AKM Asset Tags

AKM Asset Tags

Device Asset Tag Data Structure

The Asset Tag data structure for a physical device is shown here. Relevant fields include, but are not limited to:

$AssetTag_{Device}$
AKM Physical Device Provisioning Relationship Identifier: TBD
AKM Device Identifier: TBD
Device Manufacturer Globally Unique Identifier (GUID): TBD
Device Type
Manufacturer Date
Manufacturer

- Asset Tag Type – This is an enumerated value, that is the first field in every asset tag.
- Hardware Device AKM Identifier – This is the AKM Identifier for the physical device. The AKM identifier is always the second field in every asset tag. For now, the default length of a physical device AKM identifier is 128-bits. Coincidentally, this is also the address associated with the device for AKM communication relationships.
- Device Manufacturer – This is an enumerated type of different manufacturers of the chipset (ex., NXP, Intel, STM, TI, etc.).
- Device Manufacturer Globally Unique Identifier (GUID) – This is a unique value that can be read internally within the CPU processor chipset, that provides a unique identifier for the specific .
- Device Type – This is an enumerated type that represents the type of device, where the device type uses locally defined delineations (ex., within an automotive environment, it would be the type of ECU).
- Manufacturer Release Date – This details the precise date of the release of the hardware revision, not necessarily the date the hardware was released.

Electronic Image Asset Tag Data Structure (1st Stage Bootloader with BSP)

The Asset Tag data structure for an electronic image is shown here. Relevant fields include, but are not limited to:

Asset Tag Type: AssetTag$_{ElectronicImage}$
AKM Electronic Image Identifier: TBD
Electronic Image Type: 1st Stage Bootloader with BSP
Electronic Image Length (in bytes): TBD
Electronic Image Revision Number: TBD
Electronic Image Release Date: TBD

- Asset Tag Type – This is an enumerated value, that is the first field in every asset tag.
- Electronic Image AKM Identifier – This is the AKM Identifier for the physical device. The AKM identifier is always the second field in every asset tag. For now, the default length of an electronic image AKM identifier is, 128-bits
- Electronic Image Type – 1st Stage Bootloader.
- Electronic Image Length – This is a four byte field representing the length of the electronic image in bytes.
- Electronic Image Revision Number – This is of an implementation dependent format and value, with the default size being 64-bits.
- Electronic Image Release Date and Time – This is a 12-byte field, expressed in a combined UTC-based date and time format.

EI Asset Tag Data Structure (2nd Stage Bootloader & Application)

Asset Tag Type: AssetTag$_{ElectronicImage}$
AKM Electronic Image Identifier: TBD
Electronic Image Type: 2nd Stage Bootloader
Electronic Image Length (in bytes): TBD
Electronic Image Revision Number: TBD
Electronic Image Release Date: TBD

Asset Tag Type: AssetTag$_{ElectronicImage}$
AKM Electronic Image Identifier: TBD
Electronic Image Type: Application Release Image
Electronic Image Length (in bytes): TBD
Electronic Image Revision Number: TBD
Electronic Image Release Date: TBD

This is the same data structure as defined for the 1st Stage Bootloader with BSP.

Asset Tags and Asset Tag Types

An asset tag is a digital twin of a physical or virtual entity.

Types of Asset Tags are:

- **Electronic Image** – represents a data-at-rest image
- **Container** – represents a data-in-use software object
- **Physical Device** – Static physical entity
- **Subsystem of Devices and/or other Subsystems** – Virtual Object representing a specific group of devices.
- **System of Devices and/or Subsystems** – Virtual Object representing a group of groups of devices.

