Setting Hardware Root-of-Trust from Edge to Cloud, and How to Use it

Florent Chabaud [1]

[1] *Atos Big Data & Cybersecurity, Rue du Gros Caillou – 78340 Les Clayes-sous-Bois – France*

Abstract

For decades, Trusted Computing has tried to anchor trust in the hardware, and the existence of Trusted Platform Modules (TPM) in most modern design is evidence that this approach is now well understood. The default behavior of recent Operating Systems like Windows 11 is even to deny booting if this security feature is absent. But this approach is not sufficient in a modern world where one needs to trust remote platforms. To preserve confidence in security, one needs to limit the trusted computing base (TCB) of a system at a level where an assessment can make sense. Trusted Execution Architecture (TEA) is the result of a partnership with ProvenRun to implement a TCB in Atos servers in a consistent way, from Edge to High Performance Computing. This allows to envision security features based on some common Root-of-Trust known to different platforms, at different scales and levels of interaction.

Keywords

Trusted Computing, Edge Computing, High Performance Computing, Remote Attestation, Operating System

1. Introduction

In 1993, the NSA tried to introduce the Clipper Chip to promote Key Escrowed Encryption [1]. Even if this attempt failed [2] and backfired in promoting open-source encryption [3], it showed the importance of hardware in computer security, and paved the way to trusted computing. Soon, the Trusted Computing Platform Alliance, renamed as Trusted Computing Group [4], will emerge and promote another piece of hardware, the Trusted Platform Module (TPM), now standardized [5] and embedded in most platforms. But this concept is now revisited by another industry consortium, the Open Compute Project [6], which adds to the TPM some other security chips. Even if adding security hardware can make sense, it is always raising the question of how this new hardware can be trusted. Understanding the alleged improvement in terms of security is also important to assess the security benefit ratio, and in the end, other options can be envisioned.

In this paper, we will bring a quick survey of the state-of-the-art of trust in hardware in section 2. We will discuss the rationale of Atos Trusted Execution Architecture (TEA) and the pros and cons of this software-oriented approach in section 3. We will then detail some aspects of the implementation in section 4. In the end, section 5 will sketch future innovative security in Atos HPC architectures, as allowed by the Atos TEA.

2. Hardware Trust State-of-the-Art Overview

2.1. Smart Cards

Long before NSA tried to promote its Clipper Chip, the ideas of using small pieces of hardware to secure secrets arose in several places. Several patents were filed around this invention, but the seminal industrialization patent was the creation of the first portable support with both a processor and a

C&ESAR'22: Computer & Electronics Security Application Rendezvous, Nov. 15-16, 2022, Rennes, France
EMAIL: florent.chabaud@atos.net
ORCID: 0000-0002-5007-6025

memory, allowing the small piece of plastic to cryptographically interact with its environment in an active way. Embedding the processor and the memory in the same single ship came rapidly after this, innovating the reign of smart cards. It is worth recalling that Michel Ugon, a French engineer of the Bull company later acquired by Atos was at the core of these inventions[8][9][10].

The large dissemination of smart cards makes them a primary target of a new class of cryptographic attacks: if the secret remains in the chip, maybe the way it is used leaks some information on the secret. A seminal attack of this kind is due to P. Kocher and al. who invented Differential Power Analysis (DPA) [11] which remains one of the threats a cryptoprocessor needs to deal with, among other types of side channel attacks.

2.2. Hardware Security Modules

Hardware Security Modules (HSM) are another example of devices which were designed to protect secrets. HSMs usually embeds features to physically protect their internals, and provide tamper evidence at physical (labels, screws…) and logical levels (logs, alarms…). Certification standards such as Common Criteria [12] or FIPS-140 [13] are developed with HSM in mind to evaluate the robustness of these security mechanisms.

As usual, it is worth noting that being certified is not a guarantee of security. Depending on the security model, a certified HSM can be proven vulnerable to threats which are out of its protective scope. Interestingly, a recent example proved the need for HSMs to be self-protected against firmware tampering, not only on their crypto processors, but also on their applicative part [14]. Said differently, HSMs firmware also needs some Hardware Root-of-Trust!

2.3. Trusted Platform Module (TPM)

The Trusted Computing Group (TCG) promotes Trusted Computing concepts across personal computers around the use of a Trusted Platform Module (TPM). It has now become an international standard [5] for a secure cryptoprocessor providing several security functions:

- Unique device keys: the TPM embeds some private keys which are normally certified by its manufacturer.
- Measurement: the TPM securely stores some Platform Configuration Registers (PCR) which are obtained by chaining the cryptographic hash of several memory areas in a specific order. Usually, the memory areas are the successive codes used during the booting sequence, therefore building a chain-of-trust. The PCR values can be locally verified by the operating system to check that the boot sequence wasn't tampered.
- Remote attestation: using its unique device keys, the TPM can sign its PCRs to remotely attest that the boot sequence was not tampered. This signature can be verified against the TPM manufacturer public key certification infrastructure.
- Key wrapping: using its unique device keys, the TPM can encrypt other cryptographic keys to ensure their secure storage. This ensures that the locally encrypted keys cannot be decrypted without the TPM.
- Random number generator: the TPM usually embeds some hardware random number generator suitable for cryptographic usage.

It is important to understand that the TPM cannot guarantee the security of the CPU booting process by itself. It must be completed by some bootstrapping process to kick-off the measurements and take their results in account.

2.4. Trusted Execution Environment (TEE)

Following M. Sabt and al. [15] we take as a definition of a Trusted Execution Environment (TEE) "a tamper-resistant processing environment that runs on a separation kernel". It aims at providing on a single CPU an isolation between a "normal" kernel and a "trusted" one, protected against software and

hardware attacks. Several TEE solutions exist but all of them are based on some hardware technologies such as Intel TXT [16] or ARM TrustZone [17]. The latter is widely used in mobile environments as stated by M. Sabt and al.

The security of a TEE solution results from the hardware technology used, but it depends a lot more on the usage of this technology at software level. Even if the TEE is fully isolated at hardware level, its purpose is to exchange data with the normal world and process it in a secured environment. Any vulnerability in the driver which ensure communication between the two worlds can ruin the overall security of the TEE [18][19].

2.5. Secure Chips

Other secure chips have been developed in different industries to ensure firmware integrity. Examples of solution are found in the Set-top-box area where control access system vendors ensure digital rights management (DRM) on video streams through hardware security features. For instance, Nagra On-Chip Security (NOCS) "brings the hardware "root of trust" that ensures platform security" [20]. Another example is the ARM-based cryptographic embedded controller [21] which proposes all the features to implement a TEE.

2.6. Open Compute Project (OCP)

The Open Compute Project (OCP) is an organization [6] that shares designs of data center products and best practices among several companies. It leads several projects around datacenter design. As usual in those types of organization, a sponsoring program is in place with different levels [7]. Being able to claim a product is OCP Inspired™ requires at least a Silver subscription. Curiously, the annual fee is decreasing from Silver to Platinum level, but this is compensated by the obligation to contribute to events and overall activity of the project. This may explain why the list of members is roughly split in two between Community members at lowest rates, and Platinum members. Platinum members encompass companies such as Alibaba, AMD, ARM, Cisco, Deutsche Telekom, Google, HPE, Huawei, IBM, Intel, Meta, Microsoft, Nokia, Nvidia, or Schneider Electric, among others.

