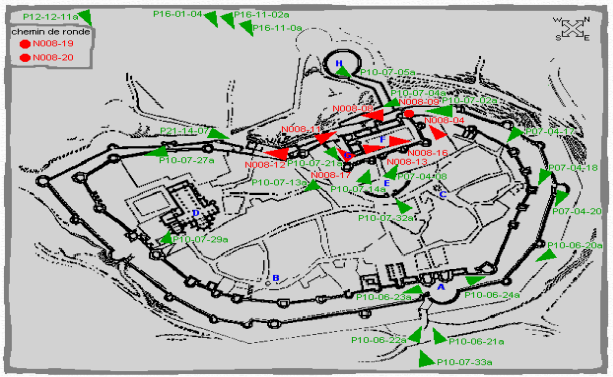
Sécurité des Mobiles
2021

Pascal Urien Télécom Paris

Rappels de Sécurité

La Sécurité est une Construction (design)



2 lignes de remparts

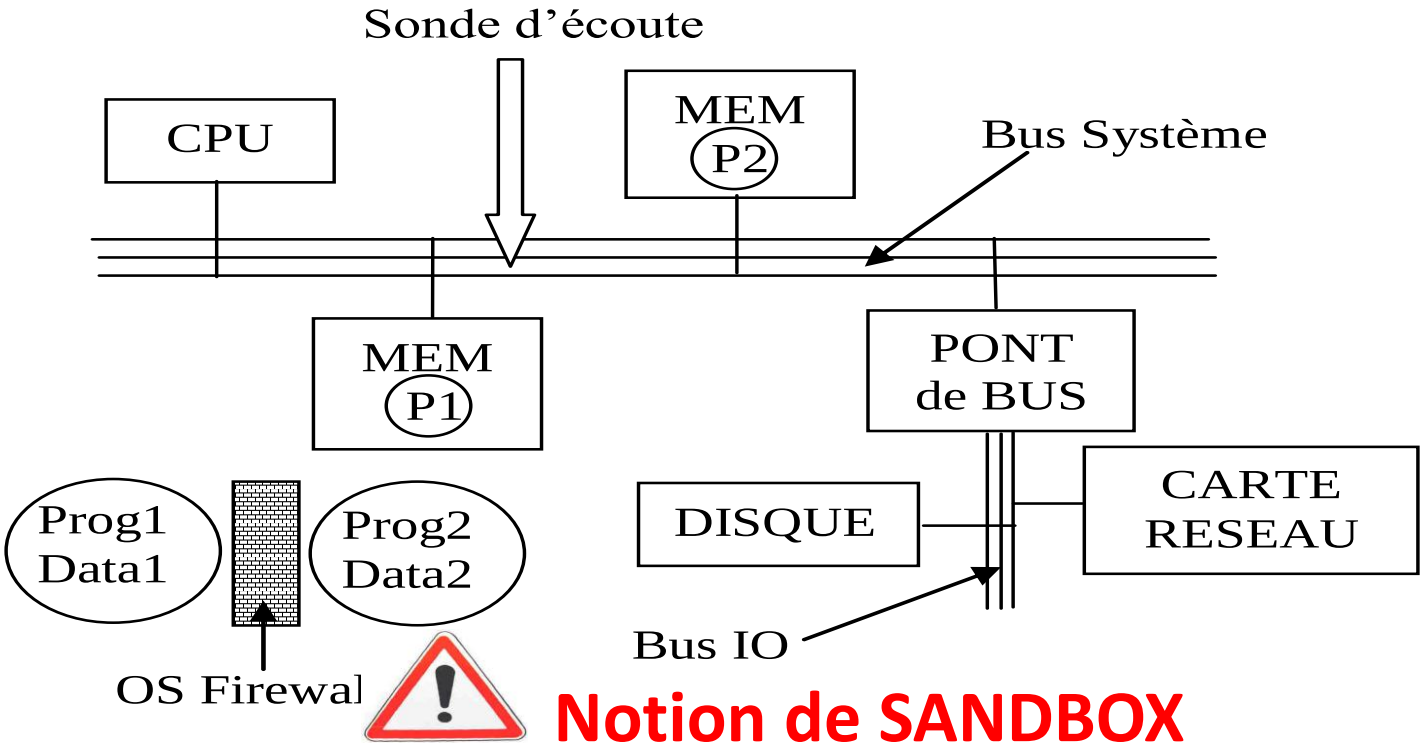
1 bastion

*La ville de Carcasonne

Sécurité des Applications Distribuées

- Une application distribuée est un ensemble d'entités logicielles, logiquement autonomes, qui produisent, consomment et échangent des informations
  - OUTi = PROG(INi))
- Dans un premier temps les composants logiciels des applications étaient logés dans un même système informatique, constituant de fait leur média de communication (parfois dénommé gluware).
  - Le bus système permet le transfert des informations stockées en mémoire, les modules logiciels sont réalisés par des processus gérés par le système d'exploitation.
  - La sécurité est uniquement dépendante des caractéristiques du système d'exploitation, par exemple en terme de gestion des droits utilisateurs, ou d'isolement des processus.

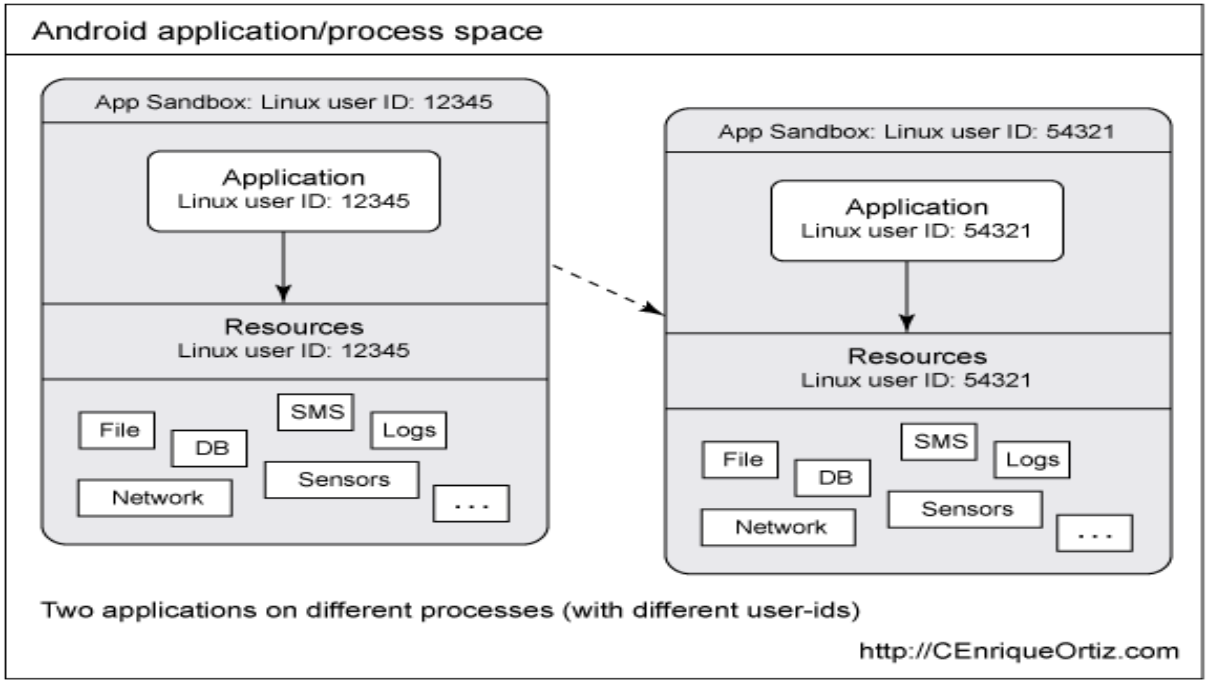
Notion de GlueWare



Notion de SANDBOX

Isolation: Notion de SandBox

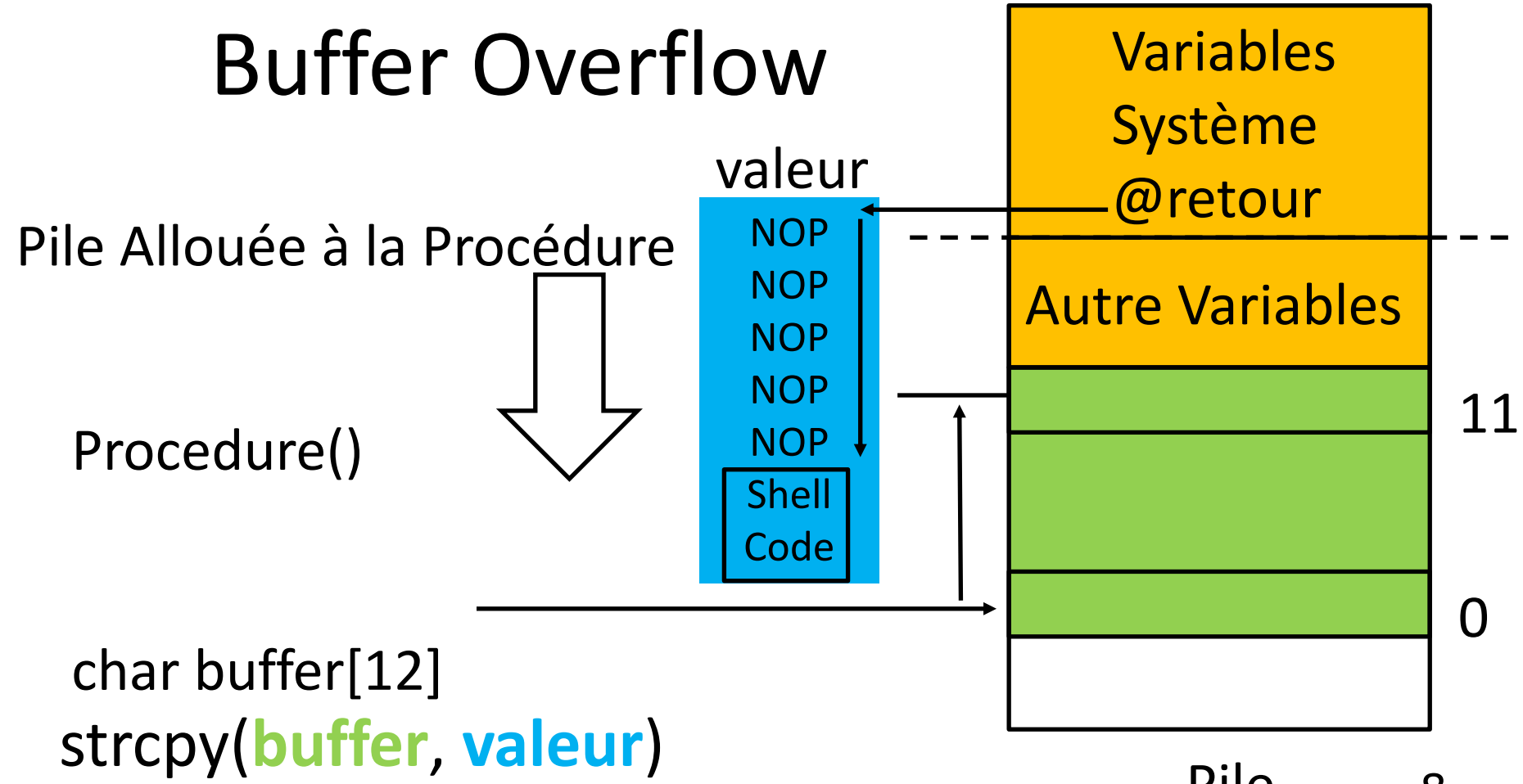
- Un SANDBOX est un environnement logiciel qui contrôle les accès d'une application aux ressources d'un système informatique géré par un système d'exploitation



