Understanding Verified Uboot on Embedded Board

Ask Question

Asked 7 years, 4 months ago Modified 6 years, 7 months ago Viewed 4k times

1

I've scoured the presentations and documentation for verified u-boot and have several questions. I'll try to walk any users through where I am as I suspect I am not the only one who is having some slight difficulty understanding the process for verified u-boot.

I have a compiled zImage that has a working external DTB for use without verification. It boots and works (let's call this normal-board.dts)

Secondly, I have u-boot compiled with the following config entries:

```
CONFIG_ARM=y
CONFIG_ARCH_AT91=y
CONFIG_TARGET_AT91SAM9260EK=y
CONFIG_SYS_EXTRA_OPTIONS="AT91SAM9G20,SYS_USE_DATAFLASH_CS1"
CONFIG_SYS_PROMPT="#> "
# CONFIG_CMD_BDI is not set
CONFIG_CMD_IMI=y
# CONFIG_CMD_IMLS is not set
# CONFIG_CMD_LOADS is not set
# CONFIG_CMD_FPGA is not set
# CONFIG_CMD_SOURCE is not set
CONFIG_CMD_SETEXPR=y
CONFIG_DEFAULT_DEVICE_TREE="myboard"
CONFIG_CMD_MMC=y
CONFIG_CMD_FAT=y
CONFIG_MTD_CMDLINE_PARTS=y
CONFIG_RSA=y
CONFIG_FIT=y
CONFIG_FIT_SIGNATURE=y
CONFIG_OF_CONTROL=y
```

My board has a partition scheme similar to:

```
... boot strap, uboot and env
0xD00084000  (zImage)
0xD0020AA00  (normal-board.dtb)
The rootfs is on NAND (external to this chip)
```

The device can be booted in a standard configuration using a command such as:

```
cp.b 0xD0084000 0x22000000 0x186A00;cp.b 0xD020AA00 0x28000000 0x61A8;bootm 0x220
```

At this point, I've recompiled u-boot, but the nomenclature is a bit confusing as there are several elements.

- FIT Control DTS (I assume that this is one used by u-boot and needs to be in its own partition)
- FIT DTB (the same DTB more or less as the non-FIT one (normal-board.dtb), but with FIT magic somewhere in it)
- FIT kernel image (I assume that some magic gets added to the zImage here too?)

Having seen that there is a uboot control FIT FDT, will this need its own partition? and will the FIT DTB be the same as the working kernel DTB (just flash this instead of the non-FIT one)???

Next, given this script I started hashing out from various documentation and slides, we can see that u-boot.{dts,dtb} is the control FDT, and the ITS file is the one with the fit (I assume that its the same as normal-board.dts, BUT has a FIT node added).

Eg. u-boot.dts

```
/dts-v1/;

/ {
        model = "Keys";
        compatible = "myboard";
        signature {
                dev_key {
                        required = "conf";
                        algo = "sha1,rsa2048";
                        key-name-hint = "dev_key";
                };
        };
};
```

Now the example DTS for myboard WITH THE FIT section:

```
/dts-v1/;

/ {
    description = "Linux kernel2";
    #address-cells = <1>;
    images {
        kernel@1 {
            description = "Linux kernel";
            data = /incbin/("../linux/arch/arm/boot/zImage");
            arch = "arm";
            os = "linux";
            type = "kernel_noload";
            compression = "none";
            load = <0x80080000>;
            entry = <0x80080000>;
            kernel-version = <1>;
```

```
            hash@1 {
                algo = "sha1";
            };
        };
    };
    configurations {
        default = "conf@1";
        conf@1 {
            description = "Boot Linux kernel";
            kernel = "kernel@1";
            signature@1 {
                algo = "sha1, rsa2048 ";
                key-name-hint = "dev_key";
                sign-images = "kernel";
            };
        };
    };
};
```

However, what the heck is fitImage (see below script - this is from the examples)? is it zImage? I couldn't find any documentation describing its first mention - what it is where it comes from etc... or is it an output generated by the reference from within the ITS for an incbin?

```
#!/bin/bash

key_dir=/tmp/keys
key_name=dev_key
FIT_IMG="fitImage"

rm -rf ${key_dir}
mkdir ${key_dir}

MKIMG="/home/dev/lede/staging_dir/host/bin/mkimage"
DTC="/usr/bin/dtc"

#Generate a private signing key (RSA2048):
openssl genrsa -F4 -out \
    "${key_dir}"/"${key_name}".key 2048

# Generate a public key:
openssl req -batch -new -x509 \
-key "${key_dir}"/"${key_name}".key \
-out "${key_dir}"/"${key_name}".crt

# Control FDT (u-boot.dts) - hits uboot to have keys etc...
CTRL_FDT="u-boot.dts"

# FIT image ITS - describes the node
FIT_ITS="fit-image.its"

#Assemble control FDT for U-Boot with space for public key:
$DTC -p 0x1000 $CTRL_FDT -O dtb -o u-boot.dtb

# Generate fitImage with space for signature:
$MKIMG -D "-I dts -O dtb -p 2000" \
-f f$FIT_ITS $FIT_IMG
```

```
# Sign fitImage and add public key into u-boot dtb:
```

Iminfo gets me this far:

```
#> iminfo

## Checking Image at 20000000 ...
   FIT image found
   FIT description: Configuration to load a Basic Kernel
    Image 0 (linux_kernel@1)
     Description:  Linux zImage
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x200000dc
     Data Size:    1465544 Bytes = 1.4 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x20000000
     Entry Point:  0x20008000
     Hash node:    'hash@1'
     Hash algo:    sha256
     Hash value:   bf1d62a9ac777310746c443f2500cf197967f1e7c9cb56ff5c33206670e1
     Hash len:     32
    Image 1 (fdt@1)
     Description:  FDT blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x20165ea4
     Data Size:    21681 Bytes = 21.2 KiB
     Architecture: ARM
     Hash node:    'hash@1'
     Hash algo:    sha256
     Hash value:   c7f32d039871d858dda8d397c3b6a685bc914c78cf70f03d1860f61ecfe9
     Hash len:     32
    Default Configuration: 'config@1'
    Configuration 0 (config@1)
     Description:  Plain Linux
     Kernel:       linux_kernel@1
```

The zImage is prepared (and this is likely the wrong way)

```
mkimage -A arm -O linux -C none -T kernel -a 0x22000000 -e 0x22008000 -n linux-4.
    -d $(KDIR)/zImage $(BIN_DIR)/$(IMG_PREFIX)-zImage-nDTB
```

Even along the lines of the following (I seem to get this, what do I do for addresses - is the reallocation part of the issue? such as the fdt_high variables?)

```
#> bootm 0x23000000
## Current stack ends at 0x23f119b8 *  kernel: cmdline image address = 0x230000
## Loading kernel from FIT Image at 23000000 ...
No configuration specified, trying default...
Found default configuration: 'config@1'
```

```
   Using 'config@1' configuration
   Trying 'linux_kernel@1' kernel subimage
     Description:  Linux zImage
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x230000dc
     Data Size:    1465544 Bytes = 1.4 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x23000000
     Entry Point:  0x23000000
     Hash node:    'hash@1'
     Hash algo:    sha256
     Hash value:   bb397db1ec90ec8526c6d215c9ded2a1357a258c2145f97fda9898e810e8
     Hash len:     32
   Verifying Hash Integrity ... sha256+ OK
   kernel data at 0x230000dc, len = 0x00165cc8 (1465544)
*  ramdisk: using config 'config@1' from image at 0x23000000
*  ramdisk: no 'ramdisk' in config
*  fdt: using config 'config@1' from image at 0x23000000
## Checking for 'FDT'/'FDT Image' at 23000000
## Loading fdt from FIT Image at 23000000 ...
   Using 'config@1' configuration
   Trying 'fdt@1' fdt subimage
     Description:  FDT blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x23165ea4
```

linux embedded-linux u-boot

Share Follow

edited Mar 1, 2017 at 20:36

asked Feb 12, 2017 at 20:47

mcdoomington

560 ● 1 ● 6 ● 20

Add a comment

2 Answers

Sorted by: Highest score (default)

3

So, there's a lot in the above question, and I'll try and answer a few things which should help clear things up:

1. The U-Boot binary needs to contain the pubkey. So in this case, the "myboard" device tree you've listed is where that needs to end up. It is within the binary and not a separate partition in flash.
2. The next thing is that the FIT image is a container with lets say different ways to open it. The fitImage contains zImage and normal-board.dtb and logic so that you can say that each of these pieces needs to be signed by a particular public key. So in this case, instead of a flash partition for the zImage and another for normal-board.dtb you have a

single partition for fitImage to reside in. It is the output from providing the "its" file to mkimage. This is similar to how "uImage" is the output from providing "zImage" to mkimage.

Share Follow

answered Feb 16, 2017 at 22:39
Tom Rini
2,153 ● 9 ● 11

Thanks Tom. The next question is I have is about addresses, loading etc.. I updated my answer to show how I got this far. – mcdoomington Mar 1, 2017 at 17:19

Add a comment

2

After alot of man-hours studying, reading, and trying - I created a full blog article about how verified uboot works, and how DTBs (both forms) come together when building the final images.

This article can be found here

However, the key things to note are indeed what Tom said and here are a few more (after quoting my article):

- There are two kinds of DTBs (the kernel & uboot DTBS)
- There is a file called an ITS - this describes the FIT image to be built
- You will need an asynchronous key pair
- You will need a version of mkimage that supports verified uboot/DTBs
- Your bootloader will need support Verified uboot enabled
- Uboot and the Linux kernel need to know about DTBs
- Even though you sign your images, a copy of the public key and other cryptographic info will be needed in the **final** bootloader

It was a fun process :)

Share Follow

answered Nov 23, 2017 at 22:34
mcdoomington
560 ● 1 ● 6 ● 20

2 Original link appears broken. Here's a cached version from the wayback machine. – Omer Anson Mar 5, 2020 at 12:33

Add a comment

Not the answer you're looking for? Browse other questions tagged linux embedded-linux u-boot or ask your own question.