Through its Security project, the objective of the OCP could be summarized as an effort to gather all previous security technologies like TPM and secure chips in an organized standard able to ensure secure computing. Also all documentation is shared according to a Creative Commons license [22] allowing to share and adapt the material.

2.6.1. OCP Platform Security Overview

The overall organization of the OCP is somehow fuzzy, but two parallel approaches are identified which should eventually converge:

1. The Datacenter Secure Control Module (DC-SCM) specification [23].
2. The OCP Platform Security Overview [24].

The main outcome of this last document can be summarized in the excerpted **Figure 1**. It introduces a new piece of hardware, the Platform Active Root-of-Trust (PA RoT).

The role of this PA RoT, which could be ensured by the DC-SCM, is aligned with the NIST Platform Firmware Resiliency Guidelines [25]. This Special Publication was issued in May 2018, and its guidelines have soon become a *de facto* standard of what a platform needs to implement to improve their resiliency against a variety of known attacks, both at software and/or hardware level. It proposes a progressive approach with three different platform security levels: Protected, Recoverable, and Resilient.

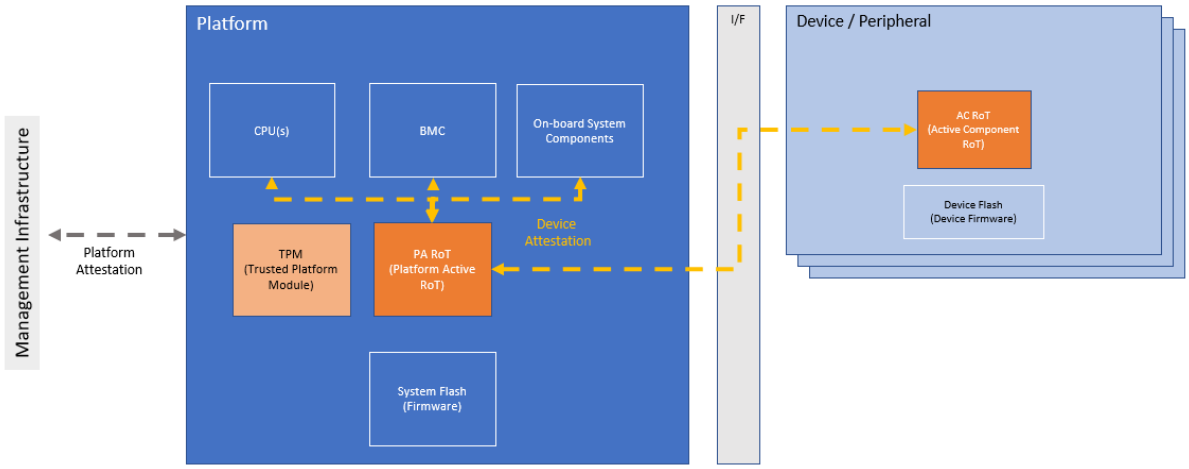

Figure 1: Overview of Secured Platform Architecture according to OCP [24] – CC BY-SA 4.0 license [22]

2.6.2. Attestation of System Components

The OCP has issued some requirements and recommendations around attestation of system components [26]. The document goal is to allow a platform (verifier) to build its platform inventory containing a list of all security-relevant devices, whether they support authentication and attestation or not. Attestations are secured by a set of cryptographic keys and protocols which are used in attestation mechanisms. Cryptographic requirements refer to standard NIST documents.

Because of these requirements, each device must be provided with a set of cryptographic keys:

1. A Unique Device Secret (UDS) which is used to characterize the attester device.
2. A private authentication key unique to each device. The corresponding certificate is allowed for digital signature usage. This key is intended to be immutable and certified by the provisioner.
3. A private signature key unique to each device. The corresponding certificate is allowed for digital signature and content commitment usages. This key is intended to be updated and certified by the device owner.

For the provisioner, the specification also requires some key management infrastructure using HSM to protect:

1. The keys of the Provisioner's Certificate Authority.
2. The keys of the Updater role.
3. The keys of the Firmware signer role.

To be noted that there is a notion of ownership transfer that implies that the Updater and Firmware signer keys can be changed by the Device owner.

Also, the requirement implies the existence of a root of trust within each attester, able to perform cryptographic operations, including random number generation with sufficient entropy. References to NIST publications and FIPS 140-3 [13] at level 2 is recommended.

Once this is set, attester devices must be capable of communicating their authentication and attestation capabilities to the platform, and platforms must be capable of interrogating potential attester devices and recording their authentication and attestation capabilities. The two references used to implement the corresponding protocols are described in DMTF's SPDM [27] (see 2.3.9) and Microsoft's Cerberus [29]. A sample implementation of the DMTF's SPDM specification is also a reference [28].

3. Atos' Approach

3.1. Threat Model

When dealing with firmware security, the threat model can drastically impact the level of protection needed. It is indeed a different story to protect the firmware integrity of a server lying in a physically isolated datacenter, or to address the same problem on a smart card which can be easily replaced by a copy. Also of importance is the scope of the intended protection. In our case, and in this paper, we focus on the security of the platform with an agnostic approach of the CPU/GPU components. We aim to ensure some security independently from the existing technologies at OS level. In particular, and as an example, the operating system can still use the TPM when it is present and leverage the CPU technologies to ensure it's booted in a secured way. This will be further explained in section 4.4. But we want to ensure a certain level of security of the platform even if none of these security features is used. So, let's first identify the type of attack scenario which one would like to prevent in this context.

3.1.1. Physical RAM Access

The first scenario of attack is basic. If one has physical access to the server, he or she could leverage this access to reprogram the memories of the hardware and have the platform firmware execute unwanted operations. For instance, in the context of an HSM which would protect secret keys, reprogramming the firmware could be a simple way to get a given secret key copied on an external interface, hence compromising it.

3.1.2. Supply Chain Attack

Physical RAM access may be assumed as limited in time. If the physical access is possible for days, like during shipment, the attack possibilities in altering firmware are much more important. Components could be replaced which would try to mimic the behavior of the original ones while preserving some backdoor, for instance.

3.1.3. The Persistent Remote Attack

As the firmware will have some critical vulnerabilities discovered, the security objective is to make the platform able to recover from such attack and to avoid its persistency. If such an attack can change every piece of data in the platform memories, then it is pretty clear that the attack can remain, since the platform relies on its memories to boot. The use of some immutable data seems therefore mandatory to ensure security.

3.1.4. Rogue Developer

The internal threat remains a possibility for any vendor, and whatever the source code tainting approach. The effects are the same in case of an intrusion on the development infrastructure. Controlling the code can mitigate this risk only if these controls are not by-passed. A good way to ensure that the code is controlled at least once is to have it signed with a properly protected cryptographic key. This ensure as well a resilient posture in case of late discovery of some rogue activity. In this case, the root-of-trust remains the cryptographic keys which are used to sign the firmware.

