FUD
Internet-Draft
Intended status: Informational
Expires: January 19, 2018

B. Moran
M. Meriac
H. Tschofenig
ARM Limited

July 18, 2017

Firmware Manifest Format
draft-moran-fud-manifest-00

Abstract

This specification describes the format of a manifest. A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 19, 2018.

Copyright Notice

Table of Contents

1. Introduction

A firmware update mechanism is an essential security feature for IoT devices to deal with vulnerabilities. While the transport of firmware images to the devices themselves is important there are already various techniques available, such as the Lightweight Machine-to-Machine (LwM2M) protocol offering device management of IoT

devices. Equally important is the inclusion of meta-data about the conveyed firmware image (in the form of a manifest) and the use of end-to-end security protection to detect modifications and (optionally) to make reverse engineering more difficult. End-to-end security allows the author, who builds the firmware image, to be sure

that no other party (including potential adversaries) to install firmware updates on IoT devices with adequate privileges. This authorization process is ensured by the use of dedicated asymmetric keys installed on the IoT device: for use cases where only integrity protection is required it is sufficient to install a trust anchor on the IoT device. For confidentiality protected firmware images it is additionally required to install either one or multiple symmetric or asymmetric keys on the IoT device. Starting security protection by the author is a risk mitigation technique so firmware images and manifests can be stored on untrusted respositories.

It is assumed that the reader is familiar with the high-level firmware update architecture [Architecture]. This document is structured as follows: In Section 3 we describe the main building blocks of the manifest and Section 4 contains the description of the ASN.1 of the manifest.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

To describe the components of the manifest we use the terms structures and attributes. The manifest has a hierarchical structure and top level components are called structures and the attributes are the components within them.

3. Components

The key components of a manifest are shown in Figure 1 and are explained in the sub-sections below.

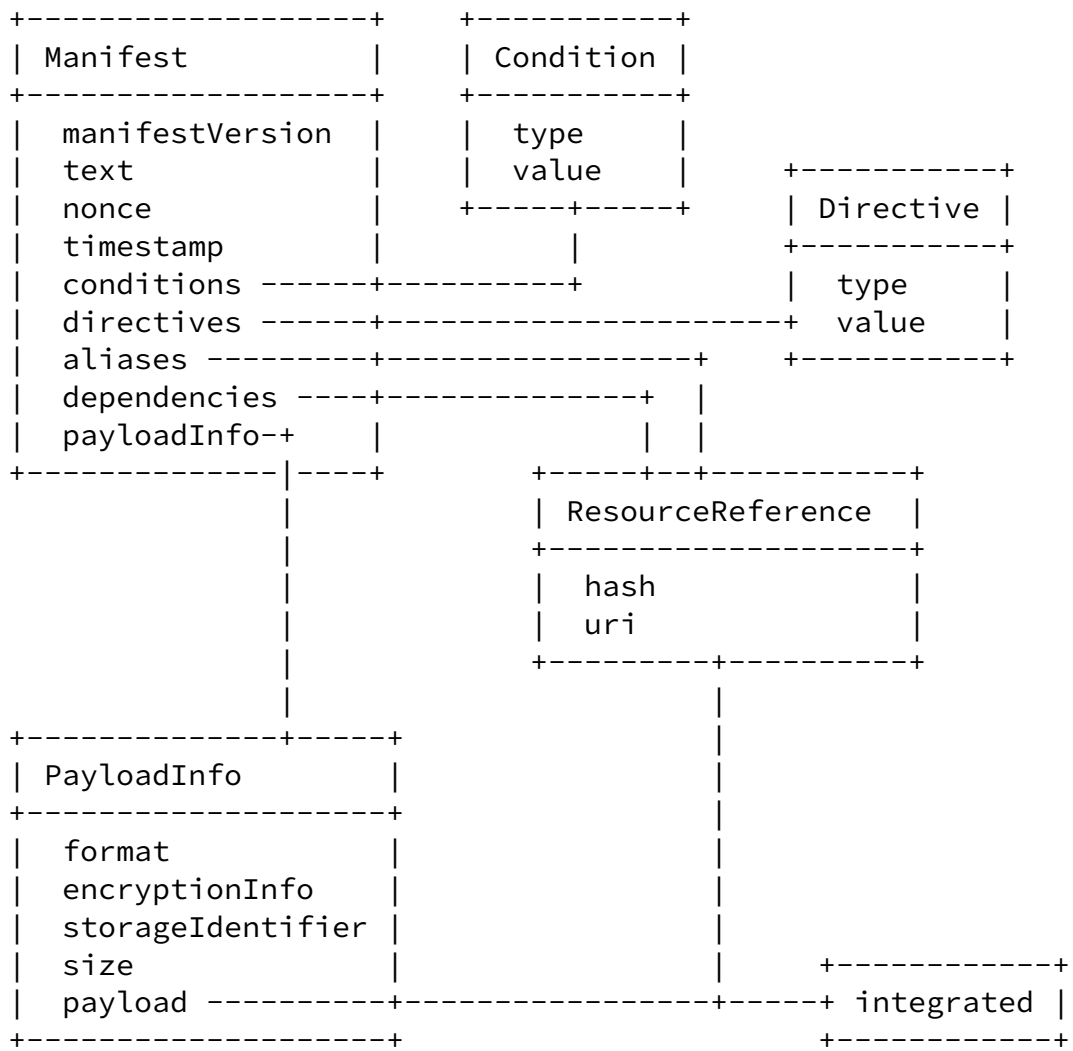
Manifest
manifestVersion
text
nonce
timestamp
conditions
directives
aliases
dependencies
payloadInfo
Condition
type
value
Directive
type
value
ResourceReference
hash
uri
PayloadInfo
format
encryptionInfo
storageIdentifier
size
payload
integrated

Figure 1: Components of a Manifest.

3.1. Manifest

The Manifest structure is the top-level construct that ties all other structures together. In addition to the structures explained in subsections below it contains:

- a version number (in the 'manifestVersion' attribute)
- a textual description about the update, including the version / vendor / model of the device (in the 'text' attribute). This information is optional.
- a timestamp indicating when the manifest was created (in the 'timestamp' attribute).

3.2. PayloadInfo

The PayloadInfo structure contains information about the firmware image. The 'format' attribute contains the firmware image type (such as rawBinary, hexLocationLengthData, ELF). The 'size' attribute offers information about the size of the firmware image in bytes. If the size of the obtained firmware image differs from the size stated in the manifest then the obtained image MUST be consider corrupted. The 'nonce' attribute contains a (short) random value to ensure that a given manifest is unique. This separates the function of the timestamp, which is provided for rollback protection, from the function of the nonce, which is for uniqueness. Keeping these functions separate ensures that a number of edge cases are catered for, for example: the creation of manifests quickly enough that they have the same timestamp. The 'storageIdentifier' attribute indicates where the image should be placed on the device. This value useful, for example, when an IoT device contains multiple microcontrollers (MCUs) and the decision needs to be made to which MCU to send which firmware image.

Most importantly, however, the PayloadInfo structure contains a reference to the firmware image (in the 'reference' attribute) or the

image is embedded inside the PayloadInfo structure (within the 'integrated' attribute). A referenced image first needs to be fetched by the device before the update can be applied. The 'reference' attribute contains a 'hash' and a 'uri' attribute: the value in the 'hash' attribute allows the device to determine whether it has already obtained this firmware image and, since it is included in the digitally signed manifest, it protects the firmware image against modifications. The 'uri' attribute references the image.

Finally, a firmware image may be encrypted and information about how to decrypt is provided in this payload in the 'encryptionInfo' attribute. The following options are provided:

- No encryption (mode="none"). In this case the firmware image is not encrypted and only integrity protected.

