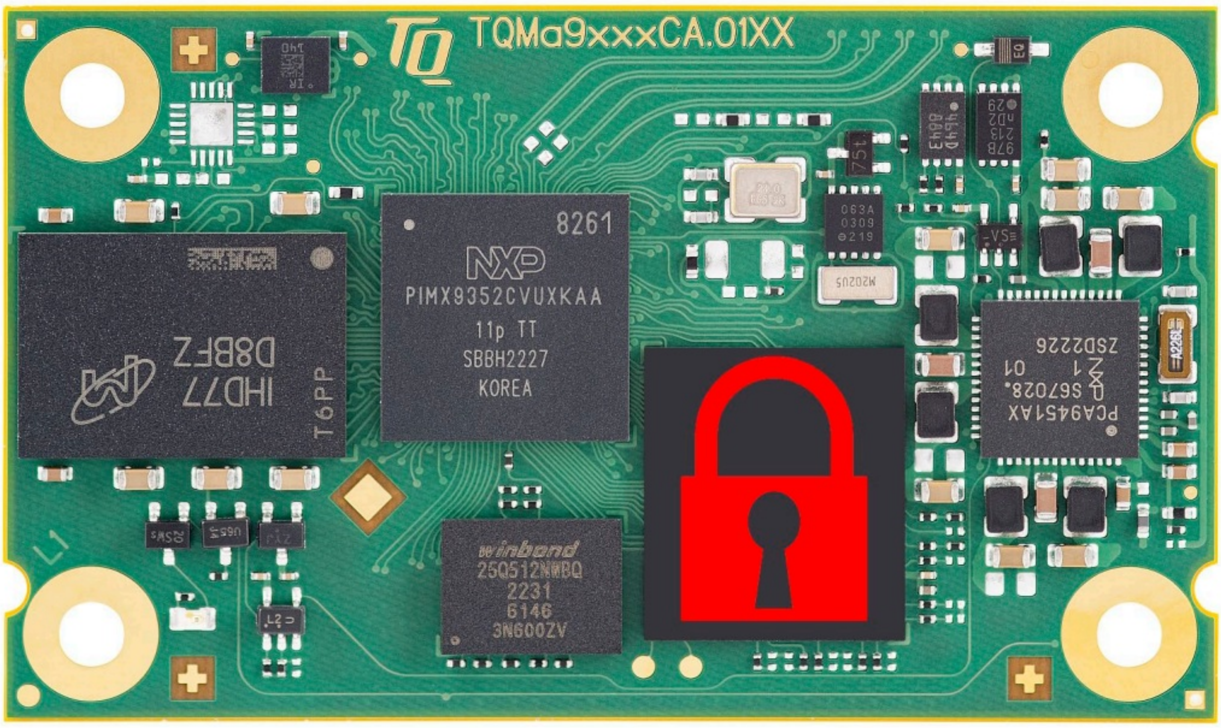
Contents [hide]

