SUIT
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

B. Moran
Arm Limited
M. Meriac
Consultant
H. Tschofenig
Arm Limited
D. Brown
Linaro

July 02, 2018

A Firmware Update Architecture for Internet of Things Devices
draft-ietf-suit-architecture-01

Abstract

Vulnerabilities with Internet of Things (IoT) devices have raised the need for a solid and secure firmware update mechanism that is also suitable for constrained devices. Incorporating such update mechanism to fix vulnerabilities, to update configuration settings as well as adding new functionality is recommended by security experts.

This document lists requirements and describes an architecture for a firmware update mechanism suitable for IoT devices. The architecture is agnostic to the transport of the firmware images and associated meta-data.

This version of the document assumes asymmetric cryptography and a public key infrastructure. Future versions may also describe a symmetric key approach for very constrained devices.

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 3, 2019.

Table of Contents

1. Introduction

When developing IoT devices, one of the most difficult problems to solve is how to update the firmware on the device. Once the device is deployed, firmware updates play a critical part in its lifetime, particularly when devices have a long lifetime, are deployed in remote or inaccessible areas or where manual intervention is cost prohibitive or otherwise difficult. The need for a firmware update may be to fix bugs in software, to add new functionality, or to re-configure the device.

The firmware update process, among other goals, has to ensure that

- The firmware image is authenticated and attempts to flash a malicious firmware image are prevented.

- The firmware image can be confidentiality protected so that attempts by an adversary to recover the plaintext binary can be prevented. Obtaining the plaintext binary is often one of the first steps for an attack to mount an attack.

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].

This document uses the following terms:

- Manifest: The manifest contains meta-data about the firmware image. The manifest is protected against modification and provides information about the author.

- Firmware Image: The firmware image is a binary that may contain the complete software of a device or a subset of it. The firmware image may consist of multiple images, if the device contains more than one microcontroller. The image may consist of a differential update for performance reasons. Firmware is the more universal term. Both terms are used in this document and are interchangeable.

- Bootloader: A bootloader is a piece of software that is executed once a microcontroller has been reset. It is responsible for deciding whether to boot a firmware image that is present or whether to obtain and verify a new firmware image. Since the bootloader is a security critical component its functionality may be split into separate stages. Such a multi-stage bootloader may offer very basic functionality in the first stage and resides in ROM whereas the second stage may implement more complex functionality and resides in flash memory so that it can be updated in the future (in case bugs have been found). The exact split of components into the different stages, the number of firmware images stored by an IoT device, and the detailed functionality varies throughout different implementations.

The following entities are used:

- Author: The author is the entity that creates the firmware image. There may be multiple authors in a system either when a device consists of multiple micro-controllers or when the the final firmware image consists of software components from multiple companies.

- Device: The device is the recipient of the firmware image and the manifest. The goal is to update the firmware of the device. A single device may need to obtain more than one firmware image and manifest to succesfully perform an update.

- Communicator: The communicator component of the device interacts with the firmware update server. It receives firmware images and triggers an update, if needed. The communicator either polls a firmware update server for the most recent manifest/firmware or manifests/firmware images are pushed to it. Note that the firmware update process may involve multiple stages since one or multiple manifests may need to be downloaded before the communicator can fetch one or multiple firmware images/software components.

- Status Tracker: The status tracker offers device management functionality that includes keep track of the firmware update process. This includes fine-grained monitoring of changes at the

device, for example, what state of the firmware update cycle the device is currently in.

- Firmware Server: Entity that stores firmware images and manifests. Some deployments may require storage of the firmware images/ manifests on more than one entities before they reach the device.

- Device Operator: The actor responsible for the day-to-day operation of a fleet of IoT devices.

- Network Operator: The actor responsible for the operation of a network to which IoT devices connect.

In addition to the entities in the list above there is an orthogonal infrastructure with a Trust Provisioning Authority (TPA) distributing trust anchors and authorization permissions to various entities in the system. The TPA may also delegate rights to install, update, enhance, or delete trust anchors and authorization permissions to other parties in the system. This infrastructure overlaps the communication architecture and different deployments may empower certain entities while other deployments may not. For example, in some cases, the Original Design Manufacturer (ODM), which is a company that designs and manufactures a product, may act as a TPA and may decide to remain in full control over the firmware update process of their products.

The terms ’trust anchor’ and ’trust anchor store’ are defined in [RFC6024]:

- "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative."

- "A trust anchor store is a set of one or more trust anchors stored in a device. A device may have more than one trust anchor store, each of which may be used by one or more applications."