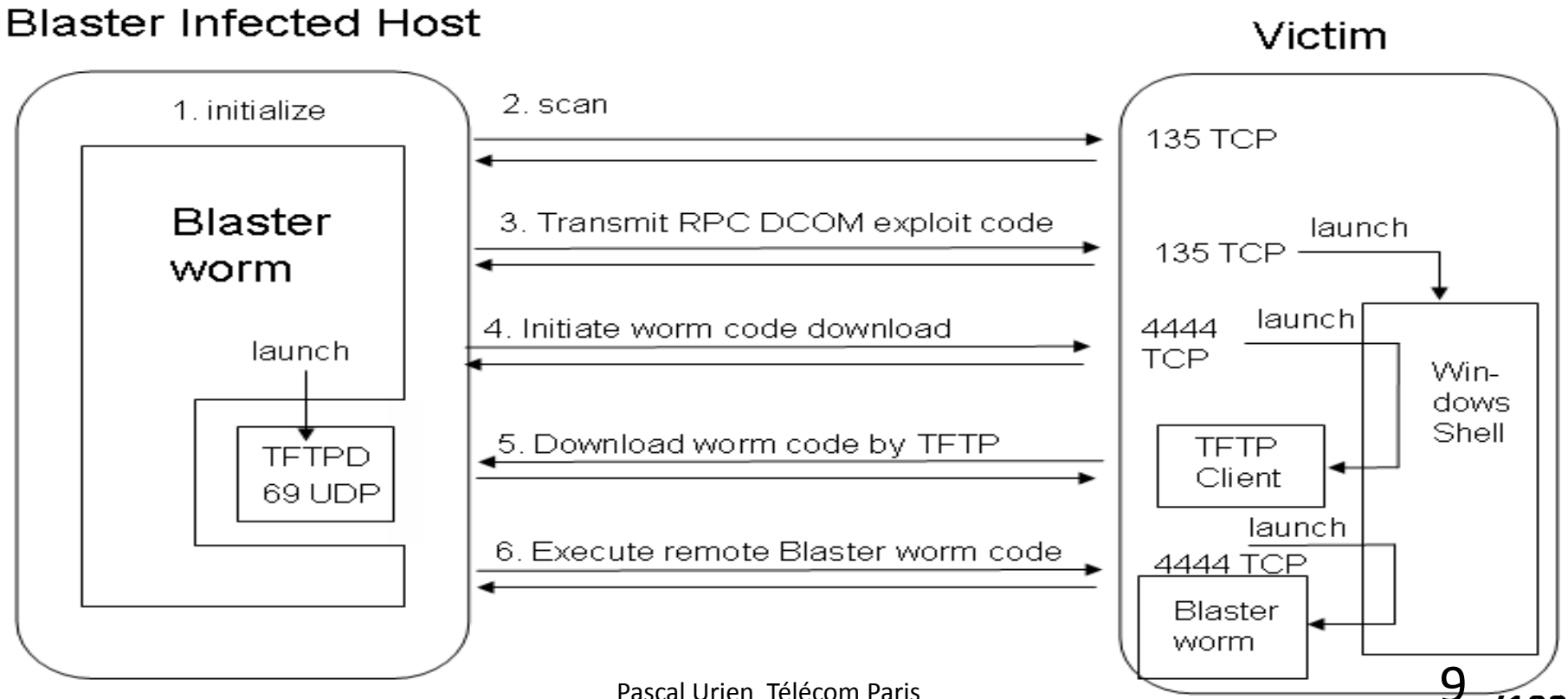
Exemple d'attaque visant l'isolation: Buffer Overflow

- En 1988, un étudiant du MIT, Robert Tappan Morris, réalisa la première mise en œuvre offensive sur internet des techniques buffers overflow, nommée Morris worm.
- **Le buffer overflow** consiste à modifier malicieusement la mémoire d'un programme, typiquement lors de l'écriture d'une information localisée dans un paramètre d'appel du programme, ou lors de l'écriture de données reçues via le réseau (c'est-à-dire à l'aide des bibliothèques de sockets).
- Les effets espérés par les *buffer overflow* dans la pile d'exécution sont répartis en trois classes principales
  - La *modification d'une variable mémoire*, proche de la zone mémoire occupée par le buffer.
  - La *modification de l'adresse de retour du programme* localisée dans la pile, le but étant l'exécution d'un code malveillant localisé dans un paramètre d'appel ou dans une information reçue via le réseau.
  - La *modification d'un pointeur de fonction* afin d'exécuter un code préalablement injecté.

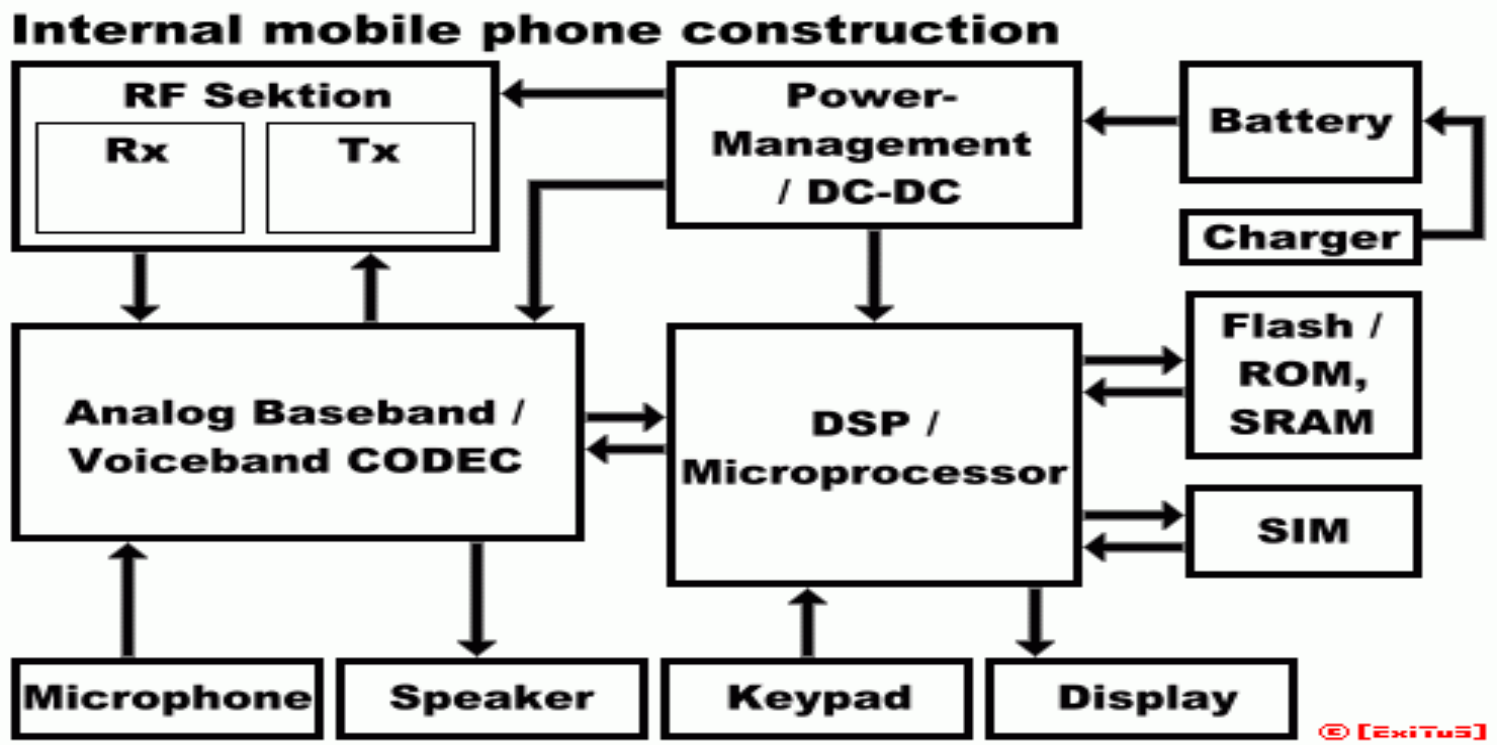
Buffer Overflow



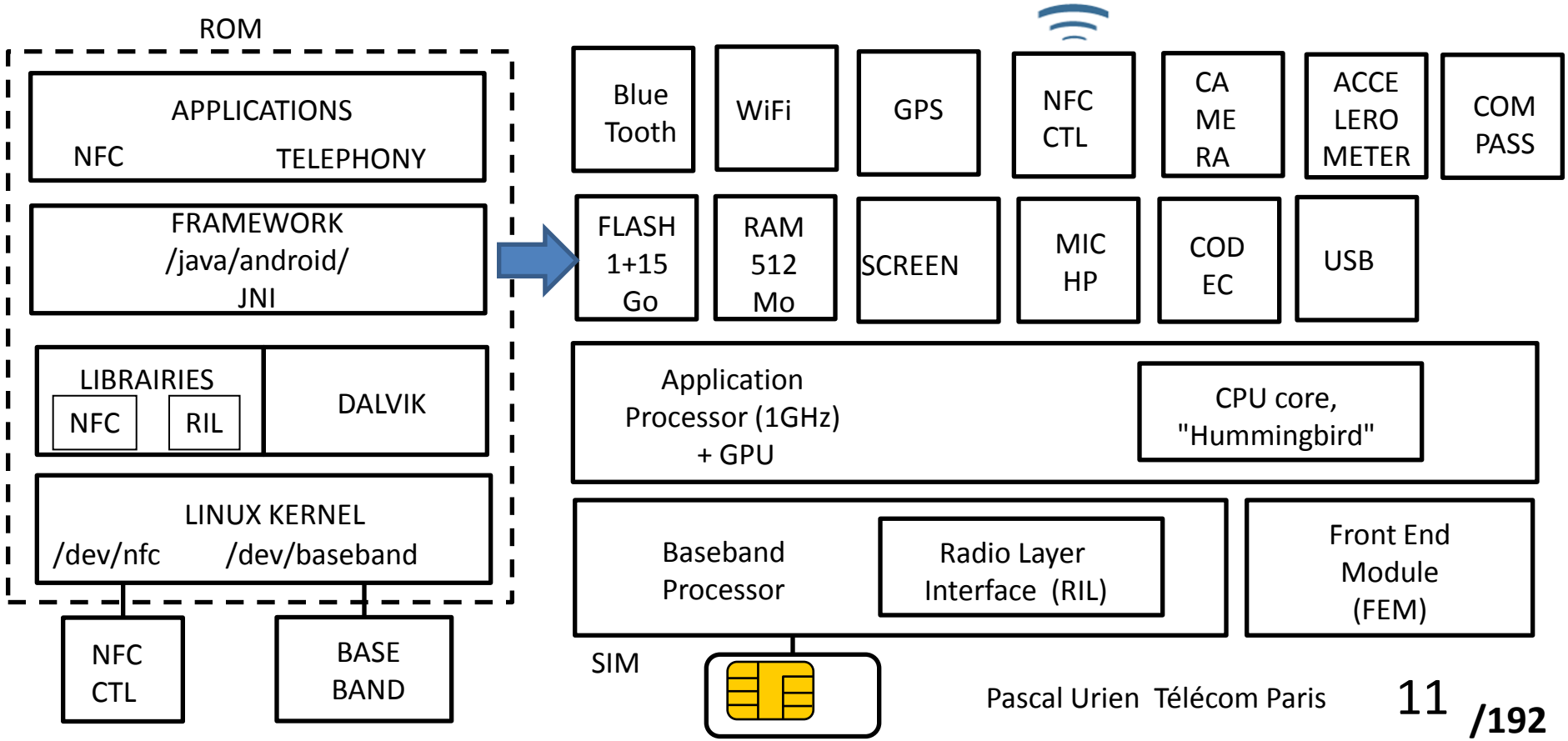
Un exemple de vers: BLASTER (2003)



