https://community.nxp.com/docs/DOC-100847

Setting up of image using yocto project:

https://community.nxp.com/docs/DOC-104422 → very very important

$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libsdl1.2-dev xterm picocom

$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=${PATH}:~/bin

$ export IMX6SOLOX=~/imx6solox
$ mkdir -p $IMX6SOLOX

$ cd $IMX6SOLOX
$ mkdir -p fsl-release-bsp && cd fsl-release-bsp
$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-3.10.53-1.1.0_ga
$ repo sync

$ MACHINE=imx6sxsabresd source fsl-setup-release.sh -b build
$ bitbake core-image-minimal

$ cd tmp/deploy/images/imx6sxsabresd/
$ sudo dd if=core-image-minimal-imx6sxsabresd.sdcard of=/dev/sdX bs=1M && sync

Preparations: - Clear all the partitions in the SD card that is used to boot the system. *NOTE: You might want to copy
the root file system partition and the other partition with the kernel image and .dtb files already existing within the
SD card. Then apply this command below.

- once the image has been built, look at the temp/deploy/images/imx6sxsabresd folder. There will be two files
with .sdcard extensions and one of the files is a linker file. Do not use that file for this step.

1) sudo dd
if=/home/dheeraj/imx6solox/fsl-release-bsp/build/tmp/deploy/images/imx6sxsabresd/core-image-minimal-imx6sxsabresd-20180214081709.rootfs.sdcard of=/dev/mmcblk0 bs=1M && sync

Problems encountered in this step: Once the dd command has been executed, there will be two

partitions automatically created. I used GParted to check the partitions. Both the partitions have some form of error where the mmcblk0p1 (points to the partition containing the .dtb files and the kernel image - Zimage) and mmcblk0p2 files (points to the partition containing the root file system) do not exist in the /dev folder. I deleted partition 2 (containing the rootfs) and the mmcblk0p1 file was generated and the partition with the .dtb files and kernel image was generated properly.

This partition was of the file type FAT16 and this file type was automatically created by the system. Then i created a partition of filetype ext3 right behind the FAT16 partition (without any free space in between). Then i ran the above dd command once again and the partition for the root file system was created successfully. This file system is of the type ext3.

2) Mount points set at media/dheeraj/boot (for partition 1) and media/dheeraj/rootfs (for partition 2). Note that the uboot is included in the SD card. It might not be visible but the u-boot is located in the unallocated space of 4MB right infront of partition 1 at the beginning of the SD card.

3) Unmount the SD card and all the partitions.

4) Now, put the SD card into the imx6sx sabre board and boot up the system using the ZOC7 terminal.

5) To access the u-boot command, press any key when the prompt says: "Press any key to stop auto-boot"

6) Once properly booted and checked, generate the fuse table and fuse binary files using the instructions given at the HAB document [“fuse prog 3 <0-8> <address>”]. This is done using the cst 2.3.3 file downloaded from github.

7) Then program the fuse table into the board's non-volatile memory using the fuse command found in the u-boot cli.

https://boundarydevices.com/high-assurance-boot-hab-dummies/

Building and configuring the u-boot to enable secure boot:

Description: *Problem with the previous process (building the core-image-minimal) is that it will create u-boot without having the config_secure_boot option enabled. Upon researching online, there has been no direct way of building sdcard image with secure u-boot. Therefore, we need to build the u-boot all over again with secure u-boot enabled.*

1) The u-boot has to be configured such that it enables HAB. However, when we build the

.sdcard image from the YOCTO project, the u-boot by default does not enable secure boot.

2) Therefore, the u-boot is built manually and configured to enable secure SD boot.
3) Download this U-boot file system: u-boot-imx6-boundary-v2017.07. The earlier versions of this u-boot file as given in this step-by-step HAB implementation guide (https://boundarydevices.com/high-assurance-boot-hab-dummies/) is outdated and has bugs within it → therefore, doesn't execute properly even when the toolchain path is exported properly and all the steps are followed properly.
4) Now download the linarno toolchain and unpack the files. This is required for cross compilation purposes. Do this by executing this command: cd
   wget -c
   https://releases.linaro.org/components/toolchain/binaries/5.2-2015.11-2/arm-linux-gnueabihf/gcc-linaro-5.2-2015.11-2-x86_64_arm-linux-gnueabihf.tar.xz
   tar xvf gcc-linaro-5.2-2015.11-2-x86_64_arm-linux-gnueabihf.tar.xz
   ln -s gcc-linaro-5.2-2015.11-2-x86_64_arm-linux-gnueabihf gcc-linaro
5) The execute these 3 commands:
   - export ARCH=arm
   - export PATH=~/gcc-linaro/bin/:$PATH → This path details where the executable files are.
   - export CROSS_COMPILE=arm-linux-gnueabihf- → Setting the cross compiler

6) Open this file from this faile path: u-boot-imx6-boundary-v2017.07/configs/mx6sxsabresd_defconfig. Add the line 'CONFIG_SECURE_BOOT=y' within this file. This is how the secure boot configuration is enabled.
7) Until step 6 is what is required to get the secure boot activated. However, the secure boot will not be activated as there are several patches that are required to be made to certain files. The patches are as given in this website: https://freescale.jiveon.com/docs/DOC-330622. However, not all the patches stated in this website needs to be applied:
   - nano arch/arm/imx-common/timer.c
   - unsigned long usec2ticks(unsigned long usec)
   - {
   - ulong ticks;
   - if (usec < 1000)
   - ticks = ((usec * (get_tbclk()/1000)) + 500) / 1000;
   - else
   - ticks = ((usec / 10) * (get_tbclk() / 100000));
   - return ticks;
   - }
   - Find this usec2ticks function and replace the function content with that given above to enable secure boot.