TQ

TQMa93 Secure Boot



Product Information

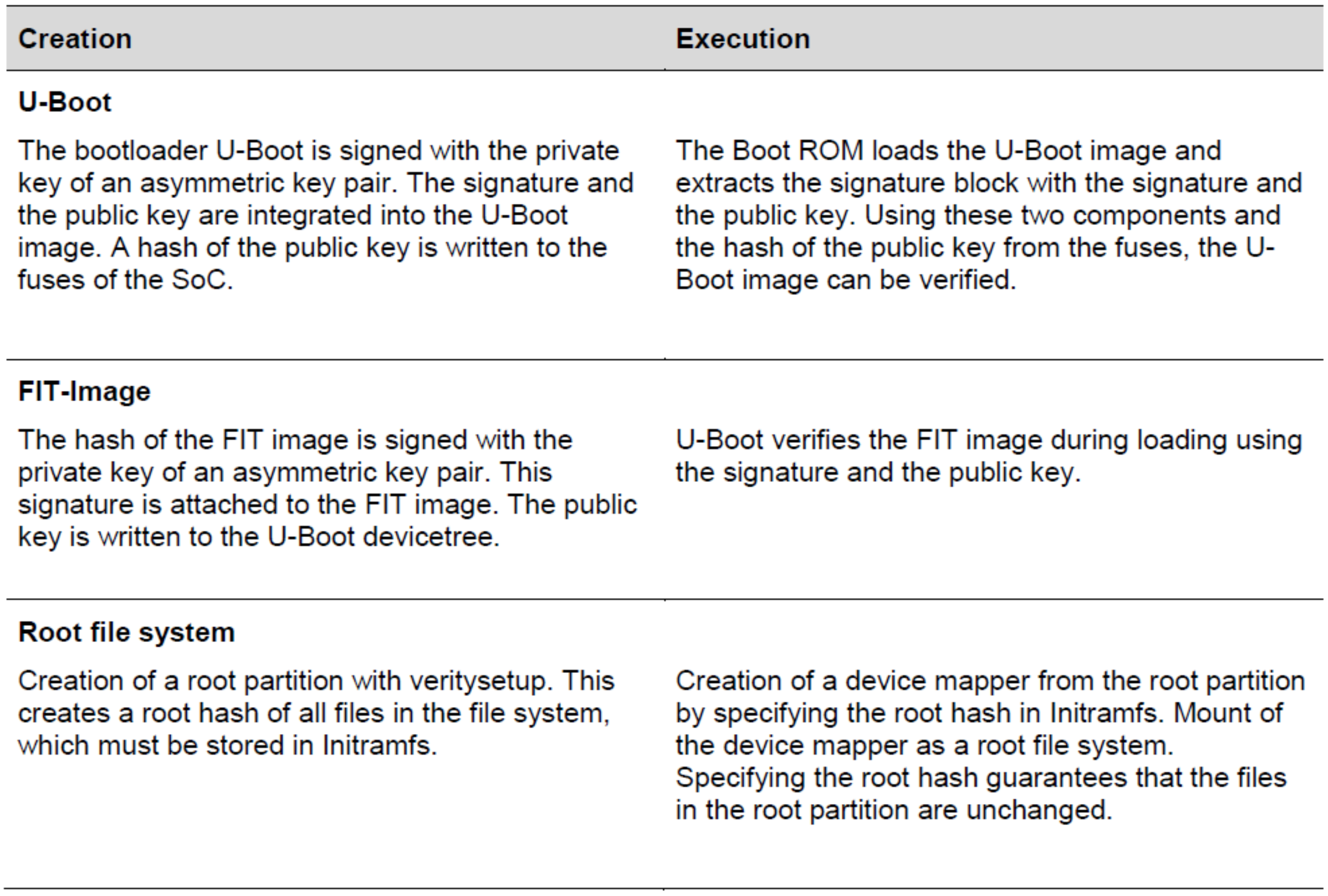
Specifications

- Model: TQMa93xx
- Operating System: Linux (Ubuntu 22.04)
- Security Feature: Secure Boot

ATTENTION: Fuses (One Time Programmable) are set in this How-to, this process is irreversible. It is therefore strongly recommended to use a development pattern for this guide.

Procedure

This guide explains how a chain of trust can be established from the boot loader via the Linux kernel to a root partition with dm-verity.
The following table provides a simplified description of the steps involved in creating the chain of trust and verification during the boot process:



Creation	Execution
U-Boot	
The bootloader U-Boot is signed with the private key of an asymmetric key pair. The signature and the public key are integrated into the U-Boot image. A hash of the public key is written to the fuses of the SoC.	The Boot ROM loads the U-Boot image and extracts the signature block with the signature and the public key. Using these two components and the hash of the public key from the fuses, the U-Boot image can be verified.
FIT-Image	
The hash of the FIT image is signed with the private key of an asymmetric key pair. This signature is attached to the FIT image. The public key is written to the U-Boot devicetree.	U-Boot verifies the FIT image during loading using the signature and the public key.
Root file system	
Creation of a root partition with veritysetup. This creates a root hash of all files in the file system, which must be stored in Initramfs.	Creation of a device mapper from the root partition by specifying the root hash in Initramfs. Mount of the device mapper as a root file system. Specifying the root hash guarantees that the files in the root partition are unchanged.