Architecture d'un mobile "classique"



Android Nexus S, 2011



Trinité, Paul Kocher & ALL (2004)

Extensibilité

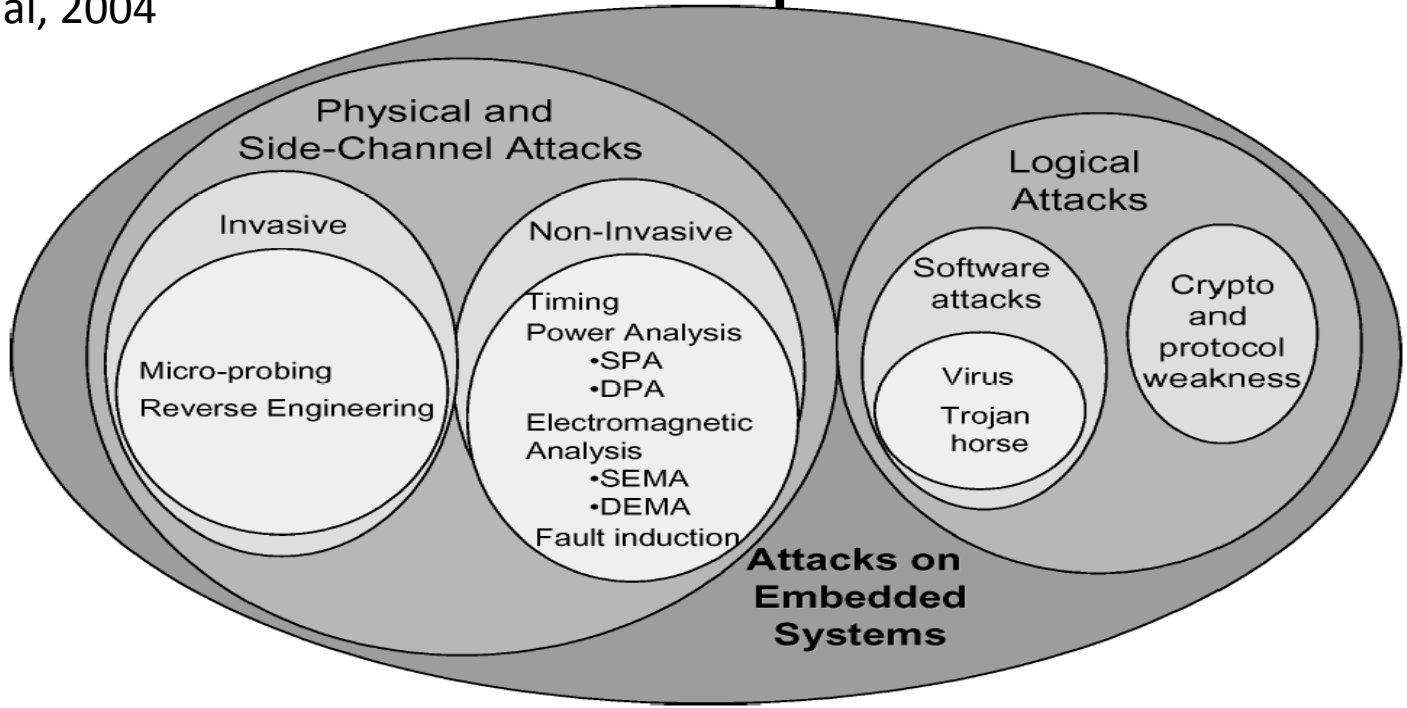
Sécurité

Complexité

Connectivité

Attaques adressant les systèmes embarqués

S. Ravi et al, 2004



Examples of attack threats faced by embedded systems.

Plateformes Sécurisées, Bill Gates, RSA Security Conference, 2006

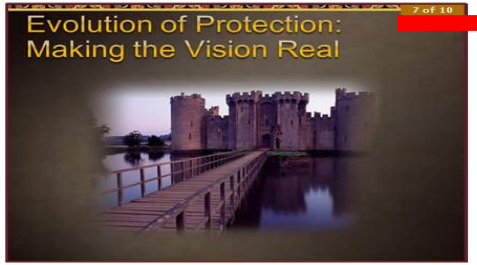
Network
Firewall

Processes
Firewall

User
User's' Right

Secure Device
Carte à puce

La stratégie Microsoft – Bill Gates
RSA Security Conference 2007



- Le mot de passe est LE problème de sécurité
- Microsoft propose l'usage généralisé de certificats

- IPSEC
- Trustworthy Computing
- **Des composants qui résistent aux attaques**
  - TPM
  - Cartes à puce
- Identité

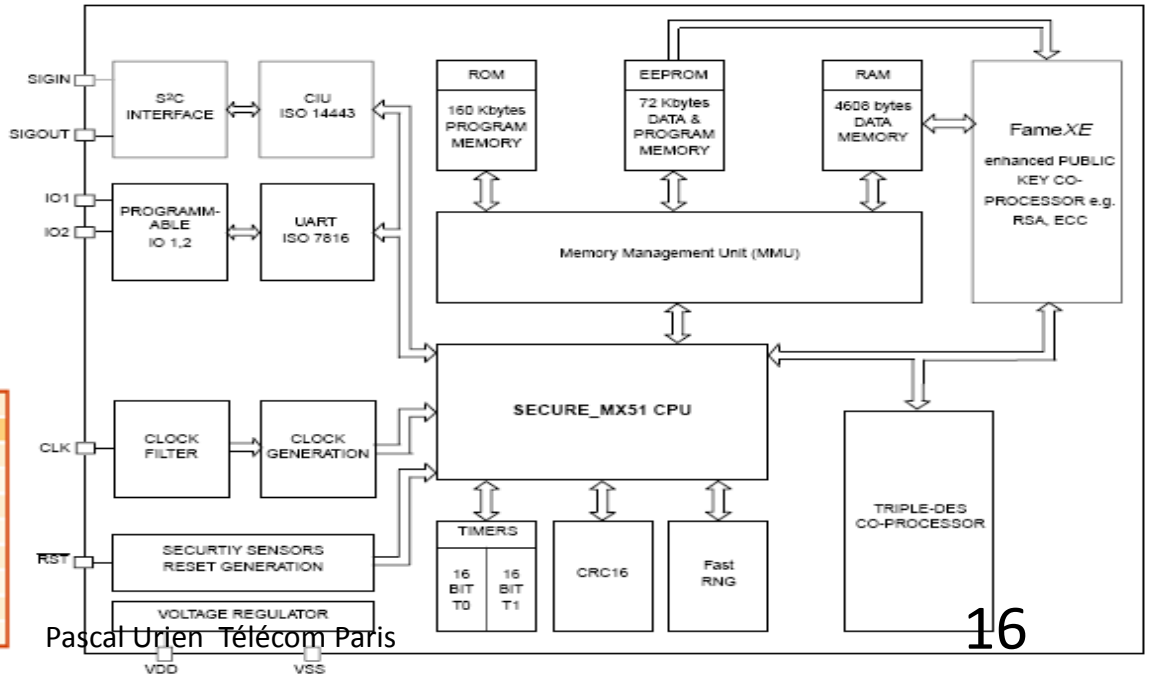
Definition d'un Secure Element.

Un Secure Element (SE) est un microcontrôleur sécurisé muni d'interfaces électriques et de communication, telles que ISO7816, SPI or I^2C .

OS JAVACARD JCOP
GP (Global Platform)
ROM 160 KB
EEPROM 72 KB
RAM 4KB
Crypto-processor
3xDES, AES, RSA, ECC
Certification CC EAL5+
Security Certificates EMVCo

	NFC secure modules
Product features	PN65L
Embedded NFC IC	PN532
Available host interfaces	serial, SPI, I²C
Embedded Secure IC	P5CN072
OS for secure device	JCOP or 3rd party
Stacked passive component IC	yes
Package thickness	1.2 mm
Package size	7x7 mm²
Package type	HLQFN48

EXAMPLE: NXP PN532



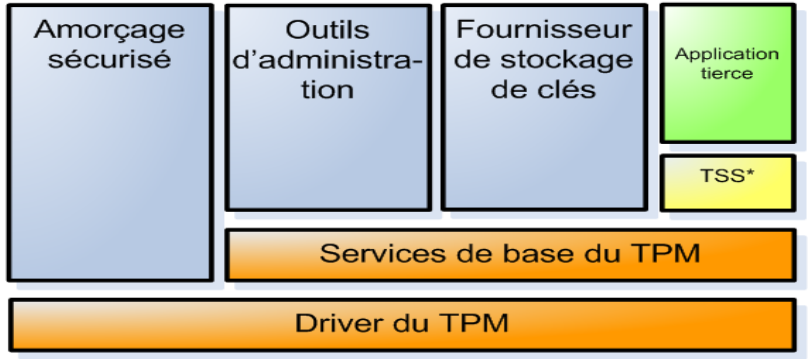
Contrôle d'intégrité

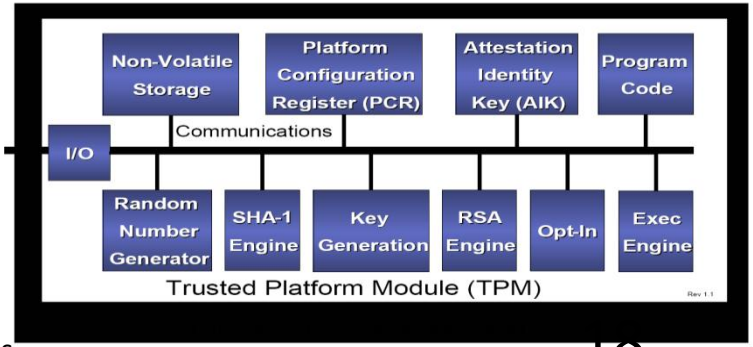
- Certaines attaques intrusives sont liées à la possibilité de modifier le code d'un programme soit avant son exécution (modification du fichier exécutable) soit au cours de son exécution (point d'arrêts en mode debug, modification de certaines zones mémoire, pose de bretelles….).
- L'analyse d'un code lors de son exécution révèle la clé associée à un algorithme cryptographique. Cependant des techniques d'embrouillage de code (obfuscation) peuvent rendre ces attaques plus difficiles.
- Le groupe Trusted Computing Group (TCG) a défini une architecture sécurisée pour les ordinateurs personnels basés sur un module hardware sécurisé le TPM (Trusted Platform Module). Dans ce modèle l'intégrité du système (bibliothèques essentielles du système d'exploitation,…) est mesurée (integrity measurement) par une empreinte (SHA-1, 160 bits) stockée dans une puce de silicium résistante aux attaques (tamper resistant). L'accès à ce dispositif est contrôlé par des secrets partagés symétriques. Le TPM s'appuie sur un arbre de clés RSA, dont l'accès à chaque nœud est protégé par une clé symétrique.