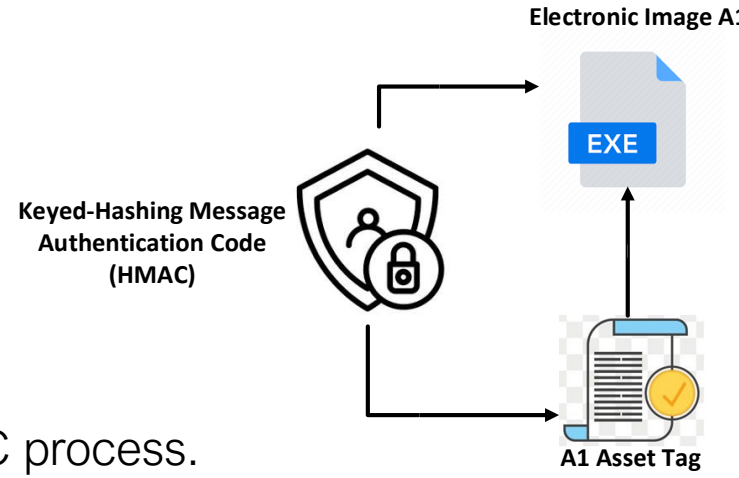
An asset tag is the foundation for an asset based AKM Security Relationship (ASR). Asset based ASRs are used to establish, maintain, and monitor, integrity and authenticity of system components. The Asset Tag, together with the asset(s) it represents are used to calculate and derive the digital signature representing the Asset.

File Integrity Code (FIC) Description

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:

- $FIC_{ImgX} = HMAC\ (SHA(ImgX) \mid\mid SHA(AssetTag_{ImgX}), HMAC\text{-}SECRET_{FIC})$



Where:

- ImgX, represents the Image labeled, component X.
- $AssetTag_{ImgX}$, represents the Asset Tag of Image X.
- $HMAC\text{-}SECRET_{FIC}$ 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).

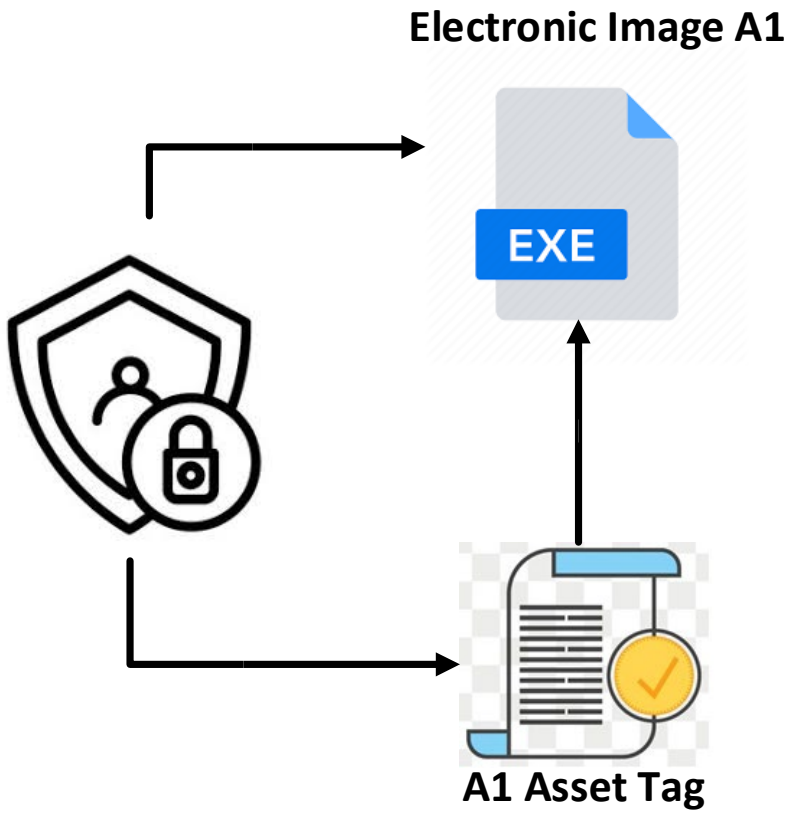
File Integrity Code (FIC) Illustration

So, keeping in mind that:

- FIC_{ImgX} = HMAC (SHA(ImgX) || SHA($AssetTag_{ImgX}$), HMAC-$SECRET_{FIC}$)

Implies:

File Integrity Code (FIC_{A1}) = **Keyed-Hashing Message Authentication Code (HMAC)**



Endpoint Integrity Code (EPIC) Description

An Endpoint Integrity Code (EPIC) is derived by taking the digital signature of the coupling of an HMAC of the Endpoint Device's Asset Tag (which includes, amongst other things, the HW GUID from the chipset manufacturer) and the Device SDS, plus 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:

- $EPIC = HMAC\ (SHA(AssetTag_{EP1} \,||\, FIC_{IMG1} \,||\, FIC_{IMG2} \,||\, FIC_{IMG3} \ldots, HMAC\text{-}SECRET_{EP1})$

Where:

- $AssetTag_{EP1}$, represents the Asset Tag of the Device (includes the Hardware Identifier).
- FIC_{ImgX}, represents the File Integrity Code of Image X.
- $HMAC\text{-}SECRET_{EP1}$ 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).