Furthermore, the following abbreviations are used in this document:

- Microcontroller (MCU for microcontroller unit) is a small computer on a single integrated circuit, which is often used for mass volumne IoT devices.

- System on Chip (SoC) is an integrated circuit that integrates all components of a computer, such as CPU, memory, input/output ports, secondary storage, etc.

- Homogeneous Storage Architecture (HoSA): A device that stores all firmware components in the same way, for example in a file system or in flash memory.

- Heterogeneous Storage Architecture (HeSA): A device that stores at least one firmware component differently from the rest, for example a device with an external, updatable radio, or a device with internal and external flash memory.

3. Requirements

The firmware update mechanism described in this specification was designed with the following requirements in mind:

- Agnostic to how firmware images are distributed

- Friendly to broadcast delivery

- Use state-of-the-art security mechanisms

- Rollback attacks must be prevented

- High reliability

- Operate with a small bootloader

- Small Parsers

- Minimal impact on existing firmware formats

- Robust permissions

- Diverse modes of operation

3.1. Agnostic to how firmware images are distributed

Firmware images can be conveyed to devices in a variety of ways, including USB, UART, WiFi, BLE, low-power WAN technologies, etc. and use different protocols (e.g., CoAP, HTTP). The specified mechanism needs to be agnostic to the distribution of the firmware images and manifests.

3.2. Friendly to broadcast delivery

This architecture does not specify any specific broadcast protocol however, given that broadcast may be desirable for some networks, updates must cause the least disruption possible both in metadata and payload transmission.

For an update to be broadcast friendly, it cannot rely on link layer, network layer, or transport layer security. In addition, the same message must be deliverable to many devices, both those to which it applies and those to which it does not, without a chance that the wrong device will accept the update. Considerations that apply to network broadcasts apply equally to the use of third-party content distribution networks for payload distribution.

3.3. Use state-of-the-art security mechanisms

End-to-end security between the author and the device, as shown in Section 5, is used to ensure that the device can verify firmware images and manifests produced by authorized authors.

The use of post-quantum secure signature mechanisms, such as hash-based signatures, should be explored. A migration to post-quantum secure signatures would require significant effort, therefore, mandatory-to-implement support for post-quantum secure signatures is a goal.

A mandatory-to-implement set of algorithms has to be defined offering a key length of 112-bit symmetric key or security or more, as outlined in Section 20 of RFC 7925 [RFC7925]. This corresponds to a 233 bit ECC key or a 2048 bit RSA key.

If the firmware image is to be encrypted, it must be done in such a way that every intended recipient can decrypt it. The information that is encrypted individually for each device must be an absolute minimum, for example AES Key Wrap [RFC5649], in order to maintain friendliness to Content Distribution Networks, bulk storage, and broadcast protocols.

3.4. Rollback attacks must be prevented

A device presented with an old, but valid manifest and firmware must not be tricked into installing such firmware since a vulnerability in the old firmware image may allow an attacker to gain control of the device.

3.5. High reliability

A power failure at any time must not cause a failure of the device. A failure to validate any part of an update must not cause a failure of the device. One way to achieve this functionality is to provide a minimum of two storage locations for firmware and one bootable location for firmware. An alternative approach is to use a 2nd stage bootloader with build-in full featured firmware update functionality

such that it is possible to return to the update process after power down.

Note: This is an implementation requirement rather than a requirement on the manifest format.

3.6. Operate with a small bootloader

The bootloader must be minimal, containing only flash support, cryptographic primitives and optionally a recovery mechanism. The recovery mechanism is used in case the update process failed and may include support for firmware updates over serial, USB or even a limited version of wireless connectivity standard like a limited Bluetooth Smart. Such a recovery mechanism must provide security at least at the same level as the full featured firmware update functionalities.

The bootloader needs to verify the received manifest and to install the bootable firmware image. The bootloader should not require updating since a failed update poses a risk in reliability. If more functionality is required in the bootloader, it must use a two-stage bootloader, with the first stage comprising the functionality defined above.

All information necessary for a device to make a decision about the installation of a firmware update must fit into the available RAM of a constrained IoT device. This prevents flash write exhaustion.

Note: This is an implementation requirement.

3.7. Small Parsers

Since parsers are known sources of bugs they must be minimal. Additionally, it must be easy to parse only those fields that are required to validate at least one signature or MAC with minimal exposure.

3.8. Minimal impact on existing firmware formats

The design of the firmware update mechanism must not require changes to existing firmware formats.

3.9. Robust permissions

When a device obtains a monolithic firmware image from a single author without any additional approval steps then the authorization flow is relatively simple. There are, however, other cases where

more complex policy decisions need to be made before updating a device.

