

# Formation fitImage

## Introduction

Mickaël Tansorier

Retours d'expérience sur le fonctionnement des fitImage et  
de la signatures des images incluses

# Plan

- 1 Introduction
- 2 Construire fitImage
- 3 Gestion dans Buildroot

# Les formats d'images Linux

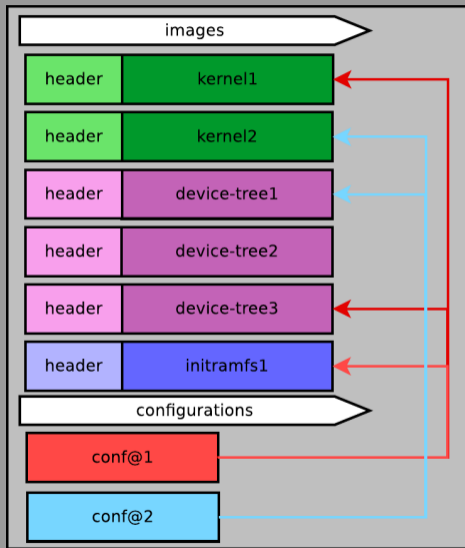
- **Image** : image générique binaire
- **zImage** : image générique binaire compressé
- **ulImage** : image avec une entête d'information utilisé par U-Boot
- **fitImage** : enveloppe d'image pouvant contenir plusieurs noyaux, devicetree, firmware. Chaque image peut être signé, et d'autres choses

# En détails

- zImage :
  - Sujet à la corruption de donnée silencieuse, ce qui peut passer inaperçu
  - Contient seulement une image
  - Utilisation répandue
- uImage :
  - somme de contrôle CRC32 faible
  - Contient seulement une image
  - Utilisation répandue

# En détails

- fitImage
  - Somme de contrôle configurable
  - Peut être signé
  - Peut contenir de multiples images (kernel, DTB, firmware. . . )
  - N'est pas beaucoup utilisé
  - Est le successeur de ulmage
  - Le descripteur de contenu est basé sur un DTS
  - Peut contenir de multiples configurations
  - De nouvelles fonctionnalités d'image peuvent être ajoutées au besoin
  - Supporte de forts checksums (SHA1, SHA256. . . ), Ce qui protège des corruptions silencieuses
  - U-Boot peut vérifier le fitImage avec une clé publique, ce qui protège contre la falsification
  - Le système de construction de Linux ne permet pas de générer une fitImage
  - Yocto peut maintenant générer une fitImage



# Construire une fitImage



## Descripteur fitImage

```
KERNEL=/path/to/zImage
KEYNAME=my_key
```

## fitImage.its

```
/dts-v1/;

/ {
    description = "fitImage for sign Kernel image
                  and DTB";
    #address-cells = <1>;

    images {
        kernel@1 {
            description = "Linux Kenel";
            data = /incbin/("%KERNEL%");
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x12000000>;
            entry = <0x12000000>;
            signature@1 {
                algo = "sha256,rsa4096";
                key-name-hint = "%KEYNAME%";
            };
        };
    };
};
```

```
fdt@1 {
    description = "Devicetree";
    data = /incbin/("%DTB%");
    type = "flat_dt";
    arch = "arm";
    compression = "none";
    load = <0x18000000>;
    entry = <0x18000000>;
    signature@1 {
        algo = "sha256,rsa4096";
        key-name-hint = "%KEYNAME%";
    };
};

};

configurations {
    default = "conf@1";
    conf@1 {
        kernel = "kernel@1";
        fdt = "fdt@1";
    };
};

};
```

# Comment choisir son algo ?

Avant novembre 2016 : Plusieurs type de signature sont disponible dans uboot.

common/image-sig.c

```
struct image_sig_algo image_sig_algos[] = {
    {
        "sha1,rsa2048",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[0],
    },
    {
        "sha256,rsa2048",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[1],
    },
    {
        "sha256,rsa4096",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[2],
    }
};
```