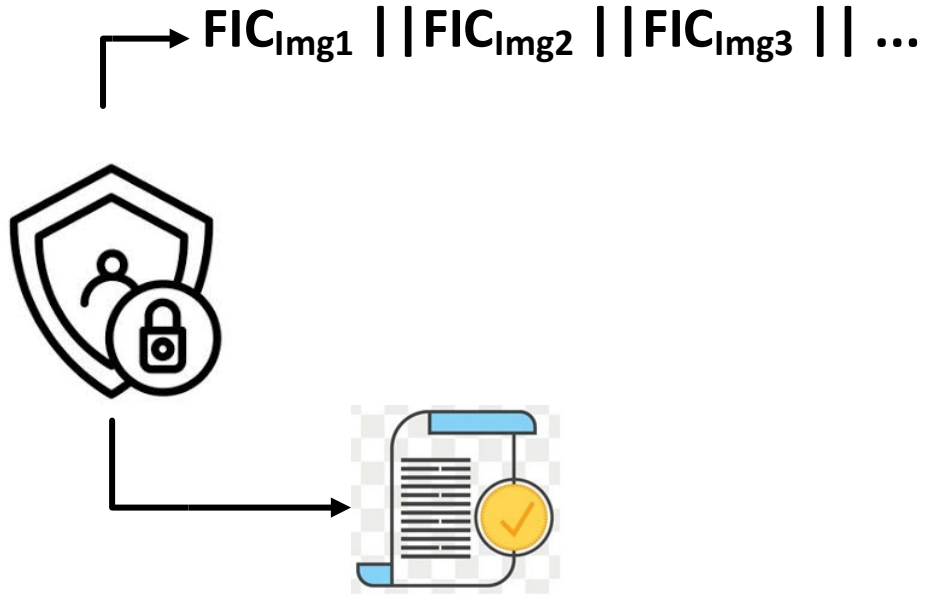
Device Integrity Code (DIC) Illustration

An Endpoint Integrity Code (EPIC) is derived by taking the digital signature of the coupling of an HMAC of the Device's Asset Tag (which includes, amongst other things, the HW GUID from the chipset manufacturer) 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 a DIC:

- $EPIC = HMAC\ (SHA(AssetTag_{EP1} \,||\, FIC_{IMG1} \,||\, FIC_{IMG2} \,||\, FIC_{IMG3} \ldots, HMAC\text{-}SECRET_{EP1})$

EndPoint Integrity Code ($EPIC_{EP1}$) =

Keyed-Hashing Message Authentication Code (HMAC)



AKM Security Relationship (ASR)

The Provisioning AKM Trust Relationship for a physical device is used for two things:

1) It is used to establish an AKM connection with the AKM infrastructure. Meaning, this is the relationship that connects the physical device to the root-of-trust network.
2) It is used to manage the aggregation of components comprising a physical device asset, including keys, seeds, and vectors used for both the encryption and digital signature algorithms ensuring the security and authentication of the physical device's digital twin and the and virtual assets (i.e., electronic images) belonging to the physical device.

The manifestation of the Physical Device's Provisioning ATR is its Synchronized Data Set (SDS).

The elements within the SDS of the provisioning ATR for the physical device include, but are not limited to the following:

- AKM Trust Relationship (ATR) Identifier
- Current Session Security Credentials
- Next Session Security Credentials
- Fallback Session Security Credentials
- Failsafe Session Security Credentials
- Arbiter Mode Session Credentials (if applicable)
- Ordered List of AKM Component (i.e., electronic image) Identifiers for Physical Device and Container Nodes.

