Secure Boot

Powody...

- Producent
  - Upewnienie się że urządzenie używane jest do działań dla których jest przeznaczone
  - Uniemożliwienie innych zastosowań niż przewidziane
  - Umożliwienie uruchomienia tylko autoryzowanego oprogramowania
  - Zabezpieczenie klientów
- Użytkownik
  - Upewnienie się że oprogramowanie nie zostało zmienione
- Najprościej… upewnienie się że binaria ładowane/boot’owane/wykonywane pochodzą z autoryzowanego źródła

Jak to działa?

- Bazowanie na weryfikacji podpisu (to nie jest jednoznaczne z szyfrowaniem)
- Zapewnienie że w akcji poprzedzającej, następuje uwierzytelnienie akcji następnej
- Wykonanie procesu boot’owania odbywa się w łańcuchu bezpiecznych operacji

Łańcuch zaufania

- Każdy z komponentów uwierzytelniony jest z użyciem podpisu i klucza publicznego
- Integralność systemu plików root, weryfikowana przez badanie funkcji skrótu

Wpływ na proces wytwarzania

- Koszty
  - Wymagane kreowanie i utrzymanie kluczy, ich wzajemne podpisanie i utrzymywanie w procesie wytwarzania
  - Komplikacja dla tworzących system (potencjalne tworzenie nowego obrazu, wymaga powtórzenia operacji podpisywania/obliczania skrótu na danym etapie)
  - Czas uruchamiania ulega wydłużeniu
- Wymagane utrzymywanie w sekrecie kluczy prywatnych wykorzystanych w łańcuchu

Kod w ROM – początek łańcucha

Specyficzne dla platformy

- Wymaganie umieszczenie klucza publicznego wykorzystanego do rozszyfrowania podpisu bootloadera
- Każdy z producentów samodzielnie decyduje w jaki sposób (na jakim medium) zapisze klucz publiczny
- Kod platformy sprawdza podpis w ROM

- Klucz publiczny zapisany w pamięci nieulotnej (NVM) dostępnej dla ROM
- OTP (ang. One-Time-Programmable) zabezpiecza dostęp do tejże pamięci
- Klucz publiczny zawiera się w ~ 1KB pamięci
- Dostępny wariant zapisania w OTP wyłącznie skrótu z klucza publicznego (mniejsza konsumpcja pamięci), umieszczonego w binarium
- Stosowana idea przechowywania kliku skrótów w przypadku ujawnienia klucza prywatnego:
  - Daje możliwość wykonania uruchomienia w trybie awaryjnym
  - Broni przed „tworzeniem cegieł” (ang. brick-device)

Sekwencja startu

- Kod ROM:
  - Ładuje bootloader w bezpieczną przestrzeń utrudniającą atak fizyczny
  - Ładuje wbudowany klucz publiczny
  - Sprawdza skrót klucza publicznego (z OTP) z kluczem publicznym (z binarium)
  - Używa zweryfikowanego klucza publicznego do weryfikacji sygnatury bootloadera
  - Uruchamia binarium bootloadera

Przygotowanie urządzenia

- Tworzenie kluczy narzędziami producenta (ang. Code Signing Tool)
- Flash niezweryfikowanego kodu U-Boot
- Podpisanie skrótem kodu U-Boot z użyciem CST
- Zablokowanie (fuse) możliwości modyfikacji bootloadera

Bootloader

- Nie nastąpi przejście do bootloadera jeśli skrót nie będzie zweryfikowany
- Blokowanie dostępu do konsoli U-Boot (`gd→flags |= GD_FLG_DISABLE_CONSOLE z funkcji board_early_init_f()`)
- Środowisko jest zaufane tylko dla binarium zweryfikowanego U-Boot (`ENV_IS_NOWHERE`)
- Przy tej procedurze, zalecane są także inne flagi producenta.. przykład (https://patchwork.ozlabs.org/patch/855542/)

Kernel

- U-Boot, wczytuje DTB (ang. DeviceTree Blob)
- DTB zawiera klucz publiczny
- Domyka to etap zaufania łańcucha
- Program mkimage posiada opcje naliczenia podpisu jądra

Generowanie klucza

```
openssl genrsa -out my_key.key 4096
openssl req -batch -new -x509 -key\
my_key.key -out my_key.crt
```