3.2. Sovereignty Principle

From a security perspective, a platform MUST use a Hardware Root-of-Trust (HW RoT). It is the only way to ensure some protection against software attacks and to achieve a level of resilience. Without it, any of the above listed attacks, if successful, would achieve a state where trust would be damaged in

an irreversible way. On the other hand, if the Hardware RoT exists, the platform may be rebuilt from this basis. This is the approach specified by OCP with its Platform Active Root-of-Trust and promoted by the NIST Platform Firmware Resiliency Guidelines [25].

But Atos also wanted to limit unknown hardware to increase trust and confidence in the solution. Even if some of the proposed PA RoT are open source like the Open Titan [31], adding new hardware increases the attack surface and the complexity of the server. And one also must take in account the delays to qualify and stabilize the new hardware [32]. In the end, the trust in the resulting hardware is disputable and will eventually come from wide usage, as the story of the TPM told us.

We therefore limited the HW RoT to some public cryptographic key anchored in the silicon, whose corresponding private keys are handled in an HSM developed by Atos: the Trustway Proteccio netHSM [30]. This HW RoT is then propagated through a Chain-of-Trust applied to:

1. A secure boot sequence (Chain-of-Trust for Detection – CTD).
2. A secure firmware update (Chain-of-Trust for Upgrade – CTU).

These chains-of-trust will be later detailed in section 4.1.

3.3. Baseboard Management Controller

Most of the modern servers have a Baseboard Management Controller (BMC), which is responsible for the power-on of the main CPUs, and the management of the firmware. From a security perspective, it is already a piece of the platform you need to trust. And it has already been proven that it can be a source of weakness for a server [33].

For the servers developed by Atos, the BMC is hosted in a System-on-Chip (SOC). We decided to leverage this existing hardware and to elevate it as the Platform Active Root-of-Trust for the platform. The **Figure 2** modifies the original figure from OCP (see **Figure 1**) to illustrate the approach.

The server's CPUs are therefore seen as symbiont devices relative to the BMC. The advantage of the approach is that this hardware is mandatory in all servers, as it is the interface to the management infrastructure to power-on the platform or upgrade its firmware. It also plays a key role in the overall integrity of the platform, and its security should be hardened.

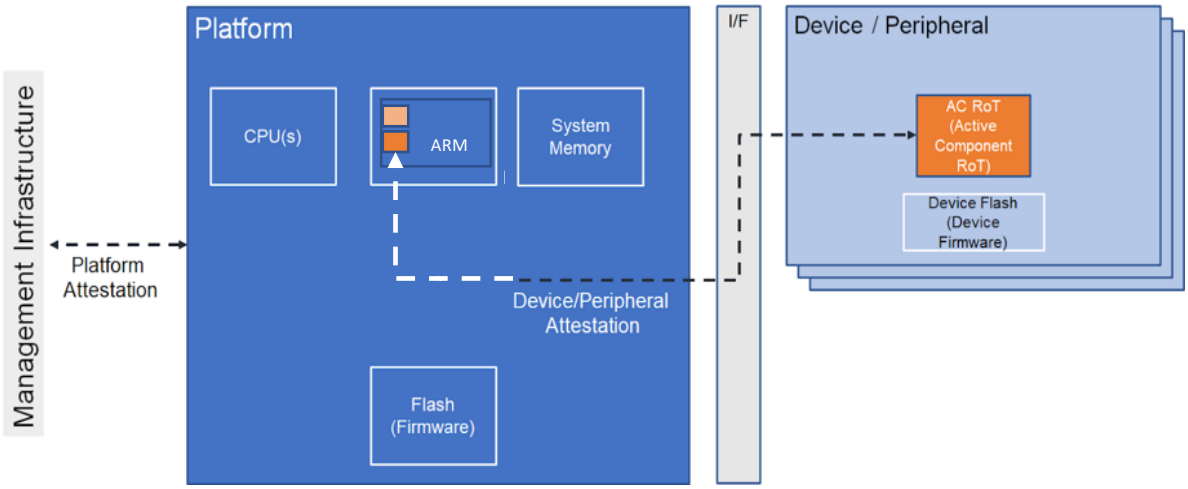

Figure 2 – Trusted Execution Architecture (TEA) of Atos servers – adapted from a CC BY-SA [22] licensed material by OCP [24]

3.4. Security Implications

The current implementation of the Atos BMC is based on Open BMC [34], a Linux Foundation collaborative open-source project whose goal is to produce an open-source implementation of the BMC Firmware Stack. The Open BMC project has already security in mind with firmware signature

verification during secure boot [35]. But this doesn't appear sufficient to reach the security level needed for a PA RoT. And even hardening Open BMC Linux kernel would not achieve hardware-like security. But the underlying hardware embeds an ARM core with TrustZone technology [17]. Leveraging this technology makes possible to achieve a decent level of security, even without a dedicated security component. This is the implementation we will detail in section 4. Assuming this technology is efficiently implemented, what are the impacts on the above threat model:

1. Reprogramming the firmware of the server assumes the possibility to reboot the server with a rogue firmware. This possibility is prevented as the BMC will verify the signature of firmware during boot sequence. And the cryptographic keys used for this verification are out of reach for a standard physical access.
2. Changing components of the platform is the threat covered by the device/peripheral attestation mechanism. Of course, the security of this mechanism depends on the existence of the PA RoT, which could be replaced in our case by another BMC. This rogue SoC would have to implement a backdoor in a way that resist subsequent firmware upgrade of the BMC using Atos firmware. This seems an acceptable residual risk.
3. The persistent remote attack risk is covered in the same way as the direct reprogramming of the firmware memories. The BMC will verify the signature of the firmware during boot sequence. The firmware upgrade feature which is present in the BMC will also verify signature of the firmware before authorizing the upgrade.
4. The public keys anchored in the hardware make possible to recover from a situation of a trapped development as long as the private keys are duly protected.

Of course, the use of a non-dedicated hardware for security has some drawbacks. For instance, it is envisioned to implement a firmware TPM in the Atos BMC. This would allow to add this security feature in HPC environment where no TPM is usually implemented for physical space reason. But it cannot be claimed the same level of security, since the TPM chips are usually certified at high level of security (see for instance [36]). For the threat model we described, dedicated to Platform security and Firmware integrity, there is no significant change in the risks. However, for cryptographic storage of user keys, which is one of the key features of a TPM for the end user, the risk assessment would have to be considered accordingly.

4. Atos's PA RoT Implementation

4.1. Ownership Transfer Preparation

Allowing the change of cryptographic keys to the platform owner is difficult to ensure while preserving the overall security, since the purpose of anchoring RoT in the hardware is to prevent software-based attacks which could change the keys used for firmware verification. Even if these keys are public, their integrity is of utmost importance to the security objectives.

Signed firmware is used to authenticate firmware before critical functions:

1. CTD: During boot sequence to ensure that the next step of the boot sequence will activate an authenticated binary code.
2. CTU: During updating process to ensure that the firmware image is authenticated before flashing it in RAMs.

