BCM63XX/BCM68XX/BCM49XX

BCA Linux Secure Boot Image Toolkit Support

Application Note

Table of Contents

Chapter 1: Introduction

Implementing secure boot with the BCM63XX/BCM68XX/BCM49XX devices that support secure boot requires building, signing, and assembling highly customized images. For devices that support "Gen3" secure boot, starting with the 5.02L.05 SDK, some features are available that will help customers sign and package their images after they have completed building all the components of their code.

Secure boot is almost always associated with a read-only root file system and is not a field upgrade from non-secure to secure. So, everything described herein will assume the use of SQUASHFS for the root file system and the "puresquibi" flash layout for NAND. As eMMC support is added, eMMC layouts with a SQUASHFS root file system are nearly identical to puresqubi except that the UBI volumes are replaced with eMMC partitions.

Disclaimer
The components described herein are expected to help customers more quickly deploy and maintain secure boot software releases. Ultimately, each customer will still need to take additional steps to lock down their systems and customize additional pieces of the system. By providing some components, this should reduce the implementation and maintenance effort required, but does not reduce the critical need for the customers' developers to fully understand the mechanisms at play and how they interact.

Chapter 2: New Menuconfig Items

In order to use these mechanisms, CFEROM and CFERAM must be conditionally compiled with flags passed down from the build PROFILE. In menuconfig, set:

```
Bootloader and Secure Boot Options  --->
    Build Bootloader With Linux (NEW)
```

Having done this, new options will become enabled under the next menu:

```
Security Features  --->
    DEBUG (Dangerous) Open JTAG on Key Abort (NEW)
    Support signed hash block in secure modes (NEW)
    Drop security privileges to non-secure mode when entering CFE console in MFG mode (NEW)
    Drop security privileges to non-secure mode when entering CFE console in FLD mode (NEW)
```

2.1 Open JTAG on Key Abort

During development, it is not uncommon to “brick” boards. Since secure boot normally disabled the JTAG interface, this can make recovery difficult. So, this flag will enable JTAG in CFEROM even in secure modes and it is vital that this option never be enabled in actual production releases.

2.2 Support Signed Hash Block in Secure Modes

When creating flash images, every component (CFERAM, Kernel, DeviceTree, Rootfs, and potentially others) needs to be authenticated before use. In order to avoid the need for a plethora of signatures (which can be difficult in environments where signing keys are tightly controlled), and to prevent unauthorized mixing/matching of components, the system can include a block of SHA256 hashes for all of the components that it signed with a single RSA operation.

2.3 Drop Security Privileges to Non-Secure Mode When Entering CFE Console in MFG/FLD Mode

In the CFE console, the user has access to commands that can manipulate both the credentials stored in memory and the contents of flash. In either MFG or FLD secure modes, it may be beneficial to restrict user access in the CFE console in order to prevent unauthorized access. These two flags will enable the following:

- The CPU privilege level is dropped to non-secure. This prevents access to secure peripherals.
- All credentials are scrubbed.
- CFE console commands are restricted to ones that provide image update functionality.

Chapter 3: Image Assembly

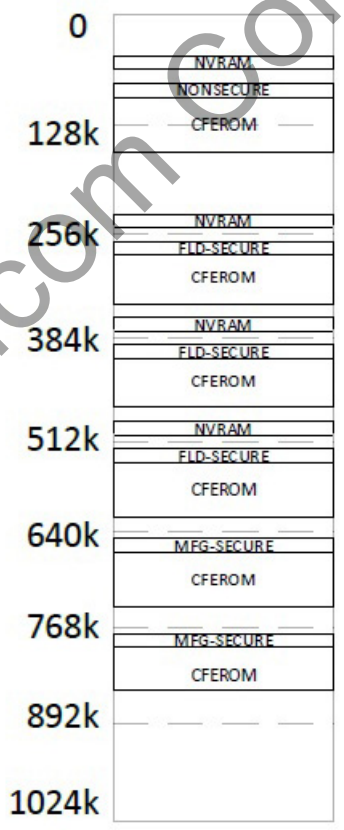
3.1 CFEROM Region Format

In all of Gen2 and Gen3 secure boot devices, the boot ROM searches the first 1M of flash one 1K boundaries for CFEROM headers. In reference software builds, this region is normally populated with multiple redundant copies of CFEROM (with its headers) and NVRAM (preceded by its magic number).

When deploying secure boot, the layout of this region can require quite a bit of customization.

