Manifest

draft-moran-suit-manifest-01

draft-moran-suit-architecture-01

Design Decision

SECURITY ARCHITECTURE

Firmware update over TLS

- Developers put firmware image on update server.
- Devices fetch firmware from that update server.
- Each device trusts the update server.
- The update server manages access control.
  - The developer logs in to the update server and uploads a firmware.
  - The update server decides whether or not to accept the uploaded firmware, based on the developer's permissions
- Devices only need to trust one set of credentials.

- A lot of trust is placed into the update server.

Firmware update with code signing

- An author can sign the firmware image before it is distributed.
  - The devices trust the developer directly.
  - The device verifies the signature of the firmware image before installing it.
  - The risks posed by a firmware repository are reduced.
  - The author can perform signing on a dedicated devices, which further reduces risk.

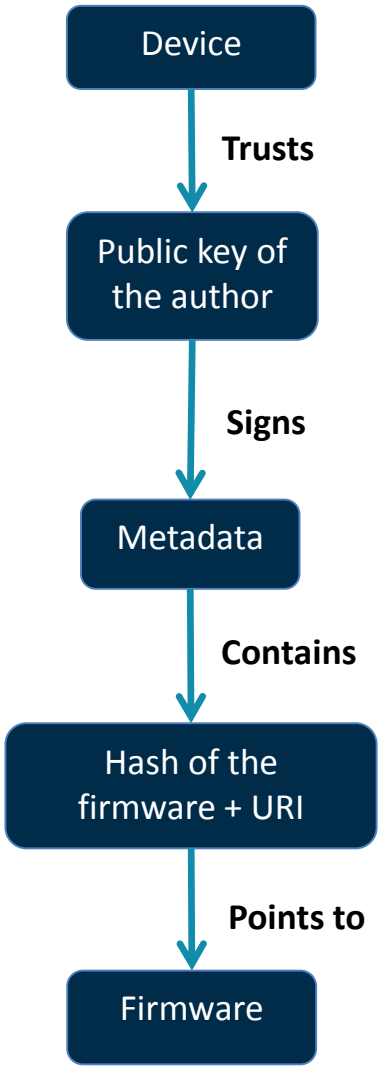
- Devices are now responsible for access control.
- Authors are now responsible for security.
- Devices must perform public key operations for each update.

Firmware update: transport security or code signing?

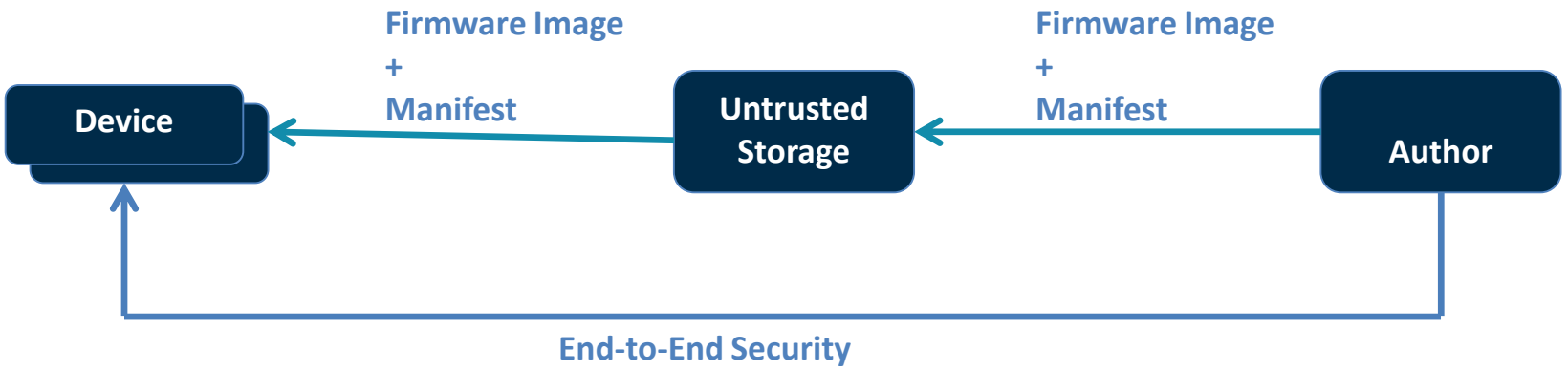
- Code signing has significant benefits for security.
  - Widely accepted practice in software, and device driver distribution.
  - Signed metadata takes this one step further, offering early validation.
  - Devices need to manage access control.
- Transport security offloads the burden of access control.
  - Devices aren't required to handle access rights of individual firmware authors.
  - They place the burden of maintaining security on the server.