Preparation

The following projects are required to create a signed boot stream for TQMa93xx:

- imx-mkimage: https://github.com/nxp-imx/imx-mkimage (required)
- NXP Code Signing Tool 3.4.x (with NXP Account): https://www.nxp.com/webapp/Download?colCode=IMX_CST_TOOL_NEW (required)
- TQ Yocto-Workspace: https://github.com/tq-systems/ci-meta-tq (recommended)

The bootstream for TQMa93xx consists of several artifacts. To obtain all these artifacts from the same source, it is recommended to use the TQ Yocto workspace ci-meta-tq. The instructions included there can be followed to build a complete image (tq-image-weston-debug or tq-image-generic-debug) for one of the following TQMa93xx-based devices:

- tqma93xx-mba91xxca.conf
- tqma93xx-mba93xxca.conf
- tqma93xxla-mba93xxla.conf

ATTENTION: To create U-Boot with secure boot functionality (AHAB), the following line must be added to local.conf:
DISTRO_FEATURES:append = ” secure”
Next, the boot stream must be recreated:
$ bitbake imx-boot

The TQ Yocto workspace can also be used to create an image of the complete chain of trust presented here. The settings required for this are described in section 5.2 .
The sources for the Linux kernel and U-Boot are optional but recommended. They can be downloaded from Github:

- Linux: https://github.com/tq-systems/linux-tqmaxx/tree/TQMa-fslc-6.6-2.0.x-imx
- U-Boot: https://github.com/tq-systems/u-boot-tqmaxx/tree/TQM-lf_v2023.04
- Linux and U-Boot should already be compiled for a variant of TQMa93xx in preparation.

U-Boot

Generating keys

Signing and verification of the boot stream are carried out using a public key infrastructure (PKI). If not already available, the Code Signing Tool can be used to create a suitable PKI. The CST 3.4.x is a tar.gz archive that only needs to be unpacked. No further installation is necessary. The following steps can be used to generate the sample keys for this guide:

ATTENTION: Paths are relative to the folder extracted from the archive.

1. Enter the serial number of the first certificate in keys/serial (file must be created):
   12345678
2. Enter the passphrase twice in keys/key_pass.txt (file must be created):
   my_passphrase my_passphrase
3. Create PKI tree:
   $ keys/ahab_pki_tree.sh -existing-ca n -kt ecc -kl p521 -da sha512 -duration 10 -srk-ca n
   For an explanation of the options, please refer to the User Guide contained in the CST

(in the docs subfolder) or the –help option of the above script.

Alternatively, the script can also be called without options and configured in interactive mode.

The script generates keys in keys/ and certificates in crts/.

4. Create SRK table and SRK hash table:

   ```
   $ linux64/bin/srktool -a -s sha512 -d sha256 -t SRK_1_2_3_4_table.bin \
   -e SRK_1_2_3_4_fuse.bin -f 1 -c
   crts/SRK1_sha512_secp521r1_v3_usr_crt.pem,crts/SRK2_sha512_secp521r1_v3_usr_
   pem,crts/SRK3_sha512_secp521r1_v3_usr_crt.pem,crts/SRK4_sha512_secp521r1_v3_
   crt.pem
   ```

5. Write SRK hash table in fuses:

   ATTENTION: This step is only possible once and is irreversible. The following values are only examples and must be replaced by your own values.

   - **Display hashes:**

     ```
     $ hexdump -e '/4 "0x"' -e '/4 "%X""\n"' SRK_1_2_3_4_fuse.bin
     0x00000000
     0x11111111
     0x22222222
     0x33333333
     0x44444444
     0x55555555
     0x66666666
     0x77777777
     ```

   - **Write hashes in fuses (TQMa93xx U-Boot):**

     ```
     => fuse prog 16 0 0x00000000
     => fuse prog 16 1 0x11111111
     => fuse prog 16 2 0x22222222
     => fuse prog 16 3 0x33333333
     => fuse prog 16 4 0x44444444
     => fuse prog 16 5 0x55555555
     => fuse prog 16 6 0x66666666
     => fuse prog 16 7 0x77777777
     ```

