AN12900

Secure Over-the-Air Prototype for Linux Using CAAM and Mender or SWUpdate

Rev. 1 — 11/2020 Application Note

Contents

1 Introduction

Over-the-Air (OTA) updates are an efficient way for OEMs to update any software on the devices that are programmed and operational in the field. By performing OTA updates, the software quality and the lifetime of the devices in the field increases.

This document provides a prototype implementation for Secure OTA for Linux images, specifically for NXP devices (in this case i.MX 8M/MM families) using Cryptographic Acceleration and Assurance Module (CAAM) accelerator and Mender or SWUpdate.

Its purpose is to perform U-Boot, rootfs, and software application updates.

1.1 Intended audience and scope

This document is intended for software, hardware, and system engineers who are ready to integrate OTA updates on their devices.

1.2 References

- i.MX 8MM Security Reference Manual
- Mender documentation
- Secure boot
- SWUpdate documentation

2 Overview

The first solution described in this document is based on Mender, a fully robust and secure client-server open source project for updating IoT devices. On top of Mender, we have integrated the Manufacturing Protection mechanism for a device – server authentication and U-Boot update feature.

Secondly, we used SWUpdate for upgrading the system. SWUpdate is an open source framework, highly configurable that is focused on updating an embedded system.

All the patches, deep dive steps, and suggested changes can be found in Setup.

The purpose of this system implementation can be summed up by the following main properties:

- Reliability
  - The update does not impact the current working of the device
  - The update does not fail the device
- Authentication
  - The initial handshake messages are signed with the valid/reliable keys
  - The server can authenticate the chip to see if it is NXP legitimate
- Authorization

  - The server can check if the chip has the right authorization to request updates and it is running the right security configuration settings
- Confidentiality
  - The keys and the OTA update package are encrypted
  - The keys are kept encrypted using red blobs

2.1 Mender

Mender is an open-source project for updating embedded Linux devices (IoT). Mender is based on client-server architecture. The server is using the microservices design pattern, which is multiple small and isolated services that make up the server. The Docker Compose environment brings all these components up and connects them to store and control the deployment of software updates OTA to the device (client). Mender's Web User Interface (UI) or REST APIs can be used for device management and also for uploading and deploying updates. Mender client runs on the device and periodically checks for updates. If a software update is available for that device, the client downloads and installs it.

The Mender client is compatible with Yocto Linux and the integration with i.MX 8M family is already available on the Mender project website.

The main properties of Mender are:

- Roll-back if the update fails
- Atomic updates (no partial install)
- Compatibility check (verifies if the artifact was created for that device type)
- The authenticity of update images (update signature verification)
- Persistent data storage