These two chains are independent and complementary.

Three types of keys can therefore be identified to verify the signature of a firmware:

1. At the beginning of the boot sequence, to benefit from Hardware root-of-trust (red key in the **Figure 3**).
2. During boot sequence where a public key embedded in firmware can be used to pursue the chain-of-trust in a flexible way (orange key in the **Figure 3**).

3. Before flashing, where a public key can be used to verify the signature of the firmware payload. The corresponding public key can be stored either in a hardware secured part of the SOC, or in a firmware provided it is protected by another chain-of-trust (see green public key in the **Figure 3** as an example).

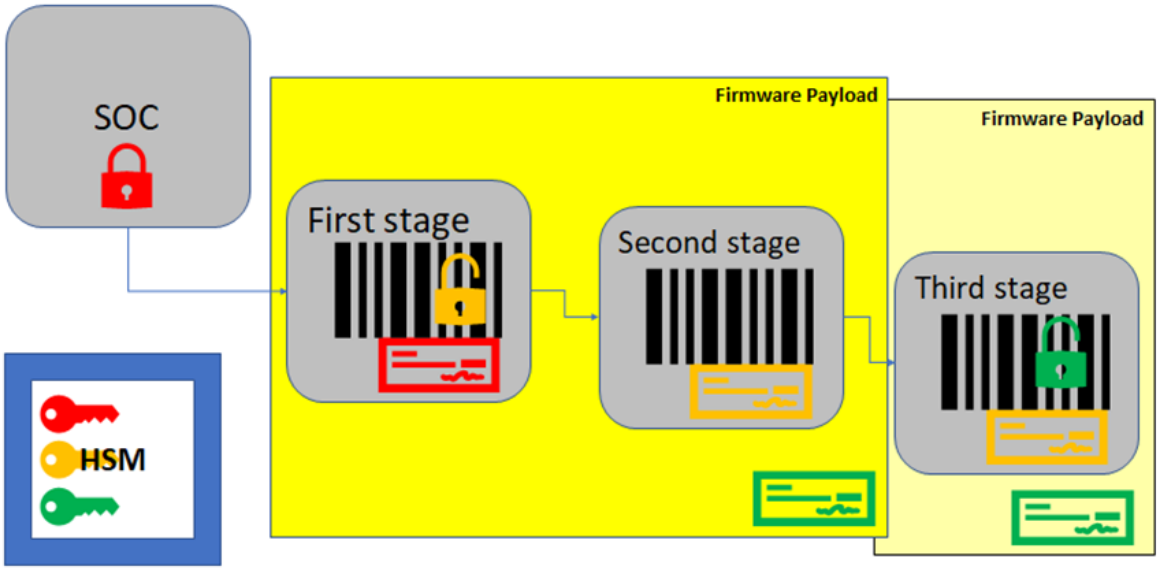

Figure 3 - Type of firmware signing cryptographic keys

Except the root-of-trust key secured at hardware level, all public keys used to verify firmware integrity and/or boot chain integrity must be part of a signed firmware. This ensures that the modification of the verification keys is authenticated provided the key store is properly implemented.

It is important to understand that the hardware secure boot is very limited in practice. It only secures the first stage of booting, a little program which cannot exceed a few kilobytes of code (63 K for ARM), because it will be loaded in ARM memory for signature verification. All the other operations of a chain-of-trust for detection (CTD) or chain-of-trust for upgrade (CTU) will exceed this limit and will therefore rely on software-based security.

Yet, this also allows ownership transfer provided the customer trusts its vendor, which seems a legitimate hypothesis. Indeed, at least the CTD key is included in the first stage booting code, which ensures the possibility to change it, while preserving the overall security of the scheme. Depending on the needs, CTD and CTU keys may be owned by the vendor or not. Consequently, in theory, the chain-of-trust keys can be changed for each owner provided at least one signature is performed involving the hardware RoT key owned by Atos.

4.2. Firmware Key Management

The root of trust is the initial public key which must be inserted and secured by hardware security measures. The root-of-trust must be immutable once the hardware security measure is in place. Therefore, the corresponding private key is of critical importance in a production environment. Due to the hardware hardening it is not possible to update a root-of-trust key by firmware upgrade. The only allowed operation may be to invalidate a compromised key. It is therefore mandatory to anticipate the compromise of such key by organizational measures and by generating several backup keys that will be injected in the hardware in case of a compromise (see 4.2.2).

4.2.1. ARM Secure Boot Specification

For intellectual property reason, we cannot here reproduce the precise way the ARM Secure Boot is implemented. We will therefore just sketch the main points to help understanding how the TEA root-of-trust key is managed.

By default, secure boot is not enabled. This is mandatory in the design process since initialization of the ARM component will need to boot the CPU. The usual chicken-and-egg situation mandates the component to be initially insecure. Several ways to activate secure boot are available. To simplify, let's say that we have:

1. A reversible way to activate hardware secure booting through jumpers on hardware pins.
2. An irreversible way to activate hardware secure booting through one-time-programmable hardware memories in the SoC.

The first option is intended for development, testing, and qualification. In the end of the production process, the second option will be used to set the component in secure mode.

Once set in secure mode, only a signed first stage can be used to boot the SoC (see **Figure 3**). The signature will be verified against the keys which have been inserted in the component.

Consequently, it is possible and recommended to introduce the public keys in the ARM core as soon as possible in the factory process. It has no immediate impact and no risk to brick the SoC provided the secure boot is left in reversible mode. It has the advantage to personalize the component early in the process, making it more difficult to tamper with (see Table 1).

Table 1
PA-RoT states

State	DEV Key	PROD Keys	Secure Boot	Usage
OPEN	Non present	Non present	Disabled	SoC reception
DEV	Activated	Activated	Reversibly activated	Development
DEV-PROD	Reversibly deactivated	Activated	Reversibly activated	Qualification Validation
CLOSED	Irreversibly deactivated	Activated	Irreversibly activated	Production

4.2.2. Secure Boot Spare Keys

For obvious security reason, the hardware secure boot public keys are also injected through OTP memories and cannot be changed once in secure mode. For a given component, the same keys will therefore be in use from day 1 of its production until its end-of-life. If the component is to be used for ten years, the corresponding private keys must be protected during this time. And on such long period, one must anticipate risks such as key loss, key compromise, identity usurpation on the firmware signing chain, etc.

As a first consequence, HSM should always be used to protect hardware secure boot private keys. This seems consistent with the sensitivity of keys which cannot be changed in the field if an incident occurs.

Secondly, the set of hardware secure boot keys should not be limited to one key. It is therefore separated in one production key, and several spare keys. Any of these keys could sign a firmware recognized by the BMC. But the only one in use is the production key. The spare keys are created just in case something weird happens to the production key. But they must be created at the same time because their public part will be injected in production. And the protection of their private part is of utmost importance.

