AKM Device Authentication for Secure Boot

AKM CYBER CORP
AUTOMATING CYBERSECURITY

AKM Secure Boot Implementation

The implementation is divided into three parts:

1) HW dependent Secure Boot (optional);
2) Bootloader with main AKM Secure Boot functionality;
3) Hardware Secure Enclave.

The Boot Procedure uses an AKM Encrypted File to store all required system configuration parameters (example: the host device’s unique “hardware identifier” (could be via a PUF) and digital signatures of user application images).

AKM Secure Boot Implementation

Hardware Dependent Secure Boot

The HW dependent Secure Boot is an optional part of the Secure Boot Process and must be directly supported by the target microprocessor. Details of the implementation of a secure boot process, varies from manufacturer to manufacturer, but the main concept is virtually always the same – that is, it (the host processor unit) delivers functionality of a hardware Root-of-Trust that can be used to initiate a chain of trusted and encrypted binary images running on the host device. In the solution presented within this document, the HW dependent Secure Boot is responsible for confidentiality and integrity of the stage-1 bootloader binary image.

In the event that the selected microprocessor does not support HW Secure Boot, then this step will be omitted.

Bootloader with AIM Secure Boot Functionality

The standard bootloader that is supported by the target microprocessor has the additional functionality that it is responsible for checking the confidentiality and integrity of selected components within the host device. Thus, simultaneously, providing implicit device authentication of the AIM Identifier and critical system digital components.

Hardware Secure Enclave

This is a specialized hardware device that is responsible for performing cryptographic operations to validate system components or software images. A hardware secure enclave stores all cryptographic credentials within a physically protected external memory.

AKM Secure Boot Procedure

The AKM Secure Boot Procedure with a hardware secure enclave is presented both as text found immediately below, and then again subsequently within the diagram on Slide 6 depicting the same process.

After initialization of both the hardware secure enclave and the host device itself, the hardware secure enclave uses the secret password value (i.e., the PASS) to create a cryptographic key. This cryptographic key is then used to decrypt the AKM Encrypted File.

The following steps are preformed after device power-up:

1) If the microprocessor is equipped with a hardware secure boot mechanism, the bootloader image is decrypted and authenticated at the hardware level. If this authentication procedure ends with failure, then the device will enter into a Secure Stop State, as defined according to the predefined Security Policy. However, in the nominal case of success by the authentication procedure, control is passed to the bootloader.

2) The Bootloader reads or calculates the Hardware Identifier (HWID) with one of four methods:

   i. CPU Chipset comes with a readable Globally Unique Identifier (GUID).

   ii. Applying the SHA2 hash function to a concatenation of serial numbers from selected system physical components (or other similar strategy).

   iii. Using a Physically Unclonable Function (PUF).

   iv. Using some other mechanism that is capable of generating a repeatable, unique hardware identifier of a specified size.

AKM Secure Boot Procedure

3) The HWSE decrypts the AKM Encrypted File using the cryptographic key that is calculated by the equation: HMAC-SHA2(HWID, PASS) where PASS is a secret password stored within the HWSE.

4) If the decryption procedure ends with failure, the device enters into a Secure Stop State defined according to the predefined Security Policy.

5) In the next step, bootloader calculate SHA2 hash code of each protected image and checks it with code stored in AKM Encrypted File.