Creating a signed boot stream

U-Boot Proper and ATF

1. Copy the required files (successful build of a TQ image, see above, or the U-Boot sources is assumed):
   - ARM Trusted Firmware: ${DEPLOY_DIR_IMAGE}/bl31-imx93.bin, rename to bl31.bin
   - U-Boot Proper:
     ${DEPLOY_DIR_IMAGE}/u-boot.bin
     This file is a link, so copy it with cp —-dereference or display and copy the original file with ls —-long
     or
     from self-compiled U-Boot sources
     These files must be copied to imx-mkimage/iMX9/. imx-mkimage can be obtained from the Github repository mentioned above, no installation is necessary.
2. Build container with U-Boot Proper and ATF (execute in folder imx-mkimage):
   $ make SOC=iMX9 REV=A1 u-boot-atf-container.img include autobuild.mak
   - …
   - CST: CONTAINER 0 offset: 0x0
   - CST: CONTAINER 0: Signature Block: offset is at 0x110
   - Offsets = 0x0 0x110
   - DONE.
   - Note: Please copy image to offset: IVT_OFFSET + IMAGE_OFFSET
     ATTENTION: The offsets for the container and signature block are required in the next step.
   - The artifact imx-mkimage/iMX9/u-boot-atf-container.img must then be copied to the CST folder unpacked in step “3.1 Generating keys”.
3. Transfer offset of container and signature block to Command Sequence File (CSF):
   - [Header]
   - Target = AHAB
   - Version = 1.0
   - [Install SRK]
   - File = “SRK_1_2_3_4_table.bin”
   - Source = “crts/SRK1_sha512_secp521r1_v3_usr_crt.pem”
   - Source index = 0

- Source set = OEM
- Revocations = 0x0
- [Authenticate Data]
- File = “u-boot-atf-container.img”
- Offsets = 0x0 0x110
- CSF based on: https://github.com/nxp-imx/uboot-imx/blob/lf_v2024.04/doc/imx/ahab/csf_examples/csf_uboot_atf.txt
- The CSF is also stored in the CST folder with the name csf_uboot_atf.txt that was unpacked in step “3.1 Generating keys”.

4. Sign container (path relative to the CST folder):
   $ linux64/bin/cst -i csf_uboot_atf.txt -o signed-u-boot-atf-container.img The signed container must then be copied back to imx-mkimage/iMX9/u-boot-atf-container.img. Note the renaming to u-boot-atf-container.img.

Complete bootstream

1. Copy the required files (successful build of a TQ image, see above, or of the U-Boot sources is assumed):
   - Edgelock Secure Enclave Firmware: ${DEPLOY_DIR_IMAGE}/mx93a1-ahab-container.img
   - RAM Firmware: ${DEPLOY_DIR_IMAGE}/lpddr4*.bin
   - U-Boot SPL:
     ${DEPLOY_DIR_IMAGE}/u-boot-spl.bin
     This file is a link, so copy it with cp –-dereference or display and copy the original file with ls –-long
     or
     from self-compiled U-Boot sources
   - Signed container with U-Boot Proper and ATF from step „3.2.1 U-Boot Proper and ATF“
     These files must also be copied to imx-mkimage/iMX9/.
2. Build bootstream
   - $ make -j8 SOC=iMX9 REV=A1 flash_singleboot
     include autobuild.mak
   - …

- CST: CONTAINER 0 offset: 0x400
- CST: CONTAINER 0: Signature Block: offset is at 0x490
- Offsets = 0x400 0x490
- DONE.
- **Note**: Please copy image to offset: IVT_OFFSET + IMAGE_OFFSET append u-boot-atf-container.img at 379 KB, psize=1024
- 1145+0 records in
- 1145+0 records out
- 1172480 bytes (1.2 MB, 1.1 MiB) copied, 0.00266906 s, 439 MB/s
- **ATTENTION:** The offsets for the container and signature block are required in the next step.
  The artifact imx-mkimage/iMX9/flash.bin must then be copied to the CST folder unpacked in step "3.1 Generating keys".

3. Transfer offset of container and signature block to Command Sequence File (CSF):
   - [Header]
   - Target = AHAB
   - Version = 1.0
   - [Install SRK]
   - File = "SRK_1_2_3_4_table.bin"
   - Source = "crts/SRK1_sha512_secp521r1_v3_usr_crt.pem"
   - Source index = 0
   - Source set = OEM
   - Revocations = 0x0
   - [Authenticate Data]
   - File = "flash.bin"
   - Offsets = 0x400 0x490
   - CSF based on: https://github.com/nxp-imx/uboot-imx/blob/lf_v2024.04/doc/imx/ahab/csf_examples/csf_boot_image.txt
   - The CSF is stored in the CST folder with the name csf_boot_image.txt unpacked in step "3.1 Generating keys".
4. **Sign the bootstream**
   - linux64/bin/cst -i csf_boot_image.txt -o signed-flash.bin

The steps for replacing the boot stream can be found in the BSP layer (https://github.com/tq-systems/meta-tq) under meta-tq/doc.