4.2.3. Private Key Protection

All the keys used to ensure trust in the platform do not deserve the same level of protection. For instance, DEV keys are considered insensitive. They will be deactivated in production and are therefore considered extractable from the HSM. This allows to have some outsourced development without having to give access to the HSM signature mechanisms.

This is obviously not the case for PROD keys which are generated, stored, and used in an Atos Trustway Proteccio HSM [30] configured in RGS mode [37]. All production keys are saved in backups protected by a 3-out-of-6 Shamir scheme [38].

Besides, the access to the signature function is controlled:

- By logical measures for Chain-of-trust keys.
- By the use of a smart card for the HW RoT production key.
- Using a smart card AND the possession of the key backup for the HW Spare production keys.

4.3. Trusted Execution Architecture (TEA)

The System-on-Chip (SoC) used in the BMC embeds the TrustZone technology which is part of its ARM Core [17]. This is used to host a hardened Operating System TeaCore provided by ProvenRun to implement a Trusted Execution Environment (TEE) as seen in section 2.4. The TEE is then used to secure the two Chains-of-trust related to secure boot and firmware upgrade (see **Figure 4**).

TeaCore is based on the use of a proven operating system ProvenCore which has been certified by ANSSI at EAL7 level in a different context [39]. It ensures a better flexibility to later add new security features such as cryptographic keys secure storage, firmware TPM, and/or flash runtime monitoring.

Together with the HW RoT key anchored in the silicium of the BMC, the TeaCore provides the architecture to implement a full Platform Active Root-of-Trust as proposed by OCP. As of today, the chains of trust for development and upgrade are implemented. Next steps could introduce attestation mechanisms.

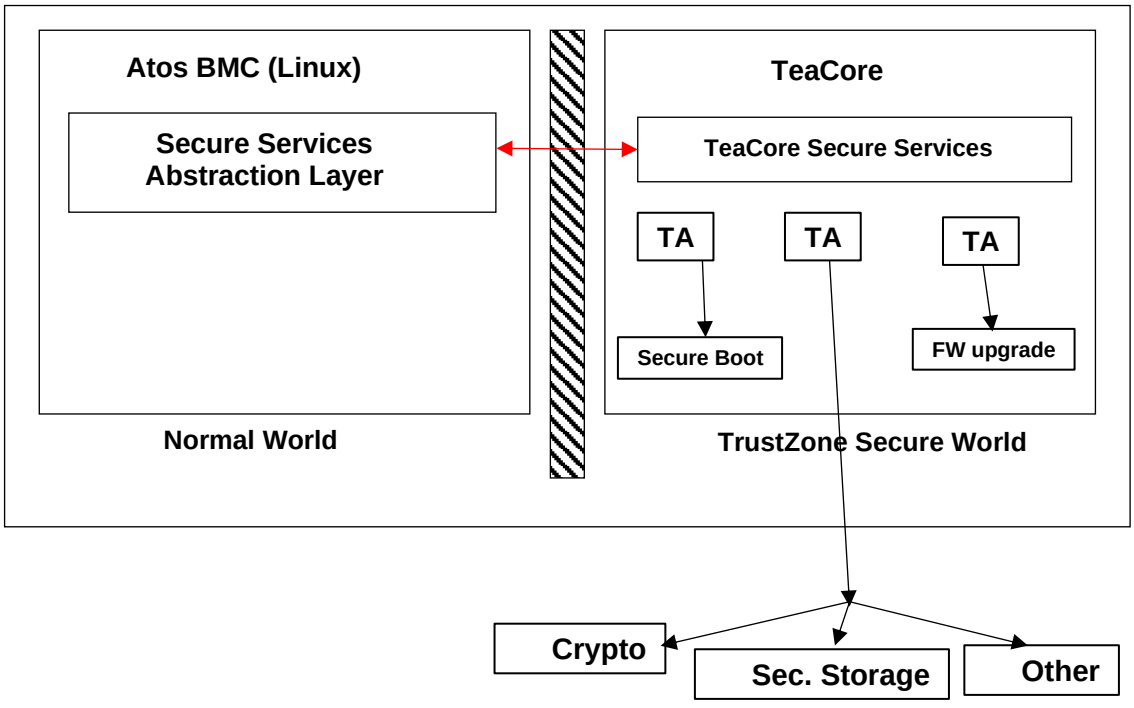

Figure 4 - Trusted Execution Environment of the Chains-of-Trust

4.4. Full Secure Boot Sequence

We have now all the information to understand the full boot sequence of an Atos server implementing the new Trusted Execution Architecture.

For this example, we will use the case of a server based on an Intel CPU implementing the TXT technology [16]. This technology implements its own RoT which signs the Initial Boot Block (IBB), the first step of the Intel TXT secure boot sequence. If activated, the Intel RoT will prevent any change of the IBB which is not properly signed. The secure boot sequence of the CPU can also imply the TPM, either a physical one if present, or the firmware implementation by Intel [40], or the one Atos could add in TEA using the TrustZone technology. This boot process will end at Operating System level. In the case of Windows 11, BitLocker can use the TPM and check through the measurements that the boot process was sane.

Atos TEA do not interfere with this process. It only adds at the beginning a preliminary verification of the IBB. Since the BMC is responsible for powering on the CPU, it will use its TrustZone to perform a signature verification of the IBB and won't power-on the CPU in case of a signature error. The whole sequence will therefore start from the ARM secure boot sequence of the BMC and ensure firmware integrity through the different existing mechanisms (see **Figure 5**). The approach would work the same way for another type of CPU.

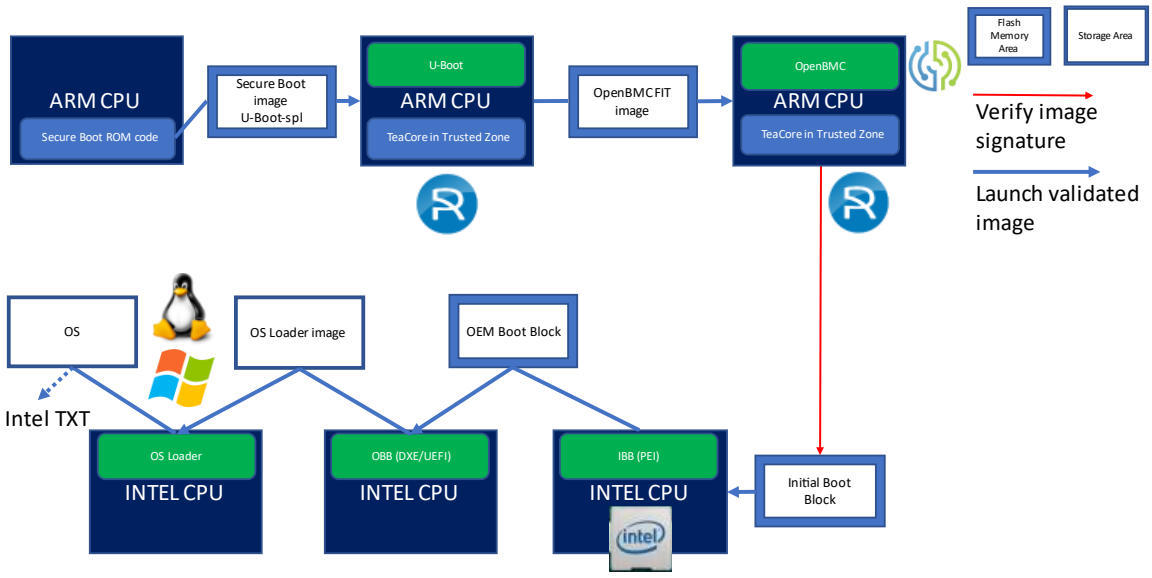

Figure 5 - Chains-of-Trust for Detection

5. Potential Application in HPC

Now that we have a TEE enabled in our servers, from Edge to Enterprise servers, let's see the type of application we could envision in a High Performance Computing (HPC) environment.