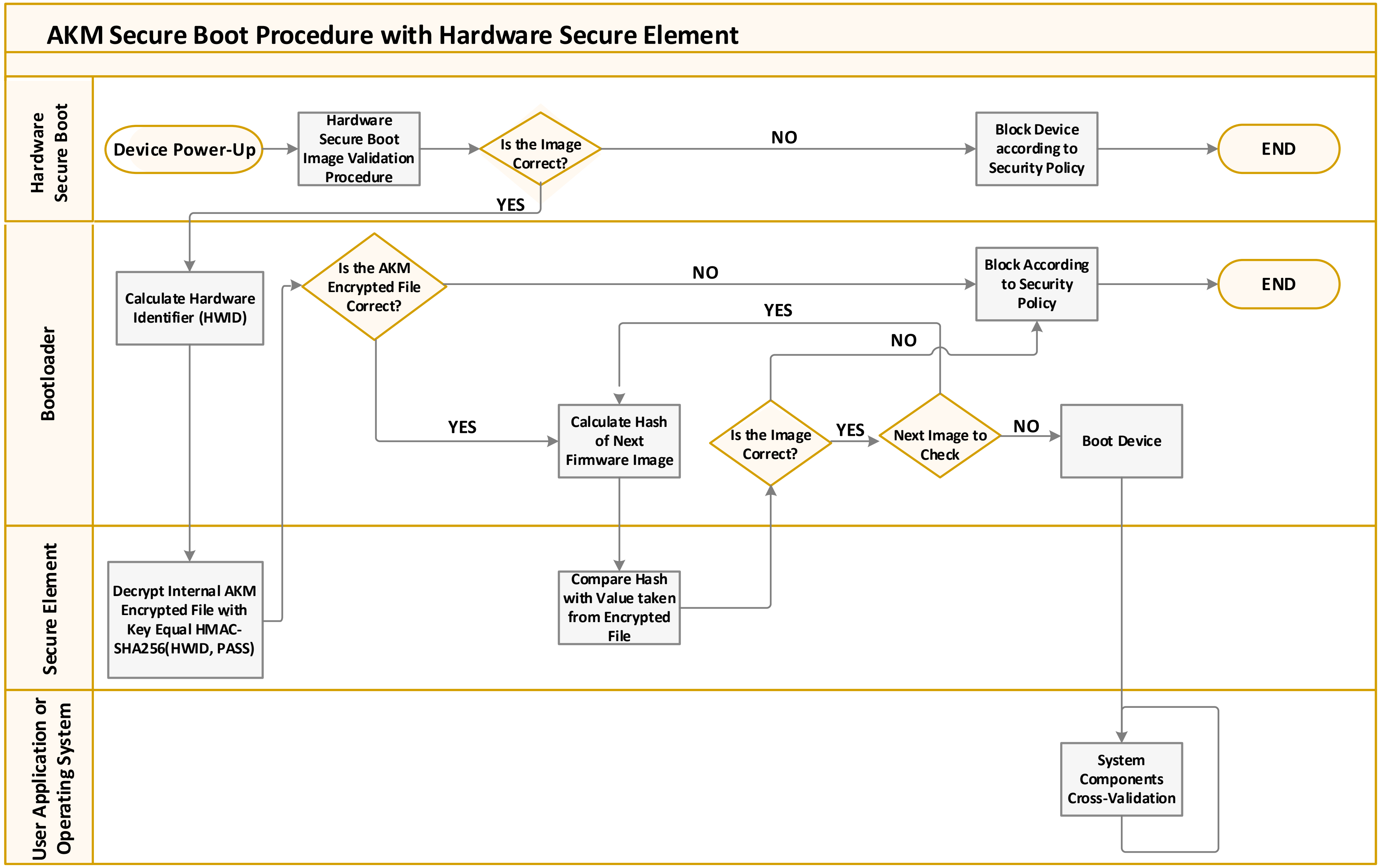
6) If authentication procedure ends with failure, device enters Secure Stop State defined according to Security Policy.

7) Successful authentication of all available images finishes boot process.

8) After the primary system image is loaded and executing, a second stage AKM authentication application running within the primary system image will perform cross-authentication of system external components. The process will use the AKM security communication protocol to securely exchange information between devices for the purpose of device identification and authentication as part of a larger and encompassing subsystem.

9) After all devices within a subsystem have been identified and authenticated, one device that has been previously designated to represent the subsystem will communicate with other subsystems to cross-authenticate the individual subsystems, thus ensuring the identity, integrity, and authenticity of the system as whole.

AKM Secure Boot Procedure (Diagram)


AKM Secure Boot Procedure with Hardware Secure Element
Hardware Secure Boot
Bootloader
Secure Element
User Application or Operating System
Device Power-Up
Hardware Secure Boot Image Validation Procedure
Is the Image Correct?
NO
YES
Block Device according to Security Policy
END
Calculate Hardware Identifier (HWID)
Is the AKM Encrypted File Correct?
NO
Block According to Security Policy
END
YES
NO
YES
Calculate Hash of Next Firmware Image
Is the Image Correct?
YES
Next Image to Check
NO
Boot Device
Decrypt Internal AKM Encrypted File with Key Equal HMAC-SHA256(HWID, PASS)
Compare Hash with Value taken from Encrypted File
System Components Cross-Validation

Device Integrity Code (DIC)

An Endpoint Integrity Code (EPIC) is derived by taking the digital signature of the coupling of an HMAC of the Hardware Identifier, the Device's Asset Tag, and an aggregation of the File Integrity Codes of all of the electronic images comprising the firmware release for the specific device. The below mathematical function specifies the calculation for an EPIC:

- $EPIC = HMAC\ (SHA(AssetTag_{Device}\ ||\ SHA(SDS_{Device})\ ||\ FIC_{IMG1}\ ||\ FIC_{IMG2}\ ||\ FIC_{IMG3}\ \ldots,\ HMAC\text{-}SECRET_{DIC})$

Where:

- ❑ $AssetTag_{Device}$, represents the Asset Tag of the Device.
- ❑ $SHA(SDS_{Device})$, represents the Synchronized Data Set (SDS) of the device.
- ❑ FIC_{ImgX}, represents the File Integrity Code of Image X.
- ❑ $HMAC\text{-}SECRET_{DIC}$ is the secret key used during the first pass of the HMAC process.

Values unique to individual devices will be used to validate the authenticity of an individual device. If the value on the device does not equal the expected value, then the hardware can be considered to be compromised (and not the device it claims to be).

By including the Device SDS as part of the calculation of the EPIC, this ensures the integrity of the Device SDS is implicitly verified by other devices within the same subsystem (as part of the SSIC).

File Integrity Code (FIC)

A File Integrity Code (FIC) is derived by taking the digital signature of the coupling of an HMAC of electronic image asset and the electronic image's asset tag. The electronic image, its asset tag, and the digital signature are all encrypted, thus preventing the tampering of the electronic image asset. If the decrypted value of the combined electronic image and asset tag, do not match the expected value, the electronic image will be considered compromised.

This is the 256-bit SHA256, resultant HMAC run over the electronic image component and its associated asset tag and maintained as part of the BackEnd Server's configuration management system. The below mathematical function specifies the calculation for a FIC:

- FIC = HMAC (SHA(IMGx) || SHA($\text{AssetTag}_{\text{ImgX}}$), $\text{HMAC-SECRET}_{\text{FIC}}$)

Where:

- ❑ IMGx, represents the Image labeled, component 'x'.
- ❑ $\text{AssetTag}_{\text{ImgX}}$, represents the Asset Tag of Image X.
- ❑ $\text{HMAC-SECRET}_{\text{FIC}}$ is the secret key used during the first pass of the HMAC process.

Values unique to individual devices will be used to validate the authenticity of an individual device. If the value on the device does not equal the expected value, then the hardware can be considered to be compromised (and not the device it claims to be).

Asset Tag Definitions

Asset Tags are the methodology for how Assets are tracked and monitored. Below is a partial list of Asset Tag type definitions, with the first two specific to what is used within the secure boot process.

The **Asset Tag of a Hardware Device**, includes, but is not limited to:

- ❑ The companywide unique AKM identifier associated with the hardware device;
- ❑ The Unique Value associated with the hardware device;
- ❑ Device Type;
- ❑ Manufacturer Date;
- ❑ Other related information.

The **Asset Tag of an Electronic Image**, includes, but is not limited to:

- ❑ The companywide unique AKM identifier associated with the Electronic Image;
- ❑ File Type;
- ❑ File Length.

The **Asset Tag of a Subsystem**, includes, but is not limited to:

- ❑ The companywide unique AKM identifier associated with the AKM Subsystem;
- ❑ Subsystem Type;
- ❑ Number of Nodes within Subsystem (this does not include the nodes representing encapsulated subsystems, this refers to only nodes at the subsystem's level).

Hash-based Message Authentication Code (HMAC)

HMAC uses two passes of hash computation. The secret key is first used to derive two keys – inner and outer. The first pass of the algorithm produces an internal hash derived from the message and the inner key. The second pass produces the final HMAC code derived from the inner hash result and the outer key. Thus, the algorithm provides better immunity against length extension attacks.