Verification

To check if the signed boot stream is valid, use the ahab_status command in U-Boot:

- => ahab_status
- Lifecycle: 0x00000008, OEM Open
- No Events Found!
- If an event is found, the boot stream is invalid and would not be able to boot on a locked device.

For falsification, an unsigned bootstream can be booted and then ahab_status can be called:

- => ahab_status
- Lifecycle: 0x00000008, OEM Open
- 0x0287fad6
- IPC = MU APD (0x2)
- CMD = ELE_OEM_CNTN_AUTH_REQ (0x87)
- IND = ELE_BAD_KEY_HASH_FAILURE_IND (0xFA)
- STA = ELE_SUCCESS_IND (0xD6)
- 0x0287fad6
- IPC = MU APD (0x2)
- CMD = ELE_OEM_CNTN_AUTH_REQ (0x87)
- IND = ELE_BAD_KEY_HASH_FAILURE_IND (0xFA)
- STA = ELE_SUCCESS_IND (0xD6)

Lock the device

ATTENTION: This step is irreversible and should only be carried out if necessary. If the configuration is incorrect, this step will result in an unusable device.

The device can be locked in the U-Boot with the command ahab_close. This means that only valid boot streams verified by the Boot ROM will boot. The following status is displayed after rebooting:

- => ahab_status
- Lifecycle: 0x00000020, OEM Closed
- No Events Found!

FIT-Image

ATTENTION: Path information is relative to a new, empty folder, e.g. fit_image_work, or the kernel sources, if self-compiled. Hereafter referred to as the working directory.

Generating a key pair

An asymmetric key pair is used to sign the FIT image. Such a pair can be generated with OpenSSL:

```
$ openssl genpkey -algorithm RSA -out dev.key -pkeyopt rsa_keygen_bits:2048
$ openssl req -batch -new -x509 -key dev.key -out dev.crt
```

Create image tree source

- Create image tree source sign.its for the FIT image.
- /dts-v1/;
- / {
- description = “Kernel fitImage for TQMa93xx”;
- #address-cells = <1>;
- images {
- kernel-1 {
- description = “Linux kernel”;
- data = /incbin/(“Image”);
- type = “kernel”;
- arch = “arm64”;
- os = “linux”;
- compression = “gzip”;
- load = <0x90000000>;
- entry = <0x90000000>;
- hash-1 {

- algo = “sha256”;
- };
- };
- fdt-1 {
- description = “Flattened Device Tree blob”;
- data = /incbin/(“<path/to/Devicetree.dtb>”);
- type = “flat_dt”;
- arch = “arm64”;
- compression = “none”;
- load = <0x97000000>;
- hash-1 {
- algo = “sha256”;
- };
- };
- };
- configurations {
- default = “conf-1”;
- conf-1 {
- description = “Linux kernel, FDT blob”;
- kernel = “kernel-1”;
- fdt = “fdt-1”;
- hash-1 {
- algo = “sha256”;
- };
- signature-1 {
- algo = “sha256,rsa2048”;
- key-name-hint = “dev”;
- padding = “pkcs-1.5”;
- sign-images = “kernel”, “fdt”;
- };
- };
- };
- };

Creating a signed FIT image

Note: The devicetree binary for U-Boot is required for this step. Ready-made devicetree binaries can be found in the Yocto workspace in the U-Boot build directory. The path to the build directory can be displayed with bitbake virtual/bootloader –e | grep ^B=.

