AKRAINO

WHITE PAPER:

Akraino Platform Security Architecture

1st edition - March 2023

www.akraino.org

LF EDGE

AKRAINO

Table of Contents

Figures

Acknowledgment

The following materials were used in the creation of this whitepaper:

- The PSA Certified™ documents numbered [3], [4] and [7] in the references section of this whitepaper, all © Copyright Arm Limited 2017-2022.
- The Arm Aarch64 Exceptions Model document numbered [2] in the references section of this whitepaper, all © Copyright Arm Limited.
- The "Intel® Hardware Shield – Below-the-OS Security" documents numbered [8] and [9] in the references section of this whitepaper, all © Copyright Intel® Corporation.

Such content is used with permission. Your use of such content is governed by the terms set out at the front of the applicable document (links provided in the references section). This document does not provide you with any legal rights to any intellectual property in either Arm or Intel® technology or products.

This White Paper is issued for information only. It does not constitute an official or agreed position of Akraino, nor of its members. The views expressed are entirely those of the author(s) and contributor(s).

Akraino declines all responsibility for any errors and any loss or damage resulting from use of the contents of this White Paper.

Akraino also declines responsibility for any infringement of any third party's Intellectual Property Rights (IPR) but will be pleased to acknowledge any IPR and correct any infringement of which it is advised.

Copyright Notification

Trademark Notification

1 Abstract

During Akraino blueprint development, blueprint owners may put a lot of effort into analyzing security threats and implementing security features in their projects. However, in many cases, blueprint owners assume that the blueprint execution environment is well protected and does not require their attention. Such assumptions may lead to attacks using platform-level vulnerabilities that interfere with the blueprint functionality and cause the loss of private or critical data. For this reason, the requirements for platform-level security should be considered an important part of blueprint requirements.

The Akraino PSA, Platform Security Architecture, defines core security requirements for Akraino platforms and blueprint execution environments.

Akraino PSA requirements are platform agnostic and define security requirements for platform hardware and system software. Appendices at the end of the document provide information about platform specific implementations of these requirements by Arm and Intel.

1.1 Target Audience

This Whitepaper is written for the following audience:

1. Akraino Blueprint owners and developers
2. Akraino platform owners
3. Cloud environment providers
4. Akraino Blueprint integrators

2 Terms and Abbreviations

TERM	DESCRIPTION
ACM	Authenticated Compute Module
ACRAM	Authenticated Code RAM
AIK	(TPM) Attestation Identity Key
AP	Application Processor
BL	Boot Loader
CPLD	Complex Programmable Logic Device
CoT	Chain of Trust
EK	(TPM) Endorsement Key
EL	Exception Level
FPF	Field Programming Fuses
hRoT	Hardware Root of Trust
IBB	Initial Boot Block
mTLS	Mutual Transport Layer Security
OEM/ODM	Original Equipment Manufacturer/Original Design Manufacturer
NSPE	Non-Secure Processing Environment
PCH	Platform Controller Hub
PFR	Platform Firmware Resilience
PK	Public Key
PKCS	Public-Key-Cryptography Standards
PRoT	Platform Root of Trust
RMA	Return Merchandise Authorization
ROM	Read Only Memory
RoT	Root of Trust
ROTPK	Root of Trust Public Key
SCP	System Control Processor
SiP	Silicon Provider (manufacturer of SoC)
SMM	System Management Mode
SoC	System on a Chip
SPE	Secure Processing Environment
SPS	Server Platform Services
TBB	Trusted Board Boot
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TLS	Transport Layer Security
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface
VMM	Virtual Machine Manager

3 Introduction

3.1 Problem Statement

Platform security refers to the security architecture, tools and processes that ensure the security of an entire computing platform. It uses bundled/unified security software, systems, and processes to enable the security of a computing platform's hardware, software, network, storage, and other components[3]. **Figure 1** depicts the areas/layers involved with platform security.

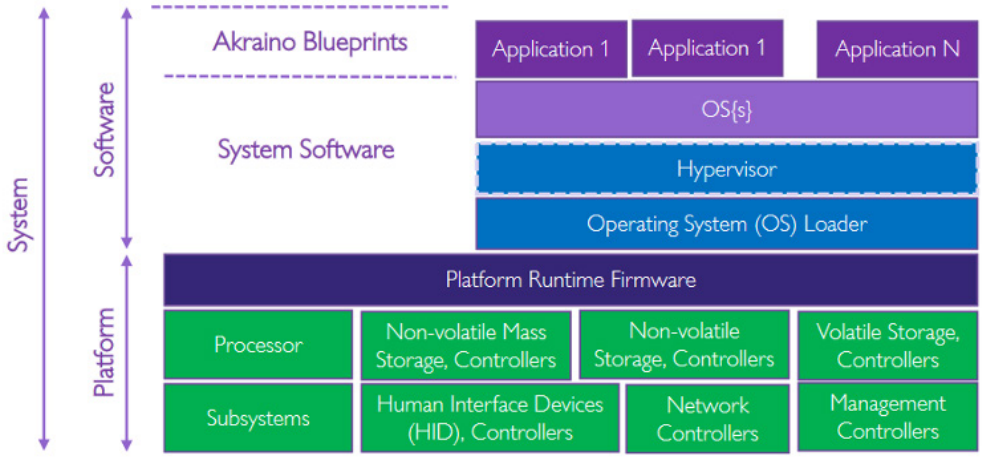

FIGURE 1 SYSTEM PLATFORM AND SOFTWARE LAYERS

The goal of platform security is to secure all layers and components within a platform. This enables securing an entire platform by using unified security requirements.

The objectives of platform security for the Akraino project are to:

- Be architecture agnostic.
- Maintain the integrity of the platform layer and provide a safe execution environment for Akraino Blueprint software stacks.
- Define secure boot requirements based on the platform Root-of-Trust.
- Attesting of the platform's secure state of integrity.
- Protection of key assets in the platform:
  - Platform critical data (platform ID, encryption keys, configuration data, etc.).
  - Mutable firmware components.
  - Secure platform firmware update.
- Protect platform runtime environment and data.
- Utilize platform security capabilities and security services in the Akraino Blueprint software stacks.

3.2 Akraino Platform Security Goals

The platform security goals include:

1. Unique identification. Devices shall be uniquely identifiable.

2. Security lifecycle. Devices shall support a security lifecycle. The device states shall be attestable and may impact access to data that is bound to the device.

3. Attestation. Devices shall be securely attestable.

4. Software authorization, devices shall ensure that only authorized software is executed. Secure boot and secure loading processes are necessary to prevent unauthorized software from being executed.

5. Secure update. Devices shall support secure update of software, or platform critical data such as hardware configuration.

6. Anti-rollback. Devices shall prevent unauthorized rollback of updates.

7. Isolation. Devices shall support isolation. Isolation of trusted services from one another and from less trusted services is essential to protect confidentiality and integrity of those services.

8. Interaction. Devices shall support interaction over isolation boundaries. The interfaces must not be used to compromise confidentiality and integrity of the interacting services or of the device.

9. Cryptographic and trusted services. All devices shall support a minimum set of trusted services and cryptographic operations that are necessary to support other security goals.