- Encryption using a symmetric key (mode="preSharedKey"). The assumption is that the symmetric key is pre-provisioned (in an out-of-band fashion) on the IoT device and also available to the developer.

- Encryption using a symmetric key derived via a key derivation function (mode="preSharedKeyKdf"). This option is a variation of the symmetric key encryption mode whereby a key derivation function is applied to the pre-provisioned key before it is used for encrypting the firmware image.

- Encryption using a symmetric key found in the 'KeyTable' attribute (mode="keyTable"). This mode is tailored to use cases where a single encrypted firmware image is transmitted to many IoT devices.

Depending on the selected mode different information has to be conveyed in the manifest.

- When encryption using a symmetric key is selected then the 'KeyId' attributes provides information for identifying the appropriate symmetric key.

- When encryption using a symmetric key derived via a key derivation function is selected then the following three parameters are provided by the 'KdfParameters' attribute: KDF algorithm, nonce,

and a key id. The computed function KDF(key, nonce).

- When encryption using the key table is selected then the 'KeyTable' attribute is used. Figure 2 shows the concept graphically where the firmware image is encrypted by a symmetric key and this symmetric key is encrypted with the public key of each of the devices.

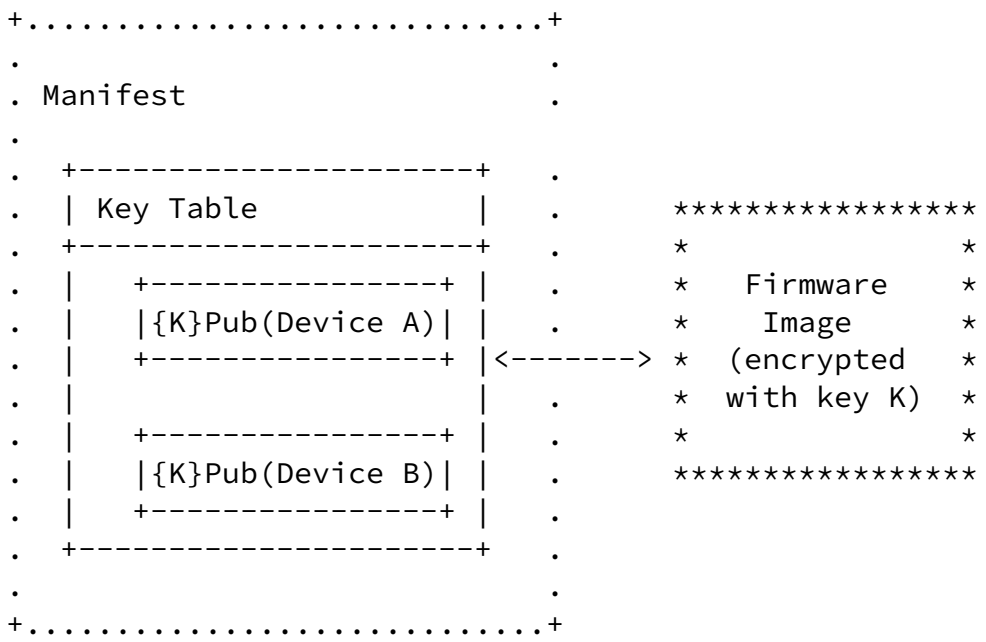

Figure 2: Key Table.

3.3. Condition and Directive

The Condition and the Directive structures together allow "If <...> Then <...>" rules to be expressed.

It offers the following functionality:

- Apply an update before a given date only (Directive.applyAfter)

- Apply an update immediately (Directive.applyImmediately)

- Apply an update only to devices that match the vendorId, classId, deviceId attributes

- Apply an update only if the device system time is before the time indicated in the Condition.lastApplicationTime.

3.4. Aliases and Dependencies

In some situations an IoT device may require more than a single firmware update image. To express the requirement that more than a single image has to be installed on a device the dependencies structure is used, which is of type ResourceReference (as used by the PayloadInfo structure).