AKM Synchronized Data Set (SDS) Parameters Data Structure

```
typedef struct _PROGRAM_DATA_VECTOR {
      U8  PDV_elements [128];
} PROGRAM_DATA_VECTOR, *PPROGRAM_DATA_VECTOR;

typedef struct _SESSION_CREDENTIALS { // SizeOf (52-bytes)
    AES_128_BIT_ENCRYPTION_KEY      Session_Encryption_Key;    // Offset:  0 - 15 ( 16-bytes)
    U32                             Session_Seed_Value;        // Offset: 16 - 19 (  4-bytes)
    HMAC_KEY                        Session_HMAC_Key;          // Offset: 20 - 51 ( 32-bytes)
} AKM_SESSION_CREDENTIALS, *PAKM_SESSION_CREDENTIALS;

typedef struct _SDS_PARAMETERS { // SizeOf (404-bytes)
    AKM_TRUST_RELATIONSHIP_IDENTIFIER       ATR_Relationship_ID;       // Offset:   0 -  15 ( 16-bytes)

    AKM_SESSION_CREDENTIALS                 Current_Session;           // Offset:  16 -  67 ( 52-bytes)
    AKM_SESSION_CREDENTIALS                 Next_Session;              // Offset:  68 - 119 ( 52-bytes)
    AKM_SESSION_CREDENTIALS                 Fallback_Session;          // Offset: 120 - 171 ( 52-bytes)
    AKM_SESSION_CREDENTIALS                 Failsafe_Session;          // Offset: 172 - 223 ( 52-bytes)

    // Optional Arbiter Session Security Credentials -- used for dynamic provisioning
    AKM_SESSION_CREDENTIALS                 Arbiter_Session;           // Offset: 224 - 275 ( 52-bytes)

    // Parameter Data Vector (PDV)
    PROGRAM_DATA_VECTOR                     Parameter_Data_Vector;     // Offset: 276 - 403 (128-bytes) */
} SDS_PARAMETERS, *PSDS_PARAMETERS;
```


AKM Component Object Data Structure

```
  /* Enumeration Types */
typedef enum _ATR_OBJECT_DELINEATOR_ENUM {
    Ultimate_AKM_Root_Of_Trust_Backend_Server_object,          // Backend Server
    Intermediate_AKM_Chain_of_Trust_Proxy_Server_object,       // Intermediate Proxy Server, and always points to the Intermediate Proxy Server
                                                               // that is closest to the affected node within the AKM Root-of-Trust hierarchy
    Local_AKM_Chain_of_Trust_management_module_object,         // Local AKM Management Module -- this includes both permanent and temporary local
                                                               // AKM Management Modules.  An AKM node's security relationship can only have one
                                                               // local AKM Management Module at a time.  Typically, temporary local AKM Management
                                                               // Modules are used in "Air  Gapped" AKM security relationships. */
    Portable_AKM_Chain_of_Trust_provisioning_device_object,    // Used by maintenance personnel to update and/or repair existing AKM networks.
    AKM_Communication_Edge_Node_object,                        // The is the nominal AKM node and represents a logically distinct AKM Module,
                                                               // usually a physical device, but could also be a logical partition of a physical
                                                               // device.
    AKM_Electronic_Image_object,                               // Electronic Image

    AKM_Security_Relationship_object                           // An AKM Security Relationship object represents an AKM Security Relationship
}  ATR_OBJECT_DELINEATOR_ENUM;

typedef struct _BALANCED_BINARY_TREE { // SizeOf: 16-bytes
    // Fields needed for keeping a balanced binary tree.
    S32  balance_factor;  //  -1, 0, or +1      // Offset:  0 -  3 ( 4-bytes)

    // NOTE: the user of this structure should be very careful in setting and/or accessing the owning data structure via these fields
    // A NULL_PTR value for the "parent_node_ptr", indicates this is the ROOT node.
    PVOID  parent_node_ptr;                 // Offset:  4 -  7 (4-bytes)
    PVOID  left_subtree_ptr;                // Offset:  8 - 11 (4-bytes)
    PVOID  right_subtree_ptr;                   // Offset: 12 - 15 (4-bytes)
} BALANCED_BINARY_TREE, *PBALANCED_BINARY_TREE;
```