4 Platform Security

The Akraino Platform Security Architecture is platform agnostic and defines the following:

- Verified boot (also known as Secure boot) is when each SW/FW component is loaded and cryptographically verified by the previous component, aka a Chain of Trust. The first component is verified by a HW based Root of Trust. The boot is terminated if any step in the Chain of Trust fails verification.
- Measured boot is when each SW and FW component is loaded and measured by the previous component. All measurements are recorded in secured storage, but not verified. The first component in the process is an exception. First, it is loaded and verified by a HW based Root of Trust. Second, the component measures itself. The boot is terminated if this initial verification fails.
- Isolation of the trusted processes.

Depending on the platform architecture and configuration, trusted hardware can enforce protection of data from unauthorized access, implement cryptographic functionality, and protect data integrity including cryptographic keys, data measurements, platform configuration data and system runtime variables.

The combination of hardware, firmware and software that is secured, verified, and trusted provides the Trusted Computing Base (TCB) for a platform. The TCB enforces the security policy and provides protection mechanisms from breaching the policy defined restrictions, obtaining authorized privileges, and tampering with hardware.

Since edge locations often have lower physical security than either private or public cloud, this Platform Security Architecture requires an immutable HW based root of trust for all platform boot types.

The following sections provide a high-level description of verified boot, measured boot, platform root of trust and platform boot flows.

4.1 Platform Roots of Trust

The Platform Roots of Trust (PRoT) include the hardware and software components that are responsible for system level security configuration, anchoring the verified (secure) boot process to establish a chain of trust for the platform, providing trusted measurements and platform verifiable attestation data.

The Trusted Computing Group (TCG) defines a Root of Trust as a component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update[10]. A RoT is trusted always to behave in the expected manner, because its misbehavior cannot be detected such as by measurement, by attestation or observation.

There are two architectural types of Roots of Trust: Immutable and Mutable. They differ in implementation, maintainability, and trust properties. An Immutable RoT is unmodifiable by owners, administrators, or users after delivery by the RoT manufacturer.

The trust properties for an Immutable RoT are predicated by the RoT hardware and unchangeable code deployed by the vendor's manufacturing process. An Immutable RoT is expected to remain identical across all devices within a set of a device models based on a defined threat model. It is also expected not to change across time and, therefore, will behave the same over the device lifespan. An interested party may evaluate a sample of devices to verify correct behavior then choose to trust the manufacturer not to change the device's manufacturing process.

A Mutable RoT is changeable by an authorized entity. The purpose of deploying a Mutable RoT includes bug and vulnerability fixes and the addition of enhanced features.

The following Root of Trusts are typically included as part of various definitions of a Platform Root of Trust:

1. Root of Trust for Measurement

- A computing engine capable of making inherently reliable integrity measurements.
- Provides measurements of firmware, software or configuration data that can be used in attestations by the Root of Trust for Reporting.

2. Root of Trust for Reporting

- A computing engine capable of reliably reporting information held by the Root of Trust for Storage.
- Manages identities and attestation assertions.

3. Root of Trust for Storage

- A computing engine capable of maintaining an accurate summary of values of integrity digests and the sequence of digests.
- Protects the confidentiality and integrity of assets by preventing access.

4. Root of Trust for Update

- An ultimate authority for the update of a Mutable RoT.
- Manages the secure firmware and software update procedure.

5. Root of Trust for Verification

- A computing engine capable of verifying data based on cryptographic mechanisms or against platform specific values provisioned into the platform protected storage.
- Manages integrity of firmware/software and data verification.

6. Root of Trust for Recovery

- A computing engine capable of recovering platform configuration to the well-known state.
- Manages recovery when firmware is corrupted or unusable.

Depending on the platform architecture, additional Root of Trusts can by defined based the platform needs. The following sections provide examples of the platform Root of Trust services.

4.2 Platform Verified Boot

Verified Boot (aka. Secure Boot) is the process of verifying and validating the integrity and authenticity of mutable firmware and software components before their execution. Depending on the platform implementation, firmware verification can be initiated at different stages of the boot process. The Verified Boot shall be based on the Root of Trust of Verification, which shall be anchored to a Hardware Platform Root of Trust or to immutable platform firmware.

The verification process involves checking firmware integrity using cryptographic algorithms. This requires the hash value to be calculated over platform firmware images and metadata. The computed hashes can be verified against cryptographic certificates or using a hash value protected from modification by hardware and anchored to the Hardware Root of Trust.

This verification process must be successfully completed before the verified firmware image can be executed.

Depending on the platform security policy, verification failure may lead to system halt, recovery of system firmware or starting the system with limited functionality.

Figure 2 illustrates an example of the firmware verification process starting from immutable platform firmware and implementing a chain of verification for firmware components.

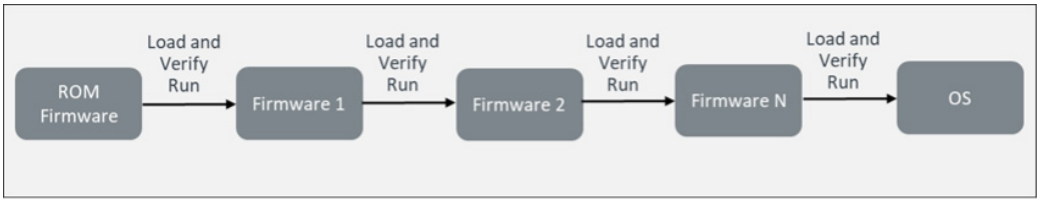

FIGURE 2 VERIFIED BOOT. CHAIN OF VERIFICATION.

Figure 3 illustrates an example of the firmware verification process initiated by the first verified mutable firmware or by trusted hardware, which controls firmware loading and the verification process.

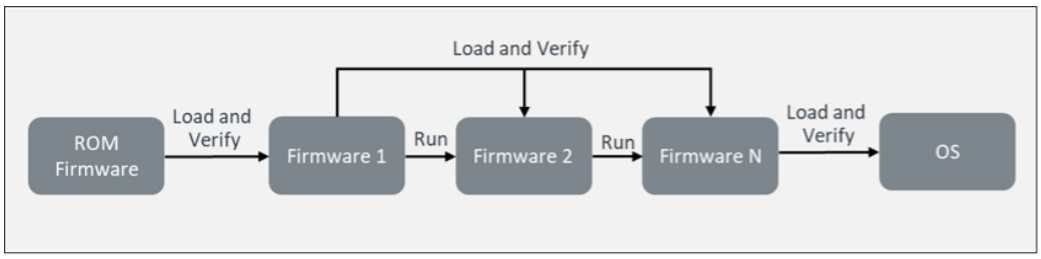

FIGURE 3 VERIFIED BOOT. VERIFICATION BY TRUSTED ENVIRONMENT.

See **Appendix B** and **Appendix C** for the platform specific implementation of Verified Boot and platform boot flows using Verified Boot.

4.3 Platform Measured Boot

Measured Boot is the process of cryptographically measuring the code and critical data so that the security state can be attested to later. Measured Boot shall be based on the Root of Trust of Measurement which shall be anchored to a Hardware Platform Root of Trust or to immutable platform firmware.

