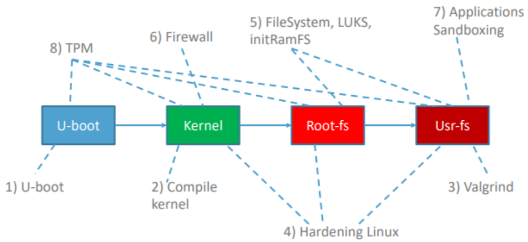
1 Introduction



Attaques de surface : Utilisateurs des ports de debug, connecteurs, Alimentations

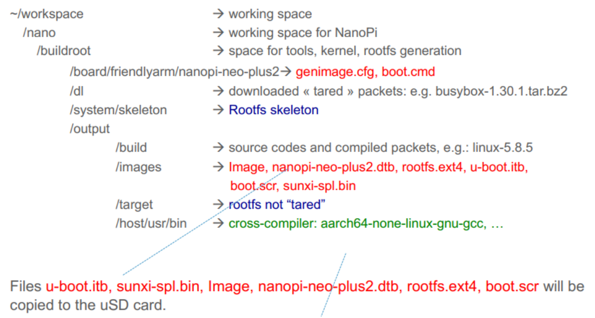
Vecteurs d'attaque : Réseau (Ethernet, Wifi), application, port série, USB, I2C, Flash, Bluetooth, GPS, etc...

1.1 Compilation pour nanopi

Cross-compilation (ARM) effectuée sur un système x86/x64. Buildroot est le toolchain utilisé. Les éléments suivants sont compilés : Bootloader, Kernel, Rootfs
Puis les images sont copiées sur la carte SD

2 Buildroot

2.1 Répertoires

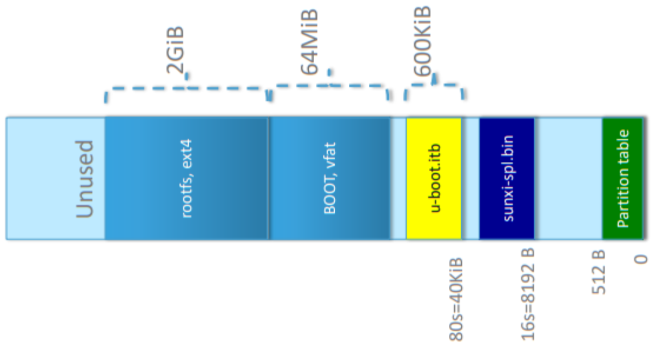


Ce qui est manquant dans le dossier output sera recompilé lorsque la commande `make` est lancée (ou alors en faisant la commande
`make <package>-rebuild`.
Le dossier `rootfs_overlay` permet d'ajouter des fichiers au `rootfs`
(`/workspace/nano/buildrootboard/friendlyarm/nanopi-neo-plus2/rootfs_overlay`)

2.2 Compilation

Dans le répertoire `buildroot`, effectuer la commande `make menuconfig` puis `make`. `make clean` pour effacer tous les fichiers compilés.
La configuration permet notamment de
1. Modifier le rootfs

2.3 Carte SD



Sector size = 512 Bytes

Area Name	From (Sector #)	To (Sector #)	Size
rootfs, ext4			2GiB
BOOT, vfat			64MiB
U-boot-itb	80		600KiB
sunxi-spl.bin	16	79	32KiB
MBR (partition table)	0	15	512B

`genimage.cfg`⟶`genimage`⟶`sdcard.img`⟶`dd`⟶ carte SD

Les fichiers pour l'initialisation sont

`rootfs.ext`	Root file system
`Image`	Noyau Linux
`nanopo-neo-plus2.dtb`	Flattened device tree
`boot.scr`	Commandes boot compilées utilisées par u-boot
`boot.vfat`	Partition boot
`u-boot.itb`	Boot loader
`sunxi-spl.bin`	Secondary Program Loader

`boot.vfat` contient `Image`, `nanopi-neo-plus2.dtb` et `boot.scr`. `boot.vfat` (ou `boot.ext4`) permet de créer `BOOT` sur la carte SD

2.3.1 rootfs

Contient `/bin`, `/sbin`, `/root`, `/etc`, etc...

2.3.2 boot.scr

Le fichier `boot.scr` est utilisé par u-boot pour charger le kernel Linux. Il est créé avec la commande `mkimage`

2.3.3 boot.cmd

`boot.cmd` contient des informations de démarrage, notamment les emplacements des différents l'emplacement de `nanopi-neo-plus2.dtb`, du kernel et (si présent) de l'initramfs

2.4 Séquence de démarrage (6 phases)

1. Lorsque le μP est mis sous tension, le code stocké dans son BROM va charger dans ses 32KiB de SRAM interne le firmware `sunxi-spl` stocké dans le secteur no 16 de la carte SD / eMMC et l'exécuter.
2. Le firmware `sunxi-spl` (Secondary Program Loader) initialise les couches basses du μP, puis charge l'U-Boot dans la RAM du μP avant de le lancer.
3. L'U-Boot va effectuer les initialisations hardware nécessaires (horloges, contrôleurs, ...) avant de charger l'image non compressées du noyau Linux dans la RAM, le fichier `Image`, ainsi que le fichier de configuration FDT (flattened device tree).
4. L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (bootargs)
5. Le noyau Linux procédera à son initialisation sur la base des bootargs et des éléments de configuration contenus dans le fichier FDT (sun50i-h5-nanopi-neo plus2.dtb).
6. Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, ...) et poursuivra son exécution.

Voir 7.10.1 pour l'illustration

3 U-boot

3.1 Compilation

On configure avec `make uboot-menuconfig` puis on effectue la compilation avec une des deux manières :
1. `make uboot-rebuild`
2. supprimer les fichiers puis `make`

La configuration de u-boot est stockée dans `.config`

3.1.1 Amélioration

Il est possible d'utiliser l'argument `-fstack-protector-all` pour ajouter des vérifications contre les buffer overflows (ou autres attaques sur le stack). Dans ce cas, un canary. Si le canary est écrasé lors de l'éxécution d'un morceau de code. On sais qu'il y a eut un dépassement dans le stack.

3.2 Démarrage

Si on appuie sur une touche, on entre en mode u-boot. La commande `booti` permet de lancer l'image linux (`boot` tout court va aussi lancer l'image Linux). Il existe aussi bootz pour charger une image compressée et bootm pour une image fit

Avec les commandes présentes dans `boot.cmd`, on indique l'emplacement dans la ram de `Image` et `nanopi-neo-plus.dtb`

Lors du démarrage, le Secondary Program Loader (sunxi-spl) va charger le fichier `u-boot.itb`

3.3 FDT (Flattened Device-Tree)

Le FDT contient une description hardware du système utilisée par Linux pour sa configuration. le FDT utilise deux fichiers :

- `.dts` : Device Tree Source (fichier ascii)
- `.dtb` : Device Tree Blob (fichier binaire)

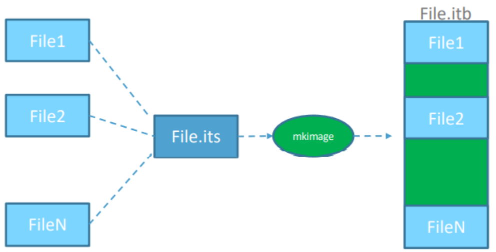
Possibilité de passer de `.dts` à `.dtb` avec la commande `dtc`.

U-boot utilise le fichier `sun50i-h5-nanopi-neo-plus2.dts` pour configurer Linux sur le NanoPi (information sur le port série, le processeur, etc...).

Le FDT est stocké dans le fichier `u-boot.itb`

3.4 FIT (Flattened Image Tree)

Nouveau format qui permet d'insérer plusieurs fichiers dans un seul



La commande `mkimage` permet de convertir un fichier `.its` en un fichier `.itb`.

Le fichier `u-boot.itb` est construit avec la commande

```
mkimage -f u-boot.its -E u-boot.itb
```

4 Kernel

4.1 Compilation

On configure avec `make linux-menuconfig` (ou `make linux-xconfig`) puis on lance une compilation avec `make linux-rebuild`

4.1.1 Amélioration

Comme pour u-boot, on peut utiliser `-fstack-protector-all` pour ajouter une protection sur le stack.

Dans le menu, cela se traduit par l'activation de "Strong Stack Protector" et "Stack Protector buffer overflow protection".

Le "Randomize va space" permet de placer les éléments à des emplacements mémoire aléatoires (pour éviter d'en cibler un facilement).

Il est également possible d'optimiser le kernel pour la place OU pour les performances.

Il faut absolument strip avant de déployer (il existe une option pour strip le linux dans le menuconfig).

On peut également restreindre l'accès au syslog (system logs).

La mise à 0 lors de l'allocation et/ou free de mémoire permet aussi d'éviter certaines attaques

4.2 Busybox

Busybox est un éxécutable qui combine beaucoup de fonctions de base (ls, mv, rm, cat, etc...). En mettant toutes ces commandes dans un seul programme, on réduit énormément les redondances et par conséquent la taille de l'éxécutable.

On peut également configurer busybox avec `make busybox-menuconfig` puis le compiler avec `make busybox-rebuild`

4.3 File systems

Les filesystems doivent être activés dans la configuration de Linux afin d'être utilisables

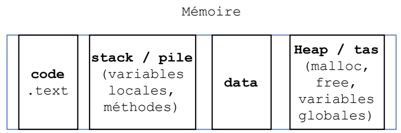
4.4 Réseau

Si le système n'est pas un routeur, on peut choisir de désactiver le routage et le `rp_filter` doit être activé sur toutes les interfaces.

4.5 Outils

- Générateurs de nombres aléatoires

4.6 Attaques



Éxecution sur le stack : Ii le stack est éxécutable, il est possible d'y placer du code puis de l'éxécuter (ce qui est de moins en moins le cas).

ret2libc : Permet de bypasser la non-éxécution du stack. Consiste à éxécuter du code dans une librairie comme libc.

ROP : Return-Oriented Programming. Éxécution de code malveillant à l'intérieur du programme lui-même

4.7 Protections

ASLR : Address Space Layout Randomization (changement des adresses de stack et heap) afin d'éviter les attaques `ret2libc`

PIE : Position de l'exécutable modifiée pour éviter qu'une attaque soit faite (ou la rendre plus difficile)

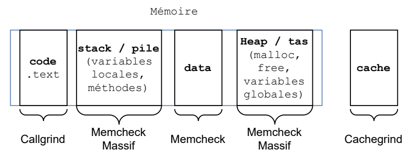
Canary : Variable qui permet de détecter un dépassement dans le stack

5 Valgrind

5.1 Outils de Valgrind

- Memcheck : Détection d'erreur mémoire
- Cachegrind : Profiler de mémoire cache
- Callgrind : Profiler de cache avec infos supp et graph
- Helgrind : Détection d'erreur de threads 1
- DRD : Détection d'erreur de threads 2
- Massif : Profiler de heap et stack (memory leak)
- DHAT : Profiler de bloc dans le heap

5.2 Utilisation des outils



5.3 Trouver le bon outil

L'outil Memcheck regroupe beaucoup de fonctionnalité. C'est lui qu'il faut utilisé en priorité

6 Hardening

6.1 Intégrité package, programme

Aller voir la section 9.1.4.

```
gpg --verify "package"
gpg --keyserver keyserver.ubuntu.com --search-
    keys "KEY"
```

6.2 Configurer un package, programme

```
tar xvzf package1.tar.gz
cd package1
less INSTALL or less README #Analyze the
    different options
./configure --help
```

6.3 Cross-compiler un programme

Ajout de host et prefix

```
./configure --host=aarch64-none-linux-gnu --
    prefix=/home/dir
make
make install
```

6.4 Contrôler les services, les ports ouverts

- ps -ale : montre tous les process
- ps -aux : montre les droits des process
- lsof : montre les ports ouverts
- nmap : montre les ports ouvert sur à une IP

6.5 De contrôler les file systems

A FAIRE

6.6 Permissions des fichiers, dossiers

```
ls -al => -rwxrwxrwx usr grp ..... t.txt
chmod 755 t.txt => -rwxr-xr-x usr grp .....t.txt
```

6.7 Sécuriser le réseau

- Désactiver l'IPv6
- Désactiver le routage source IP
- Désactiver le port forwarding
- Bloquer la redirection des msg ICMP
- Activer la vérification de routage source
- Log paquet erroné et ignore bogus ICMP
- Désactiver ICMP echo et temps
- Activer syn cookies (pour TCP)

6.8 Contrôler-sécuriser user

Modifier le umask à 0027 réduit les droits $Droit = umask \& 0777$

6.9 Limiter le login root

```
chmod 700 /root #limite l'accès au dossier root
sudo #pour avoir les droits root
```

Ne pas mettre le . dans la path

6.10 Sécuriser le noyau

Aller voir dans la section 4.7

6.11 Sécuriser une application

Activer l'option de compilation $-fstack-protector-all$ et $noexecstack$

```
gcc -Wall -Wextra -z noexecstack -pie -fPIE -
    fstack-protector-all -Wl,-z,relro,-z,now -O
    -D_FORTIFY_SOURCE=2 -ftrapv -o test test.c
```

7 File system

7.1 Génération

Squelette de rootfs dans `workspace/nano/buildroot/system/skeleton`. Il est ensuite copié dans `buildroot/output/target` et les fichiers nécessaires y sont ensuite ajoutés.
Une fois que tous les fichiers sont ajoutés, une image `rootfs.xxx` est créé (xxx est ext4, squashfs, etc...)

7.2 Types de systèmes de fichier et leurs applications

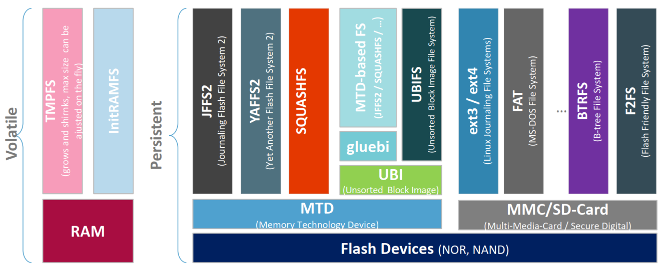
Pour les systèmes embarqués, il existe deux catégories de systèmes de fichiers :

1. Volatiles en RAM
2. Persitants sur des Flash (NOR et de plus en plus NAND)

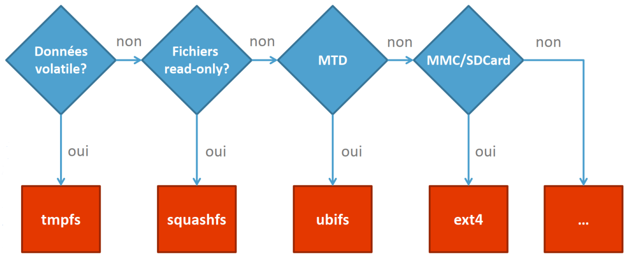
Deux technologies principales sont disponible sur les Flash :

- MTD (Memory Technology Device)
- MMC/SD-Card (Multi-Media-Card / Secure Digital Card)

7.2.1 FS types



7.2.2 Choix d'un FS



7.2.3 MMC technologies

MMC-eMMC-SD Card is composed by 3 elements

- MMC interface: handle communication with host
- FTL (Flash translation layer)
- Storage area: array of NAND chips

7.2.4 FTL

FTL is a small controller running a firmware. Its main purpose is to transform logical sector addressing into NAND addressing. It also handles:

- Bad block management
- Garbage collection.
- Wear levelling

7.3 Caractéristiques des filesystems ext2-3-4 et commande

"Filesystem considerations for embedded devices" is a good study about filesystems used on embedded systems. This file system is very used in different Linux distribution.

- EXT filesystem was created in April 1992 and is a file system for the Linux kernel
  - Ext2 is not a journaled file system
  - Ext2 uses block mapping in order to reduce file fragmentation (it allocates several free blocks)
  - After an unexpected power failure or system crash (also called an unclean system shutdown), each mounted ext2 file system on the machine must be checked for consistency with the e2fsck program
- EXT2 replaced it in 1993
  - It was merged in the 2.4.15 kernel on November 2001
  - Ext3 is compatible with ext2
  - Ext3 is a journaled file system
  - The ext3 file system prevents loss of data integrity even when an unclean system shutdown occurs
- EXT4 arrived as a stable version in the Linux kernel in 2008

- ext4 is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 as ext4
- Ext4 is included in the kernel 2.6.28
- Ext4 supports Large file system:
  - Volume max: 2^{60} bytes
  - File max: 2^{40} bytes
- Ext4 uses extents (as opposed to the traditional block mapping scheme used by ext2 and ext3), which improves performance when using large files and reduces metadata overhead for large files

7.3.1 Ext4 commands

```
# Create a partition (rootfs), start 64MB,
length 256MB
sudo parted /dev/sdb mkpart primary ext4
131072s 655359s
# Format the partition with the volume label
 = rootfs
sudo mkfs.ext4 /dev/sdb1 -L rootfs
# Modify (on the fly) the ext4 configuration
sudo tune2fs <options> /dev/sdb1
# check the ext4 configuration
mount
sudo tune2fs -l /dev/sdb1
sudo dumpe2fs /dev/sdb1
# mount an ext4 file system
mount -t ext4 /dev/sdb1 /mnt/test // with
default options
mount -t ext4 -o defaults,noatime,discard,
nodiratime,data=writeback,acl,user_xattr
/dev/sdb1 /mnt/test
```

7.3.2 Ext4 mount options and MMC/SD-Card

- filesystem options can be activated with the mount command (or to the /etc/fstab file)
- These options can be modified with tune2fs command
- Journaling: the journaling guarantees the data consistency, but it reduces the file system performances
- MMC/SD-Card constraints: In order to improve the longevity of MMC/SDCard, it is necessary to reduce the unnecessary writes
- Mount options to reduce the unnecessary writes (man mount) :
  - noatime: Do not update inode access times on this filesystem (e.g., for faster access on the news spool to speed up news servers)
  - nodiratime: Do not update directory inode access times on this filesystem
  - relatime: this option can replace the noatime and nodiratime if an application needs the access time information (like mutt)

Mount options for the journaling (man ext4):

- Data=journal: All data is committed into the journal prior to being written into the main filesystem (It is the safest option in terms of data integrity and reliability, though maybe not so much for performance
- Data=ordered: This is the default mode. All data is forced directly out to the main file system before the metadata being committed to the journal
- Data=writeback: Data ordering is not preserved - data may be written into the main filesystem after its metadata has been committed to the journal. It guarantees internal filesystem integrity, however it can allow old data to appear in files after a crash and journal recovery.
- Discard: Use discard requests to inform the storage that a given range of blocks is no longer in use. A MMC/SD-Card can use this information to free up space internally, using the free blocks for wear-levelling.
- acl: Support POSIX Access Control Lists
- user_xattr: Support ”user.” extended attributes
- default : rw, suid, dev, exec, auto, nouser, and async
  - rw : read-write
  - suid : Allow set-user-identifier or set-group-identifier bits
  - dev : Interpret character or block special devices on the filesystem
  - exec : Permit execution of binaries
  - auto : Can be mounted with the -a option (mount –a)
  - nouser : Forbid an ordinary (i.e., non-root) user to mount the filesystem
  - async : All I/O to the filesystem should be done asynchronously

7.3.3 /etc/fstab file

File **/etc/fstab** contains descriptive information about the filesystems the system can mount

- file system : block special device or remote filesystem to be mounted
- mount pt : mount point for the filesystem
- type : the filesystem type
- options : mount options associated with the filesystem
- dump : used by the dump (backup filesystem) command to determine whichfilesystems need to be dumped (0 = no backup)
- pass : used by the fsck (8) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with 1, and other filesystems should have a 2. if ¡pass¿ is not present or equal 0 -¿ fsck willassume that the filesystem is not checked.
- Field options: It contains at least the type of mount plus any additional options appropriate to the filesystem type.

Common for all types of file system are the options (man mount) :

- auto : Can be mounted with the -a option (mount -a)
- defaults : Use default options: rw, suid, dev, exec, auto, nouser, and async
- nosuid : Do not allow set-user-identifier or set-group-identifier bits to take effect
- noexec : Do not allow direct execution of any binaries on the mounted file system
- nodev : Do not interpret character or block special devices on the file system

7.4 Autre type de FS

7.4.1 BTRFS (B-Tree filesystem)

- BTRFS is a “new” file system compared to EXT. It is originally created by Oracle in 2007, it is a B-Tree filesystem
- It is considered stable since 2014
- Since 2015 BTRFS is the default rootfs for openSUSE
- BTRFS inspires from both Reiserfs and ZFS
- Theodore Ts'o (ext3-ext4 main developer) said that BTRFS has a better direction than ext4 because ”it offers improvements in scalability, reliability, and ease of management”

7.4.2 F2FS (Flash-Friendly File System)

It is a log filesystem. It can be tuned using many parameters to allow best handling on different supports.
F2FS features :

- Atomic operations
- Defragmentation
- TRIM support (reporting free blocks for reuse)

7.4.3 NILFS2 (New Implementation of a Log-structured File System)

- Developed by Nippon Telegraph and Telephone Corporation
- NILFS2 Merged in Linux kernel version 2.6.30
- NILFS2 is a log filesystem
- CoW for checkpoints and snapshots
- Userspace garbage collector

7.4.4 XFS (Flash-Friendly File System)

XFS was developed by SGI in 1993.

- Added to Linux kernel in 2001
- On disk format updated in Linux version 3.10
- XFS is a journaling filesystem
- Supports huge filesystems
- Designed for scalability
- Does not seem to be handling power loss (standby state) well

7.4.5 ZFS (Zettabyte (10^{21})File System)

ZFS is a combined file system and logical volume manager designed by Sun Microsystems.

- ZFS is a B-Tree file system
- Provides strong data integrity
- Supports huge filesystems
- Not intended for embedded systems (requires RAM)
- License not compatible with Linux

7.4.6 Conclusion

Performances:

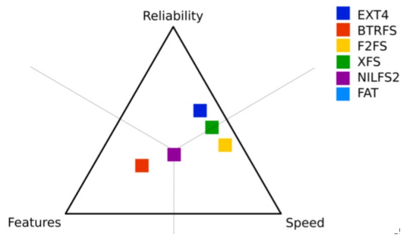
- EXT4 is currently the best solution for embedded systems using MMC
- F2FS and NILFS2 show impressive write performances

Features:

- BTRFS is a next generation filesystem
- NILFS2 provides simpler but similar features

Scalability:

- EXT4 clearly doesn't scale as well as BTRFS and F2FS



7.5 Architecture des FS

7.5.1 Journalized filesystem

A journalized filesystem keeps track of every modification in a journal in a dedicated area

- EXT3, EXT4, XFS, Reiser4
- Journal allows to restore a corrupted filesystem
- Modifications are first recorded in the journal
- Modifications are applied on the disk
- If a corruption occurs: The File System will either keep or drop the modifications
  - Journal is consistent : we replay the journal at mount time
  - Journal is not consistent : we drop the modifications

7.5.2 B-Tree filesystem

- ZFS, BTRFS, NILFS2
- B+ tree is a data structure that generalized binary trees
- CoW (Copy on Write) is used to ensure no corruption occurs at runtime :
  - The original storage is never modified. When a write request is made, data is written to a new storage area
  - Original storage is preserved until modifications are committed
  - If an interruption occurs during writing the new storage area, original storage can be used

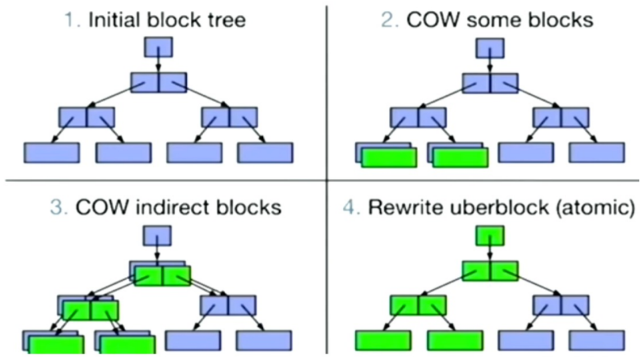
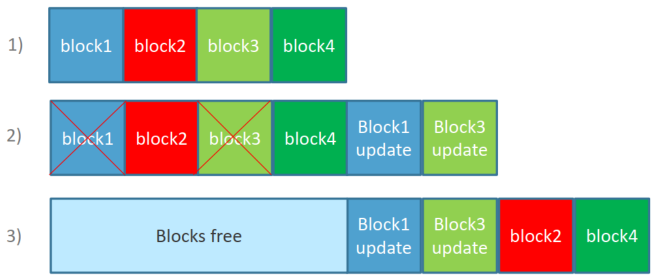


Figure 1: B-tree type FS execution

7.5.3 Log filesystem

Log-structured filesystems use the storage medium as circular buffer and new blocks are always written to the end.

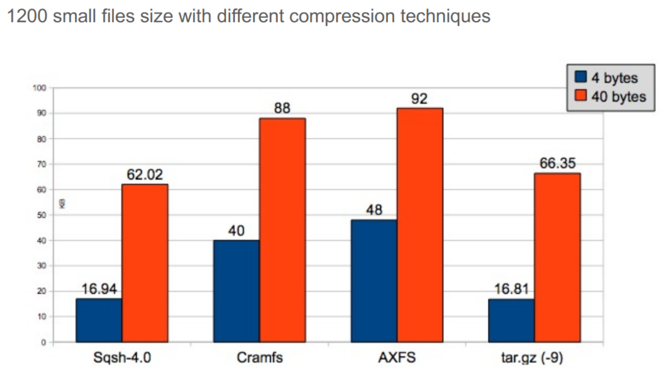
- F2FS, NILFS2, JFFS2, UBIFS
- Log-structured filesystems are often used for flash media since they will naturally perform wear-levelling
- The log-structured approach is a specific form of copy-on-write behavior



1. Initial state
2. Block 1-3 are updated, old blocks 1-3 are not used
3. Garbage copies block2 and 4, and frees old block1-2-3-4

7.6 Caractéristiques de Squashfs et commande

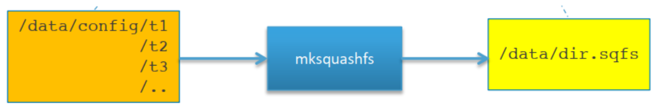
- Squashfs is a compressed read-only filesystem for Linux
- Squashfs versions
  - Squashfs 2.0 and squashfs 2.1: 2004, kernel 2.2
  - Squashfs 3.0: 2006, kernel 2.6.12
  - Squashfs 4.2: 2011, kernel 2.6.29
- It uses gzip, lzma, lzo, lz4 and xz compression to compress files, inodes and directories
- SquashFS 4.0 supports 64 bit filesystems and files (larger than 4GB), full uid/gid information, hard links and timestamps
- Squashfs is intended for general read-only filesystem use, for archival use, and in embedded systems with small processors where low overhead is needed



7.6.1 Create and use squashed file systems

1. Create the squashed file system dir.sqsh for the regular directory /data/config/ :

```
bash# mksquashfs /data/config/ /data/
dir.sqsh
```



2. The mount command is used with a loopback device in order to read the squashed file system dir.sqsh

```
bash# mkdir /mnt/dir
bash# mount -o loop -t squashfs /data
/dir.sqsh /mnt/dir
```

```
bash# ls /mnt/dir
```

3. It is possible to copy the dir.sqsh to an unmounted partition (e.g. /dev/sdb2) with the dd command and next to mount the partition as squashfs file system

```
     bash# umount /dev/sdb2
     bash# dd if=dir.sqsh of=/dev/sdb2
     bash# mount /dev/sdb2 /mnt/dir -t
squashfs
     bash# ls /mnt/dir
```

7.7 Caractéristiques de tmpfs et commandes

- Tmpfs is a file system which keeps all files in virtual memory
- Everything in tmpfs is temporary in the sense that no files will be created on your hard drive. If you unmount a tmpfs instance, everything stored therein is lost.
- tmpfs puts everything into the kernel internal caches and grows and shrinks to accommodate the files it contains and is able to swap unneeded pages out to swap space. It has maximum size limits which can be adjusted on the fly via 'mount -o remount ...'
- If you compare it to ramfs you gain swapping and limit checking. Another similar thing is the RAM disk (/dev/ram*), which simulates a fixed size hard disk in physical RAM, where you have to create an ordinary filesystem on top. Ramdisks cannot swap and you do not have the possibility to resize them

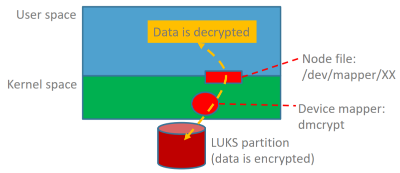
7.7.1 Devtmpfs

- devtmpfs is a file system with automatically populates nodes files (/dev/...) known by the kernel.
- This means, it is not necessary to have udev running nor to create a static /dev layout with additional, unneeded and not present device nodes.
- Instead the kernel populates the appropriate information based on the known devices.
- The kernel executes this command : mount -n -t devtmpfs devtmpfs /dev
- /dev is automatically populated by the kernel with its known devices

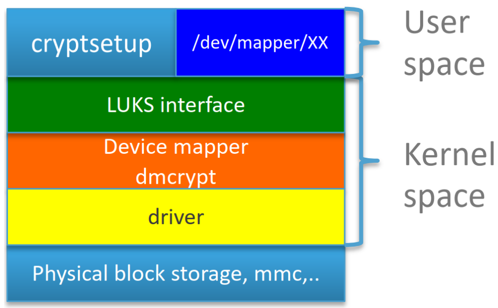
```
# ls /dev
autofs ptypf tty47
btrfs-control random tty48
bus rtc0 tty49
console shm tty5
cpu_dma_latency snapshot tty50
```

7.8 Caractéristiques de LUKSfs et commandes

1. Data in the LUKS partition is encrypted
2. Data used in the user space is decrypted by dmcrypt



- LUKS (Linux Unified Key Setup) is the standard for Linux hard disk encryption
- By providing a standard on-disk-format, it does not only facilitate compatibility among distributions, but also provides secure management of multiple user passwords
- In contrast to existing solution, LUKS stores all necessary setup information in the partition header, enabling the user to transport or migrate his data seamlessly
- LUKS - dmcrypt crypts an entire partition
- Luks features
  - compatibility via standardization
  - secure against attacks
  - support for multiple keys
  - effective passphrase revocation
  - free
- cryptsetup is a utility used to configure dmcrypt
- cryptsetup uses the /dev/random and /dev/urandom node file
- dmcrypt (Device-mapper) crypts target and provides transparent encryption of block devices using the kernel crypto API (kernel configuration, Cryptographic API)
- Device-mapper is included in the Linux 2.6 and 3.x kernel that provides a generic way to create virtual layers of block devices. It is required by LVM2 (Logical Volume Management)
- The user can basically specify one of the symmetric ciphers, an encryption mode, a key (of any allowed size), an iv generation mode and then the user can create a new block device node file in /dev/mapper
- All data written to this device will be encrypted and all data read from this device will be decrypted



7.8.1 Create LUKS partition

```
sudo cryptsetup --debug --pbkdf pbkdf2
    luksFormat $DEVICE
# Be careful, type yes in UPPERCASE
# Or create a LUKS partition for the system with
     enough memory
sudo cryptsetup --debug luksFormat $DEVICE
# Be careful, type yes in UPPERCASE
# Dump the header information of a LUKS device
sudo cryptsetup luksDump $DEVICE
# Create a mapping /dev/mapper/usrfs1 and ask
    the passphrase
sudo cryptsetup --debug open --type luks $DEVICE
     usrfs1
# Show the node file
ls /dev/mapper/
# Format the LUKS partition as ext4 partition
sudo mkfs.ext4 /dev/mapper/usrfs1
# Mount the LUKS partition to /mnt/usrfs
sudo mkdir /mnt/usrfs
sudo mount /dev/mapper/usrfs1 /mnt/usrfs
# Work with the LUKS partition
ls /mnt/usrfs
copy files to /mnt/usrfs
# Unmount the LUKS partition
sudo umount /dev/mapper/usrfs1
# Removes the existing mapping usrfs1 and wipes
    the key from kernel memory
sudo cryptosetup close usrfs1
# dmsetup: low level logical volume management
dmsetup info -C
dmsetup remove -f usrfs1
```

7.8.2 Use LUKS partition

```
# Create a mapping /dev/mapper/usrfs1 and ask
    the passphrase
sudo cryptsetup --debug open --type luks $DEVICE
     usrfs1
# Mount the LUKS partition to /mnt/usrfs
sudo mount /dev/mapper/usrfs1 /mnt/usrfs
# Unmount the LUKS partition
umount /dev/mapper/usrfs1
# Removes the existing mapping usrfs1 and wipes
    the key from kernel memory
cryptosetup close usrfs1
#It is possible to manage a luks partition with
```

```
dmsetup info -c
dmsetup remove -f usrfs1
```

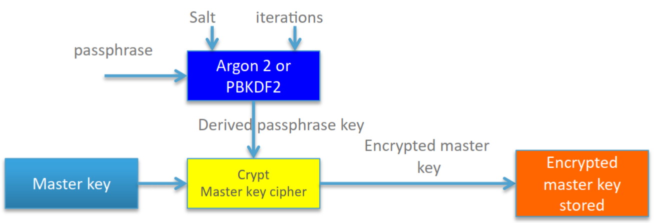
7.9 Gestion des clés de LUKS 42

7.9.1 LUKS - key generation

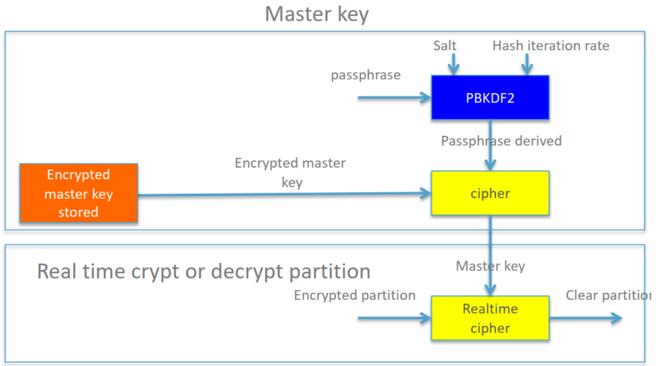
- LUKS uses the TKS1 template in order to generate secure key
- The key is derived from a passphrase
- LUKS supports multiple keys/passphrases
- TKS1 uses Argon2 or PBKDF2 (Password-Based Key Derivation Function 2) method in order to provide a better resistance against brute force attacks based on entropy weak user passphrase.
- TKS1 uses two level hierarchy of cryptographic keys to provide the ability to change passphrases

The system initialization is straight forward :

- A master key is generated
- Passphrase, Salt, iterations and other values are inputs of the functions Argon2 or PBKDF2
- A derived passphrase key is computed by Argon2 or PBKDF2
- The master key is encrypted by the derived passphrase key.
- The encrypted master key, the iteration rate and the salt are stored



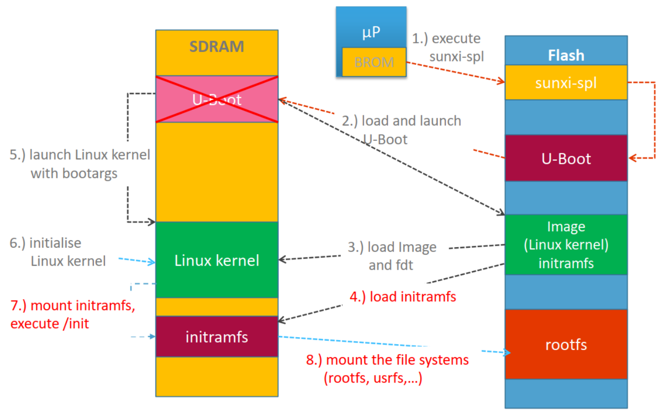

```
# Add a new passphrase to the LUKS partition(
    $DEVICE=/dev/mmcblk0p3 (nanopi) or /dev/sdc3
    (PC))
cryptsetup luksAddKey $DEVICE
# Dump the header information of a LUKS device
cryptsetup luksDump $DEVICE
# Dump the encrypted master key of a LUKS device
cryptsetup luksDump -dump-master-key $DEVICE
# Check /proc/crypto which contains supported
    ciphers and modes (but note it contains only
     currently loaded crypto API modules).
cat /proc/crypto
```



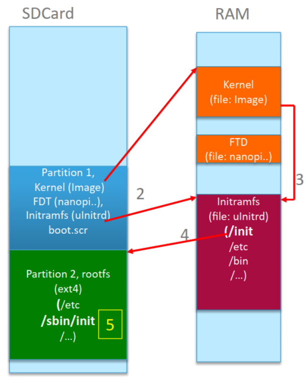
7.10 Caractéristiques de InitramFS et commandes

- initramfs is a root filesystem that is loaded at an early stage of the boot process.
- It is the successor of initrd.
- It provides early userspace commands which lets the system do things that the kernel cannot easily do by itself during the boot process.
- Using initramfs is optional.
- Boot without initramfs:
  - By default, the kernel initializes hardware using built-in drivers, mounts the specified root partition, loads the rootfs and starts the init scritps
  - Init scripts can load additional modules and starts services until it eventually allows users to log in. This is a good default behavior and sufficient for many users
- An initramfs is generally used for advanced requirements; for users who need to perform certain tasks as early as possible, even before the rootfs is mounted.

7.10.1 Boot sequence, with initramfs



1) Kernel (Image), initramfs (uInitrd), flattened device tree (Sun50i...) and boot.scr files are located in the partition 1 of the SDCard
2) Kernel, initramfs, Sun50i.. are copied to the RAM
3) Kernel mounts initramfs (uInitrd file)
4) Kernel executes init script stored in initramfs. This init script can execute early different commands
5) Init script executes the command switch_root, which switches to the standard rootfs located in the partition 2 and executes the /sbin/init command

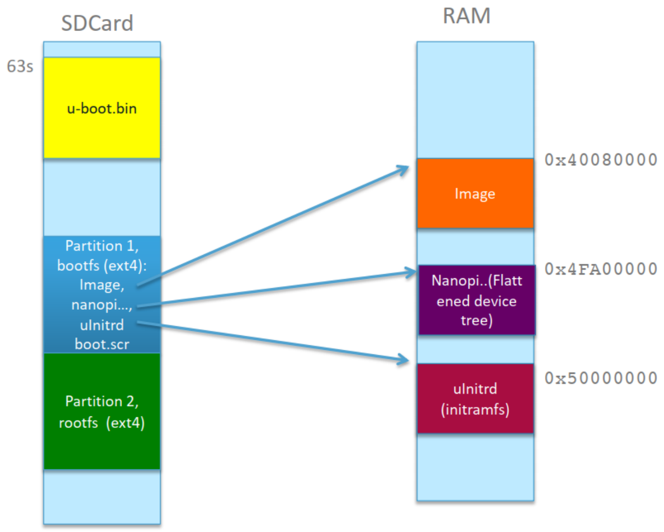


Show boot.cmd file: `cat $HOME/workspace/nano/buildroot/board/friendlyarm/nanopi-neo-plus2/boot.cmd`

```
setenv bootargs console=ttyS0,115200n8 earlyprintk root=/dev/mmcblk0p2 rootwait
ext4load mmc 0 $kernel_addr_r Image
ext4load mmc 0 $fdt_addr_r nanopi-neo-plus2.dtb
ext4load mmc 0 0x50000000 uInitrd          // Load initramfs

booti $kernel_addr_r 0x50000000 $fdt_addr_r
```

initramfs address

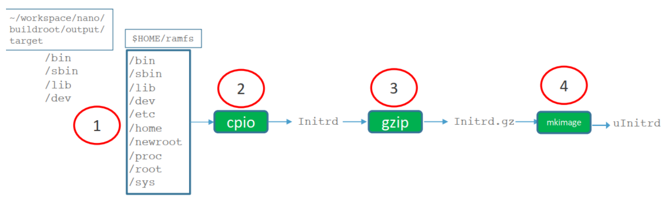


7.11 Création d'un initramFS

On PC, NanoPi rootfs is in this directory: HOME/-workspace/nano/buildroot/output/target/

Principle to build an initramfs :

1. Copy the right files into a directory ($HOME/ramfs)
2. Copy these files in a cpio archive file
3. Compress this file
4. Add the uboot header



7.12 Init script

```
#!/bin/busybox sh
# Init script in the initRamFS
mount -t proc none /proc
mount -t sysfs none /sys
mount -t ext4 /dev/mmcblk0p2 /newroot
mount -n -t devtmpfs devtmpfs /newroot/dev
exec switch_root /newroot /sbin/init
```

7.12.1 Shared library dependency

- Generally, programs are dynamically linked (other possibility: statically linked)
- A dynamically program must have all necessary libraries
- Example (on PC) for the /bin/ls program : **ldd ls**

```
#!/bin/sh
# Copie de tous les fichiers utiles (librairies)
   pour initramfs
ROOTFSLOC=ramfs
HOME=/home/lmi
cd $HOME
mkdir $ROOTFSLOC
mkdir -p $ROOTFSLOC/{bin,dev,etc,home,lib,lib64,
   newroot,proc,root,sbin,sys}
cd $ROOTFSLOC/dev
sudo mknod null c 1 3
sudo mknod mmcblk0p b 179 0
echo " Cpy /bin "
cd ../bin
cp ~/workspace/nano/buildroot/output/target/bin/
   busybox .
ln -s busybox ls
ln -s busybox dmesg
cp ~/workspace/nano/buildroot/output/target/usr/
   bin/strace .
cd ../sbin
ln -s ../bin/busybox switch_root
cd ../lib64
cp ~/workspace/nano/buildroot/output/target/
   lib64/libc-2.31.so .
ln -s libresolv-2.31.so libresolv.so.2
ln -s libc-2.31.so libc.so.6
ln -s ../lib64/ld-2.31.so ld-linux-aarch64.so.1
cd ../lib
cp ~/workspace/nano/buildroot/output/target/
   lib64/ld-2.31.so .
ln -s ../lib64/ld-2.31.so ld-linux-aarch64.so.1
cd ..
cat > init << endofinput # copie du script
   suivant dans init
# Script d'exécution après le boot (linux du
   initramfs)
#!/bin/busybox sh
mount -t proc none /proc
mount -t sysfs none /sys
mount -t ext4 /dev/mmcblk0p2 /newroot
mount -n -t devtmpfs devtmpfs /newroot/dev
exec sh # interruption du démarrage (mode manuel
   )
# Démarrage du linux général
#exec switch_root /newroot /sbin/init
endofinput
######
chmod 755 init
cd ..
sudo chown -R 0:0 $ROOTFSLOC
cd $ROOTFSLOC
find . | cpio --quiet -o -H newc > ../Initrd
cd ..
gzip -9 -c Initrd > Initrd.gz
mkimage -A arm -T ramdisk -C none -d Initrd.gz
   uInitrd
```

7.13 Files permissions

Aller voir la section 6.6

Access type	File	Directory
Read	Read the file content	Allow to read the folder content
Write	Modify (create-delete-rename) the file	Allow to create-delete-rename files in the folder
Execute	Execute the file	Allow to go in the folder

7.14 Sécurisation des comptes utilisateurs

Aller voir la section 6.8

7.15 Real-effective userID and groupID

```
chmod \textbf{7}744 file #SUID + SGID + sticky
   bits = 1
ls -al
-rwsr-Sr-T 1 test test 3438 2011-12-19 16:14
   file #s or t lower case = with x bit
```

Chaque processus possède un effective userID et un real userID pareil pour le groupID. Linux utilise seulement le effective userID. Si le bit userID est actif alors le fichier exécuter prend les droits du propriétaire du fichier.

7.16 ACL - Access Control List

Les filesystems ext3,ext4,tmpfs,btrfs autorise les ACL u: : User, g: : Group, o: : Other

```
setfacl -m u::rwx,g::r--,o:--- test
setfacl -Rm u:user1:rw TestDirectory # -R:
   Recursive
# remove
setfacl -b test #The file test has no ACL
setfacl -x u:user1,g:group1 test #The file test
   has no rights for user1
and group1
```

7.17 Attributs particuliers des FS ext2-3-4

On peut utiliser la commande `lsattr` ou `chattr`

```
chattr +i file #add i attribute
chattr -i file #del i attribute
chattr =i file #equal i attribute
lsattr file
```

- -i :file/directory can not be modified, deleted, renamed or linked symbolically, not even by root. Only root or a binary with the necessary rights can set this attribute.
- -A :Date of last access is not updated (only useful for reducing disk access)

- -S :File is synchronous, the records in the file are made immediately on the disc. (equivalent to the sync option of mount)
- -a :File can only be open in append mode for writing (log files, etc.) Only redirection >> can be used, the file can not be deleted.
- -c :File is automatically compressed before writing to disk, and unpacked before playback
- -d :File will not be saved by the dump
- -j :Ext3-ext4 :A file with the 'j' attribute has all of its data written to the ext3 or ext4 journal before being written to the file itself
- -s :When the file is destroyed, all data blocks are being released to zero.

7.18 Rechercher de permission de fichier faibles

```
find . -perm 200 #file permissions = 200
find . -perm -220 #write bit for user and group = 1
find . -perm /220 #write bit for user or group = 1
find . -perm +220 #write bit for user or group = 1 (like /220)
```

7.19 Sécuriser les répertoires temporaires

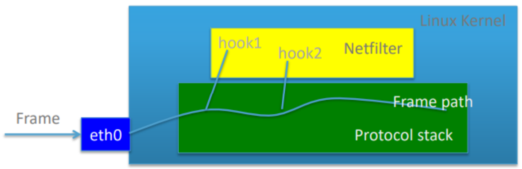
- mettre le `\tmp` dans une autre partition
- no suid programme sont permis
- rien ne peut être exécuté
- aucun node file existe dans le `\tmp`

7.20 Mémorisation des mots de passes

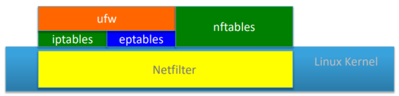
Les mot de passes sont sotckés en HASH (preuve sans connaissance)

8 Firewall iptables

Il est nécessaire d'activer netfilter dans la configuration du kernel. Un hook est une étape lors du passage d'une trame dans le stack de protocoles. Le framework netfilter sera appelé à chaque hook



On peut configurer netfilter avec la commande `iptables`. `ebtables` permet de configurer la couche 2 uniquement (Linux bridge). `nftables` vise à remplacer tout le framework



8.1 Features

1. Stateless packet filtering (table filter et ACCEPT, DROP, REJECT)
2. Stateful packet filtering
3. Translation d'adresses / ports
4. API pour autres applications

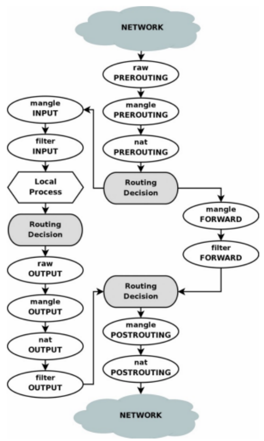
8.2 Chain

la combinaison Chain-Table constitue les hooks

Chains : INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING

8.2.1 Tables

- raw
- mangle (modification spéciales sur des paquets)
- nat (consultée lorsqu'un paquet créé une nouvelle connexion)
- filter (table de base)



8.3 Commande iptables

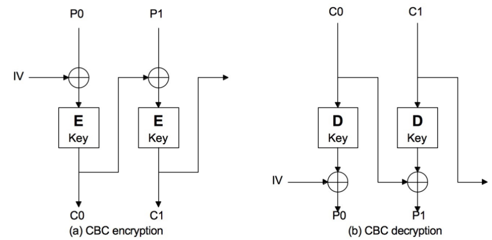
```
iptables -t table -COMMAND chain ... -j TARGET
```