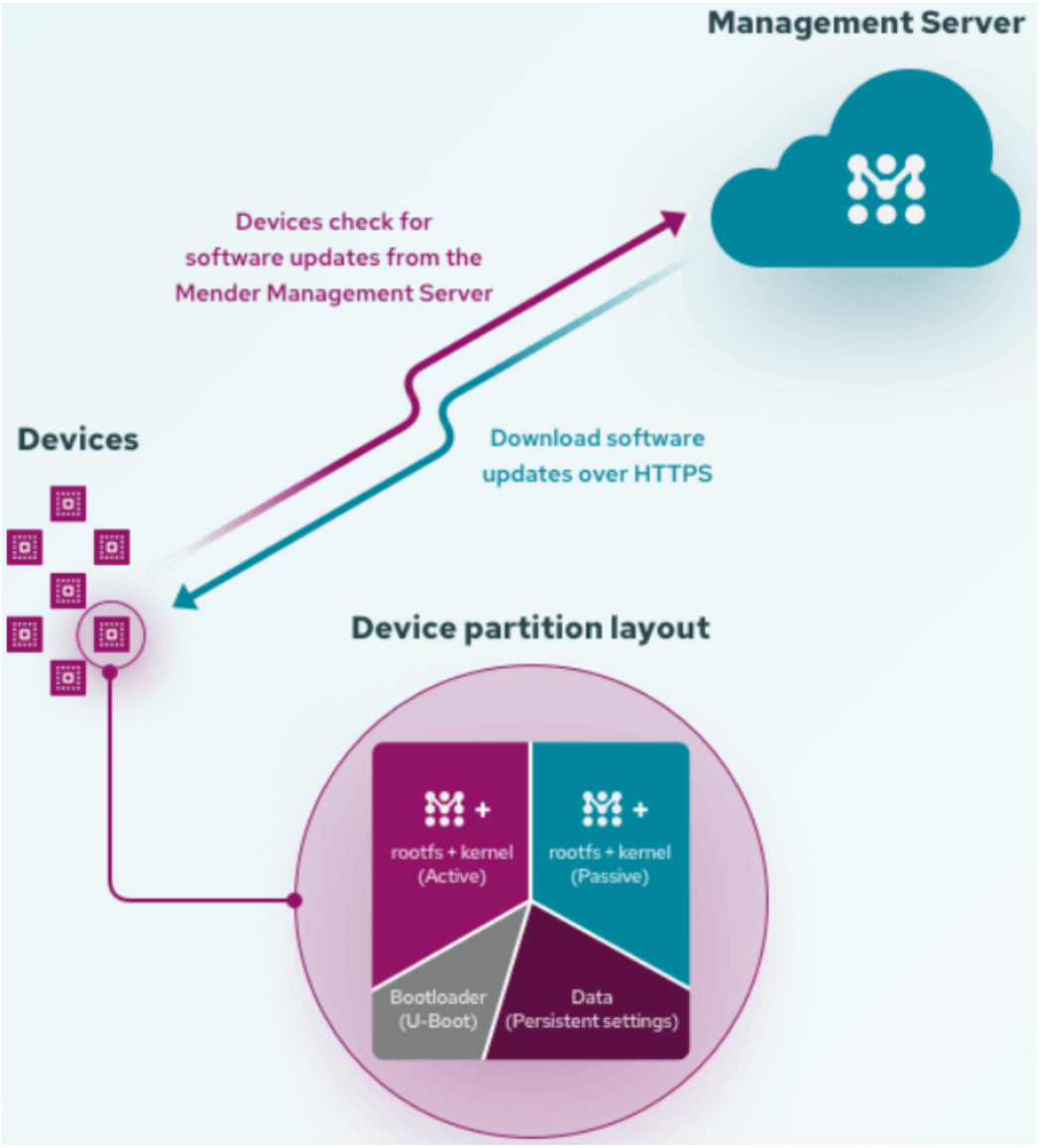

Figure 1. Mender architecture

The source of the above figure is Mender.

2.1.1 Client-server communication

Mender uses public-key cryptography for both client-server and inter-service communications. It uses several key pairs encoded in PEM format, the public key, that may also have a certificate, which is shared with other services and also a private key, which is kept secret by the service. Having different key pairs for all the services, the attack surface is limited if one private key gets compromised.

A device can authenticate to the Mender server after the set of identity attributes and the certificate are verified and the user explicitly authorizes the device from the Mender API. The set of identity attributes consists of MAC addresses or user-defined UIDs, and so on. The combination of the set of identity attributes and the public key encoded in PEM format is termed an authentication set. The authentication set sent by the client is signed with the private key, and the server uses the public key to verify the signature.

The following authentication request sent to the server:

```
type AuthReqData struct {
    // identity data
    IdData string `json:"id_data"`
    // tenant token
    TenantToken string `json:"tenant_token"`
    // client's public key
    Pubkey string `json:"pubkey"`
}

type AuthRequest struct {
    // raw request message data
    AuthReqData []byte
    // tenant's authorization token
    Token AuthToken
    // request signature
    Signature []byte
}
```

2.1.2 Rootfs and applications updates

Mender requires a dual rootfs partition layout on the device to recover, when the deployment is incomplete or corrupted during installation. Before installing an update, the Mender client verifies which rootfs partition is running and configures U-Boot to boot from the updated rootfs partition. Next, writes the updated image to the rootfs that is inactive and reboots the device. If booting the updated partition fails, use the other partition and ensure that the device does not get failed. Persistent information can be stored in the data partition and remains unchanged during the update process. Regarding application updates, the dual rootfs partition is not necessary and the update can be installed as a .deb package without rebooting the device. In case the update does not install correctly, the update script can implement the rollback method and restore the old version.

