible to retrieve transferred data. The difficulty of the attack depends on the accessibility of data buses, the dimension of the device, the number of needed connections and other protections such as the data bus scrambling.

Attack environment

- Attacker profile: To perform this attack, the attacker needs to be a **Local attacker**.

- Attacker expertise: Depending on the implemented countermeasures, this attack usually requires **Expert or Proficient** level of expertise.
- Tools:
    - De-processing tools to get access to the buses if these latter are protected by anti-probing layers;
    - Bus analysis probes and software;
    - An optical microscope to locate the buses if needed.

Countermeasures

Possible countermeasures preventing external DRAM probing include:

- Preventing access to external data buses;
- Bus-scrambling to randomize the data retrieved by the attacker. With the bus scrambling technique, the individual bus lines are randomly laid out. The scrambling can be static, chip-specific or session-specific.

3.2.4. Access to Debug Interfaces

Attack description

This attack assumes that either a debug port is not closed or that the port is closed but the closing mechanism is not secure. For instance, a JTAG interface may be disabled and a password is required to enable it, however the password comparison is vulnerable to timing or fault injection attacks. Another protection mechanism for debug interfaces consists in erasing the device memory whenever the JTAG interface is enabled. Nevertheless, this mechanism may be bypassed if it is not properly implemented.

Having a debug access, the attacker can directly access and read or modify memory contents so as to find an exploitable vulnerability or perform a privileged action without proper authorization.

Attack environment

- Attacker profile: To perform this attack, the attacker needs to be a **Local attacker**.

- Attacker expertise: The attack usually requires the **Proficient level** of expertise. If the debug interface is protected by cryptographic means and the fault injection attack is required to bypass the authentication, the attacker should need an **Expert** or a **Proficient** profile to perform the attack.

- Tools

- If the debug interface is not protected, the attacker needs several tools such as a JTAG probe and standard soldering tools to locate and connect to the JTAG debug pins.
- If the debug interface is protected by cryptographic means and the fault injection attack is required to bypass the authentication, the attacker needs tools for Physical fault injection specified in Section 3.2.2.

Countermeasures

To completely prevent debug interface attacks, the best way consists in removing the debug interface hardware. However, device manufacturers are always keen on keeping such interface for troubleshooting of devices having functional issues when on the field. Then another possible software countermeasure is to protect debugging interfaces by use of cryptographic means.
Other countermeasures such as password protection or memory erasure upon JTAG unlocking can be considered, although they need to be very carefully implemented to prevent bypass.

3.2.5. Firmware Flashing

Attack description

The goal of this attack is to physically flash a modified (malicious) or an older (vulnerable) firmware without going through the legitimate update process. A non-volatile memory can be unsoldered by using a (hot air) rework station and removed from the main board without damaging it. A modified firmware is then flashed into the memory.

This attack requires also software steps to craft a malicious firmware image with the expected format and hash. If the authenticity verification during the Boot can be defeated, the malicious firmware image will be accepted, thus implying further logical attacks.

Attack environment

- Attacker profile: To perform this attack, the attacker needs to be a **Local attacker**.

- Attacker expertise: The attack usually requires the **Proficient** level of expertise to discover boot chain vulnerabilities and craft a malicious image, however reproducing the attack can be done by a **Layman** attacker.

- Tools
  - A basic rework station is needed to unsolder the memory;
  - Tools to flash the modified firmware or an older firmware;
  - If the malicious firmware is based on a legitimate one, the attacker needs to have the legitimate firmware (e.g. via Logical Memory Dump or via physical memory dump to get the firmware image). The attacker also needs an advanced understanding of legitimate firmware, which can be acquired using reverse-engineering tools (e.g. licensed tools: IDA Pro, BinaryNinja, Tetrane Reven, free tools: Miasm, Metasm, Triton, angr, Radare 2)

Countermeasures

The memory hosting the firmware image can be soldered using a BGA package to ensure it cannot be unsoldered.

A mechanism to protect the firmware authenticity shall be used. If a hash is used for the firmware authentication, it shall be at least SHA-256.

3.2.6. Direct Memory Access

Attack description

This attack leverages DMA capabilities of devices such as network cards or PCIe cards which allows certain hardware subsystems to access the main system memory without going through the MMU. This attack may, for example, enable an unauthorized entity to access sensitive data in the TEE or ECP Controlled Content Path memory.