5.1. How an HPC Could Be a Unique Device

One drawback of the Platform Active Root-of-Trust approach is that the secure component becomes a single point of failure for the system. This is especially true when comes the definition of the Unique Device Secret (UDS). As introduced in section 2.6.2, the attestation mechanism assumes the implementation of some hierarchical certification where each vendor attests the integrity of its product using some public key infrastructure mechanism. The PA RoT will therefore use all these UDS to control the authenticity of the platform components with some signature mechanism, and the

verification of the certificates of the public keys. The trusted cryptoprocessor PA RoT is also the privileged place to store the UDS of the platform itself, allowing it to become attester to the remote management infrastructure (verifier). But this approach reaches some limits when it comes to big cloud infrastructure or high-performance computers. As an example, one of the recently deployed Atos HPC platforms counts 300 000 computing cores shared among roughly 4500 CPUs [41]. Which one of those components will host the UDS and identify the HPC in a unique way?

5.2. An HPC Architecture Overview

It is not the purpose here to detail the architecture of an HPC installation. Besides, this is an evolving matter. Schematically, an HPC framework will gather nodes of different types exchanging data through an Ethernet or Interconnect fast network. Management Nodes or Rack Management Controllers (RMC) can exist to manage a physical cluster of a hundredth of computing nodes. Each cluster is interconnected with the other clusters to form the overall HPC (see **Figure 6**).

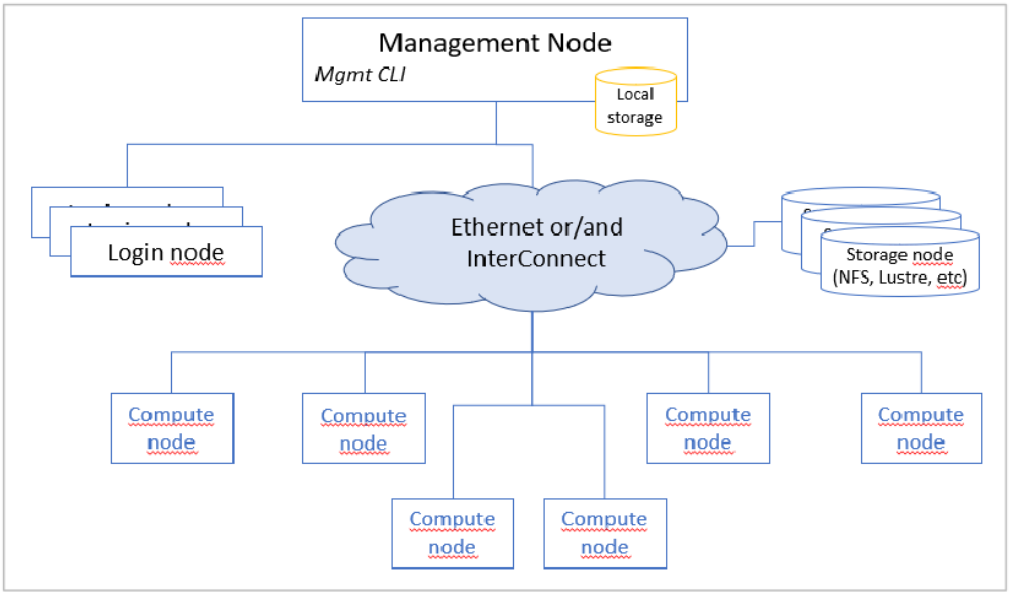

Figure 6 – Cluster architecture

From an operational point of view, the access to the computing power is devoted to some login nodes, and one of the management software roles is to schedule the job requests submitted to the login nodes to optimize the computing power. Each job will get allocated some computing nodes and storage resources for an amount of time, depending on the pre-requisites of the job request. The whole purpose of the architecture is to avoid latency in messages exchange between the computing nodes and in input/output writing on the storage nodes, while dealing with astronomically high amount of data. In a standard attestation approach, a compute node would have to check the attestation status of the storage node before sending data to it. This would mean data exchange between the nodes consuming the interconnect bandwidth. Even if it may sound marginal, keep in mind that these machines are pushing the specifications at their limits, and are subject to some avalanche effects when unwanted events occur.

On the other hand, all the development framework around HPC has already incorporated error events because the scale of the HPC makes plausible to encounter errors when the machine is in use. Hardware faults, hot swaps, are part of the normal use of an HPC computer. This can be a drawback in a classical platform firmware attestation mechanism, but it can also be turned to our advantage.

5.3. The HPC DNA: a Patented Approach

Under the hypothesis that the Trusted Execution Architecture is implemented, an immediate benefit arises from it. All nodes will get a TEE through their BMC (see **Figure 7**). Of course, the same would occur if each node would come with a TPM or any form of PA RoT secured chip. But the truth is that

adding some secure element in these nodes is not that easy for physical constraints (power alimentation, cooling, space) while a BMC is mandatory anyway. So now, we have this trusted capacity on all our nodes, and we can leverage it.