9 TPM

9.1 Chiffrements

9.1.1 symétrique

- Une seul clé pour crypter et décrypter
- Codage par bloc ou par bloc chainé




- ```
  openssl enc -aes-256-cbc -e -in t.txt -out t.enc #encrypt

  openssl enc -aes-256-cbc -d -in t.enc -out t.txt #decrypt
  ```

9.1.2 asymétrique

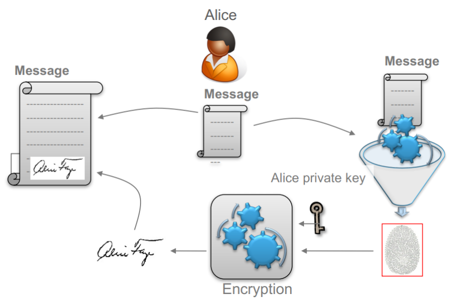
- Deux clés (publique et privée) clé publique disponible par des certificats (CMD pgp)
- Encrypt public - Decrypt private = confidentialité
- Encrypt private- Decrypt public = signature digitale

9.1.3 hash

Transforme un texte, document en un nombre de N bits unique (SHA-2, SHA-3, Blake2).
`md5sum file => a6a0e8d0522...` où
`openssl dgst -md5 file`

9.1.4 signature

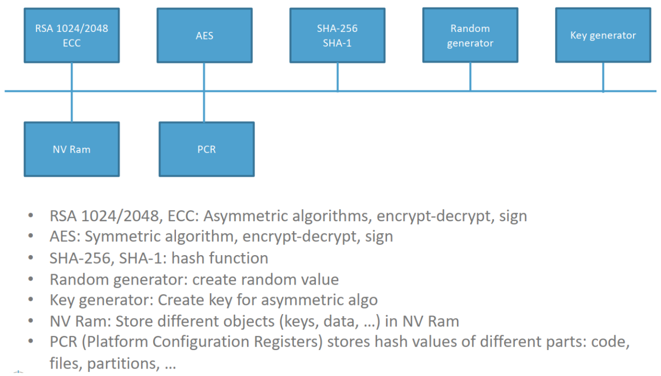
En deux parties: 1. Calcul du HASH puis encryptage avec clé privée.


  ```

9.2 Implémentations TPM

- discrete : Circuit dédié
- integrated : Partie du μC qui gère le TPM
- Hypervisor : virtuel fournis par personne fiable
- Software : virtuel pour faire des test pas sécurisé

9.3 Architecture interne



9.4 Hiérarchies

- endorsement : réservé au fabricant du TPM et fixé lors de la fabrication.
- platform : réservé au fabricant de l'hôte et peut être modifier par l'équipementier.
- owner : hiérarchie dédiée à l'utilisateur primaire du TPM peut être modifié en tout temps.
- null : réservé aux clés temporaires (RAM s'efface à chaque redémarrage)

9.5 Créer, utiliser clés

```
tpm2_createprimary -C <A> -G rsa2048 -c <B>.ctx
```

<A>	<B>
e	e_primary
p	p_primary
o	o_primary
n	n_primary

9.6 Commandes principales

```
tpm2_createprimary -C o -G rsa2048 -c o_prim #cr
    éer un clé primaire owner
tpm2_getcap handles-transient #voir clé dans la
    RAM
tpm2_getcap handles-persistent #voir clé dans la
    NV-RAM
tpm2_evictcontrol -c o_primary.ctx # sauver une
    clé en NV-RAM
tpm2_flushcontext! -t ##effacer toute la RAM
tpm2_create -C o_prim -G rsa2048 -u child_pub -r
    child_priv #créer clé enfant
tpm2_load -C o_prim -u child_pub -r child_priv -
    c child #charger clé enfant
shred passwd, rm -f passwd #supprimer de l'hôte
```

9.7 encrypter-décrypter, signer-vérifier

```
tpm2_rsaencrypt -c child -s rsaes clearfile -o
    encryptedfile
tpm2_rsadecrypt -c child -s rsaes encryptedfile
    -o clearfile
tpm2_sign -c child -g sha256 -o file.sign file
tpm2_verifysignature -c child -g sha256 -s file.
    sign -m file
```

9.8 Registres PCR

```
tpm2_pcrreset 0
tpm2_pcrextend 0:sha1=8c83...(hash)
```

9.9 Sauver des données sur le TPM

```
tpm2_evictcontrol -c passwd.ctx 0x81010000 -C o
    #sauver
tpm2_unseal -c 0x81010000 > passwd  #récuperer
```

9.10 Sauver des données et protéger avec PCR policy

```
sha1sum passwd #calcul hash
tpm2_pcrreset 0 #flush PCR0
tpm2_pcrextend 0:sha1=8c839... #sauve hash
tpm2_createprimary -C o -G rsa2048 -c primary
tpm2_startauthsession -S session
tpm2_policypcr -S session -l sha1:0 -L
    pcr0_policy #créer politique
tpm2_flushcontext session
```

```
tpm2_create -C primary -g sha256 \
-u passwd_pcr0.pub -r passwd_pcr0.priv \
-i passwd -L pcr0_policy
```

```
tpm2_evictcontrol -c passwd_pcr0 0x81010000 -C o
tpm2_flushcontext session
```

```
shred passwd
rm -f passwd
```

```
tpm2_startauthsession --policy-session -S
    session
tpm2_policypcr -S session -l sha1:0
tpm2_unseal -p session:session -c 0x81010000 >
    passwd
```

10 Autres

10.1 Commandes

netcat (nc) : Couteau suisse du TCP/IP. Permet de scanner des ports

nmap : Analyse des ports ouverts

ssh : Connexion à un système par interpréteur de commande

dd : copie byte à byte entre des streams. (`sudo dd if=/dev/zero/ of=/dev/null bs=512 count=100 seek=16`)

parted : création / modification de partiations (`sudo parted /dev/sdb mklabel msdos`

mkfs.ext4 : commandes ext4 pour créer / modifier une partition

10.2 Définitions

Honeypot : "Pot de miel" ou leurre pour faire croire qu'un système non-sécurisé est présent (à tord)

Toolchain : Codes sources et outils nécessaires pour générer une image éxécutable (sur un système embarqué)

Kernel : Coeur Linux (avec le format u-boot)

Rootfs : Root Filesystem (avec tous les dossiers et outils utilisés par Linux)

Usrfs : User Filesystem (applications spécifiques à l'utilisation du système embarqué)

Buildroot : Ensemble de makefiles et patchs qui simplifient et automatisent la création d'un Linux pour système embarqué

uClibc : Librairie c de base similaire à glibc mais plus compacte (pour systèmes MMU-less)

Busybox : Binaire unique qui contient toutes les commandes de base (ls, cat, mv)