Attack environment

- Attacker profile: To perform this attack, the attacker needs to be a **Local attacker**.
- Attacker expertise: To identify new DMA attacks, a **Proficient** attacker is required. However, even an attacker with **Layman** level can reproduce the attack.
- Tools: the attacker needs tools to directly access the device memory as a legitimate peripheral such as a PCIe or network card (e.g. the tool PCIleech[4]).

Countermeasures

To prevent DMA attacks, a possible countermeasure consists in using a well-configured IOMMU which maps physical memory addresses to logical one and acts as a firewall for all memory requests. One could also process sensitive data in an isolated environment that cannot be reached via DMA. It is also recommended to disable unneeded DMA devices/drivers.

Another countermeasure to prevent DMA attacks consists in partitioning of the DRAM by use of a dedicated hardware of the SoC, when available. The dedicated hardware monitors the DRAM accesses and enforces rules based on the initiator of the access and the address of the data accessed. When correctly configured, such dedicated hardware isolates unprotected content in DRAM from unauthorized software and hardware.

3.3. Software attacks

3.3.1. Cache Attack

Attack description

[4] https://github.com/ufrisk/pcileech

Multiple cache level bridges the gap between the latency of main memory accesses and the fast CPU clock frequencies. Cache attacks[5] exploit effects like the timing dependency of these components to learn sensitive information about executed instructions and code paths.

In the TEE context, the goal of a cache attack on a cryptographic implementation is to retrieve keys used by the TEE[6]. The attack requires some specific conditions:

- TEE and REE share the cache memory;
- The cryptographic algorithm is implemented fully in software;
- The cryptographic algorithm relies on memory accesses, for instance lookup tables.

The attacker controls the cache memory content belonging to the REE to get information on the amount of cache memory allocated to the TEE. This information makes it possible to deduce statistics on cache misses and then to get the cryptographic keys that are used.

Attack environment

- Attacker profile: This attack can be performed by a **REE-root attacker** or a **REE-user attacker**.

- Attacker expertise: To identify new cache attacks, an **Expert** attacker is required. However, even an attacker with **Layman** level can reproduce the attack.

- Tools: the attacker only needs tools so as to perform measurements of TEE cryptographic operations and deduce information on used cryptographic keys.

Countermeasures

Countermeasures to block such an attack path include:

- Cache implementation that minimizes the impact of REE behavior on TEE cache memory;
- Cryptographic software that uses the same amount and locations of the cache independently from the keys used (major architectures allow this);
- Usage of cryptographic hardware embedded in the SoC;
- Usage of tamper-resistant software implementation – this includes dedicated cryptographic instructions available in specific architectures.

3.3.2. Rowhammer Attack

Attack description

The Rowhammer bug[7] is a consequence of interactions between adjacent cells in DRAM which can cause repetitive accesses to memory cells to alter the contents of neighboring cells (bit flips).

[5] https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_lipp.pdf
[6] https://eprint.iacr.org/2016/980.pdf
[7] https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf

It is primarily caused by the high density of memory cells in modern memory (e.g. DDR3). The Rowhammer attack can be considered as a logical fault injection attack.

The Rowhammer bug can be leveraged, for example, to recover a secret RSA keys[8] such as the CI Plus Device Private keys (MDQ/HDQ) residing in the TEE from the REE in certain circumstances. The specific conditions for the attack to work include:

- The device embeds a Rowhammer vulnerable memory chip;
- At least one Rowhammer vulnerable memory location is included in the secure memory;
- The Rowhammer vulnerable memory location(s) are used to store sensitive data and are adjacent to REE memory.

Attack environment

- Attacker profile: This attack can be performed by a **TEE-user attacker**, **REE-root attacke**r or a **REE-user attacker**.

- Attacker expertise: to identify a new Rowhammer attack, an **Expert** attacker is required. Usually, even an attacker with **Layman** level can reproduce the attack.

- To perform a Rowhammer attack, the attacker needs a software tool to characterize the device and exploit the bug (e.g. Drammer[9]).

Countermeasures

Logical countermeasures can be implemented by carefully managing the secure memory to avoid sensitive data in memory areas which can be targeted by Rowhammer.

Physical countermeasures enabling to reduce or prevent the Rowhammer attack include:

- Increasing the refresh rate of memory;
- Performing tests so as to identify victim cells and refreshing vulnerable rows;
- Usage of Error Correcting Codes (ECC);
- Usage of secure DRAM chips (e.g. chips that have been certified).

3.3.3. Fuzzing

Attack description

Fuzzing[10] is a technique which aims at finding vulnerabilities in an implementation by sending invalid or randomly generated inputs. Even though it is not an attack per say, it is often one of the most important steps in a vulnerability research.

Fuzzing is usually conducted in three phases:

- Generation: the tests can be generated using different techniques such as random generation, model based tests generation and mutation;

[8] https://eprint.iacr.org/2016/618.pdf
[9] https://github.com/vusec/drammer
[10] https://msdn.microsoft.com/en-us/library/cc162782.aspx

- Execution: interface with the target, send the generated commands and record the results;
- Analysis: when a crash is identified.

In the context of CI Plus, fuzzing can be used to find vulnerabilities in the TEE itself or in the Protection Functions implementation. In this purpose, fuzzing techniques can be applied on the various physical and logical interfaces of the device including:

- Device physical interfaces (USB, HDMI, Wifi, Ethernet …);
- Trusted Applications entry points;
- TEE entry points.

Attack environment

- Attacker profile: this attack can be performed by a **TEE-user attacker** or a **REE-user attacker.**

- Attacker expertise: to identify a vulnerability in the implementation of the TEE or the protection functions, a **Proficient** attacker is required. However, even an attacker with **Layman** level can reproduce the attack.

- Tools: to perform fuzzing attacks, the attacker only needs tools that will mainly generate and send random or invalid inputs to the target. The corresponding reactions and outputs of the target will be analyzed by the tools to identify vulnerabilities and exploit them.

Countermeasures

Countermeasures aiming to reduce the success probability of fuzzing attacks include:

- Perform fuzzing tests during development process so as to reduce attack surface;
- Safe implementation of memory handling in TEE;
- Careful input parameter validation in the entry points.

3.3.4. Firmware Rollback

Attack description

The firmware implements the interface between the hardware (e.g. RAM, crypto-engine …) and the software running on the device. Firmware integrity and authenticity is a critical part of the overall security of the system, as a compromised firmware can provide unauthorized access to all secure memory and peripherals. It is thus important that the possibility of a vulnerable firmware is considered, and that when such a vulnerability is discovered, the vulnerable firmware can be updated.

Firmware rollback vulnerabilities arise when a legitimate but older, potentially vulnerable, version of the firmware can be installed on the device.

Several attack vectors may exist to perform a firmware rollback:

- Take advantage of a vulnerability in the version comparison, such as an integer overflow;
- Take advantage of a backdoor in the firmware version checking, such as a “manufacturer” debug version number always accepted;

- Take advantage of an unsecure storage of the current version.

Attack environment

- Attacker profiles: This attack can be performed by a **REE-user attacker.**

- Attacker expertise: A **Proficient** attacker is required so as to identify a new vulnerability in the firmware versions comparison procedure. However, even a **Layman** attacker can reproduce it.

- Tools: to perform a firmware rollback attack, the attacker does not need specific tools.

Countermeasures

To protect itself from firmware rollback attacks, the device should compare the version of a new firmware to be installed on the device with the current version in a secure way. A firmware update should be denied if the version is older than the current version.

The current version should be securely stored so that an attacker cannot modify it.

3.3.5. Logical Memory Dump

Attack description

Logical memory dump consists in retrieving the content of all memory areas accessible to an attacker in the REE to identify interesting targets. In particular, this technique can be used to:

- Find firmware images stored in the flash;
- Find firmware images of a running system once it has been mounted;
- Find secrets stored in the RAM if no proper data cleaning is performed.

To acquire a memory dump, the attacker needs to load and execute a program reading all the accessible memory areas, dumping the read values into a file. In some cases, the attacker can also interrupt the boot loading sequence and read memory from the boot loader shell.

Once a dump has been acquired by the attacker, reverse engineering, binary analysis and entropy analysis techniques can be used to extract firmware[11] and secrets from the dump.