Aliases are used to refer to alternative locations of firmware images. This is useful in environments where organizations cache firmware images (and their corresponding manifests) on premise to avoid the need to fetch imagines from repositories maintained by the developer's organizations (such a device manufacturer or an OEM).

3.5. Device Identification

A device is identified by at least three identifiers:

- A vendor identifier
- A device class identifier
- A device identifier

3.5.1. Vendor ID

The vendor ID is a 128-bit number that conforms to RFC-4122, type 5. This number is used by the device to verify manifests.

The Vendor ID should be derived from the manufacturer's domain name using the algorithm defined in Section 4.3 of RFC-4122.

A vendor ID is typically compiled into a firmware image since it is static for the lifetime of the firmware.

3.5.2. Device class ID

The device class is a 128-bit number that conforms to RFC-4122, type 5. This number is used by the client to verify manifests. The Device Class ID SHOULD use the Vendor ID as the namespace, but the ID within the namespace can be arbitrary.

A class ID is also typically compiled into a firmware image since it is static for the lifetime of the firmware.

3.5.3. Device ID

The device ID is also a 128-bit number that conforms to RFC-4122. The device ID can come from a variety of sources. For example, a device may obtain this identifier during the manufacturing phase (together with other configuration information and manufacturer-provided credentials). In this case, we recommend using RFC-4122, type 1, where the node ID is the factory tool ID, which provides traceability of a device back to the origin of manufacture. A device ID can also come from on-device resources, such as device unique-ID registers or device identifiers in CPUs. Our recommendation is to provide unique CPU resources to a generator function similar to the one used for the class_id. In this example, the device_info may be a combination of several components, such as:

- MAC address

- Device unique identifier

Where multiple sources of unique identity are available, they should all be provided to the UUID function, since it combines them to create a single, unique identifier.

4. Manifest ASN.1 Format

```
-- Manifest definition file in ASN.1 (v. 1.0.0-alpha)
ManifestSchema DEFINITIONS IMPLICIT TAGS ::= BEGIN

Uri ::= UTF8String
Bytes ::= OCTET STRING
UUID ::= OCTET STRING
Payload ::= OCTET STRING

AlgorithmIdentifier ::= SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }

KeyId ::= OCTET STRING
```

```
   KdfParameters ::= SEQUENCE {
       kdfAlgorithm AlgorithmIdentifier,
       kdfNonce     OCTET STRING,
       keyId        KeyId
   }

   WrappedKey ::= SEQUENCE {
              deviceSubjectKeyIdentifier OCTET STRING,
              key                        OCTET STRING
   }

   KeyTable ::= SEQUENCE {
       keyWrapAlgorithm     AlgorithmIdentifier,
       keySize              INTEGER,
       payloadKeyDigest     OCTET STRING,
       subjectKeyIdentifier OCTET STRING,
       table CHOICE {
          uri               UTF8String,
          integrated        SEQUENCE OF WrappedKey
       }
   }

   EncryptionInfo ::= SEQUENCE {
       mode ENUMERATED {
           none(0), preSharedKey(1), preSharedKeyKdf(2), keyTable(3)
       },
       config ANY DEFINED BY mode,
       encryptedPayloadHash OCTET STRING
   }

   PayloadInfo ::= SEQUENCE {
       format      CHOICE {
           enum    ENUMERATED {
               rawBinary(1), hexLocationLengthData(2), elf(3), bsdiff(4)
           },
           objectId    OBJECT IDENTIFIER
       },
       encryptionInfo EncryptionInfo OPTIONAL,
       storageIdentifier OCTET STRING,
       size INTEGER,
       payload CHOICE {
           reference    ResourceReference,
           integrated   OCTET STRING
       }
   }
```

```
   ResourceReference ::= SEQUENCE {
       hash        OCTET STRING,
```



```
       uri         Uri
   }

   ConditionValue ::= CHOICE {
       int INTEGER,
       raw OCTET STRING
   }
   Condition ::= SEQUENCE {
       type ENUMERATED {
           vendorId(1),
           classId(2),
           deviceId(3),
           lastApplicationTime(4),

           vendorSpecificMinimum(2147483648)
       },
       value ConditionValue
   }
   DirectiveRule ::= CHOICE {
       int INTEGER,
       bool BOOLEAN,
       raw OCTET STRING
   }
   Directive ::= SEQUENCE {
       type ENUMERATED {
           applyImmediately(1),
           applyAfter(2),
           restartComponent(3),
           restartSystem(4),
           installationHandler(5),

           vendorSpecificMinimum(2147483648)
       },
       rule DirectiveRule
   }

   TextField ::= SEQUENCE {
       type ENUMERATED {
           description(0), version(1), vendor(2), model(3)
```

```
      },
      value UTF8String
  }

  Manifest ::= SEQUENCE {
      manifestVersion ENUMERATED {
          v1(1)
      },
      text            SEQUENCE OF TextField OPTIONAL,
```



```
      nonce           OCTET STRING,
      digestAlgorithm AlgorithmIdentifier,
      timestamp       INTEGER,
      conditions      SEQUENCE OF Condition,
      directives      SEQUENCE OF Directive,
      aliases         SEQUENCE OF ResourceReference,
      dependencies    SEQUENCE OF ResourceReference,
      payloadInfo     PayloadInfo OPTIONAL
  }

  END
```