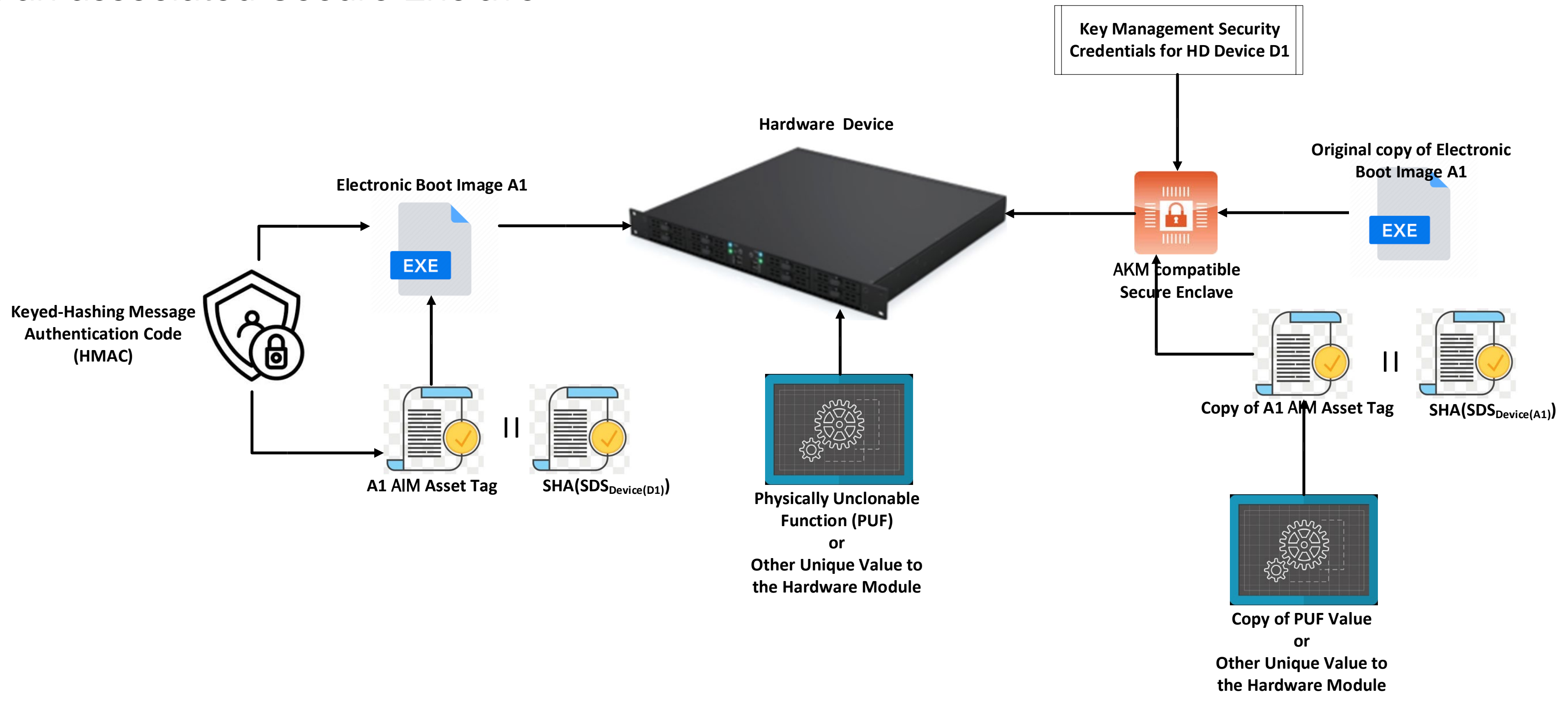
As with any Message Authentication Code, it may be used to simultaneously verify both the data integrity and the authenticity of a message.

NOTES:

1) Any cryptographic hash function, such as SHA-256 or SHA-3, may be used in the calculation of an HMAC. However, the current default algorithm for generating the HMAC code is based on the SHA256 cryptographic hash algorithm, but there are no reasons other alternative cryptographic hash algorithms could not be selected as well. Thus, which hash function is used is seen as an implementation detail.
2) SHA-256 generates an almost-unique 256-bit (32-byte) signature for a text. A hash is not 'encryption' – it cannot be decrypted back to the original text (it is a 'one-way' cryptographic function, and is a fixed size for any size of source text).

AIM Device Authentication (single image)

The AKM Secure Boot procedure is used to protect a system against unauthorized modification of hardware or software components. The protection algorithm is based on cooperation between two components – a modified version of the bootloader and an associated Secure Enclave.



Key Management Security Credentials for HD Device D1
Hardware Device
Electronic Boot Image A1
EXE
Original copy of Electronic Boot Image A1
EXE
AKM compatible Secure Enclave
Keyed-Hashing Message Authentication Code (HMAC)
A1 AIM Asset Tag
||
SHA(SDS$_{Device(D1)}$)
Physically Unclonable Function (PUF) or Other Unique Value to the Hardware Module
Copy of A1 AIM Asset Tag
||
SHA(SDS$_{Device(A1)}$)
Copy of PUF Value or Other Unique Value to the Hardware Module

AKM Provides device authentication by binding the firmware image to the physical hardware via the secure enclave as the ZKA "Verifier"

Hardware Device Integrity

Electronic images are typically a software or firmware executable or a configuration data file. Each electronic image is assigned a unique AIM identifier and is the basis for creating an AIM Trust Relationship (AIMTR) with the device within which it resides.