- `mkimage` wymaga certyfikatu i prywatnego klucza o tych samych nazwach

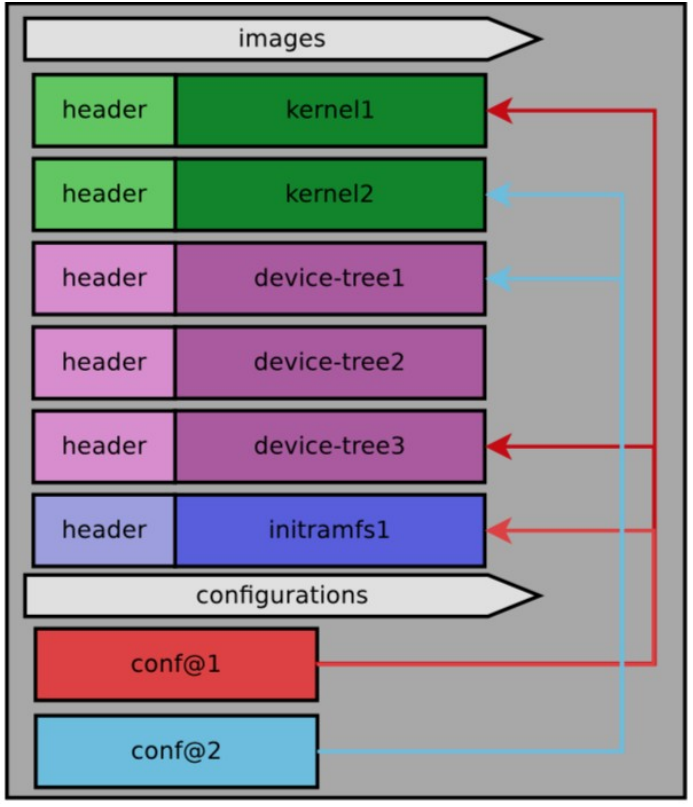
DTB

- u-boot_pubkey.dts

```
/dts-v1/;
/{
model = "Keys";
      compatible = "vendor,board";
      signature {
          key-my_key {
            required = "image";
            algo = "sha1,rsa4096";
            key-name-hint = "my_key";
}; };
};
```

- Klucz key-name-hint, powinien zawierać rdzeń nazwy klucza
- Klucz required zawiera wpis image lub conf udokumentowany w doc/uImage.FIT/signature.txt

Zawartość fitImage



- Obraz jest kontenerem wielu binariów wraz opisem (nagłówkiem) dostępu do nich
- Opisuje także skojarzenia binariów i konfiguracji
- Format wspiera wiele systemów OS oraz typów obrazów (dokumentacja w common/image.c)

fitImage.its

```
/ {
    description = "fitImage for Foo revA and revB";
    #address-cells = <1>;
    images {
        kernel@1 {
            description = "Linux kernel";
            data = /incbin/("zImage");
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x10008000>;
            entry = <0x10008000>;
            signature@1 {
                algo = "sha1,rsa4096";
                key-name-hint = "my_key";
            };
        };
        fdt@1 {
            description = "DTB for Foo revA";
            data = /incbin/("foo-reva.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
            signature@1 {
                algo = "sha1,rsa4096";
                key-name-hint = "my_key";
            };
        };
        fdt@2 {
            description = "DTB for Foo revB";
            data = /incbin/("foo-revb.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
            signature@1 {
                algo = "sha1,rsa4096";
                key-name-hint = "my_key";
            };
        };
    };
    configurations {
        default = "conf@1";
        conf@1 {
            kernel = "kernel@1";
            fdt = "fdt@1";
        };
        conf@2 {
            kernel = "kernel@1";
            fdt = "fdt@2";
        };
    };
};
```

Tworzenie fitImage i U-Boot

```
dtc u-boot_pubkey.dts -O dtb -o u-boot_pubkey.dtb
make CROSS_COMPILE=arm-linux-gnueabihf- foo_defconfig
make CROSS_COMPILE=arm-linux-gnueabihf- tools
tools/mkimage -f fitImage.its -K u-boot_pubkey.dtb \
    -k /path/to/keys -r fitImage
make CROSS_COMPILE=arm-linux-gnueabihf- \
    EXT_DTB=u-boot_pubkey.dtb
```

Wymagane opcje U-Boot

- CONFIG_SECURE_BOOT=y
- #ifdef CONFIG_SECURE_BOOT
  CSF CONFIG_CSF_SIZE
  #endif
- CONFIG_OF_CONTROL=y
- CONFIG_DM=y, CONFIG_FIT=y, CONFIG_FIT_SIGNATURE=y