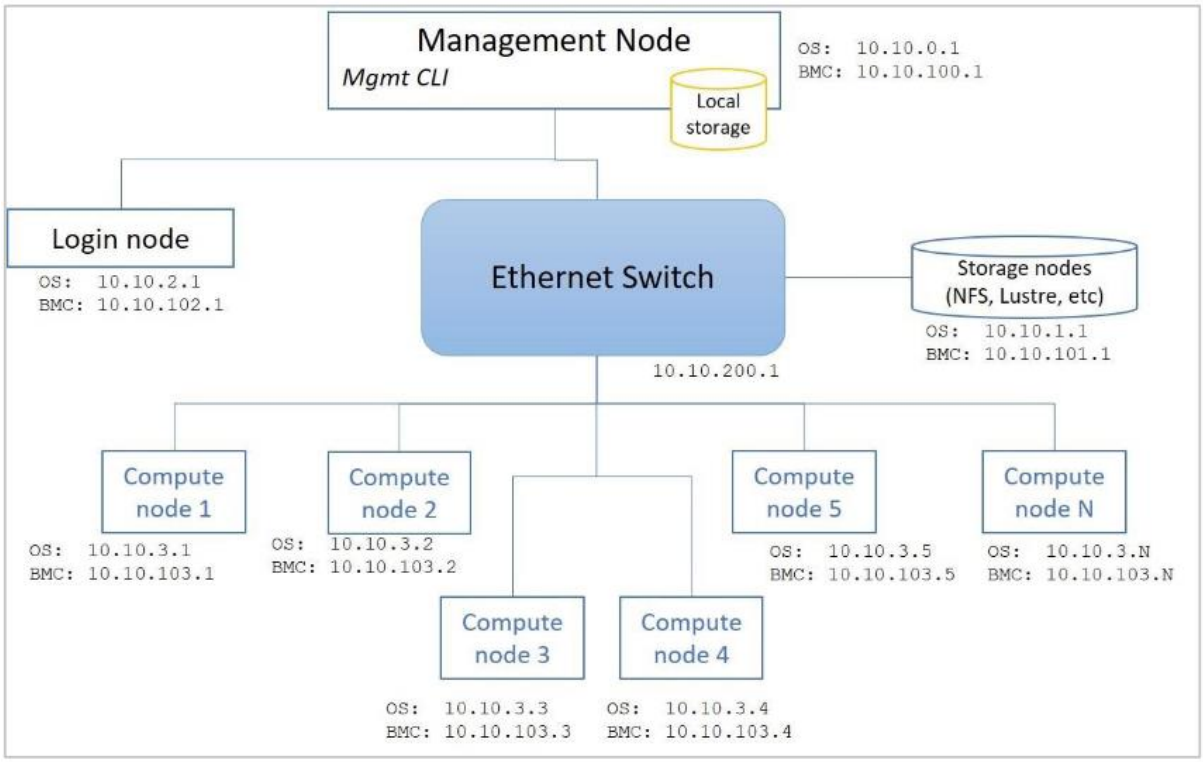

Figure 7 – Example of the management network of a cluster

Like a living body can identify its cells through characteristics determined by the DNA common to all cells, the idea of a local hardware-secured zone keeping some DNA-like secrets shared by all the machine nodes, makes possible for each node of the machine to perform the access controls, without relying on a remote server to determine if a communication is tampered or not. In other words, this generalizes the notion of Unique Device Secret (UDS) to a global platform such as a High-Performance Computer (HPC) or a Cloud-based infrastructure.

Any node of the machine will assume that its counterpart possesses the shared secret. It is therefore possible to encrypt communication under this assumption. If the counterpart fails to decrypt it, this will be treated as a glitch or hardware failure using the normal exception mechanisms of HPC development libraries.

5.4. Unique Secret Generation

The powering-on of an HPC machine is done in several steps. For instance, a rack will be powered-on before its computing blades can be powered-on. There is no guarantee on the order of the powering-on. Some computing blades can be powered-on before another rack is powered-on. The hypothesis is that all nodes will eventually establish a connection through a management network, without guarantee that all connections are feasible. For simplification, we will assume that a Rack Management Controller exists, which can be seen as a BMC dedicated to the management of all the BMCs of a rack. We will also assume that the sequence of initialization starts an RMC before the BMC it manages.

The process must therefore ensure the following properties:

- If the machine is powered-on, a new secret must be generated by the first powered on RMC.
- If the machine is powered-off, which is an unlikely event, a new secret must be generated on next power-on by the first RMC that will be powered on.

- If several RMCs are powered-on in parallel, a negotiation mechanism must converge towards a single secret.
- Any new RMC or BMC that will be powered on must get the secret in a secure way.

Generating a secret in the RMC could be a challenge from a cryptographic perspective, since we do not assume a physical random number generator is available on the BMC. It seems feasible as soon as a reliable random noise generator is available. One needs to avoid the repeatability of the boot process which could lead to the generation of the same secret on each boot. To ensure a proper source of noise, the global entropy must reach at least 256 bits. Fortunately, the RMC is gathering a lot of physical sensors information which can be leveraged to assembly enough noise in the random number generator of the device.

When a new component is added to the management network, it can be of two kinds:

- If it is a BMC, it will get the secret from its RMC.
- If it is an RMC, it will negotiate its secret against the other RMC as follows.

One cannot know in advance the order of RMC appearance in the management network. And in the life of the machine, some racks may be shut down for maintenance, then reconnected. One wants the secret to stabilize as soon as possible while preserving the history of the used keys. This is a possible application for blockchain technology and decentralized consensus making.

If RMCa and RMCb have booted and generated their secrets Sa and Sb, one cannot choose among these secrets. But a cryptographic mechanism can take place to establish a common secret Sab. This secret is timestamped in the blockchain and becomes its first block. Two blocks are then added with Sa and Sb.

If RMCc joins later with its secret Sc it will have to adopt the secret Sab. And the block Sc is added to the chain. The block chain length is therefore related to the number of racks whose secrets were changed so far.

If two racks exchange two different block chains with the same initial secret Sab, the block chain is reconciled with the missing blocks.

If the two initial secrets differ, the longer chain will be privileged. The blocks of the shorter chain will be added. If the chains have the same length the chain with the smaller hash will be kept.

If an RMC has to change its secret after negotiation, it has to propagate the new secret to its rack components. The blocks to add in the blockchain indicate the other RMCs to inform of the secret change.

5.5. Security Discussion

To prevent the secrets from being compromised when a computing blade is extracted, the corresponding secrets are stored on RAM in an encrypted way. The component extraction powers the component off by design. This ensures that at least a portion of the encrypted key is erased. The encryption mechanism can therefore guarantee the disappearance of the key if a sufficient portion of the key is lost. This security mechanism is prone to cold boot attacks [42] but this kind of scenario is mitigated if the secret is updated regularly.

Before delivering the secret to a new component of the machine, it is of course important to determine if the new component is sane. It is at this step that remote attestation protocol can be used. The UDS at node level makes perfect sense for this as it is inserted at factory time to build security upon trusted remote attestation of a component (see 2.6.2).

This pre-inserted private key should never be exposed outside its security module. Zero-knowledge protocols can use the key to attest the authenticity of the TPM-like feature remotely. This way, a newly inserted component can be checked remotely for sanity before providing the secret. And the private key is needed to decipher the secret, protecting it on first communication.

6. Conclusion

Based on well-known concepts of Product Security, Atos has implemented a Trusted Execution Architecture (TEA), common to all its servers. The trust in this implementation is founded on:

1. Public cryptographic root-of-trust keys anchored in silicon.

2. Private keys protected by an RGS certified Atos Trustway Proteccio HSM.
3. The well-known ARM TrustZone technology embedded in the existing BMC component of our platforms.
4. The hardened operating system TeaCore developed by ProvenRun on Atos specification and based on their formally proven and EAL7 certified operating system ProvenCore.

This TEA is first used to ensure some Platform Firmware Resiliency through firmware signatures verified at boot time and before any upgrade. Its generalization to all Atos-made platforms makes possible some innovative security features. As an example, we presented an innovative approach to device attestation applicable to High-Performance Computing (HPC) environments which generalizes the notion of Unique Device Secret (UDS) to a global platform such as a HPC or a Cloud-based infrastructure.