2.1.3 Update manager

When intalling an Artifact payload, the Mender client has to perform the action steps that are implemented in an Update Module. The important action step is Install. This action describes how an Artifact payload is installed on the device. Other important action that may be implemented in an Update Module is Rollback. Depending on the desired functionality and usescase, other actions may be implemented in an Update Module. More detalis can be found in Mender project documentation.

To create and inspect Mender Artifacts, mender-artifact utility must be used. Mender Artifacts are compressed with the gzip algorithm by default, and also supports lzma compression and no compression.

For example, to create an artifact with a new U-Boot binary file for an imx8mmddr4evk device, use the following commands:

```
export DEVICE_TYPE=imx8mmddr4evk./mender-artifact write module-image -t $DEVICE_TYPE -o update-
uboot.mender -T update-uboot -n updateuboot-1.0 - f signed_flash.bin
```

For signing and verifying artifacts, use a key pair. The private key is used for signing the Mender Artifact must be protected and kept outside of the build system. The public key which is used by the Mender Client to verify the signed Artifact must be included in the Mender Client build.

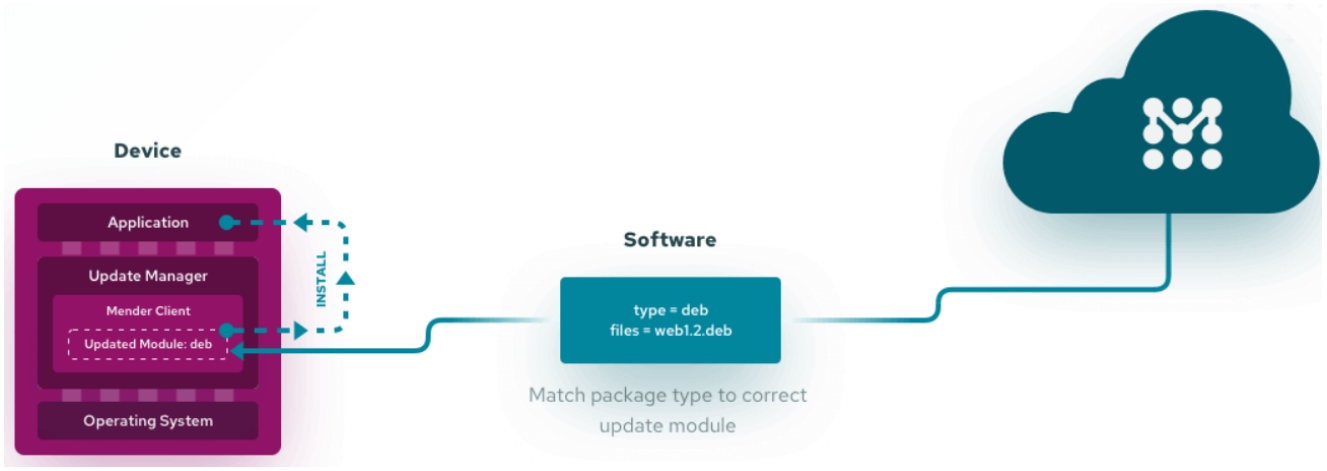

Figure 2. Update Module

The source of the above figure is Mender.

Examples of how the Artifacts are created, uploaded, deployed, and how to implement Update Modules, are available on the Mender documentation.

2.2 SWUpdate

SWUpdate is an open source project that can be used for updating an embedded device. It is highly configurable and, as mentioned in the documentation, it is considered to be a framework.

The installation of the update can be performed from local storage (for example, USB, SD), remote (Hawkbit server, integrated Web Server or pulling from remote server), or from command line. Unlike Mender, all the dual rootfs logic, rollback and other low level implementation details need to be created by the user.