- nano board/freescale/mx6sxsabresd/imximage.cfg
- Add this line after the 'BOOT_FROM sd' line: CSF 0x2000. This is to give the physical address location of the CSF data.
- nano tools/Makefile
- Delete the following strings from the Makefile:
- gpimage.0 \
- gpimage-common.o \
- omapimage.o \

8) make mx6sxsabresd_defconfig → the config files are used to build the image.
9) make menuconfig → This will open a config window. Go to the arm architecture menu and check if the 'Support i.MX HAB Features' menu is ticked/selected.
10) make V=1 → verbose output
11) This should appear and the line with the HAB Blocks is the most important line as this will be used in the CSF file in the next steps. Build result is as follows:

```
Image Type:   Freescale IMX Boot Image
Image Ver:2 (i.MX53/6/7 compatible)
Mode:       DCD
Data Size: 466944 Bytes = 456.00 KiB = 0.45 MiB
Load Address: 877ff420
Entry Point:  87800000
HAB Blocks:   877ff400 00000000 0006fc00
DCD Blocks:   00910000 0000002c 00000208
```

12) CSF file generation (highlighted data from above step is used in the last line):

```
[Header]
    Version = 4.0
    Hash Algorithm = sha256
    Engine Configuration = 0
    Certificate Format = X509
    Signature Format = CMS

[Install SRK]
    File = "../../crts/SRK_1_2_3_4_table.bin"
    Source index = 0

[Install CSFK]
    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate CSF]
#Left blank because by doing so configuration is set to default

[Unlock]
```

```
    Engine=CAAM
    Features=RNG
[Install Key]
    #authenticates and installs a public key for use in Authenticate Data command
    Verification index = 0
    Target index = 2
    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
    # Sign padded u-boot starting at the IVT through to the end with
    # length = 0x2F000 (padded u-boot length) - 0x400 (IVT offset) = 0x2EC00
    # This covers the essential parts: IVT, boot data and DCD.
    # Blocks have the following definition:
    # Image block start address on i.MX, Offset from start of image file,
    # Length of block in bytes, image data file

[Authenticate Data]
    Verification index = 2
    Blocks = 0x877ff400 0x00000000 0x0006fc00
"../../../u-boot-imx6-boundary-v2017.07/u-boot.imx"
```

13) ./cst -o csf-uboot.bin -i u-boot.csf

14) cat u-boot.imx csf-uboot.bin > u-boot.mx6sxsabresd → This is to append the signed binary csf file to u-boot.imx image and copy it to a new file (u-boot.mx6sxsabresd)

15) cp u-boot.mx6sxsabresd ../../../u-boot-imx6-boundary-v2017.07/

16) sudo dd if=/home/dheeraj/u-boot-imx6-boundary-v2017.07/u-boot.mx6sxsabresd of=/dev/mmcblk0 bs=1K seek=1;sync

17) => fuse prog 0 6 0x2 → Apply this to enable secure boot. This step will blow the SEC_CONFIG[1] fuse bit. Now the device is in closed state (or secure state). Refer to this document/website for more information regarding efuses and their structures: https://imxdev.gitlab.io/tutorial/Burning_eFuses_on_i.MX/

18) => reset → Perform reset of the CPU.

19) Problem encountered: u-boot and the kernel all boots well until a point where it says “Waiting for root device /dev/mmcblk0p2”. Go to the u-boot terminal:
=> mmcroot “/dev/mmcblk3p2” → This is where the CPU will look for the root file system

20) Sometimes, the kernel will get stuck at this part: “mxc_asrc 2034000.asrc: mxc_asrc registered” → Press the reboot button until it reboots and get to the u-boot console: =>. Now remove the sdcard and insert it again properly. Now, again press the reboot button

and get to the u-boot console: =>boot/bootd. Now the system will boot up normally.

https://community.nxp.com/thread/380527
21) objcopy -I binary -O binary --pad-to=0x54A000 --gap-fill=0x00 zImage zImage-pad.bin

Configuring the kernel with Image Vector Table and signing it to enable secure boot:

https://github.com/dheerajaraj/KeySightWork/tree/master/Kernel

Disabling unsigned kernel to boot

Description: *Signing of u-boot is compulsory. However, signing of kernel is not. Therefore, after testing with signed and unsigned kernel, it was observed that in both instances the system booted without any problem as kernel signing is not compulsory and the root of trust is established when authenticating the u-boot itself.*