Below is the manifest format in the ASN.1 2015 format.

```
-- Manifest definition file in ASN.1:2015 (v. 1.0.0-alpha)
ManifestSchema DEFINITIONS IMPLICIT TAGS ::= BEGIN

Uri ::= UTF8String
Bytes ::= OCTET STRING
UUID ::= OCTET STRING
Payload ::= OCTET STRING

KeyId ::= OCTET STRING

KdfParameters ::= SEQUENCE {
    kdfAlgorithm AlgorithmIdentifier,
    kdfNonce     OCTET STRING,
    keyId        KeyId
}

WrappedKey ::= SEQUENCE {
           deviceSubjectKeyIdentifier OCTET STRING,
```

```
           key                         OCTET STRING
}

KeyTable ::= SEQUENCE {
    keyWrapAlgorithm     AlgorithmIdentifier,
    keySize              INTEGER,
    payloadKeyDigest     OCTET STRING,
    subjectKeyIdentifier OCTET STRING,
    table CHOICE {
       uri               UTF8String,
       integrated        SEQUENCE OF WrappedKey
    }
}

PAYLOADENCRYPTION ::= CLASS {
    &mode ENUMERATED {
```



```
        none(0), pre-shared-key(1), pre-shared-key-kdf(2), key-table(3)
    } UNIQUE,
    --OpenType-- &Config
} WITH SYNTAX { MODE &mode, CONFIG &Config}

EncryptionOptions PAYLOADENCRYPTION ::= {
    {MODE none,               CONFIG NULL} |
    {MODE pre-shared-key,     CONFIG KeyId} |
    {MODE pre-shared-key-kdf, CONFIG KdfParameters} |
    {MODE key-table,          CONFIG KeyTable}
}

EncryptionInfo ::= SEQUENCE {
    mode PAYLOADENCRYPTION.&mode ({EncryptionOptions}),
    config PAYLOADENCRYPTION.&Config ({EncryptionOptions}{@mode}),
    encryptedPayloadHash OCTET STRING
}

PayloadInfo ::= SEQUENCE {
    format      CHOICE {
        enum    ENUMERATED {
            undefined(0), raw-binary(1)
        },
        objectId    OBJECT IDENTIFIER
    },
```

```
    encryptionInfo EncryptionInfo OPTIONAL,
    storageIdentifier OCTET STRING,
    size INTEGER,
    payload CHOICE {
        reference    ResourceReference,
        integrated   OCTET STRING
    }
}

ResourceReference ::= SEQUENCE {
    hash        OCTET STRING,
    uri         UTF8String
}

CONDITION ::=  CLASS {
    &type ENUMERATED {
        vendorId(1),
        classId(2),
        deviceId(3),
        lastApplicationTime(4),

        vendorSpecificMinimum(2147483648)
    },
```



```
    &Value
} WITH SYNTAX {TYPE &type, VALUE &Value}

ConditionTable CONDITION ::= {
    {TYPE vendorId,       VALUE OCTET STRING} |
    {TYPE classId,        VALUE OCTET STRING} |
    {TYPE deviceId,       VALUE OCTET STRING} |
    {TYPE applyBefore,    VALUE INTEGER} |
    {TYPE vendorSpecific, VALUE OCTET STRING}
}

Condition ::= SEQUENCE {
    type  CONDITION.&type  ({ConditionTable}),
    value CONDITION.&Value ({ConditionTable} {@type})
}

DIRECTIVE ::=  CLASS {
    &type ENUMERATED {