- The developer may elect to combine CFEROM images that operate in non-secure, mfg-secure, or field-secure modes into the same image.
- Any images that are needed after manufacturing and are not contained exclusively in erase block 0 will need to be stored redundantly in at least two blocks.
- It is advised to maintain multiple copies of NVRAM.
- It is advised to minimize the extent to which any given copy straddles multiple erase blocks, as that increases the likelihood that a given copy will encounter a bad block.
- It is important to have redundant copies avoid sharing an erase block, as there is a moment during programming where an entire block is erased and has not yet been rewritten.

Figure 1: Sample CFEROM Region Layout for 128 KB Block Size NAND Device



3.2 Upgrade Images (No CFEROM Included) Format

File system images consist of several major components.

- The root file system (SQUASHFS) image.
- “Bootfs”, with CFERAM, kernel, dtb(s), sometimes a TEE, and so on, which is stored in a TLV-like structure rather than a file system.
- Metadata (placeholders for metadata that is updated for image management).
- Image tag containing the CRC of the image and the image type.

In the case of secure boot, the bootfs is augmented by a block of hashes (called “hashes.fld” for field mode, “hashes.mfg” for mfg mode, and “hashes.bin” for non-secure mode integrity check). In the secure boot mode, this block of hashes has a signed sha256 digest prepended to it.

Once these components are available, the bootfs is packed up into its TLV-like structure (using a script, “mkfs.nada”) and all of the components are ubinized into a UBI image and then the tag is appended.

3.3 Upgrade Image (With CFEROM) Format

The format for upgrade images with CFEROM is very simple. The 1 MB “region” with CFEROM and NVRAM images is followed by the ubinized upgrade image and a tag.

Chapter 4: Building the SDK and Custom Image

In the top SDK directory, there is a new example makefile called “make.image”. After building an image with the signed hash block enabled, all of the major components needed for image assembly are sitting in various locations under targets/PROFILENAME/….

To build images, run:

```
make -f make.image
```

This will assemble the custom NAND images and leave them in targets/PROFILENAME/imagebuild/NAND/…

To further customize, make a private copy of make.image and give it a new name. Then, go through the steps of creating a custom image.

NOTE: If your private keys are not available on your build machine, you may have to replace OpenSSL commands with equivalent operations that interact with a machine that does have access to the keys, OR you may need to break make.image into three sections. The first to prepare for signature, the second to actually sign (potentially on another machine), and the third to take the signed items and package them into the flash formats.

The first thing to do in any of these makefiles is to include make.common with all of the necessary PATHS and variables defined:

```
image: nand_pureubi

BUILD_DIR = $(shell pwd)
include $(BUILD_DIR)/make.common

#
```

The first actual step is to create a directory in which to work and a directory to contain the “bootfs” files. In this example, the files being placed in that bootfs are simply the bootfs files from the just-complete build along with the correct CFERAM for NAND. If there are additional items going into the bootfs, they should be added here.

```
nand_prepare:
ifneq ($(strip $(BUILD_NAND_IMG_BLKSIZE)),)
    mkdir -p $(PROFILE_DIR)/imagebuild/NAND/head
    mkdir -p $(PROFILE_DIR)/imagebuild/NAND/bootfs
    cp $(CFE_ROM_FILE) $(PROFILE_DIR)/imagebuild/NAND/head/
    cp -r $(TARGET_BOOTFS)/* $(PROFILE_DIR)/imagebuild/NAND/bootfs
    cp $(CFE_RAM_FILE) $(PROFILE_DIR)/imagebuild/NAND/bootfs/cferam.000
ifneq ($(wildcard $(BUILD_DIR)/secureos/optee/optee.lz),)
    cp -f $(BUILD_DIR)/secureos/optee/*.lz $(PROFILE_DIR)/imagebuild/NAND/bootfs
    cp -f $(BUILD_DIR)/secureos/optee/devicetree/*.dts $(PROFILE_DIR)/imagebuild/NAND/bootfs
endif
endif
```