The hardware device, along with each specified, individual electronic image within it, will have associated with it, a set of credentials representing the AIMTR of the configuration of the hardware device and specified electronic images within the device. If these security credentials are not identical (not in sync) with the AIM Trust Relationship credentials of the other participants within the same AIMTR, then, the non-agreeing "would-be" participant is automatically excluded from the AIMTR given that different security credentials would imply different encryption seeds and keys being used between the non-agreeing "would-be" participant and the other, agreeing participants within the AIMTR.

The steps and methods used to perform the identification and authentication of a hardware device are shown below:

1) Utilize a unique value associated with the hardware device (preferably, a hardware PUF) that enables it to uniquely identify itself
2) Each hardware device will have a unique asset tag, including the companywide unique identifier of the electronic image, the unique value derived from the hardware itself and referenced in the previous step, the hardware device type, its manufacturer date, and other related information.
3) An HMAC value shall be created from the concatenated values of the hardware device's Asset Tag and the hardware device's associated release image(s) and stored within Secure Element.
4) A copy of the hardware device's Asset Tag will be stored within the SDS (which is stored within the secure enclave).

Hardware Device Integrity (continued)

5) A copy of the hardware device's AIMTR SDS will also be stored within the secure Element.
6) A copy of the HMAC calculated from the hardware device's Asset Tag and hardware device Relationship SDS, will be stored within the secure enclave. This HMAC is stored as a part of a pair, with itself plus the hardware device's security relationship SDS.
7) Using the values within the SDS, the encrypted file holding the concatenated elements of the Asset Tag (including the PUF) and the HMAC, will be unencrypted. The HMAC is then recalculated on the combination of the Asset Tag, and if it does not match the HMAC stored within the Secure Enclave for the hardware device, then, a bad device error will be triggered

Integrity validation depends heavily on the concept of security relationships. Because all nodes within an AIMTR share identical copies of the security credentials, if those security credentials are not in agreement with each other, then a AIM protected network will immediately detect the errant member of the security relationship and exclude it from further participation within the security relationship. Thus, enabling the ability to maintain, in real-time, the system integrity of both physical devices (hardware) and their associated electronic images (virtual assets of software, firmware, and data).

Electronic Image Integrity

The figure in Slide 11 includes a picture of an electronic image. For each electronic image within the hardware device, a unique AIM identifier is assigned to each electronic image. This AIM identifier is included in an Asset Tag that is specific to each individual image. Subsequently, a Keyed-Hash Message Authentication Code (HMAC), is applied to the concatenated (combined) set of the electronic image and asset tag. The default Keyed-Hash Message Authentication Code algorithm is SHA256. The concatenated set of the electronic image, asset tag, and HMAC is then encrypted using the security credentials associated for the specific electronic image.