In this architecture the authorization policy is separated from the underlying communication architecture. This is accomplished by separating the entities from their permissions. For example, an author may not have the authority to install a firmware image on a device in critical infrastructure without the authorization of a device operator. In this case, the device may be programmed to reject firmware updates unless they are signed both by the firmware author and by the device operator.

Alternatively, a device may trust precisely one entity, which does all permission management and coordination. This entity allows the device to offload complex permissions calculations for the device.

3.10. Operating modes

There are three broad classifications of update operating modes.

- Client-initiated Update

- Server-initiated Update

- Hybrid Update

Client-initiated updates take the form of a communicator on a device proactively checking for new firmware imagines provided by firmware servers.

Server-initiated updates are important to consider because timing of updates may need to be tightly controlled in some high- reliability environments. In this case the communicator, potentially in coordination with the status tracker, determines what devices qualify for a firmware update. Once those devices have been selected the firmware server distributes updates to those devices.

Note: This assumes that the firmware server is able to reach the device, which may require devices to keep reachability information at the communicator and / or at the firmware server up-to-date. This may also require keeping state at NATs and stateful packet filtering firewalls alive.

Hybrid updates are those that require an interaction between the device and the firmware server / communicator. The communicator pushes notifications of availability of an update to the device, and the device then downloads the image from the firmware server when it wants.

An alternative approach is to consider the steps a device has to go through in the course of an update:

- Notification

- Pre-authorisation

- Dependency resolution

- Download

- Installation

The notification step consists of the communicator informing the device that an update is available. This can be accomplished via polling (client-initiated), push notifications (server-initiated), or more complex mechanisms.

The pre-authorisation step involves verifying whether the entity signing the manifest is indeed authorized to perform an update. The device must also determine whether it should fetching and processing of the firmware image (unless it has been attached already to the manifest itself).

A dependency resolution phase is needed when more than one component can be updated or when a differential update is used. The necessary dependencies must be available prior to installation.

The download step is the process of acquiring a local copy of the firmware image. When the download is client-initiated, this means that the device chooses when a download occurs and initiates the download process. When a download is server-party initiated, this means that either the communicator / firmware server tells the device when to download or that it initiates the transfer directly to the device. For example, a download from an HTTP-based firmware server is client-initiated. A transfer to a LwM2M Firmware Update resource [LwM2M] is server-initiated.

If the Device has downloaded a new firmware image and is ready to install it it may need to wait for a trigger from a Communicator to install the firmware update, may trigger the update automatically, or may go through a more complex decision making process to determine the appropriate timing for an update (such as delaying the update process to a later time when end users are less impacted by the update process).

Installation is the act of processing the payload into a format that the IoT device can recognise and the bootloader is responsible for then booting from the newly installed firmware image.

Each of these steps may require different permissions.

4. Claims

Claims in the manifest offer a way to convey instructions to a device that impact the firmware update process. To have any value the manifest containing those claims must be authenticated and integrity protected. The credential used to must be directly or indirectly related to the trust anchor installed at the device by the Trust Provisioning Authority.

The baseline claims for all manifests are described in [I-D.ietf-suit-information-model]. For example, there are:

- Do not install firmware with earlier metadata than the current metadata.
- Only install firmware with a matching vendor, model, hardware revision, software version, etc.
- Only install firmware that is before its best-before timestamp.
- Only allow a firmware installation if dependencies have been met.
- Choose the mechanism to install the firmware, based on the type of firmware it is.

5. Communication Architecture

Figure 1 shows the communication architecture where a firmware image is created by an author, and uploaded to a firmware server. The firmware image/manifest is distributed to the device either in a push or pull manner using the communicator residing on the device. The device operator keeps track of the process using the status tracker. This allows the device operator to know and control what devices have received an update and which of them are still pending an update.