The measurement process includes but is not limited to measuring of platform firmware binary images, platform configuration data, and boot parameters. Measurement of platform data is based on cryptographic hashing algorithms and requires trusted hardware or secure services, for example, implementations that use a TPM can use Platform Configuration Register (PCR) storage. The platform measurements that are stored in PCRs represent the platform configuration as a unique number which can be used for the platform attestation purpose. **Appendix A** outlines the TCG TPM, though the system will determine whether such a TPM is required because a TPM is not a mandatory requirement.

Measured Boot does not affect the platform boot process.

Platform attestation tools can be implemented as part of the OS services as well as the remote attestation services located on the network.

Measured Boot does not affect the platform boot process.

Figure 4 illustrates the firmware measurement process starting from immutable platform firmware. It builds a unique measurement value by extending the current measurement in PCR using a measurement of the next firmware image.

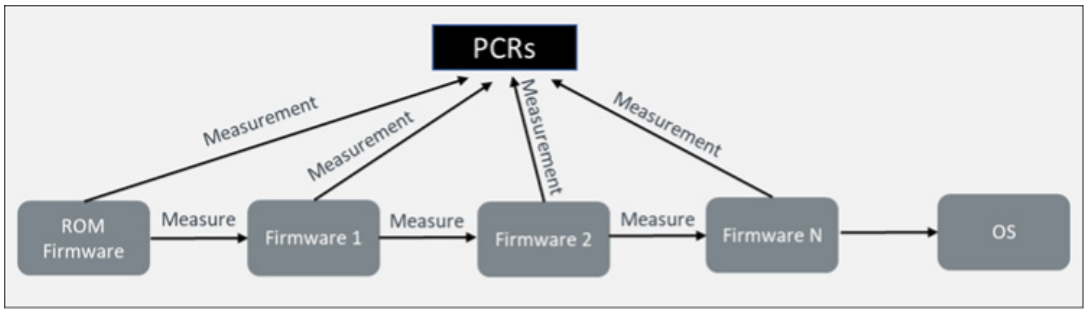

FIGURE 4 MEASURED BOOT. EACH FIRMWARE MEASURES NEXT ONE.

Figure 5 illustrates the firmware measurement process starting from immutable platform firmware. All measurements are done by the first verified mutable firmware or by trusted hardware that controls firmware loading process. This builds a unique measurement value by extending (**Appendix A**) the current measurement in PCR with measurements of the sequence of the images.

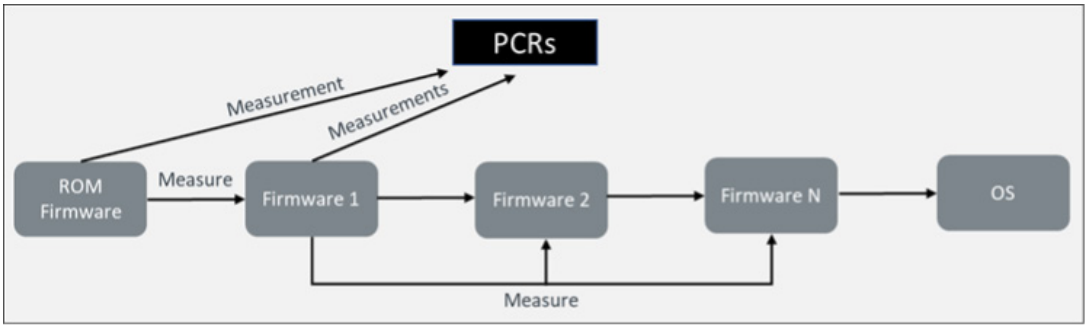

FIGURE 5 MEASURED BOOT. MEASUREMENTS BY TRUSTED ENVIRONMENT.

See **Appendix B** and **Appendix C** for the platform specific implementation of Measured Boot and platform boot flows using Measured Boot.

4.4 Platform Isolation of Trusted Processes

Mechanisms for achieving trusted process isolation on different platforms vary substantially. Many platforms include system CPU(s) that provide a secure execution environment isolated and protected from the rest of the system. Other options include isolation in the system chipset or in a dedicated security chip e.g., TPM or Secure Element, or an on-chip secure enclave. It is common for a system to utilize more than one isolation technique.

Verified and measured boot need to be trusted processes because they play a key role in establishing the root of trust and, optionally, measuring other components and providing additional services.

See **Appendix B** and **Appendix C** for the platform specific implementation of isolation of trusted processes.

4.5 Platform Boot Flows

Platform boot flow is a sequence of processes that are executed during the booting of a platform. The sequence starts from the immutable firmware and finishes at loading the platform OS. It may include multiple firmware components which initialize platform peripheral devices and a system's internal infrastructure. The sequence depends on the platform architecture and may vary between different implementations.

The boot flow must be protected from tampering during execution and implement mechanisms for the detection of platform firmware and software unauthorized modifications.

Akraino blueprints currently support execution on the multiple type of platforms. **Appendix B** and **Appendix C** provide information about the boot flows on supported platform architectures and describes security mechanisms for platform firmware integrity protection.

5 Questionnaire

The Akraino Platform Security questionnaire provides a set of questions about the platform hardware, firmware, and host software security. These questions should be used to assess the level of security implemented by a platform provider and be agnostic to the platform architecture. Akraino blueprint owners may use it to add additional security requirements for platforms which will execute an Akraino blueprint. This questionnaire is intended to be answered by a platform provider.

5.1 Use Cases

The following use cases provide guidance on how to apply the Akraino PSA questionnaire for different blueprint development and deployment scenarios.

1. Blueprint Creator:

- Blueprint User provides platform for blueprint: the Blueprint Creator can reference the PSA requirements to serve as additional security requirements that the Blueprint User should or must comply with to ensure the security of the platform running the Blueprint.
- Blueprint Creator provides the platform for the blueprint: the Blueprint Creator can provide a reference to the completed PSA requirements providing evidence to the Blueprint User that a secure platform is running the Blueprint.

2. Blueprint User:

- Blueprint Users utilize the PSA requirements to meet platform level security requirements for their specific use of the Blueprint on their provided platform. Blueprint Users can perform this step even in cases where the Blueprint Creators do not explicitly reference PSA. In this case, the Blueprint User is responsible for testing to ensure the Blueprint performs, as expected, with PSA requirements in place.
- For the public or private cloud execution environment, the Blueprint User may request a provider to answer the PSA questionnaire for ensuring that provided platforms implement the required level of security and provide a secure execution environment.

3. Chip Providers:

- Can utilize PSA requirements to ensure that their offering meets these requirements. This can provide a potential competitive advantage in the market for chips that meet PSA requirements.

4. Hardware Device Providers (i.e., server, IoT device, etc.) utilizing chips that meet the Platform Security Requirements of **section 5.2** can obtain a potential competitive advantage in the market for devices that meet PSA requirements.

5. Organizations allowing the use of opensource software can use the PSA requirements to define additional platform security requirements for Blueprints they will support. PSA requirements serve to increase the security posture of the organization.

5.2 Platform Security Questionnaire

The platform security questionnaire provides an assessment for the platform security components based on Immutable Platform Root of Trust and Platform Security Root of Trust.