## Après novembre 2016 :

```
struct crypto_algo *image_get_crypto_algo(const char *full_name);
```

```
struct checksum_algo checksum_algos[] = {
    {
        .name = "sha1",
        .checksum_len = SHA1_SUM_LEN,
        .der_len = SHA1_DER_LEN,
        .der_prefix = sha1_der_prefix,
        #if IMAGE_ENABLE_SIGN
        .calculate_sign = EVP_sha1,
        #endif
        .calculate = hash_calculate,
    },
    {
        .name = "sha256",
        .checksum_len = SHA256_SUM_LEN,
        .der_len = SHA256_DER_LEN,
        .der_prefix = sha256_der_prefix,
        #if IMAGE_ENABLE_SIGN
        .calculate_sign = EVP_sha256,
        #endif
        .calculate = hash_calculate,
    }
};
```

```
struct crypto_algo crypto_algos[] = {
    {
        .name = "rsa2048",
        .key_len = RSA2048_BYTES,
        .sign = rsa_sign,
        .add_verify_data = rsa_add_verify_data,
        .verify = rsa_verify,
    },
    {
        .name = "rsa4096",
        .key_len = RSA4096_BYTES,
        .sign = rsa_sign,
        .add_verify_data = rsa_add_verify_data,
        .verify = rsa_verify,
    }
};
```

# Ajouter les clés dans Uboot

- Générer une paire de clé (ex : avec openssl)
- Ajouter les clé à un `external dtb`
- Ajouter l'`external dtb` à la compilation d'Uboot

## Créer un devicetree<sup>1</sup> spécifique. u-boot\_pubkey.dts

```
/dts-v1/;
/ {
    model = "Keys";
    compatible = "vendor,board";
    signature {
        key-%KEYNAME% {
            required = "image";
            algo = "sha256,rsa4096";
            key-name-hint = "%KEYNAME%";
        };
    };
};
```

---

1. arborescence de périphériques

Pour le générer le dtb :

```
$ dtc -p 4096 $(@D)/u-boot_pubkey.dts -O dtb -o $(@D)/u-boot_pubkey.dtb
```

l'option `-p 4096` permet de réserver un espace pour accueillir la clé.

La clé n'est pas présente :

```
$ cat u-boot_pubkey.dtb
vendor,board signature key-my_key image sha256,rsa4096 my_key
modelcompatiblerequiredalgokey -name -hint
```



Pour y ajouter la clé public :

```
$ mkimage -D "-I dts -O dtb -p 4096" -f $(@D)/fitImage.its -K $(@D)/u-  
boot_pubkey.dtb -k $(@D) -r fitImage
```

Ce qui donne :

```
$ cat u-boot_pubkey.dtb  
vendor,board signature key-my_key  
[...]  
image sha256,rsa4096 my_key modelcompatiblerequiredalgokey-name-  
hintrsa,num-bitrsa,n0-inversersa,exponentrsa,modulusra,r-  
squaredsquared
```

Pour s'assurer que le devicetree contenant la clé public soit dans dans le binaire u-boot, il faut que cette option soit à non :

```
BR2_TARGET_UBOOT_USE_CUSTOM_CONFIG
```

Pour ajouter ce DTB spécifique dans U-boot (même s'il n'y a pas de dtb) il faut utiliser l'option EXT\_DTB de make :

```
$ make CROSS_COMPILE=arm-linux-gnueabihf - EXT_DTB=u-boot_pubkey.dtb
```

Ok maintenant il faut signer le noyau linux.

Il faut le signer avant de compiler U-boot puisque pour ajouter la clé public au devicetree il faut executer mkimage avec en entrée l'its et en sortie le fitImage :

```
$ mkimage -D "-I dts -O dtb -p 4096" -f $(@D)/fitImage.its -K $(@D)/u-  
boot_pubkey.dtb -k $(@D) -r fitImage
```

Dans BR le noyau Linux est bien construit avant U-boot, donc pas besoin d'ajouter de dépendance.