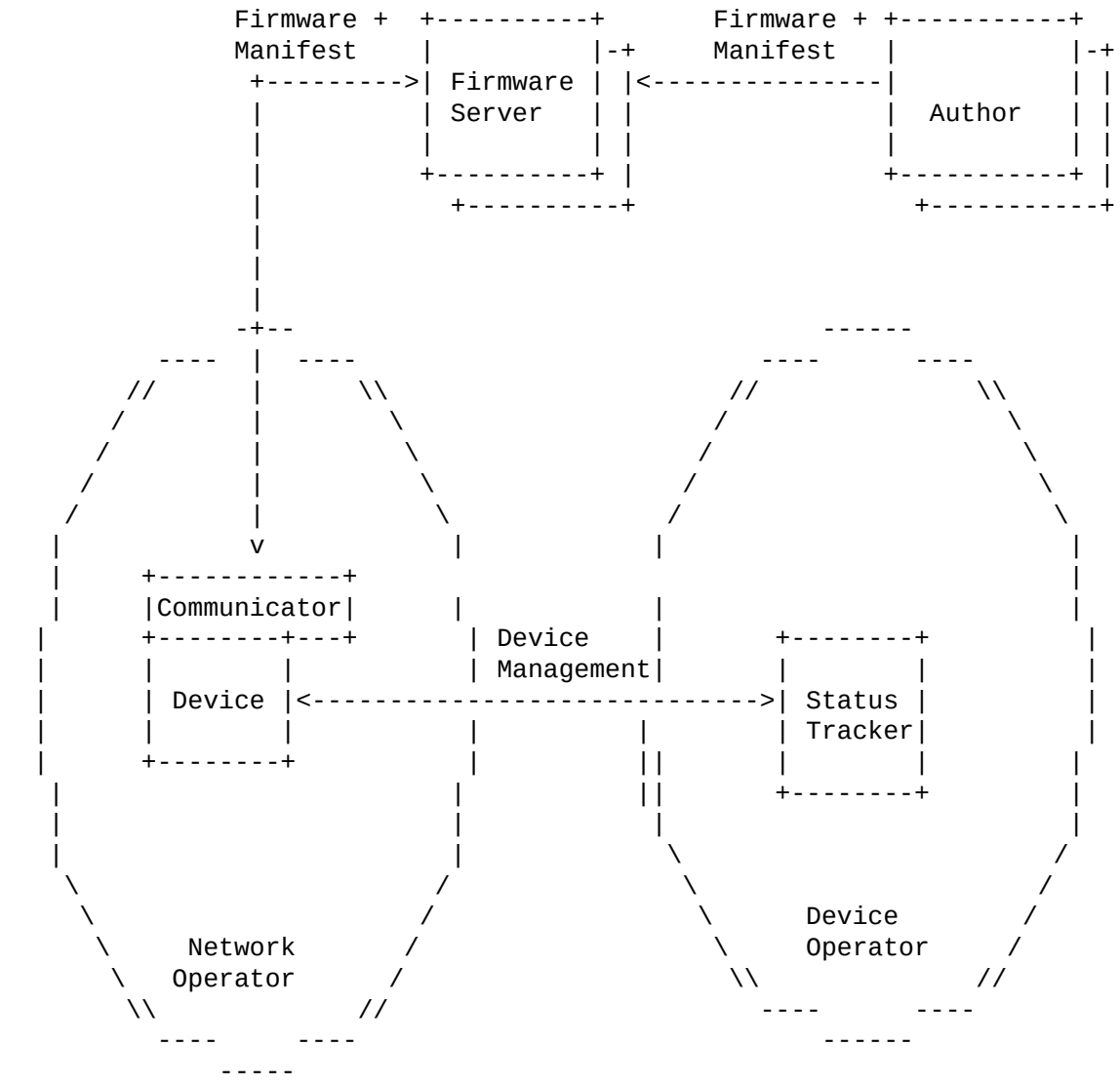


Figure 1: Architecture.

End-to-end security mechanisms are used to protect the firmware image and the manifest although Figure 2 does not show the manifest itself since it may be distributed independently.

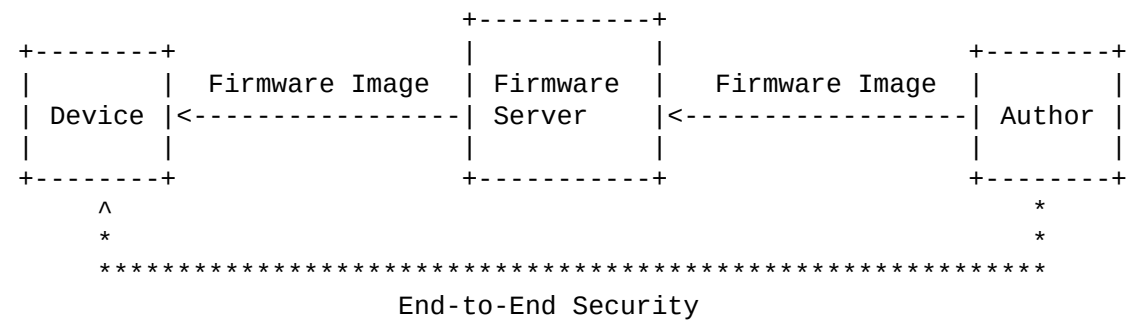


Figure 2: End-to-End Security.