#	QUESTION	DETAILS
1	The platform shall provide a hardware mechanism(s) to isolate the Secure Processing Environment (SPE) from the Non-secure Processing Environment (NSPE).	The platform data and code at different security levels should be isolated from each other.
2	The chip shall support Verified Boot, initiated from code in the immutable Platform Root of Trust, for: • All secure-processing environment (SPE) code, and • The first mutable code of the Non-Secure Processing Environment (NSPE)	The platform boot shall start from a hardware-based root of trust (hRoT) and verify next mutable firmware image.
3	The system shall support either Verified Boot or Measured Boot (or both) of NSPE code if initiated by verified boot.	The platform shall use either verified boot or measured boot after verified boot anchored to the hRoT is complete.
4	(If applicable) The Platform shall support a security lifecycle, i.e., protecting critical security parameters and sensitive data based on device lifecycle state and enforcing the rules for transition between states. Attestation reports of the platform should provide the current lifecycle state of the platform.	The platform with lifecycle support shall not expose critical data during the change of the platform lifecycle state (e.g., change from Production state to RMA/Debug state).

#	QUESTION	DETAILS
5	The Platform shall support the storage or derivation of the following minimum set (or equivalent) of critical security parameters: • A Platform RoT Public Key (ROTPK), or hash of, used for authenticating the first updateable firmware component code during secure boot. If defined by the Platform: • A secret Hardware Unique Key (HUK), used for deriving other device secrets. • A secret attestation key and identifier that uniquely identifies the attestation key. • An identifier that uniquely identifies the Platform Security RoT on the chip.	The platform shall support storage of unique keys used for different platform security operations (e.g., verified boot, attestation, etc.).
6	The Platform Security RoT shall support secure update of firmware and any Application RoTs. Updates may be delivered either from locally connected devices (such as removable media) or from remote servers.	The platform shall implement a secure firmware update mechanism.
7	The update mechanism shall prevent unauthorized rollback of updates. A mechanism may be provided to support authorized rollback for recovery reasons. Anti-rollback is strongly recommended but not mandatory in PSA.	The platform shall be protected from unauthorized rollback of firmware.
8	The Platform Security RoT shall protect unauthorized modification of platform security parameters, system software, and device sensitive data. The Platform Security RoT may enforce authorized access to platform security parameters, system software, and device sensitive data.	The platform security RoT shall be used to protect and authorize access to platform sensitive data (e.g., the platform RoT public key).
9	The Platform Security RoT shall use best practice cryptography (algorithms, key, and hash sizes, etc.) for protection of its assets. This includes the provision of a suitable source of random data with hardware based RNG.	The platform shall always use best practice cryptography. The best practice may depend on the region or country.

5.3 System Software Security Questionnaire

The table below provides additional details for the System Software Security Questions

#	QUESTION	DETAILS
1	The System Software shall be updatable either from locally connected devices (such as removable media) or from remote servers.	It is expected that security related fixes will be required during the system lifetime and hence this requirement that assumes that System Software updates are verifiable.
2	(Optional) The update mechanism shall prevent unauthorized rollback of system software and authentication data. A mechanism may be provided to support authorized rollback for recovery reasons.	A system software rollback should be either prohibited or authorized.
3	The System software shall rely only on the Platform Security RoT for all queries of the Platform Security RoT identity(s).	Here are two use case examples: • An ID is directly based on platform RoT e.g., KF Edge FDO project uses TPM for establishing platform identity. • An ID is SW defined. In such cases, the platform RoT should be used to protect against unauthorized accesses to this ID e.g., by using LUKS disk encryption with the key stored in the TPM.
4	The System software shall use secure storage to protect sensitive data and provide this functionality for application data. If supported, it shall seal the sensitive data in a specific device instance and include the security lifecycle state.	An example is using data sealing with the encryption key stored in the TPM and protected with the Extended Authorized Policy. Another example is LUKS disk encryption, with the key stored in the TPM. One more example is platform services running in a trusted environment.

#	QUESTION	DETAILS
5	The System Software shall use best practice cryptography (algorithms, key, and hash sizes, etc.) for protection of its assets. This includes the provision of a suitable source of random data with hardware based RNG. There should be no reliance on proprietary cryptographic algorithms or customization of standard cryptographic algorithms.	An appropriate industry best known practice for cryptography must be followed. These practices are changing over time and may differ between different geographies. It is a responsibility of the blueprint owner to select and indicate an appropriate source.
6	System software shall provide the ability to authenticate remote devices and servers when establishing a two-way connection to support data integrity, confidentiality, or authenticity	Examples are: • mTLS • LF Edge FDO channel established according to TO2 protocol
7	The system software shall provide the ability to encrypt and enforce integrity of data exchanged with remote devices and servers.	Examples are the same as for #6 above
8	The System software shall use secure protocols for authentication and protection of two-way communication. These protocols shall not leak data that would lead to the identification of the devices.	TLS is an example of an acceptable protocol.
9	(Optional) Functionality packages that are not needed for the intended use shall not be installed or shall be disabled if non-installation is not practical.	If an OS or UEFI packaged is not used by the blueprint (and it is practical to remove), it is recommended to remove this package from the installation to prevent its possible misuse.
10	The system software shall support an attestation method that can be used to prove the genuineness of the device and its integrity. If possible, the current security lifecycle state of the device should be included.	End-to-end support for the measured boot and corresponding platform attestation with the assistance of a software agent. The platform should have this capability, but its enabling is at the discretion of the system user and/or blueprint developer

#	QUESTION	DETAILS
11	The system software should provide logging of relevant security events and errors. The log should include sufficient details for root cause analysis and should be protected from unauthorized access.	Security logging is critical for the investigation of incidents and auditing. What to log is at the discretion of the system user and/or blueprint developer. Using disk encryption (e.g., LUKS) is an example of log protection
12	Where supported, the system software shall enable the execution of application specific software and system software with the lowest level of privileges necessary for the intended function.	The OS (e.g., Linux) shall not be used without the accounts and/or privileges set to least privilege levels and Blueprint owners must also use least privilege levels for their components and users.
13	If the system software has a mechanism to reset authentication and authorization parameters, they shall not be resettable to any universal factory default value. Such data must not be easily determined by automated means or obtained from publicly available information.	The system software should have a secure mechanism for a user to reset authentication parameters. The implementation of this mechanism and can be based on different recovery processes, such as a system reset requiring entering a password, email response verification, security questions, etc.
14	If the system software makes use of cryptographic algorithms for user authentication, the cryptography used for that feature shall comply with requirement #5 in the System Software Security section.	The system may use different methods for user authentication like certificates (e.g., SSH access), smart cards, authenticators, etc.
15	If the System software allows persistent storage of data, it shall support protection against unauthorized access and mechanisms for access rights management.	System may define different user access rights to the data on the platform storage or TPM Extended Authorization policies.

6 Conclusion

Platform Security provides the tools necessary to extend security for Akraino Blueprints all the way down to the hardware level. Akraino Platform Security takes a vendor agnostic approach and attempts to focus on fundamental Platform Security areas that all vendors should support. By implementing the Platform Security requirements defined, the Akraino Blueprint user can be assured that (1) the hardware/firmware running their software is free from known malware, (2) a secure area to store platform security sensitive keys/passwords/information is provided, (3) the secure update of firmware is ensured, and (4) the integrity of the system firmware is maintained.