1. Copy the required files into the working directory:
    - Rename U-Boot devicetree imx93-tqma9352-mba91xxca.dtb, imx93-tqma9352-mba93xxca.dtb or imx93-tqma9352-mba93xxla.dtb, in pubkey.dtb:
      From U-Boot build directory in Yocto workspace (path: bitbake virtual/bootloader –e | grep ^B=)
      or
      from self-compiled U-Boot sources
    - Linux-Kernel:
      ${DEPLOY_DIR_IMAGE}/Image
      This file is a link, so copy it with cp –-dereference or display and copy the original file with ls –-long
      or
      from self-compiled Linux sources
    - Linux devicetree:
      Depending on the variant ${DEPLOY_DIR_IMAGE}/imx93-tqma93…
      This file is a link, so copy it with cp –-dereference or display and copy the original file with ls –-long
      or
      from self-compiled Linux sources
    - The keys generated in step 4.1
    - The ITS file generated in step 4.2
2. Create FIT image with signature
   ```
   $ mkimage -f sign.its -K pubkey.dtb -k . -r image.itb
   ```
   The public key is written to the devicetree of the U-Boot. This key is used to verify the FIT image signed above.

ATTENTION: To pack the U-Boot devicetree with the public key into the signed bootstream from chapter 3.2, the steps from chapter 3.2 must be repeated with a

customized U-Boot Proper u-boot.bin. To do this, the devcietree with the public key pubkey.dtb must be specified via the EXT_DTB option when compiling the U-Boot:
make EXT_DTB=<Pfad/zu/pubkey.dtb>

Verification
In U-Boot with public keys, the signed FIT image image.itb can be booted with bootm after it has been loaded from a suitable medium (TFTP, eMMC, SD).
When booting the FIT image, U-Boot returns the information Verifying Hash Integrity … sha256,rsa2048:dev+ OK with name, algorithm and length of the key generated in chapter 4.1 on the console:

- ## Loading kernel from FIT Image at 80400000 …
  …
- Verifying Hash Integrity … sha256,rsa2048:dev+ OK
  …
- ## Loading ramdisk from FIT Image at 80400000 …
  …
- Verifying Hash Integrity … sha256,rsa2048:dev+ OK
  …
- ## Loading fdt from FIT Image at 80400000 …
  …
- Verifying Hash Integrity … sha256,rsa2048:dev+ OK
  …

For falsification, another key pair can be generated as described in section 4.1 and used to sign the FIT image. This FIT image cannot be booted without exchanging the key in the U-Boot Devicetree:
Loading kernel from FIT Image at 80400000 …

Using ‘conf-1’ configuration
Verifying Hash Integrity … sha256,rsa2048:test- error!
Verification failed for ‘<NULL>’ hash node in ‘conf-1’ config node
Failed to verify required signature ‘key-dev’
Bad Data Hash
ERROR: can’t get kernel image!

Extend Chain of Trust: root partition

The previously established chain of trust verifies the origin of the U-Boot and Linux kernel. With the mechanisms mentioned above, only the owner of the generated private key can sign his software and boot it on the device. Further links can be added to the chain. The following section outlines how the root partition can be protected against manipulation using dm-verity. For the real implementation, it is also shown how the complete chain can be created with the TQ-BSP. A step-by-step guide to dm-verity protection is omitted due to the complexity of the requirements.

Sketch: Verity Devicemapper

1. **Generate Verity hashes:**
   veritysetup calculates the hash values and stores them at the end of the root partition. The root partition can be a real file or a block device file (e.g. /dev/sdaX).
   - veritysetup \
   - –data-block-size=1024 \
   - –hash-block-size=4096 \
   - –hash-offset=<Offset> \
   - format \
   - <Root-Partition.img> \
   - <Root-Partition.img>
   - veritysetup outputs the following information (with correspondingly different values):
   - VERITY header information for data.img
   - UUID: e06ff4cb-6b56-4ad4-bd97-0104505a70a5
   - Hash type: 1
   - Data blocks: 204800
   - Data block size: 1024
   - Hash block size: 4096
   - Hash algorithm: sha256
   - Salt: 17328c48990b76fbb3e05d0ebfd236043674cf0d14c278bc875b42693621cc21
   - Root hash:

a0e1a449d452f74d041706b955794c0041e3d8ad051068df6589e08485323698

- The root hash is the sensitive value that needs to be protected. If this hash is compromised, e.g. if it can be changed by an unauthorized person, then the protection of the integrity of the root partition by dm-verity is worthless.