Therefore, to ensure that only a signed kernel boots, the u-boot environment has been modified. The u-boot environment, once configured, can be closed to prevent further configuration and the u-boot can also be password protected. However, this linux system has been built as a core-image-minimal. Therefore, many features, commands of the traditional linux OS will not be made available here. Therefore, the commands required to lock the u-boot cannot be executed with a core-image-minimal. The list of commands executed to configure the u-boot to only allow signed kernel images to boot is:

=> setenv kernelPass 0;
=> setenv habcheck 'load mmc 2:1 0x82000000 zImage;if hab_auth_img 0x82000000 0x54A000;then setenv kernelPass 0;else setenv kernelPass 1; fi'
=> setenv bootcmd 'mmc dev ${mmcdev};mmc dev ${mmcdev};run habcheck;if mmc rescan && test ${kernelPass} = 1;then if run loadbootscript; then run bootscript;else if run loadimage; then run mmcboot;else run netboot; fi; fi;else echo Unsigned Kernel! Stopping boot...; fi'
=> saveenv

- The second command is executed to load the kernel into the kernel load address

(0x82000000). The hab_auth_img function uses the u-boot to authenticate the kernel image. First argument of hab_auth_img is the load addr. The second argument is the length of the signed image to be checked from the load address.

- Based on the file hab.c in u-boot-imx (used to build HAB enabled U-boot in pg 2) folder, the function that authenticates the kernel image will return 1 if the authentication is unsuccessful and 0 if the authentication is successful. Therefore, if the authentication is unsuccessful, the variable kernelPass (which is the first command) is set to 0 and if successful, the variable is set to 1.
- The third command makes use of this value of kernelPass to decide whether to execute or not execute.

Preventive measures

Description: *Sometimes, private key can be compromised or the Certificate authority might have been invalid. Therefore, there should be a way to invalidate the keys that have been fused into the imx6 processor. This invalidation is done by revoking the selected SRK fuse.*

Based on reference from the imx6sx application processor reference manual (pg 2828) and https://www.nxp.com/docs/en/application-note/AN4581.pdf, this command needs to be executed:
=> fuse prog 5 7 0x00000001
This will invalidate the SRK. Accordingly, the SRK fuse has been revoked and the boot process stops/ failing to execute properly. However, the CSF file needs to be changed in order to allow the boot to happen with another set of keys. However, the methods for this to take place is confidential and a non-disclosure agreement has to be signed before appropriate guidance from nxp website is given. More info found here (https://community.nxp.com/thread/399255).

LITTLE BIT OF TUTORIAL

1) Export command: Means that it is part of your shell. Export command marks an environment variable to be exported with any newly forked child process, thus allowing qa child process to inherit all marked variables.
2) Any new child process that is forked from the parent's process, does not inherit the parent's variables by default.
3) Therefore, the export is a form of inheritance that is enabled for all the children of the parent bash. You exit from a child or any bash for that matter using the exit command.
4) Any process, except init, can be a parent and a child all at the same time. The init process is marked with a PID of 1.
5) $ echo $$ → Prints the PID of the current shell
6) 27861
7) $ bash → creates a new child process
8) $ echo $$
9) 28034
10) $ ps --ppid 27861 → print the child process of the parent process (27861)
11) PID TTY TIME CMD
12) 28034 pts/3 00:00:00 bash → Result

IMPORTANT DIVERSION

1) PATH variable -->Human readable address that tells the user where and which directories to search for executable files in response to commands issued by user. Therefore, when you type a command into a shell, the shell needs to find out where the files are. Therefore, the PATH variable is a list of directories separated by the colon character.

1) Functions can also be exported:
2) $ printname () { echo "Linuxcareer.com"; }
3) $ printname
4) Linuxcareer.com
5) $ export -f printname
6) $ bash
7) $ printname
8) Linuxcareer.com

2) To remove variables from the export list, you need to apply this simple command:

```
        $ export | grep MYVAR
declare -x MYVAR="10"
$ export -n MYVAR
$ export | grep MYVAR
$
```

3) cat proc/partitions/ command output explanation”
major — The major number of the device with this partition. The major number in the /proc/partitions, (3), corresponds with the block device ide0, in /proc/devices.
minor — The minor number of the device with this partition. This serves to separate the partitions into different physical devices and relates to the number at the end of the name of the partition. #blocks — Lists the number of physical disk blocks contained in a particular partition. name — The name of the partition.

4) About load addresses and entry points:
https://www.gitbook.com/book/0xax/linux-insides/details

5) The IVT (Image Vector Table) is the data structure that the ROM reads from the boot device supplying the program image containing the required data components to perform a successful boot.

IVT includes the program image entry point, a pointer to the DCD (Device Configuration Data) and other pointers used by the ROM during the boot process. ROM locates the IVT at a fixed address determined by the boot device. IVT offset from the base address and initial load region size for each boot device is defined in the table → 1kB (offset for SD cards) and 4KB (size of load region for SD cards).

Device Configuration Data (DCD)
Upon reset, the chip uses default register values for all the peripherals in the system. However, these settings typically are not ideal for achieving optimal performance and there are even some peripherals that must be configured before use.