Ładowanie fitImage

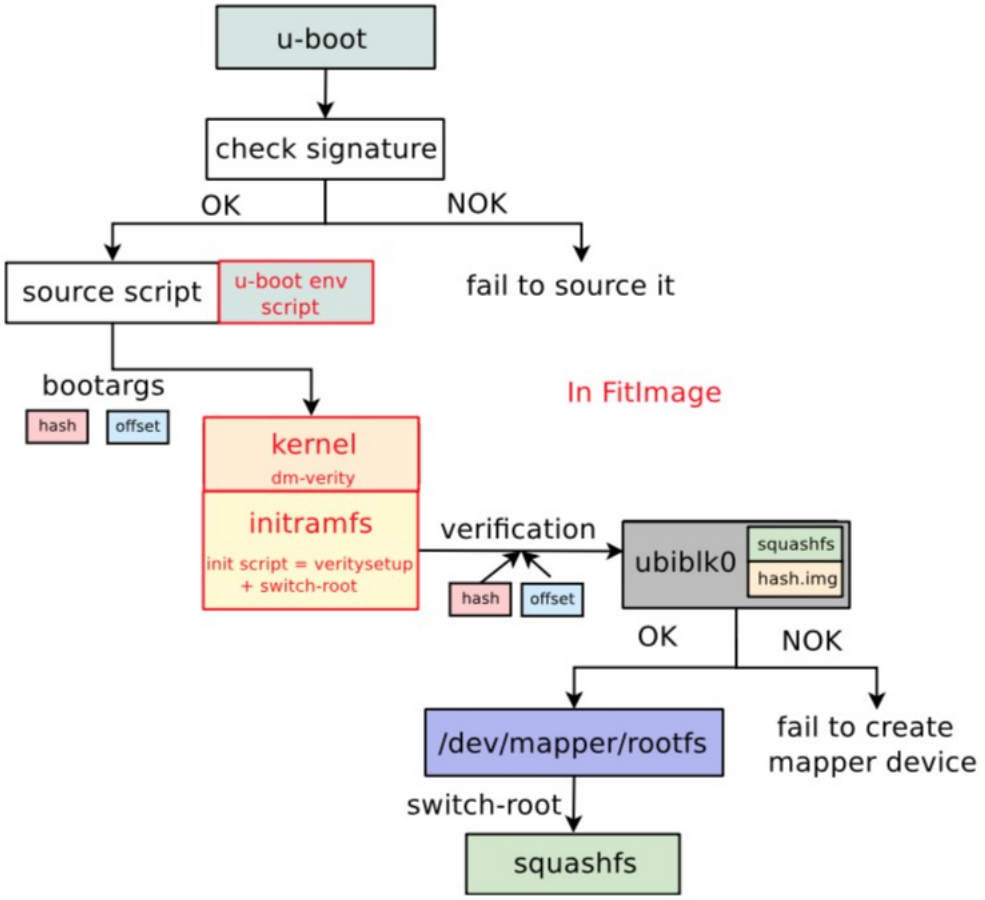
```
=> bootm 0x15000000 #or bootm 0x15000000#conf@1 since conf@1 is the default
## Loading kernel from FIT Image at 15000000 ...
   Using 'conf@1' configuration
   Verifying Hash Integrity ... OK
   Trying 'kernel@1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x150000e4
     Data Size:    7010496 Bytes = 6.7 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x10008000
     Entry Point:  0x10008000
     Hash algo:    sha1
     Hash value:   7d1fb52f2b8d1a98d555e01bc34d11550304fc26
     Sign algo:    sha1,rsa4096:my_key
     Sign value:   [redacted]
   Verifying Hash Integrity ... sha1,rsa4096:my_key+ sha1+ OK
## Loading fdt from FIT Image at 15000000 ...
   Using 'conf@1' configuration
   Trying 'fdt@1' fdt subimage
   [...]
   Verifying Hash Integrity ... sha1,rsa4096:my_key+ sha1+ OK
   Booting using the fdt blob at 0x156afd40
   Loading Kernel Image ... OK
   Loading Device Tree to 1fff2000, end 1ffff1ed ... OK

Starting kernel...
```

rootfs

- W celu utrzymania łańcucha zaufania:
  - System plików root, powinien być dostępny tylko do odczytu
    - Squashfs
  - Uwierzytelnienie rootfs
    - dm-verity:
      - Infrastruktura uwierzytelnienia danych na obrazie squashfs
      - Wymaga narzędzi w przestrzeni użytkownika (umieszczane w initramfs)
      - Zamyka łańcuch zaufania

Pełen obraz


u-boot
check signature
OK
NOK
source script
u-boot env script
fail to source it
bootargs
hash
offset
In FitImage
kernel
dm-verity
initramfs
init script = veritysetup + switch-root
verification
hash
offset
ubiblk0
squashfs
hash.img
OK
NOK
/dev/mapper/rootfs
fail to create mapper device
switch-root
squashfs

Podsumowanie