The files needed for updating the device together with the metadata about each file are compressed in a CPIO archive representing the artifact which will be deployed on the device. In the metadata file (`sw-description`) four important sections may be defined:

1. images (for example, rootfs image)
2. files
3. scripts (considered being executable; pre- and post- install scripts)
4. bootenv

Also, the update image contains details about hardware compatibility in order to make sure that the update is installed only on devices having a certain hardware revision and is compatible with the software image. Moreover, the software version can be verified to skip the installation if it is the same or if the version of new software is older than the version of installed software. For examples and steps on how to create an update, see the SWUpdate documentation.

From the security perspective, SWUpdate provides support for symmetric key encryption of images as well as for signing and verifying the signed images using a given public key.

The hash for every image or file that is part of the update image, needs to be added in the metadata file (`sw-description`). Then, the `sw-description` file will be signed. This ensures that the update image will not be tampered.

Two of the most common update strategies provided by SWUpdate are double copy and single copy. In the double copy with fall-back strategy, the required space is twice as much than in the single copy strategy but it guarantees that there is always a working copy available. The bootloader is in charge for booting the proper image.

To ease the integration of the SWUpdate project, a `meta-swupdate` layer is provided and can be added in the Yocto Project.

For more information and examples, see the SWUpdate documentation.

2.2.1 Double copy with fall-back

In the double copy with fall-back strategy, there are two copies of the kernel and the root file system.

The updated image is installed to the rootfs that is not active, then the device reboots. If booting the updated rootfs fails, the other partition is used to ensure that the device is functional. For storing persistent information, another partition can be created that will remain unchanged during the update.

Besides updating the root file system including the kernel and applications in the active rootfs, U-Boot updates can be performed as described in U-Boot update.

In order to improve the security of the update system, the Manufacturing protection mechanism described in Manufacturing Protection mechanism can be integrated for client-server authentication.

As previously mentioned, unlike Mender, SWUpdate does not provide implementation on how to set up the device, including the dual rootfs partition layout and the recovery from failure.

Here you can find a device set up example which was implemented for demo purpose only.

2.3 Manufacturing Protection mechanism

The Manufacturing Protection (MP) feature is used when authenticating the device to the OEMs server. This authentication process can be used to ensure that the device is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in secure boot. Secure boot is a key feature of the boot ROM, also known as a High Assurance Boot (HAB). The HAB uses a combination of hardware and software together with the Public Key Infrastructure (PKI) protocol to protect the system from executing unauthorized programs. Before the HAB allows the user image to execute, the image must be signed. The signing process is done during the image build process by the private key holder and the signatures are then included as a part of the final program image. If configured to do so, the ROM verifies the signatures using the public keys included in the program image.

The foundation for the MP feature is a private/public key pair generated on the device. The private key is unreadable and unexportable. It is coupled to the OEM SRK fuse data used for secure boot, and available for use after a device is enabled for secure boot.

During the product development, when a device is configured for secure boot, the public key needs to be generated and saved for use by an authentication/provisioning server. The generated public key is same for all devices of an i.MX family programmed with the same OEM SRKs.

MP is backed by the CAAM and enabled by HAB during system boot. CAAM supplies three functions to support the feature. The first function generates a non-readable and non-exportable private key. The second function sign data with the private key. The third function generates a corresponding public key used for authenticating the signatures generated by devices. To provide data to the MPSign function, the secure boot firmware writes data in the MPMR, then locks it. Additional data can be provided as ordinary message input to the MPSign function. This additional data is added to the content of the MPMR before the data is hashed and signed. Details for building the descriptors can be found in the i.MX 8MM Security Reference Manual.

To securely extract the MP Public key and for sign data using the MP Private key, OP-TEE is used. OP-TEE is an open-source project used to create a Trusted Execution Environment (TEE) in a non-secure Linux. The Client Application (CA) opens a session within the Trusted Application (TA) and requests MPPubk extraction or signing using MPPrivk. To access low-level primitives in CAAM a Pseudo TA running in the OP-TEE core is used. The CA receives the response and forwards it to the script that made the initial request.

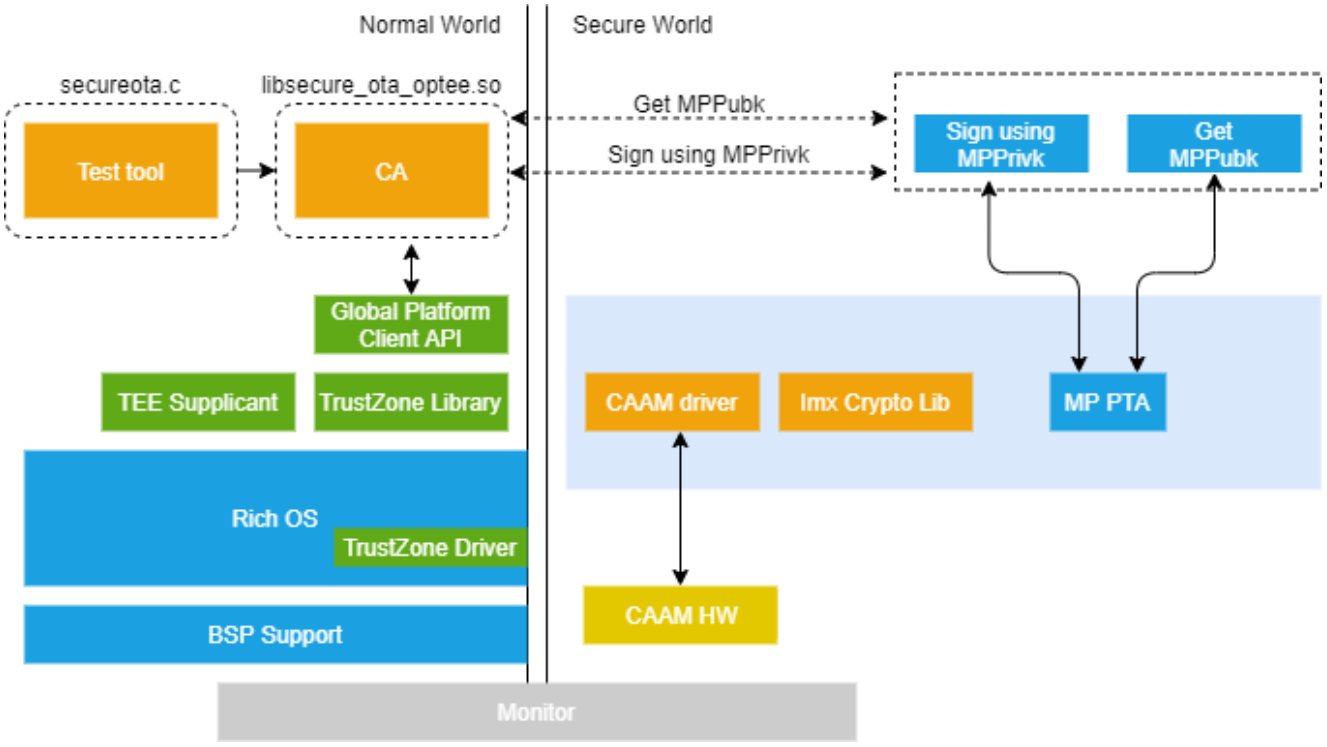

Figure 3. Overview of the OP-TEE application for MP

The MPPubk is encoded in PEM format and stored in the data partition as a red blob. A blob is a cryptographic data structure that CAAM uses to protect user-defined data across system power cycles. It provides both confidentiality and integrity protection. The data to be protected is encrypted so that it can be placed safely in non-volatile storage before the chip is powered down. Each time the blob protocol is used to protect data and a randomly generated key is used to encrypt the data. The random key is itself encrypted using key encryption and is then stored along with the encrypted data. The encryption key is derived from the chip master secret key so the encryption key can be recreated when the chip powers up again. The combination of encrypted key and encrypted data is called a blob. CAAM blobs differ depending on the data needed to be protected. Red blobs are left unencrypted when the blob is decapsulated.

As mentioned in Client-server communication, one of the steps performed for client-server authentication is sending and processing a set of identity attributes. The authentication request sent by the client is signed with the MP Private key, and the server uses the MP Public key to verify the signature. To verify the signature, instead of having the MP Public key, the server has to recreate the hash of the authentication set which is a combination of MPMR and the identity attributes.