2. **Integrate the root hash into the chain of trust**
    - The root hash generated above is stored in the signed FIT image, which protects it against manipulation. For this purpose, an initramfs is added to the FIT image in which the root hash is stored in a file.
    - The images node of the ITS file from chapter 4.2 is extended by the following section, among others:
    - ramdisk-1 {
    - description = “dm-verity-image-initramfs”;
    - data = /incbin/(“<path/to/Initramfs.cpio.gz>”);
    - type = “ramdisk”;
    - arch = “arm64”;
    - os = “linux”;
    - compression = “none”;
    - load = <0x98000000>;
    - entry = <0x98000000>;
    - hash-1 {
    - algo = “sha256”;
    - };
    - };
3. Check the integrity of the root partition
    - The initramfs contains a suitable script that generates a device mapper from the root partition and the root hash.
    - veritysetup \
    - –data-block-size=${DATA_BLOCK_SIZE} \
    - –hash-offset=${DATA_SIZE} \
    - create rootfs \
    - </dev/Root-Parititition> \
    - </dev/Root-Parititition> \
    - <Root Hash>

The device mapper is then mounted:

- mount \
- -o ro \
- /dev/mapper/rootfs \
- /rootfs

The root filesystem is read-only. To switch to the actual root filesystem, use switch-root.

Automated creation with TQ-BSP

In principle, an image with a chain of trust from the boot loader to the root partition can be created automatically with the TQ-BSP.
For TQMa93xx the following options have to be added to local.conf :

- # The DISTRO_FEATURE secure necessary config options for U-Boot and Kernel
- DISTRO_FEATURES:append = " secure"
- # Name of the key used for signing the bootloader
- IMX_HAB_KEY_NAME = "ahab"
- # Activates the signing of the FIT image in the build process
- UBOOT_SIGN_ENABLE = "1"
- # This class contains the logic for creating a protected root partition
- IMAGE_CLASSES += "dm-verity-img"
- # Name of the initramfs image for dm-verity handling
- INITRAMFS_IMAGE = "dm-verity-image-initramfs"
- # Initramfs is stored as a separate artifact in the image
- INITRAMFS_IMAGE_BUNDLE = "0"
- # Store FIT image with initramfs in boot partition
- IMAGE_BOOT_FILES:append = " fitImage-${INITRAMFS_IMAGE}-${MACHINE}-
- ${MACHINE};fitImage" # Image to be protected with dm-verity
- # Alternative: tq-image-weston-debug
- DM_VERITY_IMAGE = "tq-image-generic-debug"
- # Type oft he above image
- DM_VERITY_IMAGE_TYPE = "ext4"

ATTENTION: The exact options may change in future versions of the BSP. The latest information can be found in the BSP layer documentation (https://github.com/tq-systems/meta-tq) under meta-tq/doc.

The complete image is created with bitbake tq-image-generic-debug and can then be written to an SD card, for example.

Verification
In Linux, mount -a can be used to check if the Verity Devicemapper is mounted as the root filesystem:

- # mount
- …
- /dev/mapper/rootfs on / type ext4 (ro,relatime)
- …
- In addition, the entire root file system is read-only in this case:
- # touch test
- touch: cannot touch ‘test’: Read-only file system

For falsification, the root file system can be modified offline and the device rebooted. The modification causes a different root hash and the boot process is aborted: device-mapper: verity: 179:98: data block 1 is corrupted

More information about the TQMa93xx can be found in the TQ Support Wiki:
https://support.tq-group.com/en/arm/modules#nxp_imx_9_series

TQ-Systems GmbH
Mühlstraße 2 I Gut Delling I 82229 Seefeld Info@TQ-Group | TQ-Group

Frequently Asked Questions

Q: Is it possible to reverse the irreversible process of setting Fuses mentioned in the How-to?
A: No, setting Fuses is irreversible. It is recommended to use a development pattern.

Q: Where can I find the required sources for Linux and U-Boot?

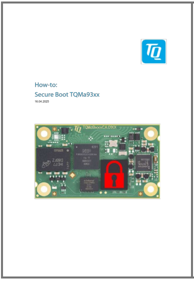
A: Linux:

Linux Repository

U-Boot: U-Boot

Repository

Documents / Resources

	TQ TQMa93 Secure Boot [pdf] User Guide TQMa93xx, TQMa93 Secure Boot, Secure Boot, Boot

References

- User Manual

Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search:

e.g. whirlpool wrf535swhz Search