Using a TPM module to secure BeagleBone Black boot

Mark Scott

February 18, 2015

1 U-Boot

2 Linux

2.1 Create partitions

As root on another linux box:

1. Create partition table (tested with msdos)
2. Create small boot partition (e.g. 20 MiB), formatted as FAT16 or FAT32 for u-boot
3. Create root partition unformatted with the type `e8` (we will do the rest with LUKS)
4. Create a second root partition, formatted as ext4

2.2 Create encrypted partition

1. Format the root partition with LUKS using a password to begin with. Change `/dev/sdc2` to match the `e8` partition created above:

```
cryptsetup -v luksFormat /dev/sdc2
```

2. Mount the encrypted partition

```
cryptsetup luksOpen /dev/sdc2 newbone
```

3. Format the partition:

```
mkfs.ext4 /dev/mapper/newbone
```

4. Mount the partition:

```
mkdir /mnt/newbone # If directory mount point does not exist
mount -t ext4 /dev/mapper/newbone /mnt/newbone
```

2.3 Install ArchLinux to unencrypted partition

Instructions from `http://archlinuxarm.org/platforms/armv7/ti/beaglebone-black`.

1. Get the ArchLinux image:

```
wget http://archlinuxarm.org/os/ArchLinuxARM-am33x-latest.tar.gz
```

2. Untar the image to the unencrypted root partition. Use `bsdtar`, and ensure you do it as root – **not just sudo**.

```
# Mount the partition if not already mounted:
mount /dev/sdc3 /mnt/sdc3
bsdtar -xpf ArchLinuxARM-am33x-latest.tar.gz -C /mnt/sdc3
```

3. Copy MLO and u-boot.img from elsewhere into boot partition (see other instructions)
4. Copy uEnv.txt into boot partition for booting ArchLinux

2.4 Initial setup

1. Boot not connected to network
2. Log in with username/password: root/root
3. Change password with `passwd`
4. Add a user account, e.g.:

```
useradd -m -G wheel -s /bin/bash mark
```

5. Change the password, e.g:

```
passwd mark
```

6. Change the hostname with:

```
hostnamectl set-hostname <newname>
```

7. Reboot.
8. We should update the system immediately. This could be done on a private network for critical systems, or by downloading critical packages separately and installing them before connecting to a network.

   As root:

```
pacman -Syu
```

9. Reboot again.

2.5 Check TPM modules are loaded

1. Check TPM modules are loaded with `lsmod`. If not, kernel modules are not available so follow instruction in `https://wiki.archlinux.org/index.php/TPM`.
2. Check useful TPM info is available, e.g. the PCRs with:

```
cat /sys/class/misc/tpm0/device/pcrs
```

2.6 Configure ArchLinux

As root:

1. You may want to use the following pacman commands to install useful packages:

```
pacman -S sudo
```

2. Then type:

```
visudo
```

3. and uncomment the following command (remove the #):

```
# %wheel ALL=(ALL) ALL
```

As a user in wheel group:

1. Install packages:

```
sudo pacman -S wget
sudo pacman -S base-devel
sudo pacman -S yajl (for package-query)
```

2. Install yaourt so AUR packages are easier:

   a) Install package-query

```
mkdir yaourt
cd yaourt
wget https://aur.archlinux.org/packages/pa/package-query/package-query.tar.gz
tar xvzf package-query.tar.gz
cd package-query
makepkg
sudo pacman -U package-query-1.5-2-armv7h.pkg.tar.xz
```

   b) Install yaourt

```
wget https://aur.archlinux.org/packages/ya/yaourt/yaourt.tar.gz
tar xvzf yaourt.tar.gz
cd yaourt
makepkg
sudo pacman -U yaourt-1.5-1-any.pkg.tar.xz
```

3. Download the latest ibmswtpm library for reading the TPM when booting (from `http://sourceforge.net/projects/ibmswtpm/files/`)

```
mkdir ibmswtpm
cd ibmswtpm
wget http://sourceforge.net/projects/ibmswtpm/files/tpm4720.tar.gz
tar xvzf tpm4720.tar.gz
cd libtpm
./comp-chardev.sh
sudo make install (to install tpm utils in /usr/local/bin and headers in /usr/local/lib)
```

4. Install trousers (for tcsd daemon):

   ...When asked if PKGBUILD should be edited, say yes and add `armv7h` to the `arch=` line, e.g.:

```
arch=('i686' 'x86_64' 'armv7h')
```

```
yaourt -S trousers
yaourt -S opencryptoki [If this fails with a 404, check the URL is correct in PKGBUILD]
yaourt -S tpm-tools
```

5. Configure opencryptoki

```
sudo groupadd pkcs11
sudo gpasswd -a root pkcs11
```

6. To use tpm-tools:

```
# Start the tcsd daemon
sudo systemctl start tcsd

# Then type tpm-tools commands, e.g.
tpm_nvread -i 0x1002 -s 312
```

7. To use ibm tools:

```
# Stop the tcsd daemon which prevents the tools from using the TPM
sudo systemctl stop tcsd

# Then use the tools installed, e.g.:
sudo nv_readvalue -in 1002 -sz 312
```

2.7 Adding encryption support to initrd

The inspiration for this approach came from tpm-luks [`https://github.com/shpedoikal/tpm-luks/`]

1. Edit `/etc/mkinitcpio.conf` to include the correct hooks: insert `encrypt-tpm` on the `HOOKS=` line before `filesystems`:

```
cd /etc
cp mkinitcpio.conf mkinitcpio.conf.backup # Backup file before editing
vi mkinitcpio.conf
HOOKS="... encrypt-tpm filesystems ..."
```

2. Include modules and binary in ramdisk with following patch:

```
cd /usr/lib/initcpio/install
cat<<\EOF|patch -o encrypt-tpm
--- encrypt 2014-09-03 00:54:24.000000000 +0000
+++ encrypt-tpm 2015-02-18 14:20:25.184656549 +0000
@@ -20,6 +20,12 @@
     add_file "/usr/lib/initcpio/udev/11-dm-initramfs.rules" "/usr/lib/udev/rules.d/11-dm-initramfs.rules"

     add_runscript
+
+    add_module tpm
+    add_module tpm_i2c_atmel
+    add_binary "/usr/local/bin/nv_readvalue"
+    add_binary "/usr/local/bin/unsealxfile"
+    add_binary "/usr/bin/shred"
 }

 help() {
EOF
```

3. Add code to decrypt partition with this patch, which does the following:

   a) Adds a variable and new function to load key to tmpfs

   b) Checks `cryptoptions` for `use-tpm` and calls new function

   c) Unmounts the tmpfs area once key has been used

```
cd /usr/lib/initcpio/hooks
cat<<\EOF|patch -o encrypt-tpm
--- encrypt 2014-09-03 00:54:24.000000000 +0000
+++ encrypt-tpm 2015-02-18 14:16:40.633779693 +0000
@@ -49,11 +49,40 @@
         echo "Use 'cryptdevice=${root}:root root=/dev/mapper/root' instead."
     }

+    tpm_getkey() {
+       modprobe tpm
+       modprobe tpm_i2c_atmel
+       nvreader=/usr/local/bin/nv_readvalue
+       unseal=/usr/local/bin/unsealxfile
+       tpm_mntdir=/mnt/tpm_tmpfs
+       ckeyfile=${tpm_mntdir}/key
+
+       # Mount a tmpfs ram disk.
+       # 8 K is the smallest we can get away with because the block size is 4 K and we have two files to store.
+       if [ ! -d ${tpm_mntdir} ]; then
+         mkdir -p ${tpm_mntdir}
+         mount -t tmpfs -o size=$((4096*2)) tmpfs ${tpm_mntdir}
+       fi
+
+       ${nvreader} -in 1002 -sz 322 -of ${tpm_mntdir}/sealedkey
+       ${unseal} -hk 0x40000000 -if ${tpm_mntdir}/sealedkey -of ${ckeyfile}
+    }
+
+    tpm_cleanup() {
+       /usr/bin/shred -u ${tpm_mntdir}/sealedkey
+       /usr/bin/shred -u ${ckeyfile}
+       umount ${tpm_mntdir}
+    }
+
     for cryptopt in ${cryptoptions//,/ }; do
         case ${cryptopt} in
             allow-discards)
                 cryptargs="${cryptargs} --allow-discards"
                 ;;
+            use-tpm)
+                usingtpm=1
+                tpm_getkey
+                ;;
             *)
                 echo "Encryption option '${cryptopt}' not known, ignoring." >&2
                 ;;
@@ -72,6 +101,9 @@
                     echo "Invalid keyfile. Reverting to passphrase."
                 fi
             fi
+            if [ "${usingtpm}" = "1" ]; then
+              tpm_cleanup
+            fi
             # Ask for a passphrase
             if [ ${dopassphrase} -gt 0 ]; then
                 echo ""
EOF
```

4. Generate the initrd:

```
mkinitcpio -g /boot/initrd-custom.img
```

5. Add the following to uEnv.txt bootargs:

   a) Add a `loadinitrd` command:

```
loadinitrd=load mmc ${mmcdev}:1 ${rdaddr} /boot/initrd-custom.img; setenv rdsize 0x${filesize}
```

b) Add `initrd=$rdaddr,$rdsize` to `mmcargs`

c) Add `$rdaddr:$rdsize` to second argument of `bootz` in `mmcboot`

d) Add `run loadinitrd` to uenvcmd before '`run mmcboot`'

e) Using the syntax '`cryptdevice=dev:name:options`', add the appropriate `cryptdevice` option to bootargs in mmcargs, e.g.:

```
cryptdevice=/dev/mmcblk0p2:myencfs
```

f) Copy `/boot` to FAT32 partition:

```
mkdir /mnt/boot # If it doesn't already exist
mount /dev/mmcblk0p1 /mnt/boot
cp /boot/zImage /mnt/boot
cp /boot/initrd-custom.img /mnt/boot
mkdir /mnt/boot/dtbs
cp /boot/dtbs/* /mnt/boot/dtbs
```

2.8 Copy Linux to encrypted partition

1. Open the encrypted partition

```
cryptsetup luksOpen /dev/mmcblk0p2 newroot
```

2. Mount the partition (you may need to create the folder):

```
mount -t ext4 /dev/mapper/newroot /mnt/newroot
```

3. Switch to `root` user:

```
su -
```

4. Put the machine into single-user mode:

```
telinit 1
```

5. Copy the root partition's files to encrypted partition:

```
rsync -aHAX --exclude={"/dev/*","/proc/*","/sys/*","/tmp/*","/run/*","/mnt/*","/media/*","/lost+found"}
/ /mnt/newroot/
```

6. Reboot. This will reboot and mount the encrypted partition with a password.

2.9 Boot encrypted Linux

1. Enter the password during boot to mount the encrypted root.

2. Check we have booted with encrypted partition:

```
# mount | grep mapper
/dev/mapper/myencfs on / type ext4 (rw,relatime,data=ordered)
```

The above shows that `/` was mounted as ext4 against `/dev/mapper/myencfs` which is the encrypted partition.

More information can be found by typing:

```
# cryptsetup luksUUID /dev/mmcblk0p2
b9d9c1c2-5038-4969-85ab-75fc0b88d099
```

And checking the UUID against the mapped partition:

```
dmsetup info myencfs
Name:              myencfs
State:             ACTIVE
Read Ahead:        256
Tables present:    LIVE
Open count:        1
Event number:      0
Major, minor:      254, 0
Number of targets: 1
UUID: CRYPT-LUKS1-b9d9c1c25038496985ab75fc0b88d099-myencfs
```