2.4 U-Boot update

To perform a U-Boot update, a backup solution is available ensuring that the device is up and running even if the update does not installed successfully or corrupted.

On the i.MX 8M family, redundant boot feature is in ROM, it uses the second image only when the first boot image (SPL) is corrupted. Therefore, by default, in case the FIT image gets corrupted or tampered (if it is not a valid image) the system can hang.

The ROM supports redundant boot if PERSIST_SECONDARY_BOOT bit is set to 1. For the secondary image support, the primary image must reserve space for the secondary image table.

Table 1. Secondary image table

Reserved (chipNum)

Table continues on the next page...

Table 1. Secondary image table (continued)

Reserved (driveType)
Tag
firstSectorNumber
Reserved (sectorCount)

Where:

• The *Tag* is used as an indication of the valid secondary image table. It must be *0x00112233*.

• The *firstSectorNumber* is the first 512-byte sector number of the secondary image.

For the Closed mode, if there are failures during primary image authentication, the boot ROM turns on the PERSIST_SECONDARY_BOOT bit and performs the software reset. After the software reset, the secondary image is used.

To update the U-Boot image and recover when the FIT image is corrupted, a customized solution need to be implemented at the SPL level. In case the HAB authentication for the primary U-Boot fails, load the secondary U-Boot from a known sector number and if the authentication succeeds, use the secondary U-Boot.

Primary U-Boot:

```
sudo dd if=signed_flash.bin of=/dev/<sdX> bs=1K seek=33 && sync
```

Secondary U-Boot:

```
sudo dd if=signed_flash.bin of=/dev/<sdX> bs=1K seek=2048 && sync
```

When a U-Boot update is available, ensure that which U-Boot image is active, before updating the primary U-Boot image. If the primary U-Boot image is running, it has to be copied over to the secondary one. In this way, make sure that a valid version is available as a secondary U-Boot. If the secondary U-Boot image is loaded, the primary one installed directly. To check which U-Boot image was loaded, 1 or 2 is written at a certain address in memory at the SPL level and this value is read from user-space.

Another option when receiving a U-Boot update is to always update the primary image and have the secondary U-Boot image at its initial version, never updated. Therefore, if a U-Boot update is available, the primary one gets updated and if it is corrupted use the secondary one.

For the current implementation, in case both U-Boot images are corrupted or tampered, the device will not be remotely usable anymore.

3 Setup

For the setup instructions and to run the demo application see here. For any problems related to Mender client-server communication, see the troubleshooting section from the Mender project documentation. For any SWUpdate related issue, see the help and support section from SWUpdate documentation.

4 Usage

After booting the device, it is recommended to verify that the necessary components for the update system are integrated. Key functionalities include:

- libubootenv is enbaled
- Dual rootfs partition layout
- The kernel is loaded from the expected rootfs
- The rootfs is mounted from the corresponding partition

- The Mender/SWUpdate daemon is started, depending on the update system used

More information about verifying the correctness of the client integration can be found in the update system documentation.

4.1 Mender User Interface

The Mender client initializes the communication with the server and does not require any open ports on the embedded device. The client can be configured to send requests to the server at a reasonable frequency. For the device authentication step, if the certificate and the authentication set are verified by the server, the device appears on the Mender server UI as pending. After getting authorized by the server owner, updates are ready to be deployed on the device.