7. References

[1] Howard S. Dakoff, The Clipper Chip Proposal: Deciphering the Unfounded Fears That Are Wrongfully Derailing Its Implementation, 29 J. Marshall L. Rev. 475 (1996).

[2] Y. Frankel and M. Yung. Escrow Encryption Systems Visited: Attacks, Analysis and Designs. Crypto 95 Proceedings, August 1995.

[3] Philip Zimmermann - Why I Wrote PGP (June 1991 – updated 1999) URL: http://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html

[4] Trusted Computing Group. URL: https://trustedcomputinggroup.org/

[5] Information technology — Trusted Platform Module, International Standards Organization ISO/IEC 11889 series (2009-2015).

[6] Open Compute Project. URL: https://www.opencompute.org

[7] OCP Membership Tiers. URL: https://www.opencompute.org/membership

[8] Michel Ugon. Support d'information portatif muni d'un microprocesseur et d'une mémoire morte programmable, CII-Honeywell-Bull patent FR77.26107. 26/8/1977.

[9] Michel Ugon. Portable data carrier including a microprocessor. CII-Honeywell-Bull patent US4.211.919A. 26/8/1977. https://patents.google.com/patent/US4211919A

[10] Michel Ugon. Single chip microprocessor with on-chip modifiable memory, Bull CP8 patent US4.382.279. 25/4/1978. https://patents.google.com/patent/US4382279A/en?oq=4.382.279

[11] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis" Advances in Cryptology - Crypto 99 Proceedings, Lecture Notes In Computer Science Vol. 1666, M. Wiener, ed., Springer-Verlag, 1999.

[12] Common Criteria for Information Technology Security (ISO/IEC 15408) https://www.commoncriteriaportal.org/

[13] Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication 140-3, National Institute of Standards and Technology, March 22, 2019. URL: https://doi.org/10.6028/NIST.FIPS.140-3.

[14] Jean-Baptiste Bédrune and Gabriel Campana. Everybody be cool, this is a robbery! SSTIC 2019 Proceedings, June 2019. URL: https://www.sstic.org/media/SSTIC2019/SSTIC-actes/hsm/SSTIC2019-Article-hsm-campana_bedrune_neNSDyL.pdf

[15] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 57-64, doi: 10.1109/Trustcom.2015.357

[16] Intel Trusted Execution Technology (Intel® TXT) Software Development Guide, rev. 017.3 March 2022 https://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf

[17] ARMLtd, "Arm security technology - building a secure system using trustzone technology," Rev. C, April 2009. https://developer.arm.com/documentation/PRD29-GENC-009492/c

[18] Di Shen. Attacking your "Trusted Core" - Exploiting TrustZone on Android. BlackHat USA 2015. https://www.blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android.pdf

[19] Laginimaineb, "Bits, Please!: Full TrustZone exploit for MSM8974" 8 octobre 2015. https://bits-please.blogspot.com/2015/08/full-trustzone-exploit-for-msm8974.html
[20] Nagra-Certified Secure Video/Audio Chipsets Surpass 80 Million Mark, 2015. https://dtv.nagra.com/nagra-certified-secure-videoaudio-chipsets-surpass-80-million-mark
[21] Microchip CEC1702 Data Sheet, 2019. https://www.microchip.com/en-us/product/CEC1702
[22] Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) https://creativecommons.org/licenses/by-sa/4.0/
[23] Datacenter Secure Control Module (DC-SCM) Specification (2021). URL: https://www.opencompute.org/documents/ocp-dc-scm-spec-rev-1-0-pdf
[24] OCP Platform Security Overview. URL: https://docs.google.com/document/d/1-bfAF86cEKcn1guF-Qj2C2HhMM2oJ2njNGdHxZeetR0/edit#
[25] Special Publication 800-193 Platform Firmware Resiliency Guidelines, National Institute of Standards and Technology, May 2018. URL: https://doi.org/10.6028/NIST.SP.800-193
[26] Attestation of System Components v1.0 Requirements and Recommendations (2020) https://www.opencompute.org/documents/attestation-v1-0-20201104-pdf
[27] Security Protocol and Data Model (SPDM) Specification https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.0.0.pdf
[28] Libspdm, a sample implementation of the DMTF SPDM specification https://github.com/DMTF/libspdm
[29] Project Cerberus Firmware Challenge Specification https://github.com/opencomputeproject/Project_Olympus/tree/master/Project_Cerberus
[30] Atos Trustway Proteccio netHSM https://atos.net/en/solutions/cyber-security/data-protection-and-governance/hardware-security-module-trustway-proteccio-nethsm
[31] Open Titan: the first open source project building a transparent, high-quality reference design and integration guidelines for silicon root of trust (RoT) chips. https://opentitan.org/
[32] Dominic Rizzo. OpenTitan at one year: the open source journey to secure silicon. Google Open Source Blog, 7 December 2020. https://opensource.googleblog.com/2020/12/opentitan-at-one-year-open-source.html
[33] Fabien Périgaud, Alexandre Gazet and Joffrey Czarny. Backdooring your server through its BMC: the HPE iLO4 case. SSTIC 2018 proceedings.
[34] Open BMC: Defining a Standard Baseboard Management Controller Firmware Stack. https://www.openbmc.org/
[35] Joel Stanley. Securing firmware: Secure and Trusted boot in OpenBMC. January 2020, LCA 2020. https://archive.org/details/lca2020-Securing_firmware_Secure_and_Trusted_boot_in_OpenBMC
[36] Infineon Techologies AG OPTIGA™ Trusted Platform Module SLB9672_2.0 v15.20.15686.00 Common Criteria Part 3 conformant EAL 4 augmented by ALC_FLR.1 and AVA_VAN.4 Certification Report. BSI-DSZ-CC-1113-2021. 21 May 2021. https://www.commoncriteriaportal.org/files/epfiles/1113a_pdf.pdf
[37] Arrêté du 13 juin 2014 portant approbation du référentiel général de sécurité et précisant les modalités de mise en œuvre de la procédure de validation des certificats électroniques, JORF n°0144 du 24 juin 2014. https://www.ssi.gouv.fr/entreprise/reglementation/confiance-numerique/le-referentiel-general-de-securite-rgs/
[38] Adi Shamir. "How to share a secret", Communications of the ACM, 22 (11): 612–613, 1979.
[39] ProvenCore secure OS achieves EAL7 Common Criteria certification, 13 September 2019. https://provenrun.com/provencore-secure-os-achieves-eal7-common-criteria-certification/
[40] Darek Fanton. Intel Platform Trust Technology – TPM for the Masses. 6 July 2022. https://www.onlogic.com/company/io-hub/intel-platform-trust-technology-ptt-tpm-for-the-masses/
[41] Patricia Pottier. The NWP systems at Météo-France. 30th ALADIN Wk & HIRLAM ASM 2020. https://www.umr-cnrm.fr/aladin/IMG/pdf/poster-france-wk2020-web.pdf
[42] Sergei Skorobogatov. Low temperature data remanence in static RAM. Technical report UCAM-CL-TR-536. University of Cambridge Computer Laboratory, June 2002. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf