Android Image Signing

Broadcom Proprietary and Confidential

Release Note: This document can be shared with customers.

ais - Android Image Signing

This document describes how one can customize the keys and signatures used during the standard android build process for a device.

Note that those processes described here are distinct from the signing of images for the purpose of secure|verified boot. The signing of images for those processes is described in the Android Verified Boot document.

The keys and signing rights described in this document are related to:

- the android platform key(s) used to sign applications.
- the android dm-verity key used to sign verity aware images (system, vendor, etc...).

Big Warning

None of the key(s) provided by Broadcom reference implementation should ever be used in a final product image; the keys are provided as a mean to show how one can customize the android environment.

Customers are responsible for generating and adapting their own key(s) based on the information provided here.

Platform Keys

Standard Android Keys

The default keys used by the android platform during standard build process are located here:

<aosp>/build/make/target/product/security

Those keys are considered test-keys and so generally are required to be updated for a device. The key used by the environment to sign various part of it are:

- media
- platform
- shared
- test

When producing a device specific set of equivalent keys, one need to create all of those.

For the broadcom reference device, we provide a set of keys equivalent available in the following directory:

<aosp>/vendor/broadcom/bcm_platform/signing

All the keys generated for the device are using the following command, password is left blank:

```
make_key \
<name> \
'/C=CA/ST=BritishColumbia/L=Vancouver/O=BcmSTB/OU=BcmSTB/CN=BcmSTB/emailAddress=pierre
@broadcom.com' rsa
```

Where <name> is one of the 4 named key listed prior. the "make_key" command is provided by android under: "development/tools/make_key".

During a device profile build, the usage of those keys is triggered by setting the device product variable:

```
PRODUCT_DEFAULT_DEV_CERTIFICATE := vendor/broadcom/bcm_platform/signing/testkey
```

Bcmstb Key

The bcmstb key, also present and generated in the same directory location and manner as listed in the prior section for the broadcom reference keys, is a special key created to allow signing of test applications or executables which require special privileges to reach the core middle ware (i.e. nexus) functionalities.

The bcmstb key is associated with the "bcmstb_app.se" policy content which allows to tailor access to nexus middle ware functions through selinux tuning.

The bcmstb key is merely a development key but it does highlight the steps that could be taken for creating a device specific key, signing applications with it and providing an associated custom selinux policy for those applications. The interest of such is that one do not need to use the existing android platform keys and so do not need to change the default intended android platform policies associated with such signing; instead we create a restricted sandbox for the sake of custom applications with special needs.

It is understood that such key is not meant to be used for a product image, however following the mechanism described here allows you to create an equivalent key and associate it with selinux policy in a similar manner.

step-1: sign your application with the bcmstb key: simply add this LOCAL_CERTIFICATE to your Android.mk

```
LOCAL_CERTIFICATE := $(BCM_VENDOR_STB_ROOT)/bcm_platform/signing/bcmstb
```

step-2: hook up the bcmstb certificate with the selinux knowledge

Ensure that the key is defined in the following modules (follow examples from "device/broadcom/common/sepolicy"):

- keys.conf
- seapp_contexts
- mac_permissions.xml

step-3: create a .te policy module associated with your signing certificate

This is the bcmstb_app.te module reference equivalent. It creates a context to be associated with applications signed with the bcmstb (or equivalent) key. Because of the hook up from step-2, selinux would automatically apply this configuration for those signed applications.

step-4: update other .te policy to make use of the new context

Finally, you may need to update other policy modules to also hook up properly with your newly created context.

dm-verity Key

This is the key used during the dm-verity process applied to the selected partitions (e.g.: system, vendor).

- For a|b mode devices which mount the system.img as rootfs, the key is also integrated in the kernel keychain by the build automatically in order to allow the kernel verity driver to authenticate the rootfs properly.
- For device which mount verity partitions via the fs_mgr (standard android mount from within init), the verity key is also integrated on the android rootfs via the verity_key public certificate. This applies to non a|b mode devices which enable dm-verity or to a|b mode devices which mount other dm-verity partitions (such as the 'vendor.img').

The key location in the android tree and in the vendor reference is:

<aosp>/build/make/target/product/security

The key is also generated in the same manner as the other platform keys as far as the broadcom reference is concerned and stored here:

<aosp>/vendor/broadcom/bcm_platform/signing

In order to fully integrate the new key, additional steps are documented here.

step-1:produce a .der formatted certificate for inclusion within kernel key ring (a|b mode devices mounting system.img as rootfs)

Produce a "der" formatted certificate which is the one added to the kernel keyring during the build of the kernel image. The following command can be used to generate the "der" certificate from the "pem" one which is the

certificate created by the make_key tool:

```
openssl x509 -in verity.x509.pem -outform der -out verity_dev_key.der.x509
```

step-2:produce a verity_key to add to the rootfs

The "verity_key" to be added to the root file system can be produced using a tool byproduct of the android build. This produces a .pub certificate from the prior generate .pem and that certificate is included on the rootfs by the build. The tool is found on the host applications when building android (note we use the [-convert] option to the tool since we provide the key input):

```
./out/host/linux-x86/bin/generate_verity_key -convert \
vendor/broadcom/bcm_platform/signing/verity.x509.pem \
vendor/broadcom/bcm_platform/signing/verity.key
```

step-3:putting it all together in the device setup

Finally, update the device configuration to ensure the build system knows to use your device key rather than the default android one, there are two things:

1. tell android build to use the device key instead of the default one.
2. copy the "verity_key" public certificate to the rootfs to allow android build to package it properly.

```
PRODUCT_VERITY_SIGNING_KEY      := vendor/broadcom/bcm_platform/signing/verity
PRODUCT_COPY_FILES              +=
vendor/broadcom/bcm_platform/signing/verity.key.pub:root/verity_key
```

Integration on Kernel Version 4.9 Onward

In kernel version 4.9 onward, the key ring is populated using a single "pem" formatted x509 certificate list which aggregates all the certificate we want the kernel to load.

This is different from prior version of the kernel (e.g. 4.1) where the kernel build would scan for existence of "pem" files at the root of the linux tree and pull them together during build on its own.

The Broadcom reference build would provide the kernel with the proper settings to know how to deal with the key ring integration, however because of the requirements to have a single aggregated certificate in kernel 4.9, care must be taken that the dm-verity certificate must be included as part of the unique file alongside other desired certificates.

An example of such is found under:

<aosp>/device/broadcom/elfin/signing/keys.pem.x509

In this example, the aggregated certificate contains 3 entries:

1. Google private certificate used for the device when google signs images for certification.

2. Android's default dm-verity certificate provided under the build tree.

3. Broadcom's private dm-verity certificate.