Whether the firmware image and the manifest is pushed to the device or fetched by the device is a deployment specific decision.

The following assumptions are made to allow the device to verify the received firmware image and manifest before updating software:

- To accept an update, a device needs to verify the signature covering the manifest. There may be one or multiple manifests that need to be validated, potentially signed by different parties. The device needs to be in possession of the trust anchors to verify those signatures. Installing trust anchors to devices via the Trust Provisioning Authority happens in an out-of-band fashion prior to the firmware update process.

- Not all entities creating and signing manifests have the same permissions. A device needs to determine whether the requested action is indeed covered by the permission of the party that signed the manifest. Informing the device about the permissions of the different parties also happens in an out-of-band fashion and is also a duty of the Trust Provisioning Authority.

- For confidentiality protection of firmware images the author needs to be in possession of the certificate/public key or a pre-shared key of a device. The use of confidentiality protection of firmware images is deployment specific.

There are different types of delivery modes, which are illustrated based on examples below.

There is an option for embedding a firmware image into a manifest. This is a useful approach for deployments where devices are not connected to the Internet and cannot contact a dedicated server for download of the firmware. It is also applicable when the firmware update happens via a USB stick or via Bluetooth Smart. Figure 3 shows this delivery mode graphically.

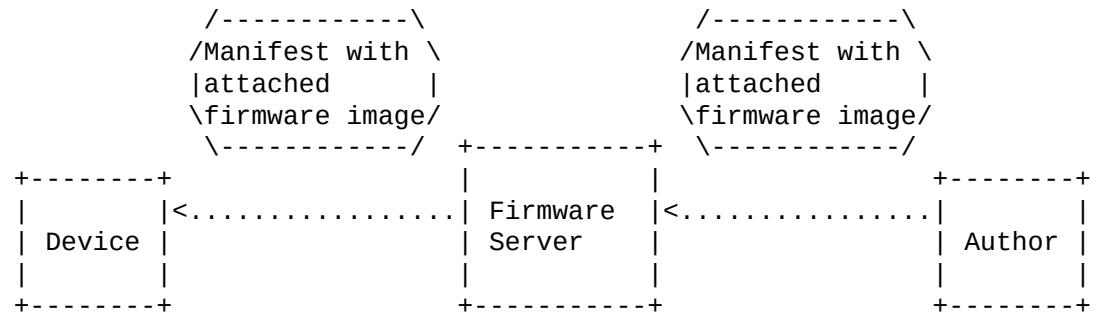


Figure 3: Manifest with attached firmware.

Figure 4 shows an option for remotely updating a device where the device fetches the firmware image from some file server. The manifest itself is delivered independently and provides information about the firmware image(s) to download.

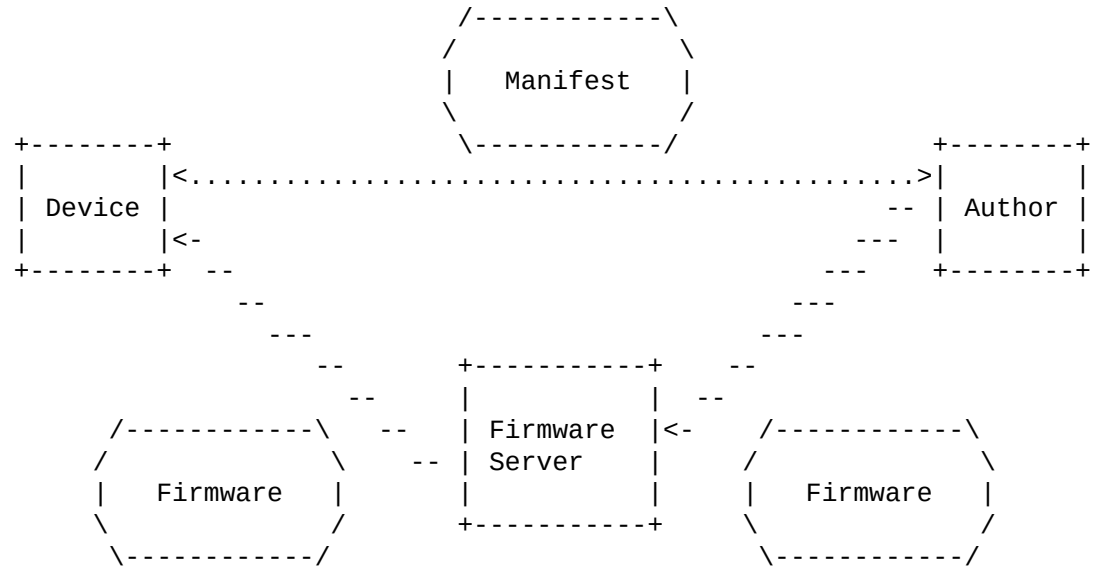


Figure 4: Independent retrieval of the firmware image.