# Gestion dans Buildroot

```
--- a/boot/uboot/Config.in
+++ b/boot/uboot/Config.in
@@ -167,6 +167,39 @@ config BR2_TARGET_UBOOT_NEEDS_OPENSSL
     typically the case when the board configuration has
     CONFIG_FIT_SIGNATURE enabled.

+if BR2_TARGET_UBOOT_NEEDS_OPENSSL
+
+config BR2_TARGET_UBOOT_SIGN_FITIMAGE
+    bool "Sign fitImage"
+    depends on BR2_PACKAGE_HOST_UBOOT_TOOLS_FIT_SIGNATURE_SUPPORT
+    help
+        Sign fitImage. This need external dtb for uboot, its file and openssl key.
+
+if BR2_TARGET_UBOOT_SIGN_FITIMAGE
+
+config BR2_TARGET_UBOOT_ITS
+    string "its file"
+    help
+        its file need to have absolute path.
+
+config BR2_TARGET_UBOOT_SIGN_DTS
+    string "dts key file"
+    help
+        dts file need to have absolute path. This will use as external dtb.
+
+config BR2_TARGET_UBOOT_KEY_NAME
+    string "Keys name"
+    help
+        Name of public and private key
+
+config BR2_TARGET_UBOOT_KEY_SERVER
+    string "Keys server"
+    help
+        Server adress to get keys
+endif
+endif
+
config BR2_TARGET_UBOOT_NEEDS_LZOP
    bool "U-Boot needs lzop"
    help
```

```

--- a/boot/uboot/uboot.mk
+++ b/boot/uboot/uboot.mk
@@ -246,5 +246,30 @@ define UBOOT_HELP_CMDS

+# Sign fitImage
+ifneq ($(call qstrip,$(BR2_TARGET_UBOOT_SIGN_FITIMAGE)),)
+UBOOT_MAKE_OPTS += EXT_DTB="$(@D)/u-boot_pubkey.dtb"
+endif
+
+ifneq ($(BR2_TARGET_UBOOT_SIGN_FITIMAGE),)
+UBOOT_ITS_PATH = $(call qstrip,$(BR2_TARGET_UBOOT_ITS))
+UBOOT_EXT_DTS = $(call qstrip,$(BR2_TARGET_UBOOT_SIGN_DTS))
+UBOOT_KEY_NAME = $(call qstrip,$(BR2_TARGET_UBOOT_KEY_NAME))
+UBOOT_KEY_SERVER = $(call qstrip,$(BR2_TARGET_UBOOT_KEY_SERVER))
+DTS_NAME = $(call qstrip,$(BR2_LINUX_KERNEL_INTREE_DTS_NAME))
+define UBOOT_SIGN_FITIMAGE
+  wget $(UBOOT_KEY_SERVER)/$(UBOOT_KEY_NAME).key -O $(@D)/$(UBOOT_KEY_NAME).key
+  wget $(UBOOT_KEY_SERVER)/$(UBOOT_KEY_NAME).cert -O $(@D)/$(UBOOT_KEY_NAME).cert
+  sed -e "s|%(KERNEL%)|$(BINARIES_DIR)/zImage|" $(UBOOT_ITS_PATH) > $(@D)/fitImage.its
+  sed -e "s|%(DTB%)|$(BINARIES_DIR)/$(DTS_NAME).dtb|" -i $(@D)/fitImage.its
+  sed -e "s|%(KEYNAME%)|$(UBOOT_KEY_NAME)|" -i $(@D)/fitImage.its
+  sed -e "s|%(KEYNAME%)|$(UBOOT_KEY_NAME)|" $(UBOOT_EXT_DTS) > $(@D)/u-boot_pubkey.dts
+  $(HOST_DIR)/bin/dtc -p 4096 $(@D)/u-boot_pubkey.dts -O dtb -o $(@D)/u-boot_pubkey.dtb
+  PATH=$(PATH):$(HOST_DIR)/bin $(HOST_DIR)/bin/mkimage -D "-I dts -O dtb -p 4096" -f $(@D)/fitImage.its -K $(@D)/u-
    boot_pubkey.dtb -k $(@D) -r $(BINARIES_DIR)/fitImage
+endif
+UBOOT_PRE_BUILD_HOOKS += UBOOT_SIGN_FITIMAGE
+endif
+
+UBOOT_CUSTOM_DTS_PATH = $(call qstrip,$(BR2_TARGET_UBOOT_CUSTOM_DTS_PATH))

define UBOOT_BUILD_CMDS
@@ -329,6 +353,11 @@ endif

+define UBOOT_REMOVE_KEY
+  rm -f $(@D)/$(UBOOT_KEY_NAME).key $(@D)/$(UBOOT_KEY_NAME).cert $(@D)/$(UBOOT_KEY_NAME).pub
+endif
+UBOOT_POST_INSTALL_IMAGES_HOOKS += UBOOT_REMOVE_KEY
+
define UBOOT_INSTALL_OMAP_IPT_IMAGE
  cp -dpf $(@D)/$(UBOOT_BIN_IPT) $(BINARIES_DIR)/
endif

```

## Documentation :

- <https://elinux.org/images/e/e0/Josserand-schulz-secure-boot.pdf>
- [https://www.denx.de/wiki/pub/U-Boot/Documentation/multi\\_image\\_booting\\_scenarios.pdf](https://www.denx.de/wiki/pub/U-Boot/Documentation/multi_image_booting_scenarios.pdf)
- [https://elinux.org/images/8/8a/Vasut--secure\\_and\\_flexible\\_boot\\_with\\_u-boot\\_bootloader.pdf](https://elinux.org/images/8/8a/Vasut--secure_and_flexible_boot_with_u-boot_bootloader.pdf)

# Des question ?

Enfin je vais essayer de répondre...

`mickael@tansorier.fr`

GNU Free Documentation License, Version 1.3