With the bootfs populated, we call mkhashes to generate a hash of everything in bootfs (except for hashes.bin and hashes.fld themselves if present) and call the resulting file hashes.bin. This hashes.bin specifies that cferam.000 is the image that CFEROM should launch (the --file specification followed by --boot). In this example, hashes.bin will be used for non-secure validation AND a signed copy will be used in secure mode, and both will specify cferam.000 as the next stage. If different modes required different versions of CFERAM, each mode could start with its own hashes file.

```
nand_pureubi: nand_prepare
    # This demonstrates how to finalize an image.   Here, one would insert any additional components
    to be included in the signature block for the boot filesystem
    # Create the block of hashes
    hostTools/imagetools/mkhashes --exclude='bootfs\/hashes.bin$$'  --exclude='bootfs\
       /hashes.fld$$' --out=$(PROFILE_DIR)/imagebuild/NAND/bootfs/hashes.bin --item
       rootfs=$(PROFILE_DIR)/squashfs.img    --file cferam.000 --boot $(PROFILE_DIR)/
       imagebuild/NAND/bootfs
    #
    #
```

Preparing the first 1 MB requires starting with a 1 MB empty file. In this case, we will insert a few copies of CFEROM (non secure). This is not a perfect example because they both depend on the same erase block being okay.

```
ifeq ($(strip $(SECURE_BOOT_ARCH)),GEN3)
    #
    # build UBI cferom
    # start with a 1meg empty region
    hostTools/imagetools/gen1meg $(PROFILE_DIR)/imagebuild/NAND/region
    #
    # and put a few copies of cferom in flash with nonsecure headers
    hostTools/imagetools/insertboot --arch $(SECURE_BOOT_ARCH) --cfe $(CFE_ROM_FILE) --nonsec --
       chip=$(BRCM_CHIP) --offset 71680  $(PROFILE_DIR)/imagebuild/NAND/region
    hostTools/imagetools/insertboot --arch $(SECURE_BOOT_ARCH) --cfe $(CFE_ROM_FILE) --nonsec --
       chip=$(BRCM_CHIP) --offset 196608  $(PROFILE_DIR)/imagebuild/NAND/region
    #
```

Createimg will insert multiple copies of NVRAM in the image. It accepts a number of options, not demonstrated here, to pre-populate NVRAM fields.

```
# insert a few copies of an essentially empty nvram
hostTools/createimg.pl --replace --offsets=32768,192512 --input=$(PROFILE_DIR)/imagebuild/NAND/
    region --outputfile=$(PROFILE_DIR)/imagebuild/NAND/region --nvram_magic
#
```

If building for secure boot, we need to prepare the items to be signed. That requires the authenticated header (containing the public credentials) AND the CFEROM itself. This call to insert boot is only asking for the header to be generated and placed in a file:

```
ifneq ($(strip $(BUILD_SECURE_BOOT)),)
    #
    # now let's make signed images
    #
    # First, generate the auth header for CFEROM
    #
    hostTools/imagetools/insertboot --arch $(SECURE_BOOT_ARCH) --cfe $(CFE_ROM_FILE) --field  --
       cred=targets/keys/demo/$(SECURE_BOOT_ARCH)/ --chip=$(BRCM_CHIP) --out=$(PROFILE_DIR)/
       imagebuild/NAND/auth_header
    #
    # Get rid of the non-hash signature
    rm -f $(PROFILE_DIR)/imagebuild/NAND/bootfs/vmlinux.sig
    #
```

Next, the header, concatenated with the CFEROM, needs to be signed with the private key. This is done locally here, but the auth_header and CFEROM could be taken to another machine, the OpenSSL commands or their equivalent run without the rest of the build environment, and then the signature (in a file, auth_sig) returned. This entire operation can be skipped if only upgrade images are being created or the same signed CFEROM is being reused from a previous signing.

```
# now concatenate auth_header and CFEROM and sign the resulting digest > auth_sig
# this is done with local openssl commands here, but could be done elsewhere
#
cat $(PROFILE_DIR)/imagebuild/NAND/auth_header $(CFE_ROM_FILE) \
    | openssl dgst -sign targets/keys/demo/$(SECURE_BOOT_ARCH)/Krot-fld.pem -keyform pem -
    sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1 -out auth_sig
```

For an upgrade image, the one component that needs a cryptographic signature is the block of hashes. The unsigned block of hashes has its own digest signed with the private key and the signature is PREpended to the hash block. The resulting signed hash block is given the appropriate name (“hashes.fld” in this case).

```
# prepend the signature to the hashes
cat  $(PROFILE_DIR)/imagebuild/NAND/bootfs/hashes.bin \
    | openssl dgst -sign targets/keys/demo/$(SECURE_BOOT_ARCH)/Krot-fld.pem -keyform pem -
    sha256 -sigopt rsa_padding_mode:pss -sigopt rsa_pss_saltlen:-1 -out  $(PROFILE_DIR)/
    imagebuild/NAND/hashes.sig
cat $(PROFILE_DIR)/imagebuild/NAND/hashes.sig $(PROFILE_DIR)/imagebuild/NAND/bootfs/
    hashes.bin > $(PROFILE_DIR)/imagebuild/NAND/bootfs/hashes.fld
#
#
```

At this point, all the components are ready and the signatures and signed hashes are available. If the signing happened on another machine, the signature and signed hashes would be brought back and the remaining portion of the build would continue.

The CFEROM file that was signed (could be the same or different than the one used for non-secure) can be inserted along with its header (from the public keys) and the signature (from a file):

```
    # insert 2 copies of the signed cferom into flash images
    hostTools/imagetools/insertboot --arch $(SECURE_BOOT_ARCH) --cfe $(CFE_ROM_FILE) --
       field=auth_sig  --cred=targets/keys/demo/$(SECURE_BOOT_ARCH)/ --chip=$(BRCM_CHIP) --offset
       262144 $(PROFILE_DIR)/imagebuild/NAND/region
    hostTools/imagetools/insertboot --arch $(SECURE_BOOT_ARCH) --cfe $(CFE_ROM_FILE) --
       field=auth_sig  --cred=targets/keys/demo/$(SECURE_BOOT_ARCH)/ --chip=$(BRCM_CHIP) --offset
       524288 $(PROFILE_DIR)/imagebuild/NAND/region
    #
endif
Endif
```

Now that the 1 MB “region” is complete, pack the signed hashes file, if present, along with all the other bootfs components into the “blob.bin” file:

```
# Create the pureubi "blob.bin" file with cferam, vmlinux, dtbs, etc...
hostTools/imagetools/mkfs.nada --out=$(PROFILE_DIR)/imagebuild/NAND/head/blob.bin $(PROFILE_DIR)/
    imagebuild/NAND/bootfs
```

The metadata placeholders are the same format. There is probably never a reason to change the following:

```
# Then create the metadata for the image
hostTools/imagetools/mkfs.nada --out=$(PROFILE_DIR)/imagebuild/NAND/head/meta.bin --extra
    cferam.000=998 --extra squash=1 --extra committed=0
```

The ubinized image will be created by the ubinize utility, but it needs a config file. So, first mkubi_ini creates the ubi.ini file, then ubinize follows the instructions in that file to build the ubinized image. Ubinize needs to know the flash geometry.

```
# generate a ubi.ini file combining the rootfs, metadata, and boot volumes
hostTools/imagetools/mkubi_ini --meta=$(PROFILE_DIR)/imagebuild/NAND/head/meta.bin  --
    boot=$(PROFILE_DIR)/imagebuild/NAND/head/blob.bin --root=$(PROFILE_DIR)/squashfs.img >
    ubi.ini
# Then ubinize it
hostTools/mtd-utils*/ubinize -v -o $(PROFILE_DIR)/imagebuild/NAND/my_rootfs128kb_puresqubi.img -m
    2048 -p $(FLASH_NAND_BLOCK_128KB) ubi.ini
```

Finally, we add the image tag identifying the type of flash image and the device for which the image was built and having a CRC to protect against transport errors.

```
# Finally, add the image tag -- hardcoded to 128K block size for the moment
hostTools/addvtoken --endian $(ARCH_ENDIAN) --chip $(BRCM_CHIP) --flashtype NAND128 --btrm 1
    $(PROFILE_DIR)/imagebuild/NAND/my_rootfs128kb_puresqubi.img $(PROFILE_DIR)/imagebuild/NAND/
    custom_$(PROFILE)_puresqubi_128.w
cat $(PROFILE_DIR)/imagebuild/NAND/region $(PROFILE_DIR)/imagebuild/NAND/
    my_rootfs128kb_puresqubi.img > $(PROFILE_DIR)/imagebuild/NAND/
    my_cferom_rootfs128kb_puresqubi.img
hostTools/addvtoken --endian $(ARCH_ENDIAN) --chip $(BRCM_CHIP) --flashtype NAND128 --btrm 1
    $(PROFILE_DIR)/imagebuild/NAND/my_cferom_rootfs128kb_puresqubi.img $(PROFILE_DIR)/imagebuild/
    NAND/custom_$(PROFILE)_cferom_puresqubi_128.w
```

Revision History

63XX-68XX-49XX-AN100-R, April 26, 2018

- Initial release