2.10 Generate key to store in TPM

1. Mount a temporary RAM disk to create the key:

```
mkdir tmpfs-mnt
mount tmpfs tmpfs-mnt -t tmpfs -o size=32m
```

2. Generate a random key file (storing it in tmpfs). In order for the IBM seal to work, it must be <149 bytes, but the unseal only works when the original blob was <61 bytes. So we will use the maximum 60 bytes.

```
cd tmpfs-mnt
dd if=/dev/urandom of=mykeyfile bs=60 count=1 iflag=fullblock
```

3. Allow the key file to unlock the encrypted partition

```
sudo cryptsetup luksAddKey /dev/mmcblk0p2 mykeyfile  # Enter passphrase above
```

4. Seal the key file against a PCR value with the SRK (Storage Root Key). We use an example value for PCR 15.

```
sealxfile -hk 0x40000000 -if mykeyfile -of sealedkeyfile -ix 15 2A221563D80EC9819B496FC80C03249448FFDAD6
```

5. Define space to store key file in TPM. Note that we are using well known passwords here at the moment and other permissions might want tweaking:

```
tpm_nvdefine -i 0x1002 -s 322 -y -z -p AUTHWRITE
```

6. Store the key file in the TPM. Key data can be sealed before writing and will only unseal with correct PCRs.

```
tpm_nvwrite -i 0x1002 -s 322 -f sealedkeyfile -z
```

7. Optionally, decide if you want to store the key file somewhere else, such as on a USB key. If so, copy the file now, otherwise it will be lost when tmpfs dismounts.

8. Add the `use-tpm` option to the `uEnv.txt`:

```
cryptdevice=/dev/mmcblk0p2:myencfs:use-tpm
```

9. Reboot and initrd will load key from TPM

A uEnv.txt contents

```
optargs=
mmcpart=1
loadfdt=load mmc ${mmcdev}:${mmcpart} ${fdtaddr} /boot/dtbs/${fdtfile}
loadinitrd=load mmc ${mmcdev}:${mmcpart} ${rdaddr} /boot/initrd-custom.img; setenv rdsize 0x${filesize}
loaduimage=mw.l 4804c134 fe1fffff; if load mmc 0:${mmcpart} ${loadaddr} /boot/zImage; then setenv mmcdev 0;
setenv mmcroot /dev/mapper/myencfs; mw.l 4804c194 01200000; echo Booting from external microSD...; else setenv
 mmcdev 1; if test $mmc0 = 1; then setenv mmcroot /dev/mmcblk1p3 rw; fi; load mmc 1:2 ${loadaddr} /boot/zImage
; mw.l 4804c194 00c00000; echo Booting from internal eMMC...; fi

# To boot with an initrd, add to mmcargs:
#       initrd=${rdaddr},${rdsize}
# To load the encryption key from the TPM (custom hook in initrd) add to mmcargs:
#       cryptdevice=/dev/mmcblk0p2:myencfs:use-tpm

mmcargs=setenv bootargs console=tty0 console=${console} ${optargs} root=${mmcroot} rootfstype=${mmcrootfstype}
 rw initrd=${rdaddr},${rdsize} cryptdevice=/dev/mmcblk0p4:myencfs:use-tpm

# To boot with an initrd, change mmcboot to the following:
#       mmcboot=run mmcargs; bootz ${loadaddr} ${rdaddr}:${rdsize} ${fdtaddr}

mmcboot=run mmcargs; bootz ${loadaddr} ${rdaddr}:${rdsize} ${fdtaddr}

# To boot with an initrd, add 'run loadinitrd' before 'run mmcboot'
uenvcmd=i2c mw 0x24 1 0x3e; run findfdt; if test $board_name = A335BNLT; then setenv mmcdev 1; mmc dev ${
mmcdev}; if mmc rescan; then setenv mmc1 1; else setenv mmc1 0; fi; fi; setenv mmcdev 0; mmc dev ${mmcdev}; if
 mmc rescan; then setenv mmc0 1; else setenv mmc0 0; fi; run loaduimage && run loadfdt && run loadinitrd &&
run mmcboot
```