This architecture does not mandate a specific delivery mode but a solution must support both types.

6. Manifest

In order for a device to apply an update, it has to make several decisions about the update:

- Does it trust the author of the update?

- Has the firmware been corrupted?
- Does the firmware update apply to this device?
- Is the update older than the active firmware?
- When should the device apply the update?
- How should the device apply the update?
- What kind of firmware binary is it?
- Where should the update be obtained?
- Where should the firmware be stored?

The manifest encodes the information that devices need in order to make these decisions. It is a data structure that contains the following information:

- information about the device(s) the firmware image is intended to be applied to,
- information about when the firmware update has to be applied,
- information about when the manifest was created,
- dependencies on other manifests,
- pointers to the firmware image and information about the format,
- information about where to store the firmware image,
- cryptographic information, such as digital signatures or message authentication codes (MACs).

The manifest information model is described in [I-D.ietf-suit-information-model].

7. Device Firmware Update Examples

Although these documents attempt to define a firmware update architecture that is applicable to both existing systems, as well as yet-to-be-conceived systems; it is still helpful to consider existing architectures.

7.1. Single CPU SoC

The simplest, and currently most common, architecture consists of a single MCU along with its own peripherals. These SoCs generally contain some amount of flash memory for code and fixed data, as well as RAM for working storage. These systems either have a single firmware image, or an immutable bootloader that runs a single image. A notable characteristic of these SoCs is that the primary code is generally execute in place (XIP). Combined with the non-relocatable nature of the code, firmware updates need to be done in place.

7.2. Single CPU with Secure - Normal Mode Partitioning

Another configuration consists of a similar architecture to the previous, with a single CPU. However, this CPU supports a security partitioning scheme that allows memory (in addition to other things) to be divided into secure and normal mode. There will generally be two images, one for secure mode, and one for normal mode. In this configuration, firmware upgrades will generally be done by the CPU in secure mode, which is able to write to both areas of the flash device. In addition, there are requirements to be able to update either image independently, as well as to update them together atomically, as specified in the associated manifests.

7.3. Dual CPU, shared memory

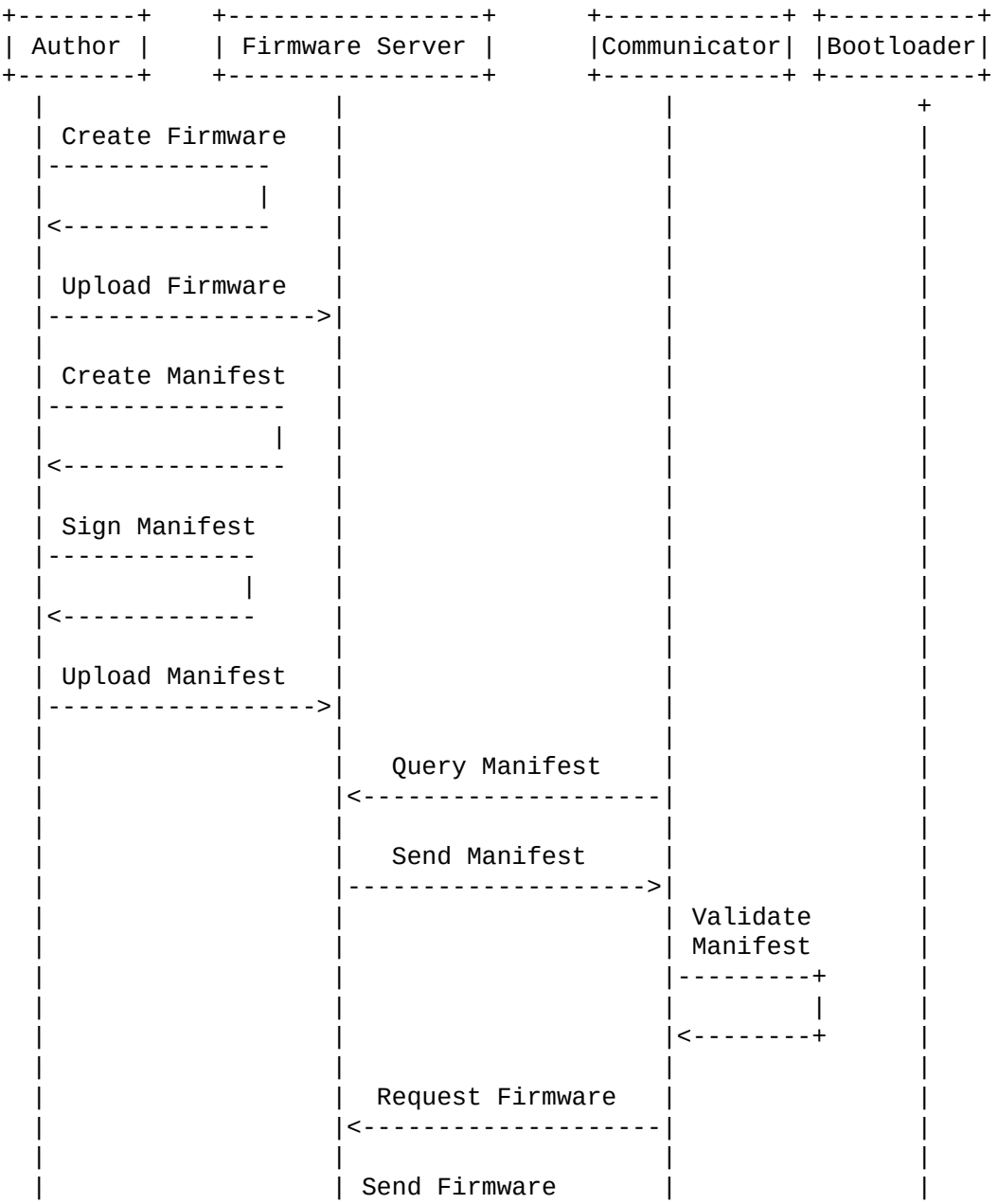
This configuration has two or more CPUs in a single SoC that share memory (flash and RAM). Generally, they will be a protection mechanism to prevent one CPU from accessing the other’s memory. Upgrades in this case will typically be done by one of the CPUs, and is similar to the single CPU with secure mode.