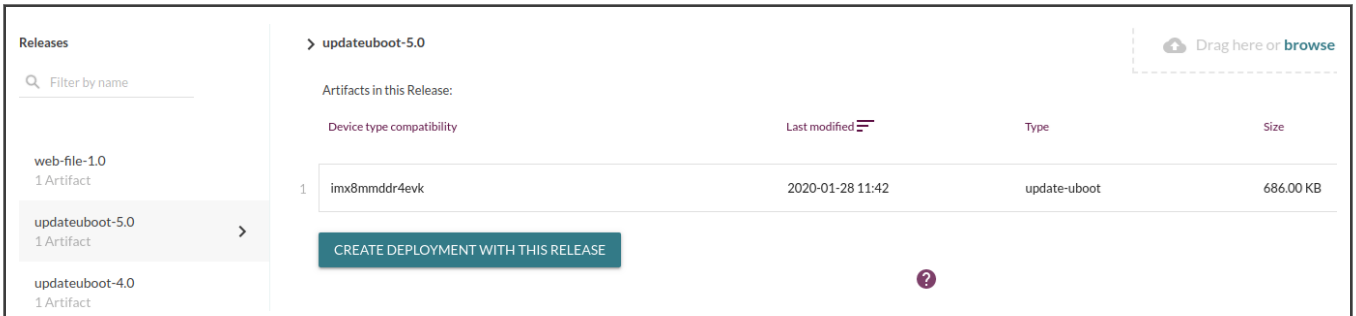

Figure 4. Mender UI – artifact deployment

The deployments status, as well as the log files, are available in the Mender UI server as shown in the following figure. If an update is already installed, it can not be reinstalled and the status as already installed. To install an update, the device has to have an Update Module implemented, as mentioned in Update manager.

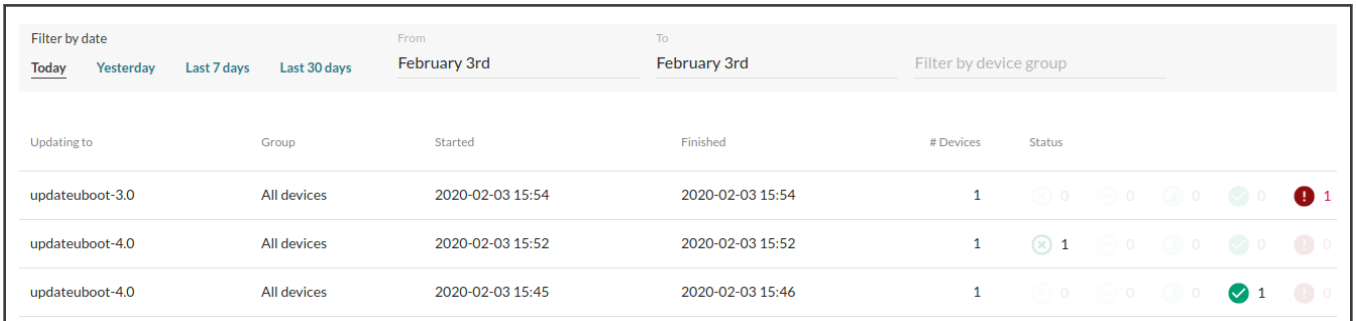

Figure 5. Mender UI – deployment status

4.2 SWUpdate User Interface

SWUpdate provides several methods for OTA updates including an integrated Web Server. When an update image is deployed on the device, SWUpdate verifies its signature and the checksums for each file included in the CPIO archive. If this stage is completed successfully, it checks the hardware and software compatibility. Next, if all the components described in the metadata file are in the archive and after running the pre-install scripts, the update will be installed. Finally, the post-install scripts are run, the bootloader environment is updated if it is specified in `sw-description` file and the update status is reported.

The deployment status and the debug information are available in the UI as shown in the following figure.