In summary, by implementing Akraino Platform Security, Blueprint owners can be assured that they are taking advantage of available platform security tools to ensure their Akraino Blueprint is running on a secure platform.

Appendix A

This appendix describes platform independent implementation of security modules and devices.

A.1 TPM as a Root of Trust for Measurements and Reporting

TCG (Trusted Computing Group) is an industry organization that published a specification defining a Trusted Platform Module (TPM). The currently defined version (or family) is TPM 2.0. Use of a TPM depends upon each specific implementation and is not a mandatory requirement for an Akraino platform.

The TPM spec defines multiple capabilities. Two of them are of primary interest for the measured boot:

TPM serves as a Root of Trust for Storage (ROTS)

- The measurements are stored in Platform Configuration Registers (PCRs).
- On each TPM boot, all PCRs are reset to an initial value.
- The only way to change a PCR value is by "extending" it. Extending means that the TPM calculates a hash value of the current PCR value concatenated with a new measurement value, typically a hash of the code or data, and stores the new hash value in the PCR. The symbol "||" below means concatenation:
  PCR(new) = HASH (PCR(old) || measurement(new))
- The TPM specifications defines multiple hashing algorithms.
- If a platform extends a PCR multiple times, all data is hashed into the PCR. Extending measurements in a different order will produce a different PCR result value.
- After the measured boot is done each PCR holds a final value that is a product of all extend operations performed by the TPM for this PCR during the measured boot.

• The TPM also serves as a Root of Trust for Reporting (ROTR). It can produce a signed quote (aka measurement boot report) in a response from a boot verifier. The quote among many other parameters contains the PCR values. There is also a TCG SCRTM (Static Core Root of Trust for Measurement) event log of all completed extend operations. The verifier may request the log and "replay" it as needed. Below is a quote signing key chain:

- A TPM comes with an asymmetric Endorsement Key (EK) provisioned during its manufacturing process. The EK private part is known to the TPM only. The EK public part is included in the certificate issued by the TPM manufacturer. The certificate is signed by the manufacture CA signing key.

- A platform owner instructs the TPM to generate an asymmetric Authentication Key pair (AK). The AK private part is known to the TPM only and the public part is signed by the EK.

The AK is used to sign the TPM quote. The EK is not used directly for signing the TPM quote due to privacy considerations.

Appendix B

This appendix describes Intel specific security models and boot flows.

B.1 Intel Platform Security Model

B.1.1 Measured Boot Support on x86 Platforms

A diagram below depicts verified boot and measured boot concepts side-by-side.

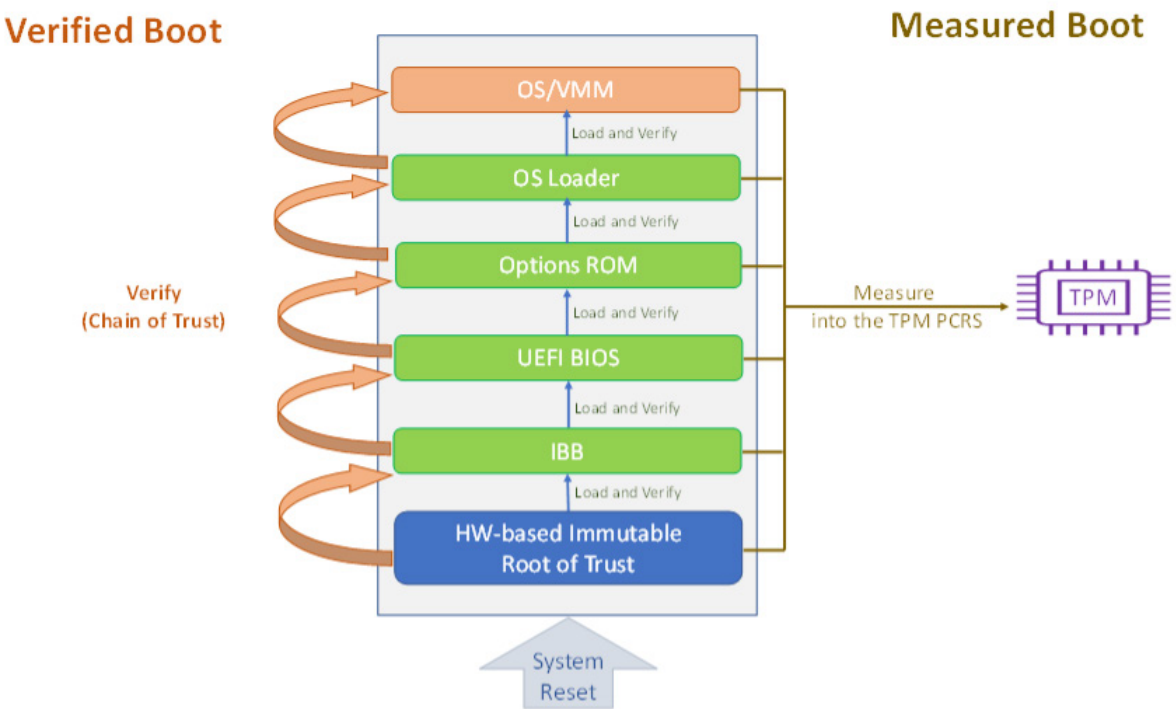

FIGURE 6 VERIFIED BOOT AND MEASURED BOOT SIDE-BY-SIDE.

B.1.2 Immutable RoT Support on x86 Platforms

Historically, x86 based platforms would often utilize a boot ROM/FLASH as a root of trust for the firmware verification, measurement, update, and recovery. This approach doesn't provide an immutable root of trust and consequently doesn't satisfy Akraino Platform Security requirements.

Fortunately, modern x86 based systems are typically shipped with HW-based security capabilities that satisfy Akraino Platform Security requirements. An OEM can utilize appropriate CPU/SoC features to deliver an end-to-end security solution for x86 based system e.g., the table below maps root of trust types to Intel® provided features.

Root of Trust	Intel® Boot Guard	Intel® Bios Guard	Intel® PFR
Verification	Yes (CPU/ACM)	-	Yes (PFR CPLD)
Measurement	Yes (CPU/ACM)	-	Yes (PFR CPLD)
Storage	Yes (TPM/PCR)	-	Yes (TPM/PCR)
Reporting	Yes (TPM/EK and AIK)	-	Yes (TPM/EK and AIK)
Firmware Update	-	Yes (SMM and ACM)	Yes (PFR CPLD)
Firmware Recovery	-	-	Yes (PFR CPLD)

ACM – Authenticated Compute Module is executed in a special mode as a part of the immutable root of trust (in conjunction with CPU microcode)

CPLD – Complex Programmable Logic Device. The CPLD is the root-of-trust in a system that is designed based on the Intel® PFR.

TPM – Trusted Platform Module

Section "Platform Boot Flow with Intel® Boot Guard" later in this document describes in detail how Intel® Boot Guard is used for measured and secure boots.

B.2 Intel platforms boot flow

B.2.1 Platform Boot Flow with Intel® Boot Guard

Intel® Boot Guard is an example of the HW based Root of Trust in the case of verified boot and an example of Root of Trust for Measurements in the case of measured boot. A platform boot starts with:

- Intel Server Platform Services, executing in the chipset, that reads the OEM Boot Policy
- CPU Microcode that authenticates Intel® Authenticated Code Module (ACM) binary.
- Intel® ACM that verifies the OEM provided Initial Boot Block (IBB).