Attack environment

- Attacker profile: the corresponding attacker profiles considered for this attack are **TEE-user attacker** and **REE-user attacker.**

- Attacker expertise: the attack usually requires the **Proficient** level of expertise, especially a good knowledge in reverse engineering and data analysis to recover valuable information from the acquired dump.

- Tools: the attacker needs software tools such as simple scripts to recover the memory dump and data analysis tools such as binwalk[11] to analyze the extracted data.

[11] https://github.com/devttys0/binwalk

Countermeasures

The memory accessible to the REE should not contain any data interesting for an attacker. Residual information should be erased as soon as possible.

3.3.6. Code Injection

Attack description

The code injection technique aims at executing attacker's code on behalf of the attacked system, for example the TEE code. If an attacker is able to execute his own code as a replacement of the TEE code, he effectively breaks the isolation and gains access to secrets stored in the secure memory.

Code injection can be performed by leveraging a vulnerability in secure primitives or Trusted Application entry points. An example of such a vulnerability in a real-life use case was found in 2015[12].

Attack environment

- Attacker profile: the corresponding attacker profiles considered for this attack are **REE-root attacker** and **REE-user attacker.**

- Attacker expertise: To identify a new vulnerability, this attack usually requires the **Proficient** level of expertise with a good knowledge of reverse engineering and TEE systems. However, the attack can be reproduced by a **Layman** attacker.

- Tools: the attacker needs software tools such as a binary disassembler like IDA Pro to perform reverse engineering of the TEE code in order to find exploitable vulnerabilities to achieve code execution inside the TEE.

Countermeasures

Secure coding practices should be followed in the TEE implementation to reduce the likelihood of code injection vulnerabilities to be present. In particular all primitives reading and writing to arbitrary memory locations should be properly reviewed. The countermeasures described in section 3.3.3 also apply.

3.3.7. REE Privilege Escalation

Attack description

Privilege escalation is the action of gaining additional privileges than what was initially available. Privilege escalation vulnerabilities fall into two broad categories:

[12] http://bits-please.blogspot.fr/2015/08/full-trustzone-exploit-for-msm8974.html

- Permanent privilege escalation: in the case of Linux-based operating systems, a common kind of privilege escalation is known as *rooting*, which consists in an unprivileged user acquiring *root* privileges by becoming the *root* user;
- Temporary privilege escalation: a simpler form of privilege escalation can be to temporary gain elevated privileges by leveraging code injection vulnerabilities into privileged processes. This is usually possible to extend into permanent privilege escalation.

Obtaining root privileges on the REE is often considered as a first step for more sophisticated attacks targeting Trusted Execution Environments, but can also be the end goal in the case of systems without a TEE.

In the case of Linux systems, root privileges allow in particular to:

- Alter the kernel behavior (e.g. by inserting or removing kernel modules);
- Change system configuration parameters (e.g. ASLR);
- Obtain full control over the execution of any running program;
- Read/write any file on the system.

There are many different ways of achieving privilege escalation on Linux systems, examples include:

- Taking advantage of race-conditions in the kernel[13];
- Exploiting vulnerable *suid* binaries such as *sudo*[14];
- Overwriting memory areas which should be out of reach of the attacker[15].

Attack environment

- Attacker profile: the corresponding attacker profile for this attack is **REE-user attacker**. Note that in case of success, the attacker automatically becomes a **REE-root attacker**.

- Attacker expertise: the attack usually requires an **Expert** attacker to find a new vulnerability, but only the **Layman** level of expertise is required to reproduce a known vulnerability (equivalent to running a script).

- Tools: The attacker needs software tools such as a debugger (e.g. GDB), a disassembler (e.g. IDA Pro) to perform reverse engineering to discover new vulnerabilities when the source code is not available. Public vulnerabilities databases such as cve.mitre.org can also be used to find all known vulnerabilities for a specific version of the rich OS.

Countermeasures

The devices should be shipped with the latest available version of the REE, in order to prevent publicly known vulnerabilities from being exploited on CI Plus products.

Additionally, the embedded version of the REE should be kept up to date with the latest security patches.

[13] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5195
[14] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000367
[15] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000364