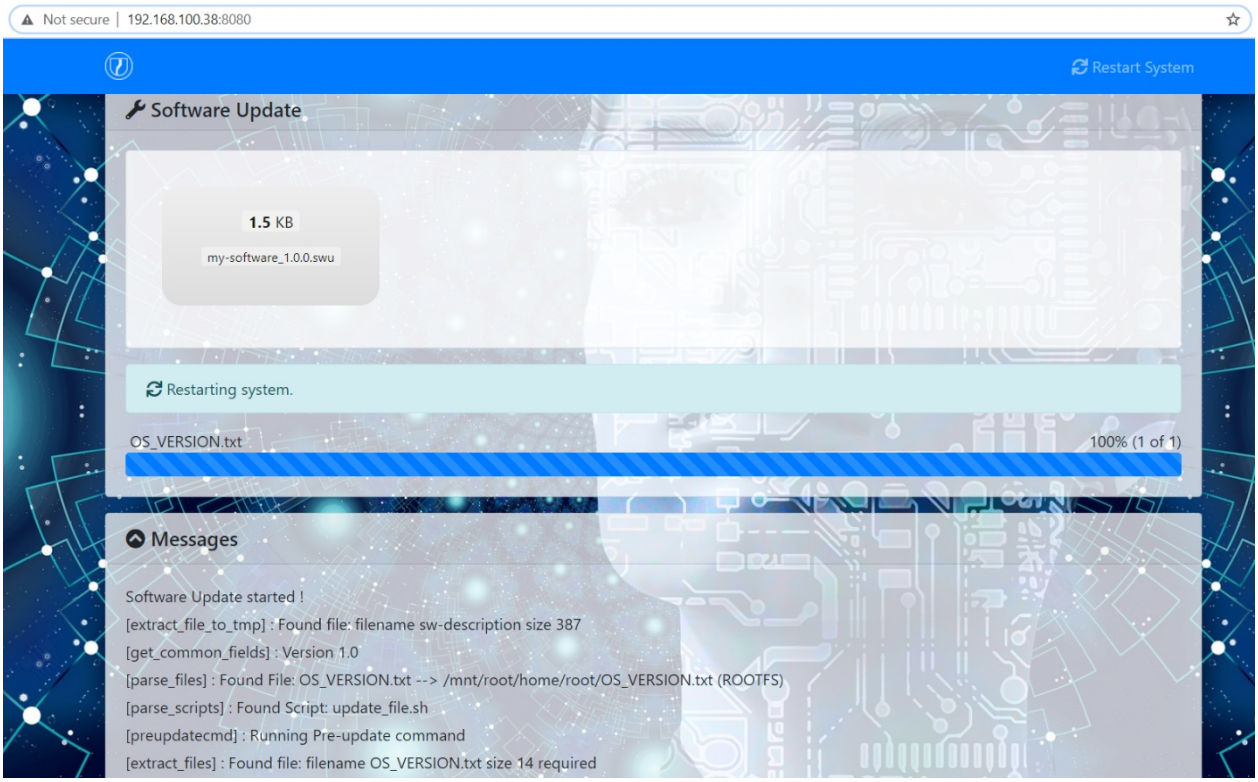

Figure 6. SWUpdate Integrated Web Server – update install passed

Besides the integrated Web Server, the update images can be deployed from Hawkbit server or can be pulled from a remote server. Further, the SWUpdate framework can be set up to communicate with other back end servers to roll out the update images.

5 Conclusion

In this document, two software update mechanisms for deploying OTA updates on iMX8M are presented. Both solutions, Mender and SWUpdate, support root file system updates, application level updates, and U-Boot updates. The manufacturing protection mechanism can be integrated into the client-server authentication process and ensures that the device is a genuine NXP part, is of a correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the secure boot.

Table 2 presents some key characteristics for Mender and SWUpdate. Users may decide which update system to use, depending on the project.

Table 2. Conclusion

	License	Yocto integration	Usability	Update strategy	Updates	Encrypted boot	Remote updates
Mender	Apache-2.0	meta-mender	Usable out of the box. Relies on some assumptions (U-Boot Count Limit, Linux env	Dual-image update solution.	Signed images delivered as a file archive in the tarball format.	Not tested yet on i.MX8M.	Web-interface and back-end update system using Docker over HTTPS.

Table continues on the next page...

Table 2. Conclusion (continued)

			tools, and so on) and implements all the necessary steps for setting up the device. It is a complete and robust solution.				
SWUpdate	GPL-2.0+	meta-swupdate	Highly configurable; all the necessary steps for setting up the device must be implemented by the user. Uses Kbuild for configuration.	Single copy and double copy with fall-back.	Signed and encrypted images delivered as a CPIO archive.	Succesfully updated the rootfs and the signed kernel.	Pulling from a remote HTTPS server or add a custom server. Currently, by default, it supports a Hawkbit server.

6 Revision history

Table 3. Revision history

Revision number	Date	Substantive changes
1	11/2020	• Added SWUpdate User Interface • Added Double copy with fall-back • Added SWUpdate User Interface • Added Mender User Interface • Updated Introduction • Updated References • Updated Overview • Updated Setup • Updated Usage • Added Conclusion
0	07/2020	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11/2020

Document identifier: AN12900

arm