Boot Guard relies on PCH (Platform Controller Hub) Field Programming Fuses (FPFs) to store the hash value of the OEM public key and Boot Guard boot policy. The FPFs can be programmed only once, and the OEM does it during the platform manufacturing process.

The ACM performs critical tasks in the Boot Guard solution. It is digitally signed by Intel and stored on the flash together with BIOS and other firmware components. The public key for verifying the signature of the ACM is hard coded in Intel's CPU. The CPU opcode loads the ACM in the internally protected L2 cache, called ACRAM (Authenticated Code RAM), which then verifies the ACM and, only in the case of successful verification, the CPU allows the ACM to execute. ACM is required to execute successfully when Boot Guard is enabled; failure to authenticate ACM results in a CPU shutdown.

The ACM loads an Initial Boot Block (IBB) provided and signed by the OEM. The IBB is a Boot Guard specific requirement for the OEM; typically, this includes silicon and memory initialization FW. The reason is to provide a small module that can be quickly measured by the ACM in the internally protected memory. The ACM loads the IBB into last-level cache and enables No-Eviction Memory, to create a secure, isolated environment for the BIOS code to run. ACM then verifies the IBB signature using the OEM Boot Guard key store in the FPF. Only after a successful verification will ACM transfer control to the IBB. Failure to verify IBB results in the CPU being shut down through SPS enforcement.

A diagram below depicts the Boot Guard ACM and IBB execution flow.

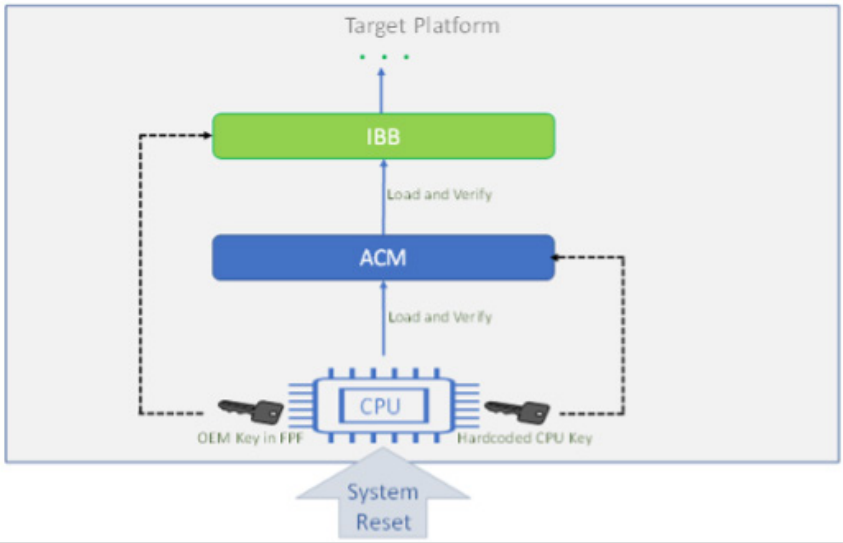

FIGURE 7 INTEL® BOOT GUARD CONCEPT

A diagram below depicts a simplified verified boot flow with Intel® Boot Guard serving as an immutable HW-based verification root of trust.

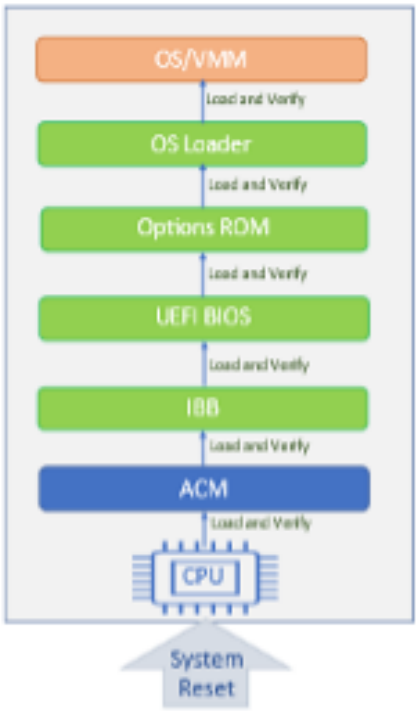

FIGURE 8 VERIFIED BOOT EXAMPLE

After the system resets, the CPU loads and verifies the ACM which loads and verifies the IBB as described in the previous section. The integrity enforcement of the subsequently loaded components is performed by a chain of trust. The following components (in green) are provided by the OEM and each component must load and verify a subsequent component before transferring control to it. A hash or a digital signature can be used to ensure the data integrity during the load and verification process. If the verification fails, the boot process is terminated. Each arrow in the diagram depicts this "load-and-verify-before-execute" pattern. Ultimately, the system loads and verifies OS/VMM provided by the OSV.

The verified boot may be combined with the measured boot. Refer to **section B.2.3** on how the verified boot may fit into a bigger picture.

B.2.2 Measured Boot Flow with Intel® Boot Guard and TPM

The measured boot differs from the verified boot. In the verified boot, each component authenticates and cryptographically verifies the next component. The verified boot terminates in case of any authentication or verification failure. In the measured boot, one component measures the next component into a TPM PCR before loading the following component. The measured boot won't fail because no verification is performed during the boot. This boot sequence is called a Chain of Trust (COT). There is one exception related to execution of the first two FW components (ACM and IBB) in the boot flow as discussed later in this section.

A diagram below depicts a simplified measured boot flow that uses Intel Boot Guard as a root of trust for the measurements and TPM as a root of trust for the storage and reporting.

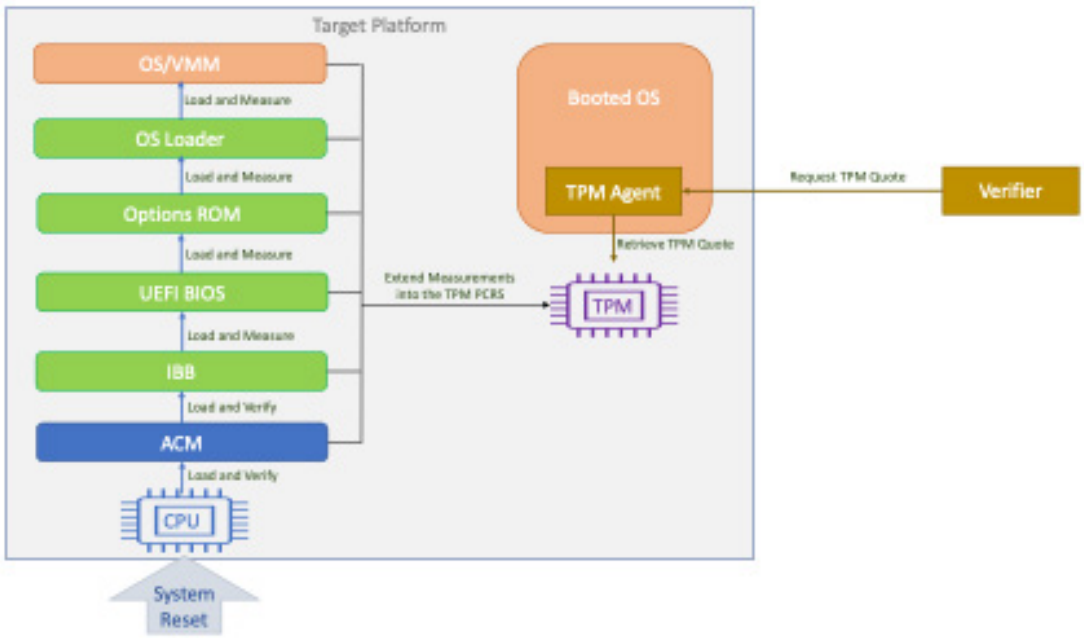

FIGURE 9 MEASURED BOOT EXAMPLE.