7.4. Dual CPU, other bus

This configuration has two or more CPUs, each having their own memory. There will be a communication channel between them, but it will be used as a peripheral, not via shared memory. In this case, each CPU will have to be responsible for its own firmware upgrade. It is likely that one of the CPUs will be considered a master, and will direct the other CPU to do the upgrade. This configuration is commonly used to offload specific work to other CPUs. Firmware dependencies are similar to the other solutions above, sometimes allowing only one image to be upgraded, other times requiring several to be upgraded atomically. Because the updates are happening on multiple CPUs, upgrading the two images atomically is challenging.

8. Example Flow

The following example message flow illustrates the interaction for distributing a firmware image to a device starting with an author uploading the new firmware to Firmware Server and creating a manifest. The firmware and manifest are stored on the same Firmware Server.



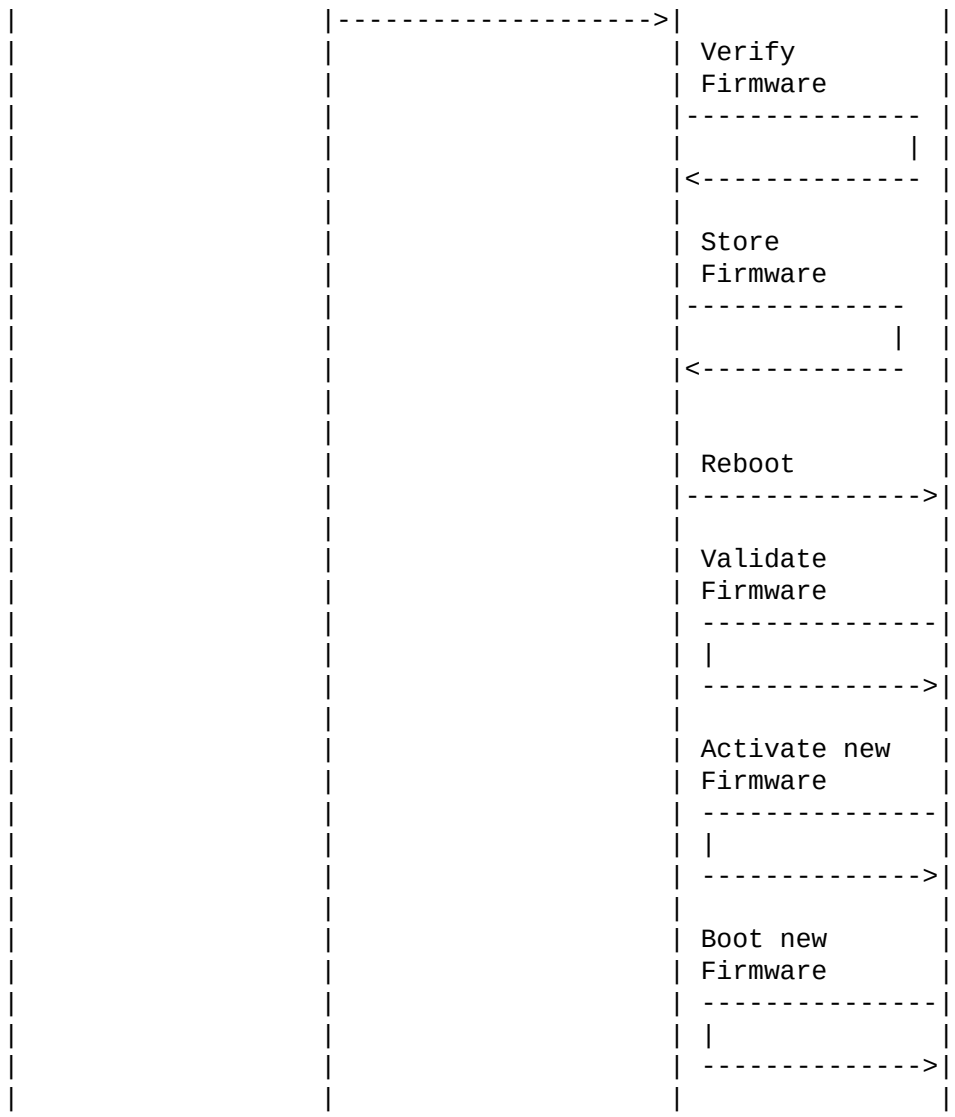


Figure 5: Example Flow for a Firmware Upate.

9. IANA Considerations

This document does not require any actions by IANA.

10. Security Considerations

Firmware updates fix security vulnerabilities and are considered to be an important building block in securing IoT devices. Due to the importance of firmware updates for IoT devices the Internet Architecture Board (IAB) organized a 'Workshop on Internet of Things

(IoT) Software Update (IOTSU)', which took place at Trinity College Dublin, Ireland on the 13th and 14th of June, 2016 to take a look at the big picture. A report about this workshop can be found at [RFC8240]. A standardized firmware manifest format providing end-to-end security from the author to the device will be specified in a separate document.

There are, however, many other considerations raised during the workshop. Many of them are outside the scope of standardization organizations since they fall into the realm of product engineering, regulatory frameworks, and business models. The following considerations are outside the scope of this document, namely

- installing firmware updates in a robust fashion so that the update does not break the device functionality of the environment this device operates in.