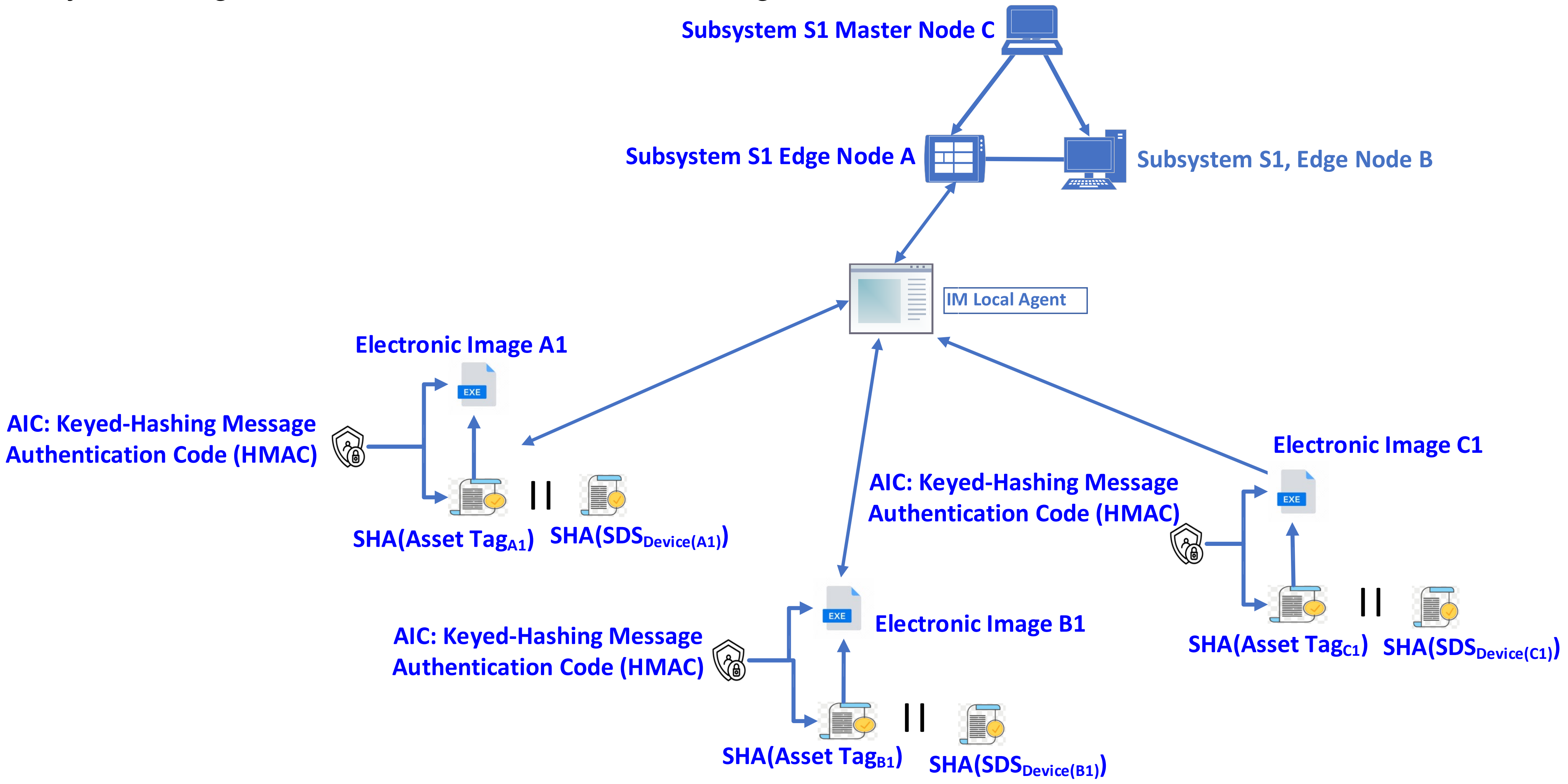
The steps and methods used to perform the identification and authentication of each electronic image are shown below:

1) Each electronic image ("EI") will have a unique asset tag, including the unique AIM identifier of the electronic image.
2) A copy of the EI's Asset Tag will be stored within the secure enclave as part of the hardware device's SDS.
3) A copy of the digital signature that is derived from calculating the HMAC of the concatenated elements consisting of the Electronic Image (EI) and the EI's Asset Tag and will be stored within the SDS as part of the section of the SDS representing the EI. This resultant digital signature is what is known as a File Integrity Code (FIC).
4) A copy of the EI's security credentials will also be stored within the secure enclave as part of the hardware device's SDS.
5) Using the values within the security credentials, the encrypted file holding the concatenated elements of the electronic image, the Asset Tag, and the FIC, will be unencrypted. The FIC is then recalculated on the concatenation of the electronic image and the asset tag, and if it does not match the FIC stored with the encrypted file and associated asset tag on the target device, a bad image error will be triggered.

This process can be repeated for every image within a particular device (as shown in the next slide, Slide 15).

Electronic Image Integrity (multiple images)

This process is repeated for every image within a particular device. The below diagram illustrates this concept for device, “Subsystem Edge Node A” and three different images, A1, B1, and C1


Subsystem S1 Master Node C
Subsystem S1 Edge Node A
Subsystem S1, Edge Node B
IM Local Agent
Electronic Image A1
AIC: Keyed-Hashing Message Authentication Code (HMAC)
SHA(Asset Tag$_{A1}$) || SHA(SDS$_{Device(A1)}$)
Electronic Image C1
AIC: Keyed-Hashing Message Authentication Code (HMAC)
SHA(Asset Tag$_{C1}$) || SHA(SDS$_{Device(C1)}$)
Electronic Image B1
AIC: Keyed-Hashing Message Authentication Code (HMAC)
SHA(Asset Tag$_{B1}$) || SHA(SDS$_{Device(B1)}$)