Envisioned Relationships

- Prerequisite: Public key of the firmware author is stored on the device.
- Metadata is signed
- Metadata contains digest of firmware



Envisioned Architecture



Design Decision

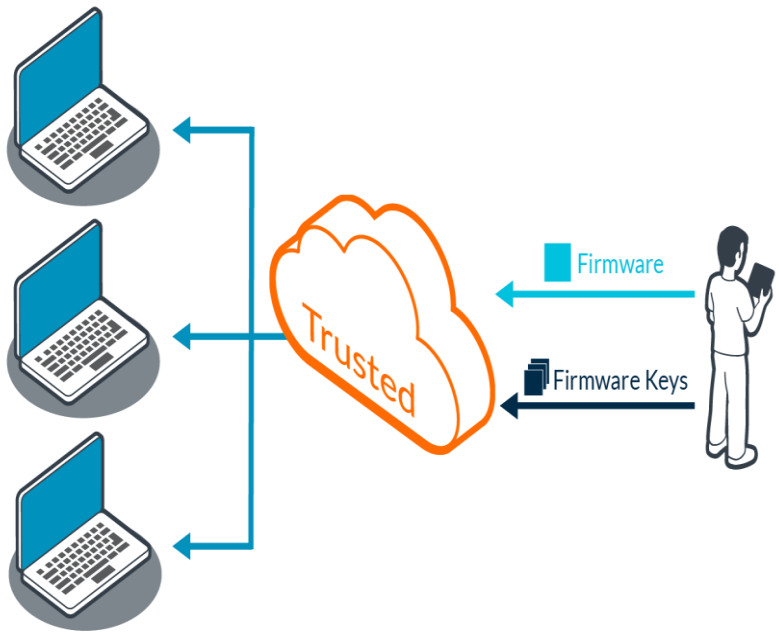
ENCRYPTION

Firmware update with per-device encryption

- The firmware author encrypts unique copy of the firmware for every recipient device.
  - The firmware author builds a new firmware image
  - They encrypt one copy of it for every device
  - They upload all of these copies to a distribution service
  - Each device downloads its own firmware image and decrypts it

Firmware update with single image encryption

- A single, encrypted firmware image is distributed.
  - Each device also receives a copy of the image decryption key, encrypted using its unique encryption key.
  - The device decrypts this with its unique encryption key.
  - The device uses the image decryption key to decrypt the image.
- Optional feature; not needed in all deployments



Design Decision

TARGETING UPDATE

Targeting Update

- The operator can select a group of devices.
  - They can select devices by a variety of parameters, such as: Vendor & Model, Current firmware version, …
- Instruct the system to update some or all devices automatically when the vendor publishes new firmware
- The operator can select a phased roll-out to minimize risk.

- Manifest includes various attributes that allow update to be tailored to specific devices/device categories.

Design Decision

MANIFEST ENCODING

Manifest Encoding

- Initially specified in ASN.1/DER. Used CMS-based security wrapper.
  - Not well received based on mailing list feedback.
- Changed to CBOR/COSE. Described in CDDL.
- Is everyone happy now?

Design Decision

MANIFEST ATTRIBUTES

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Version number of the manifest

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Optional, textual description of the Update.

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Random value to ensure that a given manifest is unique.

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Indicates when the manifest was created.

Used for rollback protection.

Manifest CDDL

1. Vendor ID
2. Class ID
3. Device ID
4. Best Before

```
= [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    oadInfo: ? PayloadInfo
```

Used to construct IF … THEN … Rules

1. Apply Immediately
2. Apply After

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

Used to refer to alternative locations of the firmware image

Manifest CDDL

```
Manifest = [
    manifestVersion : uint,
    text : {* int => tstr } / nil,
    nonce : bstr,
    timestamp : uint,
    conditions: [ * condition ],
    directives: [ * directive ] / nil,
    aliases: [ * ResourceReference ] / nil,
    dependencies: [ * ResourceReference ] / nil,
    extensions: { * int => bstr } / nil,
    payloadInfo: ? PayloadInfo
]
```

To express the requirement that more than one image has to be installed on a device

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Format of the binary

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Size of the firmware image in bytes

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Indicates where the image should be placed on the device

Useful when device contains multiple MCUs and requires multiple firmware images.

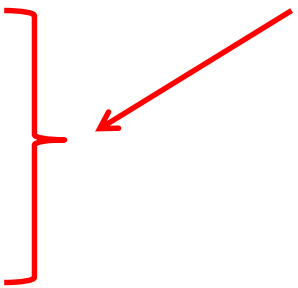
Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

A set of ranked references for where to find the payload.

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr, s
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```



Fingerprint computed over the firmware image using the indicated algorithm.

Payload CDDL

```
PayloadInfo = [
    format = [
        type: int,
        ? parameters : bstr
    ],
    size: uint,
    storageIdentifier: bstr,
    uris: [*[
        rank: int,
        uri: tstr
    ]] / nil,
    digestAlgorithm = [
        type : int,
        ? parameters: bstr
    ] / nil,
    digests = {* int => bstr} / nil,
    payload = COSE_Encrypt / bstr / nil
]
```

Attached firmware image