- installing firmware updates in a timely fashion considering the complexity of the decision making process of updating devices, potential re-certification requirements, and the need for user consent to install updates.

- the distribution of the actual firmware update, potentially in an efficient manner to a large number of devices without human involvement.

- energy efficiency and battery lifetime considerations.

- key management required for verifying the digital signature protecting the manifest.

- incentives for manufacturers to offer a firmware update mechanism as part of their IoT products.

11. Mailing List Information

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

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

12. Acknowledgements

We would like to thank the following persons for their feedback:

- Geraint Luff
- Amyas Phillips
- Dan Ros
- Thomas Eichinger
- Michael Richardson
- Emmanuel Baccelli
- Ned Smith
- Jim Schaad
- Carsten Bormann
- Cullen Jennings
- Olaf Bergmann
- Suhas Nandakumar
- Phillip Hallam-Baker
- Marti Bolivar
- Andrzej Puzdrowski
- Markus Gueller
- Henk Birkholz
- Jintao Zhu

We would also like to thank the WG chairs, Russ Housley, David Waltermire, Dave Thaler for their support and their reviews. Kathleen Moriarty was the responsible security area director when this work was started.

13. References

13.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

13.2. Informative References

[I-D.ietf-suit-information-model] Moran, B., Tschofenig, H., Birkholz, H., and J. Jimenez, "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", draft-ietf-suit-information-model-00 (work in progress), June 2018.

[LwM2M] OMA, ., "Lightweight Machine to Machine Technical Specification, Version 1.0.2", February 2018, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://www.rfc-editor.org/info/rfc5649>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/info/rfc6024>.

[RFC8240] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", RFC 8240, DOI 10.17487/RFC8240, September 2017, <https://www.rfc-editor.org/info/rfc8240>.

13.3. URIs

[1] mailto:suit@ietf.org

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Milosch Meriac
Consultant

EMail: milosch@meriac.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@gmx.net

David Brown
Linaro

EMail: david.brown@linaro.org