AKM Info Object Balanced Binary Tree Data Structure

```
typedef struct _AKM_INFO_OBJECT { // SizeOf: 36-bytes
    AKM_OBJECT_STANDARD_128_BIT_ADDRESS  AKM_Node_Address;       /* Offset:  0 - 15 (16-bytes) */

    ASR_OBJECT_DELINEATOR_ENUM                    AKM_Object_Delineator;  /* Offset: 16 - 19 ( 4-bytes) */

    // Balanced Binary Tree
    // Use Methods defined to manipulate this BT, to ensure only AKM Info
    // Objects are manipulated for nodes connected via this field.
    BALANCED_BINARY_TREE                          Balanced_Binary_Tree;   /* Offset: 20 - 35 (16-bytes) */
} AKM_INFO_OBJECT, *PAKM_INFO_OBJECT

  typedef struct _AIM_ELECTRONIC_IMAGE_OBJECT { // SizeOf: 64-bytes
      AKM_OBJECT_STANDARD_128_BIT_ADDRESS     AKM_Device_Node_Address;                    // Offset:   0 -  15 (16-bytes)

      // The below field is the calculated result of this expression: FIC_ImgX = HMAC (SHA(ImgX) || SHA(AssetTag_ImgX), HMAC-SECRET_FIC)
      SHA_256_HMAC                                AIM_Electronic_Image_Digital_Signature;     // Offset:  16 -  47 (32-bytes)

      // Balanced Binary Tree
      // Use Methods defined to manipulate this BT, to ensure only AKM
      // Electronic Image Info Objects are manipulated for nodes connected via this field.
     BALANCED_BINARY_TREE                          Balanced_Binary_Tree;                   // Offset:  48 -  63 (16-bytes)
  } AIM_ELECTRONIC_IMAGE_INFO_OBJECT, *PAIM_ELECTRONIC_IMAGE_INFO_OBJECT;
```


AKM Synchronized Data Set (SDS) Data Structure

```
// This structure represents a single Synchronized Data Set (SDS) for physical device AKM provisioning modules
// and is always organized from the perspective of the chain of trust.  The AKM Object representing the
// AKM Provisioner is always higher on the chain of trust than the object representing the AKM Provisionee.
typedef struct _ATR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET { // SizeOf (472-bytes)
    SDS_PARAMETERS                                  SDS_parameters;                 // Offset:   0 - 403 (404-bytes)

    // The below field is the calculated result of this expression: DIC = HMAC (SHA(AssetTag_Device) || SHA(SDS_Device) || FIC_IMG1 || FIC_IMG2 || FIC_IMG3 ..., HMAC-SECRET_DIC)
    SHA_256_HMAC                                    AIM_Physical_Device_Digital_Signature; // Offset: 404 - 435 (32-bytes)
    // As of this writing, this should be one of the following values:
    //
    //    Ultimate_AKM_Root_Of_Trust_Backend_Server_object,
    //    Intermediate_AKM_Chain_of_Trust_Proxy_Server_object,
    //    Local_AKM_Chain_of_Trust_management0_module_object,
    //    Portable_AKM_Chain_of_Trust_provisioning_device_object,
    ASR_OBJECT_DELINEATOR_ENUM                      AKM_Provisioner_Object_Delineator;      // Offset: 436 - 439 ( 4-bytes)
    AKM_OBJECT_STANDARD_128_BIT_ADDRESS             Provisioning_Module_AKM_Address;        // Offset: 440 - 455 (16-bytes)

    // As of this writing, this can be one of the following values:
    //
    //    Intermediate_AKM_Chain_of_Trust_Proxy_Server_object,
    //    Local_AKM_Chain_of_Trust_management_module_object,
    //    Portable_AKM_Chain_of_Trust_provisioning_device_object,
    //    AKM_Communication_Edge_Node_object
    ASR_OBJECT_DELINEATOR_ENUM                      AKM_Provisionee_Object_Delineator;      // Offset: 456 - 459 ( 4-bytes)
    AKM_OBJECT_STANDARD_128_BIT_ADDRESS             Subservient_Object_AKM_Address;         // Offset: 460 - 463 (16-bytes)

    U32                                         NumberOfVirtualComponents;          // Offset: 464 - 467 (  4-bytes)
    PAKM_INFO_OBJECT                                ComponentBinaryTree;                // Offset: 468 - 471 (  4-bytes)
} ATR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET, *PASR_PHYSICAL_DEVICE_PROVISIONING_SYNCHRONIZED_DATA_SET;
```


Example Provisioning of Physical Device Components

See Slide 14 for the field in **BOLD BLUE** representing the FIC