The measured boot starts by executing the Boot Guard ACM and IBB as defined in **section B.2.1**. The boot terminates if either the ACM verification by the CPU opcode or IBB verification by the ACM fails. The arrows between the CPU and the ACM and ACM and IBB represents the verified boot behavior even though (the rest of) the platform is configured for the measured boot (notice "Load and Verify" text instead of "Load and Measure" for other blue arrows). Additionally, the ACM extends its self-measurements into the TPM PCR0 in addition to the IBB measurements. Then the IBB and each subsequent component (except the last one) in the Chain of Trust load and measure the next component. The Chain of Trust extends all measurements into the corresponding TPM PCRs according to the TCG Platform Firmware Profile Specification.

As memory is not available at reset, ACM relies on verified BIOS to determine if Boot Guard is enabled and generate the first events for the TPM event log: the event log header, locality startup event, and SCRTM events must be created before any further events are logged by BIOS, to ensure event log replay succeeds.

After the measured boot is completed, the platform is considered trusted, but not necessarily secure e.g., the PCR values may indicate a known vulnerability in one of the components. It is up to other software (called a Verifier on the diagram above) to make a security decision based on the TPM quote and, optionally, the corresponding boot log. The Verifier must trust only the TPM (quote). The TPM Agent on the Target Platform facilitates access to the TPM, but it is not trusted because it runs as a regular OS process or system service regardless of the system trust status and before it is known. For example, one of

the implications is that the Verifier itself must generate a one-time nonce which is sent to the TPM for an inclusion in the quote to mitigate possible quote replay attacks.

B.2.3 Measured and Verified Boot

The previous sections described verified boot and measured boot in isolation, but in the real world they can be joined into a combined measured and verified boot. There are good reasons for such a combination:

- To catch configuration (aka user) errors, e.g., the platform is booted with a signed OS kernel, but the kernel has an incorrect version.
- To detect booting of an older, but still formally valid and signed, firmware that may have vulnerabilities.
- To distinguish between normal and recovery boot modes if the platform has an ability to boot in the recovery mode in cases when the verification boot fails.

The verified boot and/or measured boot is not used alone for protecting the platform, but in a combination with additional software for applying appropriate policies e.g.,

- Disk encryption software may use a disk encryption key sealed in the TPM.A combination of the TPM PCR values is configured as a TPM Extended Authentication Policy. The key is released only if the current TPM PCR state matches the policy. E.g., LUKS can be configured in such way.
- Kubernetes (K8S) POD execution policy enforcement. An external Hardware Verification Service (HVS) retrieves a platform TPM quote and verifies it. The HVS verification result is used to decide if the platform can be allowed to execute K8S pods.
- Restricting access to external resources like network devices or storage based on platform measurements.

Appendix C

C.1 Arm Platform Reference Boot Flow

The following section describes the boot flow and platform firmware verification sequence for the platform firmware reference implementation provided by Arm [6]. The firmware boot flow may be adjusted by the OEM/ODM or SiP depending on the platform configuration and requirements. However, the security requirements for platform firmware should not be affected by the platform vendor boot flow adjustments.

C.1.1 Trusted Boot

The Trusted Board Boot (TBB) [1] feature prevents malicious firmware from running on the platform by authenticating all firmware images up to and including the normal world bootloader. It does this by establishing a Chain of Trust using Public-Key-Cryptography Standards (PKCS).

C.1.2 Chain of Trust

A Chain of Trust (CoT) starts with a set of implicitly trusted components. In the Arm reference implementation, these components are:

- A SHA-256 hash of the Root of Trust Public Key (ROTPK), see [3] and [5].It is stored in the trusted root-key storage registers.
- The BL1 image, on the assumption that it resides in ROM so it cannot be tampered with.

The remaining components in the CoT are either certificates or boot loader images. The certificates follow the X.509 v3 standard. This standard enables adding custom extensions to the certificates, which are used to store essential information to establish the CoT.

The certificates are categorized as “Key” and “Content” certificates. Key certificates are used to verify public keys which have been used to sign content certificates. Content certificates are used to store the hash of a boot loader image. An image can be authenticated by calculating its hash and matching it with the hash extracted from the content certificate.

C.1.3 Execution and Security states

The current state of an Armv8-A or Armv9-A processor is determined by the Exception level [2] and the current Execution state. The current Execution state defines the standard width of the general-purpose register, and the available instruction sets.

Execution state also affects aspects of the memory models and how exceptions are managed.

The current Security state controls which Exception levels are currently valid, which areas of memory can currently be accessed, and how those accesses are represented on the system memory bus.

Figure 10 shows the Exception levels and Security states, with different Execution states being used.

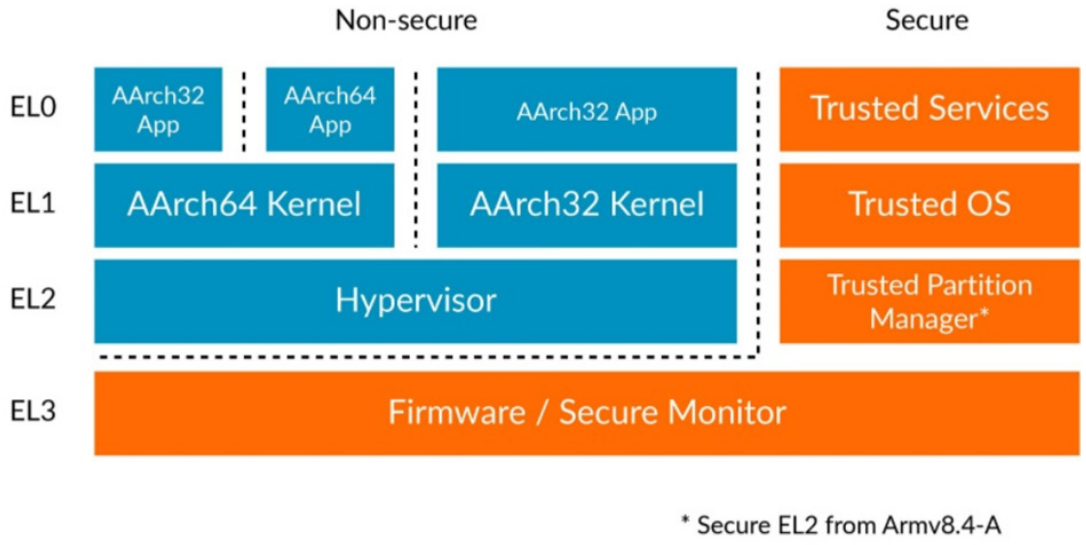

FIGURE 10 ARMV8-A AND ARMV9-A EXCEPTION MODEL.