Le TPM*

- L'intégration du TPM dans le système d'exploitation *windows* s'applique aux trois points suivants,
  - Le ***Secure Boot***. Le premier programme amorce est stocké dans le TPM. Le *boot* est une succession de programme Pi tels que Po est contenu dans le TPM, chaque Pi est associé à une empreinte Hi enregistrée dans le TPM, le programme Pi-1 charge Pi et vérifie son empreinte Hi.
  - Le **chiffrement du contenu du disque dur** (*bitlocker*^{TM}) à l'aide d'une clé maître (VEK, Volume Encryption Key) stockée dans le TPM.
  - **Le contrôle de l'intégrité du PC** lors de sa connexion réseau (*NAP, Network Access Protection*).

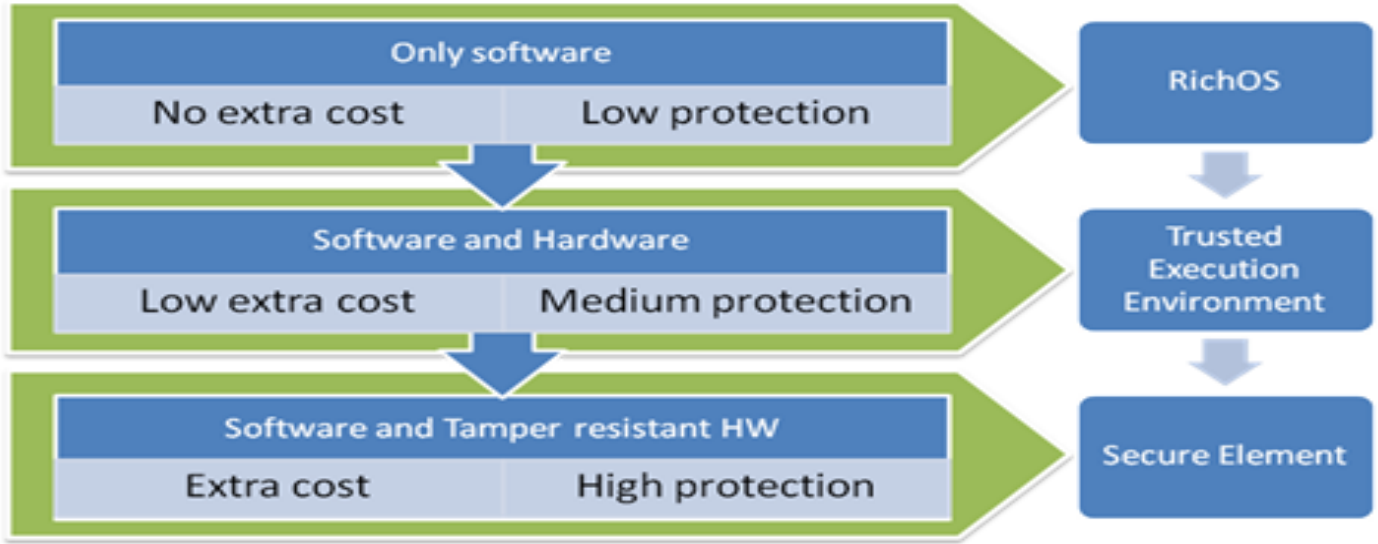
*http://www.trustedcomputinggroup.org/





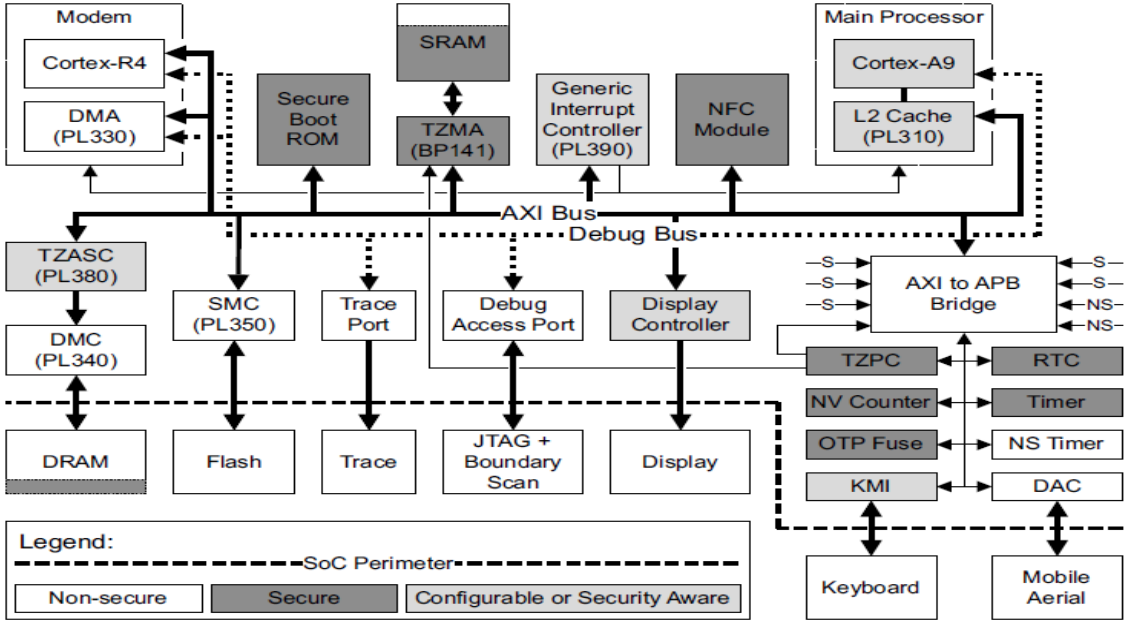
Trusted Execution Environment (TEE)

- Le TEE est un environnement d'exécution de confiance pour les mobiles.
- Il est basé sur un mécanisme d'isolation



TrustZone©

- Le concept de TrustZone propose la virtualisation d'un processeur permettant la création de deux espaces de traitement de l'information le "Normal Word" et le "Secure World".
- Chacun de ces modes de fonctionnement possède un banc de registres séparé et des mécanismes d'interruption différents.
- Cependant le CPU et les mémoires internes (ROM, SRAM) sont communs aux deux mondes.
- Une troisième entité, le Monitor gère les changements de contexte entre le "Normal Word" et le "Secure World" à l'aide d'instructions spécifiques (Secure Monitor Call, SMC); elle se comporte de fait comme un hyperviseur qui réalise grâce à la technique de virtualisation, une isolation des espaces dits normaux ou sécurisés.

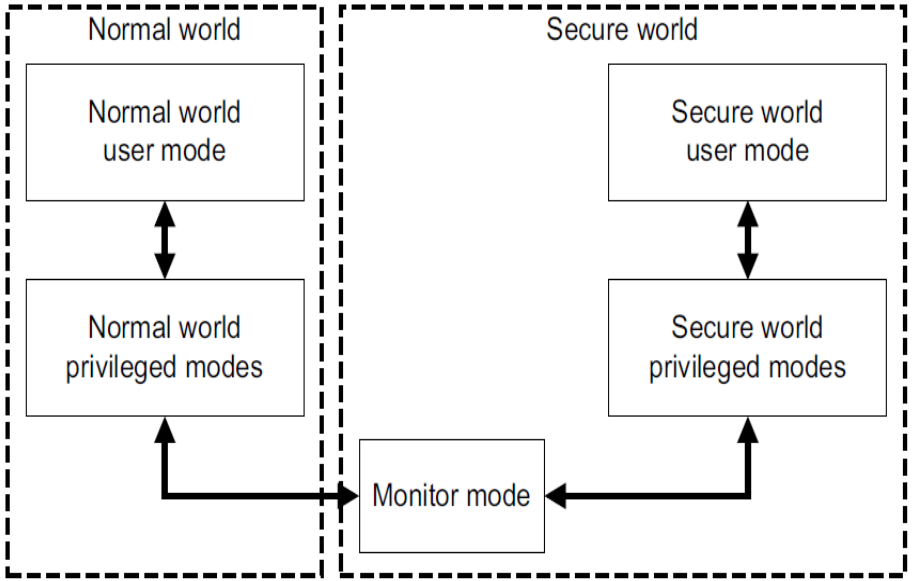


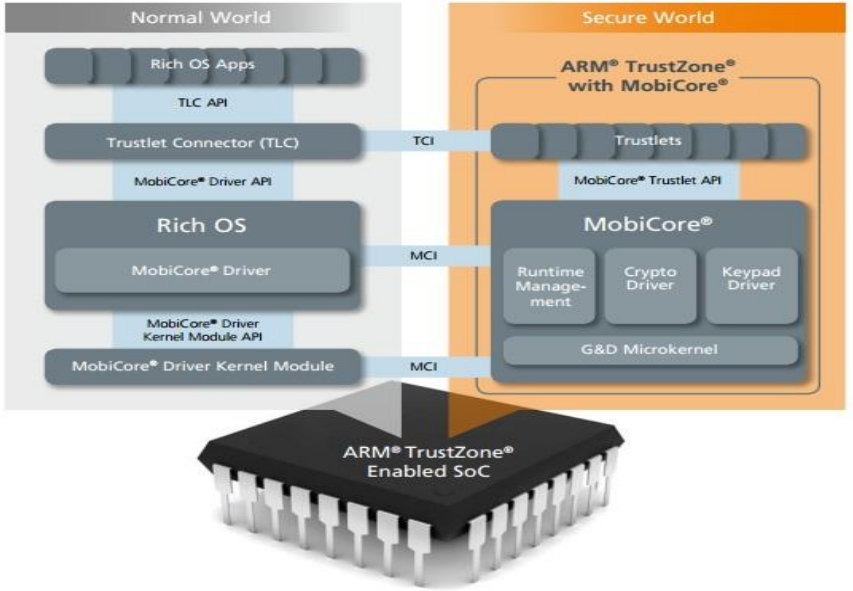
- Les tailles mémoire internes (In-SoC) sont de l'ordre de 10 Ko pour la ROM et 128 Ko pour la SRAM.

TrustZone©

- D’un point de vue physique, et contrairement aux Secure Elements, le processeur n’implémente pas de contremesures matérielles.
- Les mémoires externes (Off-Soc), non volatiles (ROM, FLASH) ou volatiles (DRAM…) sont partagées par les deux mondes. Une entité MMU (Memory Management Unit) réalise les partitions nécessaires à leur virtualisation; des protections cryptographiques (chiffrement et intégrité) sont nécessaires pour la sécurité des informations stockées par le "Secure World". Le MMU assure également les partitions mémoires internes au SoC.
- Le concept TrustZone introduit la notion d’entrée/sortie (IO) sécurisée. Un clavier sécurisé est géré par un pilote (driver) exécuté dans un SW. Un affichage sécurisé dispose deux mémoires d’affichage (FrameBuffer) distinctes, et donc implique la disponibilité d’un contrôleur particulier. Un pilote NFC exécuté en mode SW assure le traitement sécurisé de transaction de paiement.

TrustZone©


Normal world
Normal world user mode
Normal world privileged modes
Secure world
Secure world user mode
Secure world privileged modes
Monitor mode

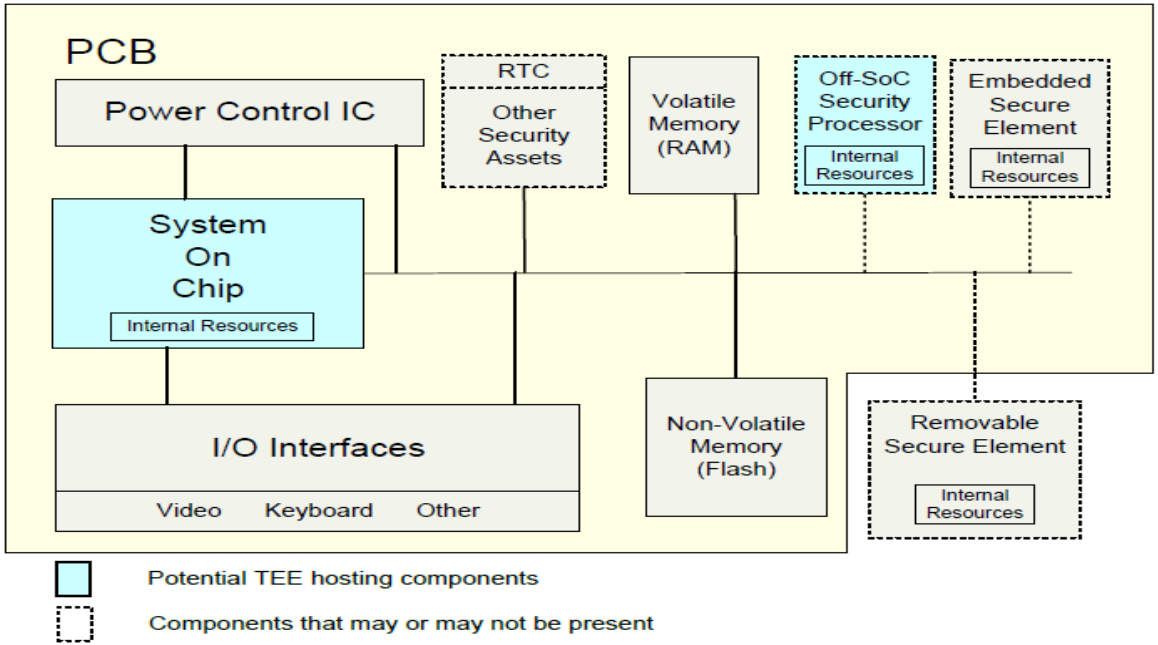

Normal World
Rich OS Apps
TLC API
Trustlet Connector (TLC)
MobiCore® Driver API
Rich OS
MobiCore® Driver
MobiCore® Driver Kernel Module API
MobiCore® Driver Kernel Module
TCI
MCI
MCI
Secure World
ARM® TrustZone® with MobiCore®
Trustlets
MobiCore® Trustlet API
MobiCore®
Runtime Manage-ment
Crypto Driver
Keypad Driver
G&D Microkernel
ARM® TrustZone® Enabled SoC

Secure Element et TEE

Printed Circuit Board

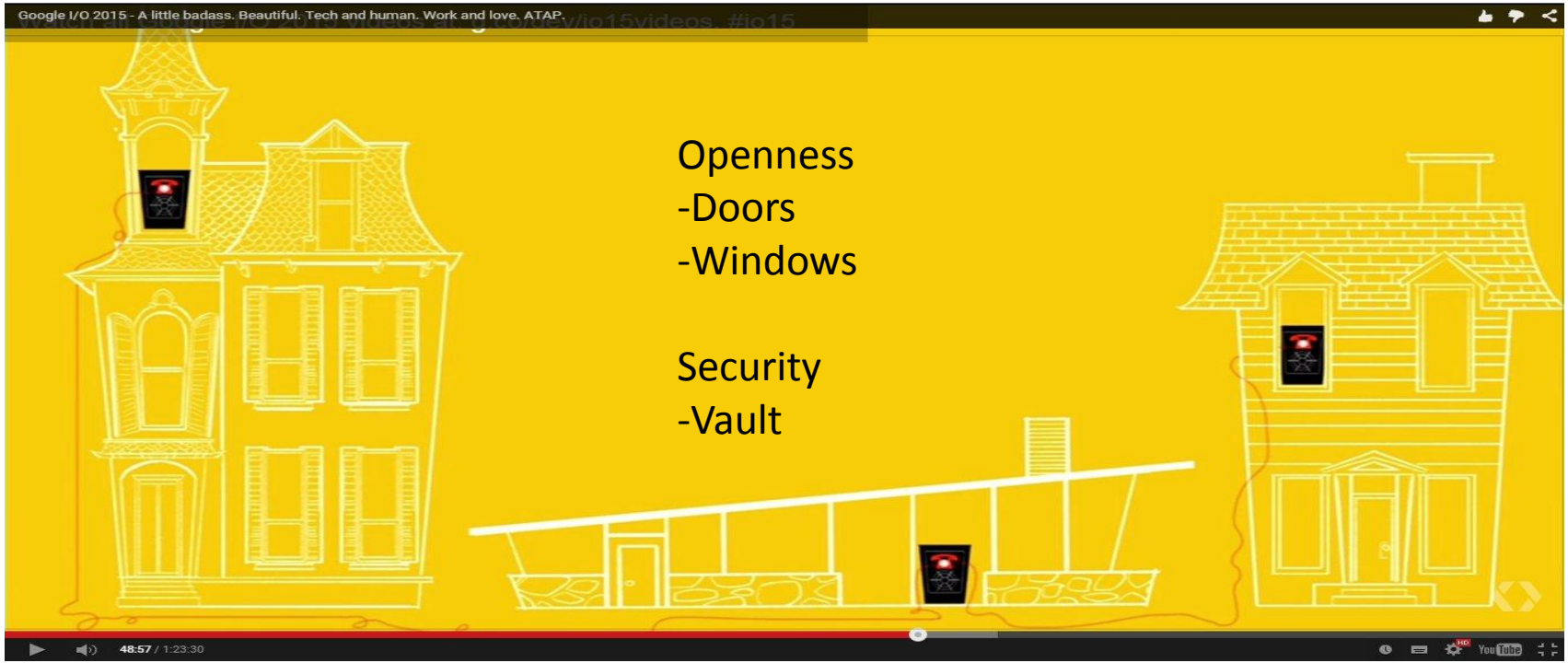
Real Time Clock

Figure 2-1: Chipset Architecture



GlobalPlatform Device Technology TEE System Architecture Version 1.0, 2011

Google Vault (2015)



Google Vault : a SD Card

Research Hardware / Development Kit

- Fully Open Source
- FPGA-based development PCB
- OpenRISC1200 Processor
- microSEL RTOS
- SD protocol
- NAND FTL
- Project Vault IDL
- HW-backed crypto

- SD card
- Only two files: WFILE and RDFILE
- Cryptographic procedures
- GB of storage
- MB of throughput
- NFC controller

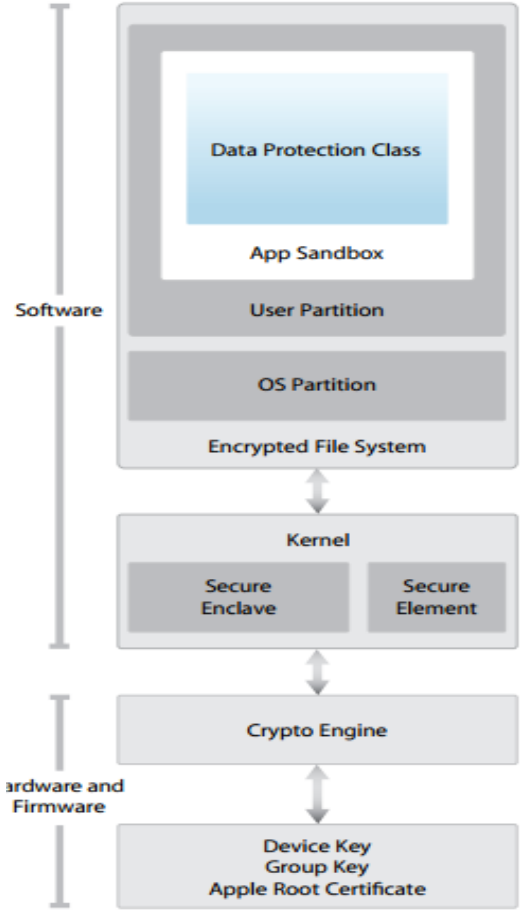
Sécurité des Mobiles
Quels Objectifs ?

Sécurité des Mobiles: Objectifs

- Isolation/protection des Applications
  - Sandbox
  - Protection des données
- Protection des ressources embarquées
  - Accès au réseau cellulaire, répertoire d'adresses, GPS, caméra, comptes de réseaux sociaux,…
- Protection des données
  - Chiffrement des fichiers

Sécurité iPhone

https://www.apple.com/privacy/docs/iOS_Security_Guide_Oct_2014.pdf



Au Sujet de l'iPhone

- Secure Boot
  - L'EEPROM de boot contient le certificat racine (CA Apple) pour la vérification d'authenticité des composants.
  - Le processeur BaseBand gère également un processus de Secure Boot
  - Un coprocesseur optionnel (Secure Enclave) gère également un processus de Secure Boot

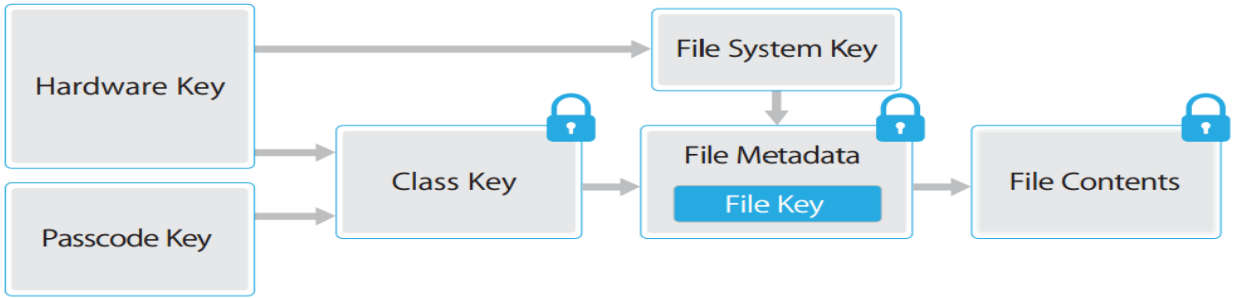
https://www.apple.com/privacy/docs/iOS_Security_Guide_Oct_2014.pdf

Divers Composants

- Secure Enclave
  - C'est un coprocesseur optionnel possédant un identifiant (UiD, Unique ID) et gérant ses propres mémoires (chiffrées)
- TouchID
  - Un capteur biométrique (empreinte digitale) contrôlé par le coprocesseur Secure Enclave
- Secure Element
  - En élément sécurisé (intégré dans un contrôleur NFC) utilisé par exemple pour le protocole de paiement ApplePay

Fichiers et Applications

- Un système de chiffrement des fichiers
  - une clé de chiffrement par fichier (AES 256)
  - Une clé hardware
  - Un passcode, c'est-à-dire un mot de passe utilisateur
- Une protection par Sandbox pour les applications