```

```
        applyImmediately(1),
        applyAfter(2),
        restartComponent(3),
        restartSystem(4),
        installationHandler(5),

        vendorSpecificMinimum(2147483648)
    },
    &Rule
} WITH SYNTAX {TYPE &type, RULE &Rule}

DirectiveTable DIRECTIVE ::= {
    {TYPE applyImmediately,    RULE BOOLEAN} |
    {TYPE applyAfter,          RULE INTEGER} |
    {TYPE restartComponent,    RULE BOOLEAN} |
    {TYPE restartSystem,       RULE BOOLEAN} |
    {TYPE installationHandler, RULE OCTET STRING} |
    {TYPE vendorSpecific,      RULE OCTET STRING}
}

Directive ::= SEQUENCE {
    type DIRECTIVE.&type ({DirectiveTable}),
    rule DIRECTIVE.&Rule ({DirectiveTable} {@type})
}

TextField ::= SEQUENCE {
    type ENUMERATED {
        description(0), version(1), vendor(2), model(3)
    },
    value UTF8String
```



```
}

Manifest ::= SEQUENCE {
    manifestVersion ENUMERATED {
        v1(1)
    },
    text            SEQUENCE OF TextField OPTIONAL,
    nonce           OCTET STRING,
    digestAlgorithm AlgorithmIdentifier,
    timestamp       INTEGER,
    conditions      SEQUENCE OF Condition,
```

```
    directives      SEQUENCE OF Directive,
    aliases         SEQUENCE OF ResourceReference,
    dependencies    SEQUENCE OF ResourceReference,
    payload         PayloadInfo OPTIONAL
}

END
```

5. IANA Considerations

Editor's Note: A few registries would be good to allow easier allocation of new features.

6. Security Considerations

This document is about a manifest format describing and protecting firmware images and as such it is part of a larger solution for offering a standardized way of delivering firmware updates to IoT devices. A more detailed discussion about security can be found in the architecture document [Architecture].

7. Mailing List Information

The discussion list for this document is located at the e-mail address fud@ietf.org [1]. Information on the group and information on how to subscribe to the list is at https://www1.ietf.org/mailman/listinfo/fud

Archives of the list can be found at: https://www.ietf.org/mail-archive/web/fud/current/index.html

8. Acknowledgements

We would like the following persons for their support in designing this mechanism

- Geraint Luff

- Amyas Phillips

- Dan Ros

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[Architecture] Moran, B., "A Firmware Update Architecture for Internet of Things Devices", July 2017.

9.3. URIs

[1] mailto:fud@ietf.org

Authors' Addresses

Brendan Moran
ARM Limited

EMail: Brendan.Moran@arm.com

Milosch Meriac
ARM Limited

EMail: Milosch.Meriac@arm.com

Hannes Tschofenig
ARM Limited

EMail: hannes.tschofenig@gmx.net