C.1.4 Trusted Board Boot Sequence

A boot sequence may contain multiple steps and may load multiple images of firmware. The Arm reference implementation [6] defines five boot stages, called Boot Loader (BL), which loads specific pieces of firmware:

- Boot Loader stage 1 (BL1) AP Trusted ROM.
- Boot Loader stage 2 (BL2) Trusted Boot Firmware.
- Boot Loader stage 3-1 (BL3-1) EL3 Runtime Firmware.
- Boot Loader stage 3-2 (BL3-2) Secure-EL1 Payload (optional).
- Boot Loader stage 3-3 (BL3-3) Non-trusted Firmware.

The CoT is verified through the following sequence of steps illustrated in **Figure 11**. The system stops booting or switches to recovery mode if any of the steps fail.

- BL1 loads and verifies the BL2 content certificate. The issuer's public key is read from the verified certificate. A hash of that key is calculated and compared with the hash of the ROTPK that is read from the trusted root-key storage registers. If they match, the BL2 hash is read from the certificate.

- BL1 loads the BL2 image. The BL2 hash is calculated and compared with the hash read from the BL2 content certificate. Control is transferred to the BL2 image if all the comparisons succeed.
- BL2 loads and verifies the trusted key certificate. The issuer's public key is read from the verified certificate. A hash of that key is calculated and compared with the hash of the ROTPK that is read from the trusted root-key storage registers. If the comparison succeeds, BL2 reads and saves the trusted and non-trusted world public keys from the verified certificate.

The next two steps are executed for each of the SCP_BL30, BL31, & BL32 images. The steps for the optional SCP_BL30 and BL32 images are skipped if these images are not present.

- BL2 loads and verifies the BL3x key certificate. The certificate signature is verified using the trusted world public key from the trusted key certificate. If the signature verification succeeds, BL2 reads and saves the BL3x public key from the certificate.
- BL2 loads and verifies the BL3x content certificate. The signature is verified using the BL3x public key. If the signature verification succeeds, BL2 reads and saves the BL3x image hash from the certificate.

The next two steps are executed only for the BL33 image.

- BL2 loads and verifies the BL33 key certificate. If the signature verification succeeds, BL2 reads and saves the BL33 public key from the certificate.
- BL2 loads and verifies the BL33 content certificate. If the signature verification succeeds, BL2 reads and saves the BL33 image hash from the certificate.

The next step is executed for all the boot loader images.

- BL2 calculates the hash of each image. It compares it with the hash obtained from the corresponding content certificate. The image authentication succeeds if the hashes match.

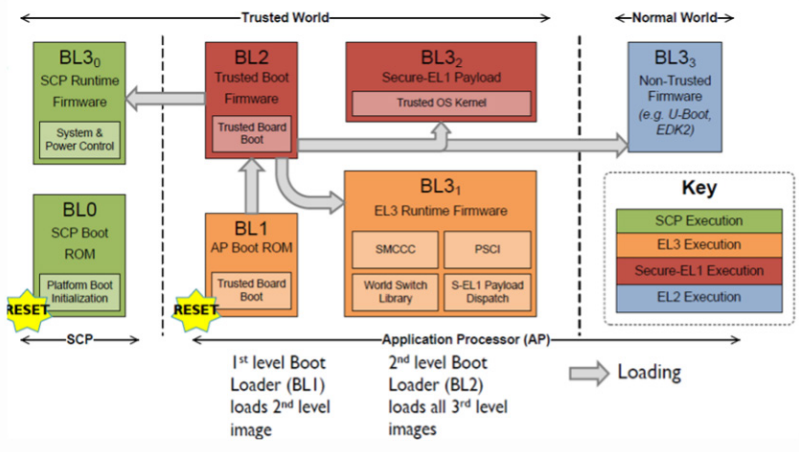

FIGURE 11 TRUSTED BOOT FLOW.

C.1.5 Measured Boot and Attestation

An SoC may need to prove the integrity of its software to a remote party or to local systems on the same board. A prerequisite for attesting the platform state is to create measurements of loaded code and data on each boot. The measurements are then securely stored in a trusted subsystem. This is known as a measured boot. The measurement report provided to local or remote attestation mechanisms can be used to assess the integrity of such firmware and makes it part of an overall chain of trust.

Each stage of the chain of trust accurately and robustly measures all the critical code and data that will be loaded. This also includes:

- Loadable modules (including dynamic patches and code loaded from peripherals)
- Parameters that influence boot behavior (for example, flags or variables that may change the control flow of the loaded program).

Each stage of the chain of trust stores the measurements in a local root of trust. The measurements may be held in a security module or in other types of trusted subsystems (e.g., TPM, **Appendix A**). A remote party can use the list of measurements to help validate the specific software identity of the platform.

The immutable bootloader can store a boot state that is accessible by the RoT runtime services. Part of the boot state includes a freshly generated number called a boot seed. The boot seed may be used by later services, for example to allow a validating entity to ensure that attestations for different attestation end points were generated in the same boot session. It must be large enough to make global collisions statistically improbable.

It is possible for systems to provide multiple images in the form of a signed firmware image package or a single image. A firmware image package allows for packing bootloader images (and potentially other payloads) into a single archive that can be loaded. Nevertheless, each component must be measured independently. This is necessary for a remote party to easily verify a remote attestation.

References

[1] Trusted Board Boot Requirements: **https://developer.arm.com/documentation/den0006/latest**

[2] AArch64 Exception Model: **https://developer.arm.com/documentation/102412/0102**

[3] Platform Security Model: **https://www.psacertified.org/app/uploads/2021/12/JSADEN014_PSA_Certified_SM_V1.1_BET0.pdf**

[4] Platform Threat Model and Security Goals: **https://www.psacertified.org/development-resources/building-in-security/platform-threat-model-and-security-goals**

[5] Platform Security Boot Guide: **https://developer.arm.com/documentation/den0072/0101/**

[6] Arm Trusted Firmware for A-class processors: **https://www.trustedfirmware.org/projects/tf-a/**

[7] PSA Certified Level 1 Questionnaire: **https://www.psacertified.org/app/uploads/2022/06/JSADEN001-PSA_Certified_Level_1-2.2-REL-01.pdf**

[8] Intel HW Shield (including Intel(R) Boot Guard): **https://www.intel.com/content/dam/www/central-libraries/us/en/documents/below-the-os-security-white-paper.pdf**

[9] Secure Boot with Intel(R) Boot Guard in context of Network Infrastructure: **https://builders.intel.com/docs/networkbuilders/secure-the-network-infrastructure-secure-boot-methodologies.pdf**

[10] TCG Glossary: **https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf**

Authors

Daniil Egranov (Arm Ltd.)
Eugene Yarmosh (Intel)
Randy Stricklin (AT&T

Contributors

Catharine West (Intel)
Don Banks (Arm Ltd.)
Rob Smart (Arm Ltd.)


AKRAINO
LF EDGE