Sécurité du Système Android

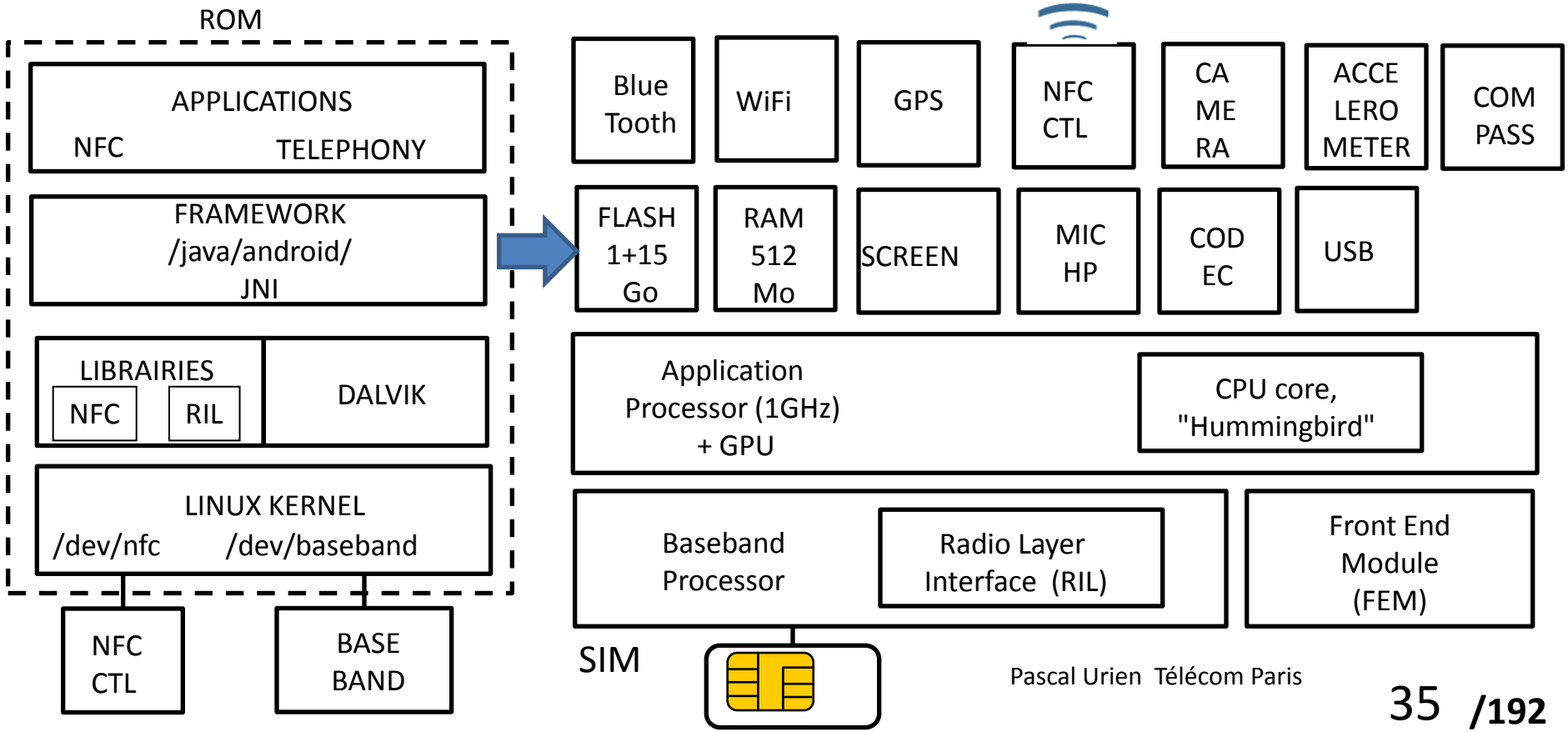
Android: Historique

Android est un système d'exploitation crée par la société Android Inc. rachetée en 2005 par GOOGLE.

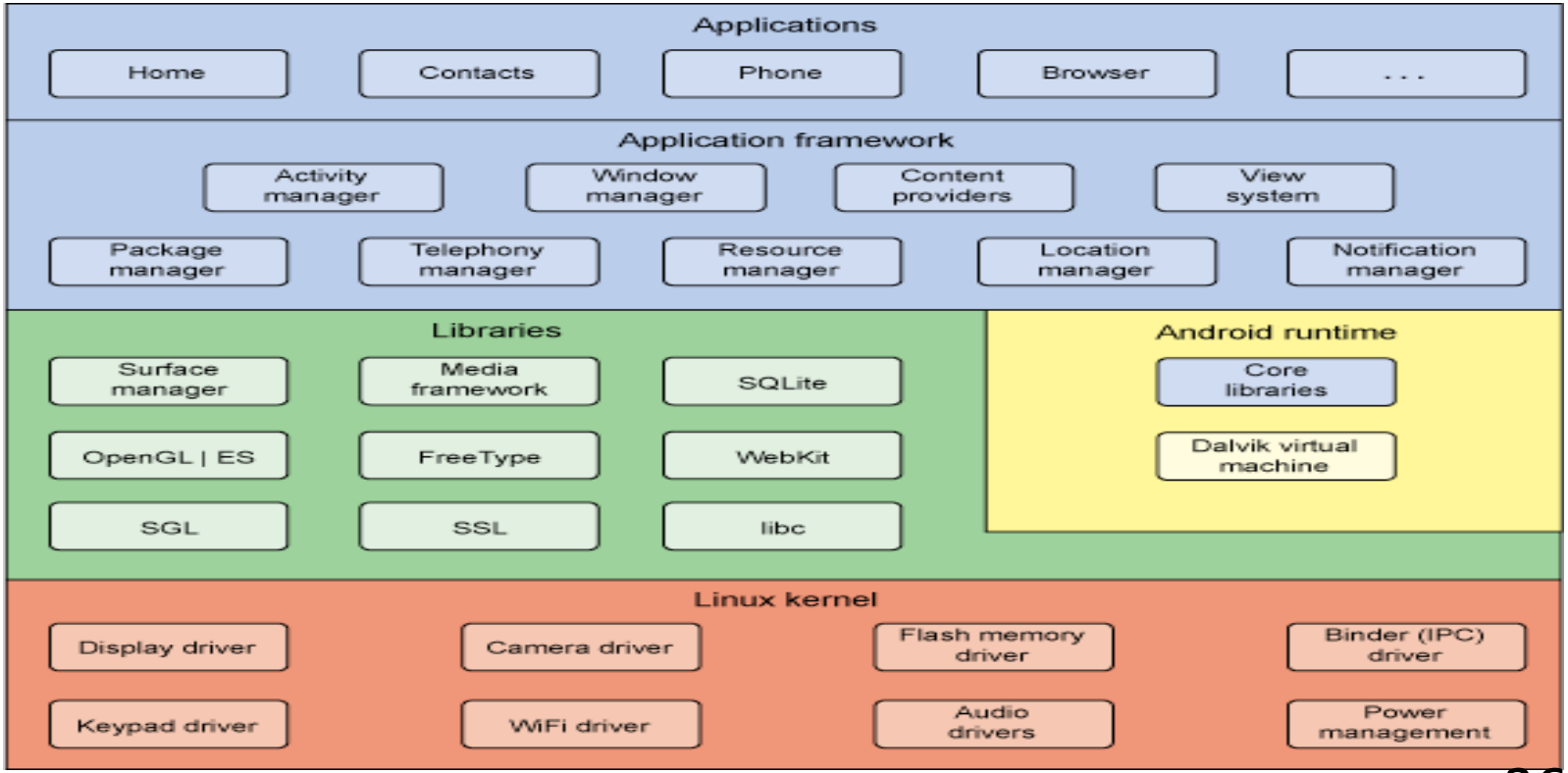
Architecture du système Android

- Un système d'exploitation Android comporte les éléments suivants
  - Un noyau Linux
  - Un ensemble de libraires, et la DALVIK Virtual Machine
  - Un framework JAVA utilisant JNI
  - Un ensemble d'applications

Exemple: Architecture du Nexus S

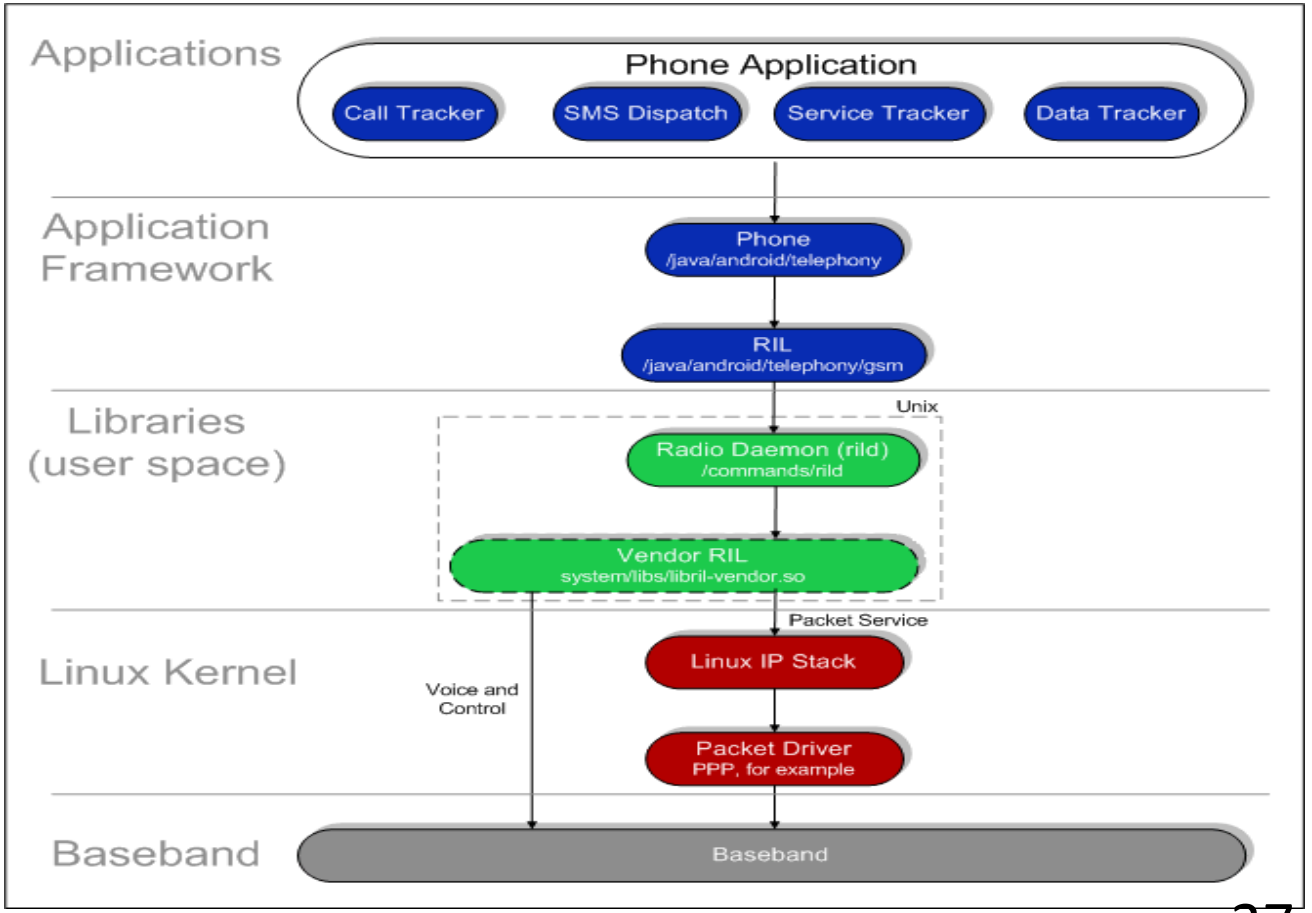


Android: Architecture Logicielle

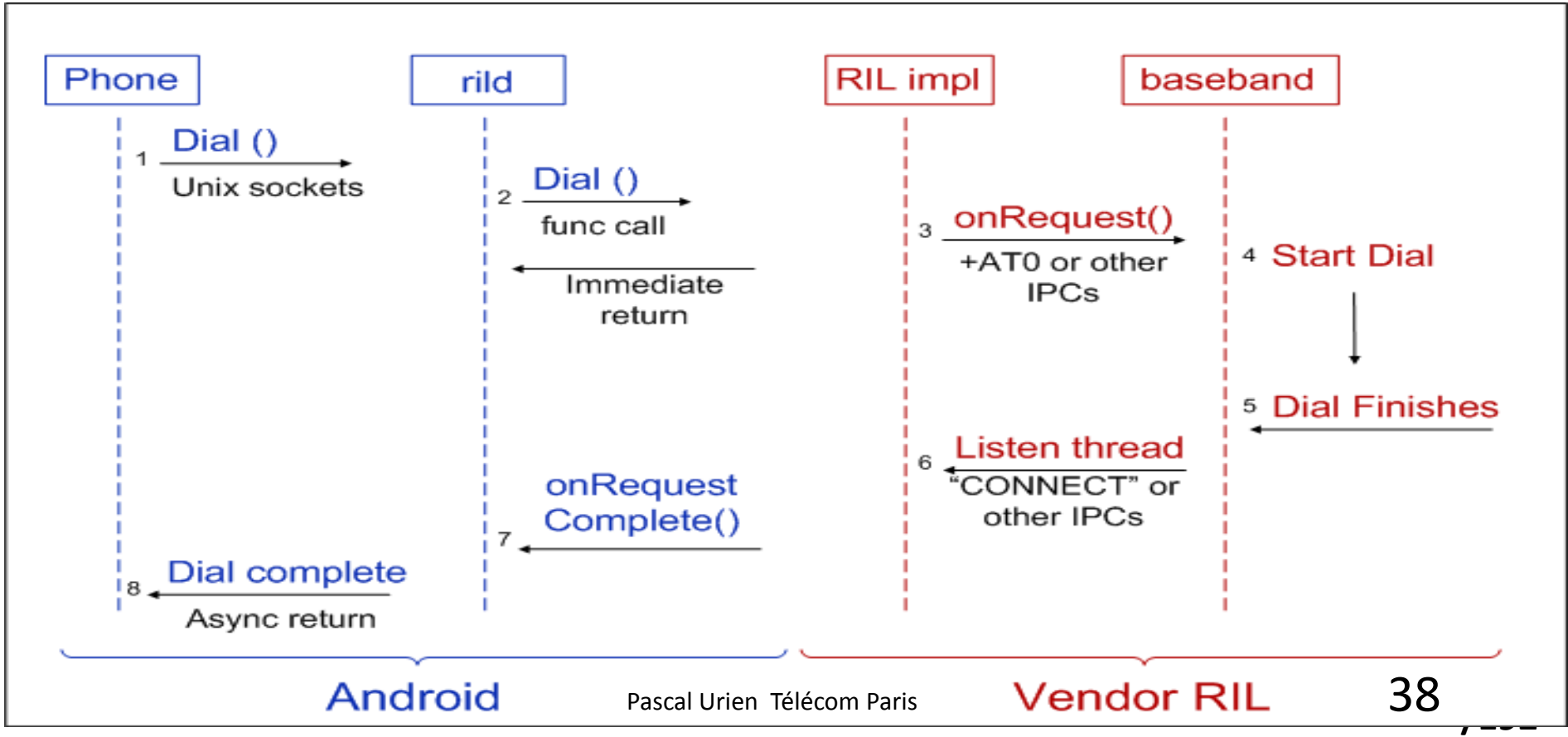


RIL 1/3 Radio Interface Layer

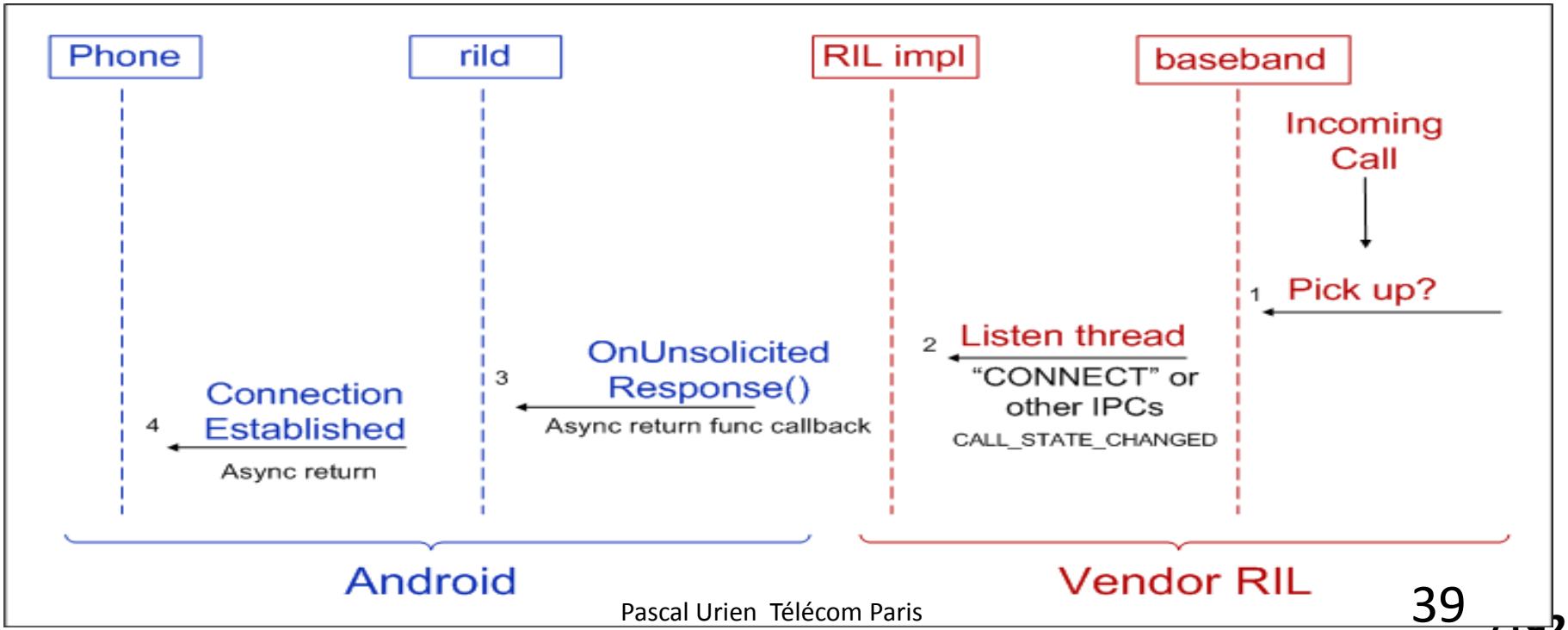
http://www.kandroid.org/online-pdk/guide/telephony.html



RIL 2/3

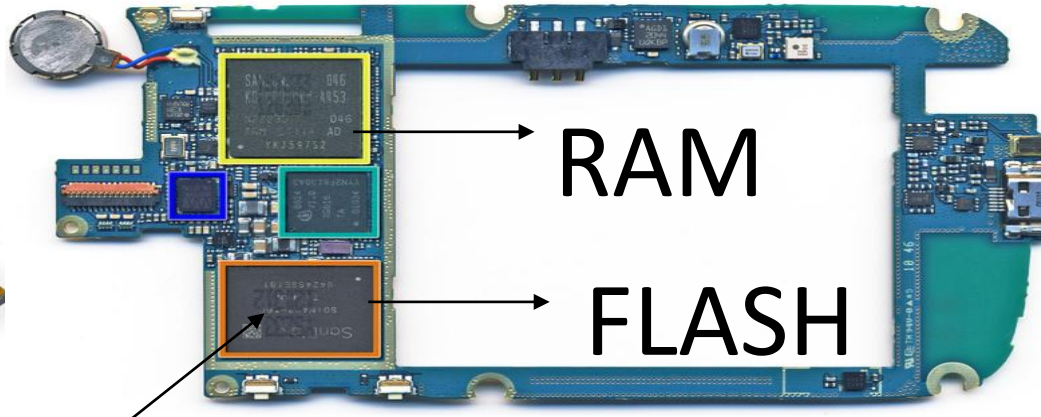

Phone
rild
RIL impl
baseband
1 Dial ()
Unix sockets
2 Dial ()
func call
Immediate return
3 onRequest()
+AT0 or other IPCs
4 Start Dial
5 Dial Finishes
6 Listen thread
"CONNECT" or other IPCs
7 onRequest Complete()
8 Dial complete
Async return
Android
Vendor RIL

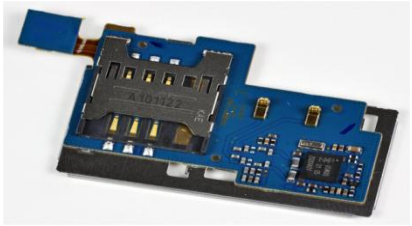
RIL 3/3


Phone
rild
RIL impl
baseband
Incoming Call
1 Pick up?
2 Listen thread
"CONNECT" or other IPCs
CALL_STATE_CHANGED
3 OnUnsolicited Response()
Async return func callback
4 Connection Established
Async return
Android
Vendor RIL

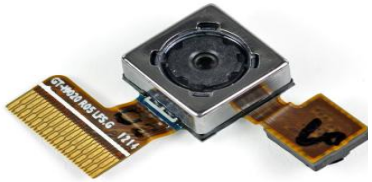
Broadband wi-fi chip

MIC, HP



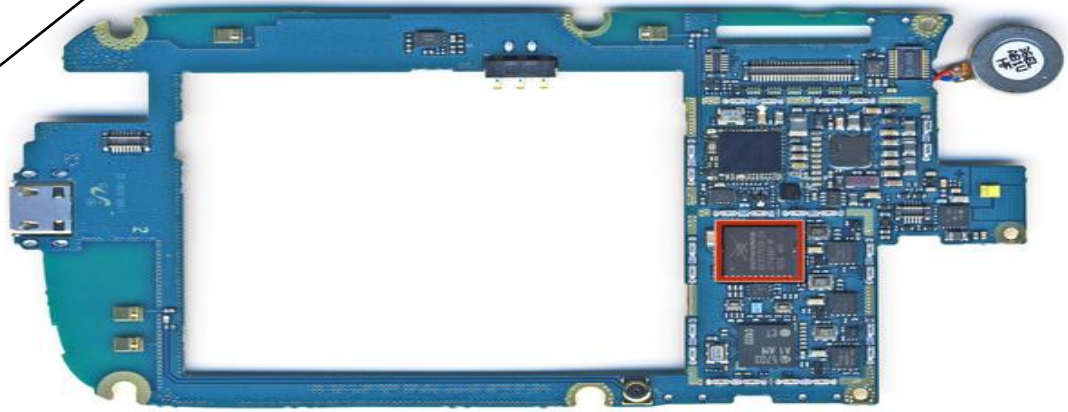


SIM, PN544



CAMERA

- Big players on the motherboard include:
  - Skyworks SKY77529 Tx Front-End Module for Dual-Band GSM / GPRS / EDGE
  - SanDisk SDIN4C2 16GB MLC NAND flash.
  - Samsung KB100D00WM-A453 memory package and S5PC110A01 1GHz Cortex A8 Hummingbird Processor.
  - Infineon 8824 XG616 X-Gold baseband processor
  - Wolfson Microelectronics WM8994 ultra-low power audio codec.



http://www.ifixit.com/Teardown/Nexus-S-Teardown/4365/1

Application Android

- Une application Android comporte au plus quatre composants :
  - l'activité (Activity)
  - le service (Service)
  - le fournisseur de contenu (Content Provider)
  - le gestionnaire de broadcast (Broadcast Receiver)
- Les composants peuvent être déclarés publics ou privés.

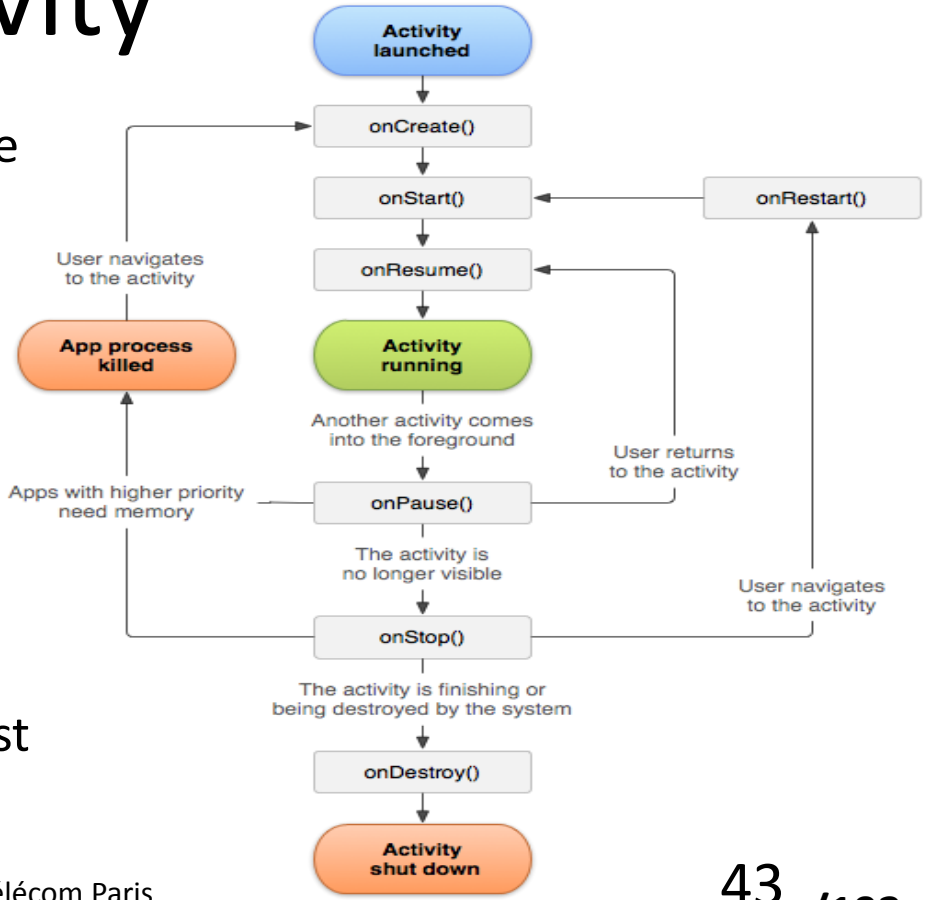
http://developer.android.com/guide/components/fundamentals.html

Application Android

- Le système d'exploitation Android est un système Linux multi-utilisateurs
  - Chaque application est associée à un compte utilisateur (user) diffèrent
- Par défaut, le système attribue à chaque application un User-ID (UID) unique et non connu de l'application.
  - Les autorisations d'accès aux fichiers sont basées sur le USER-ID.
  - C'est une politique d'accès DAC (Discretionary Access Control)
- Par défaut, chaque application s'exécute dans son propre processus Linux.
  - Android démarre le processus lorsque un composant d'une application doit être exécuté.
  - Il arrête le processus lorsque l'application n'est plus utilisée ou lorsque le système a besoin de mémoire pour d'autres applications.
- Chaque processus possède sa propre machine virtuelle (VM), est en conséquence exécuté dans un environnement isolé des autres applications.
- Le système Android met en œuvre le principe du moindre privilège. Chaque application accède par défaut uniquement aux composants nécessaire à son exécution.

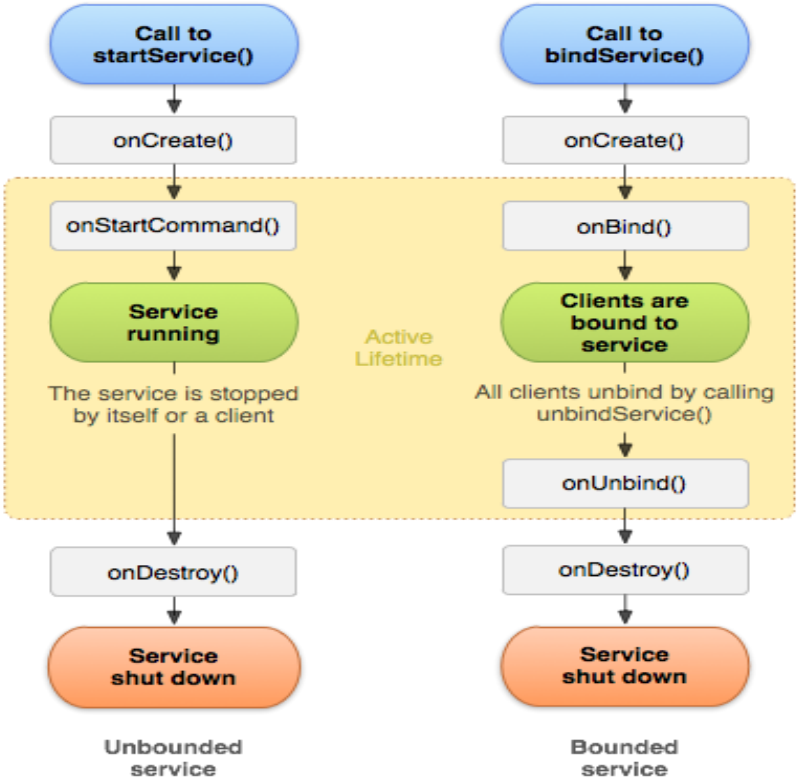
Activity

- Une activité représente un écran unique avec une interface utilisateur.
- Par exemple, une application de messagerie peut avoir une activité qui affiche une liste de nouveaux e-mails, une autre activité pour composer un courriel et une autre activité pour la lecture des emails.
- Bien que les activités travaillent ensemble pour former une expérience utilisateur cohérente dans l'application de messagerie électronique, chacune est indépendante des autres.



Service

- Un service est un composant qui s'exécute en arrière-plan.
- Un service ne gère pas une interface utilisateur.
- Par exemple, un service peut jouer de la musique en arrière-plan, ou il peut extraire des données via le réseau sans intervention de l'utilisateur.
- Un autre composant, comme une activité, peut démarrer le service et interagir avec ce dernier.



Content Provider

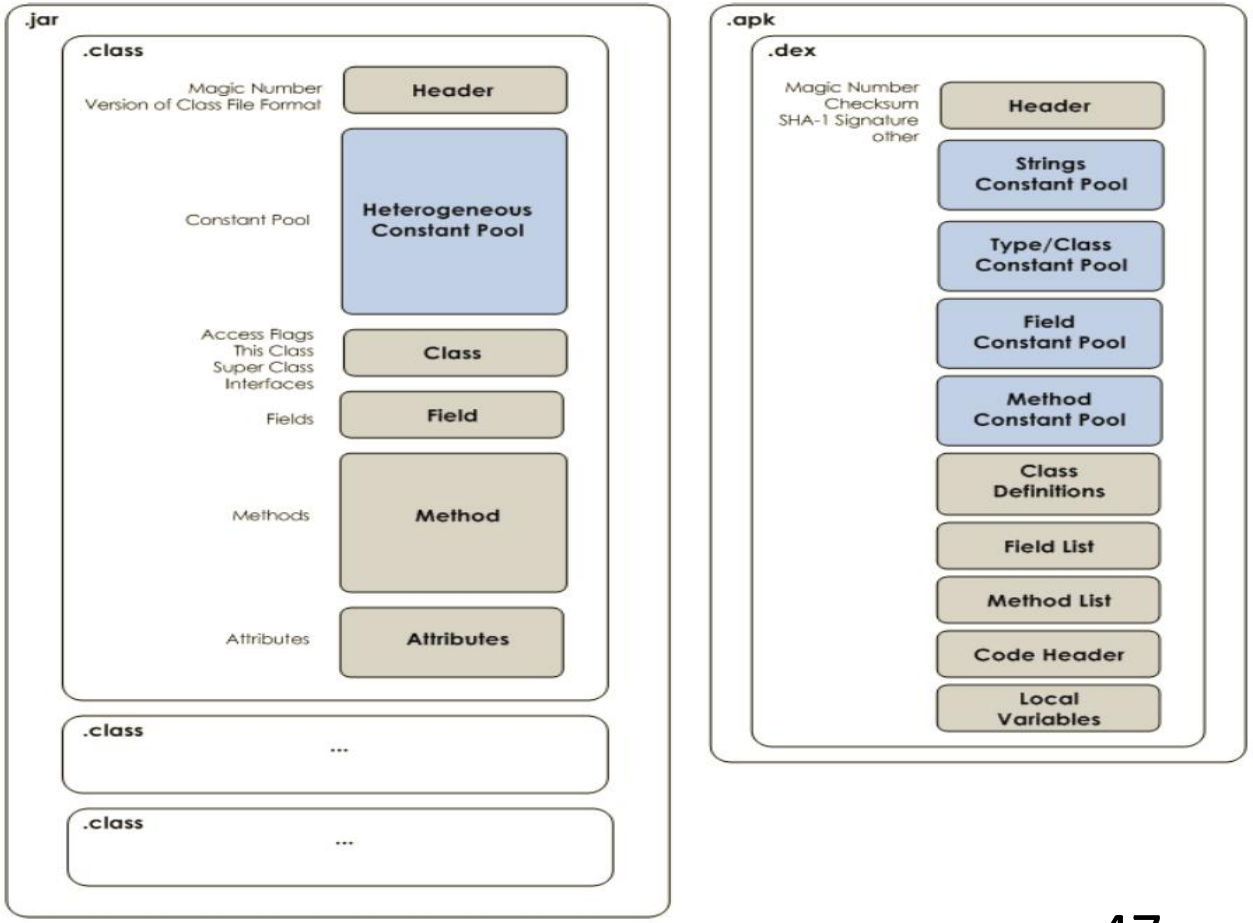
- Un Content Provider gère un ensemble d'information partagées, lié à l'application.
- Les données peuvent être stockées dans le système de fichiers, une base de données SQLite, sur le web, ou tout autre endroit de stockage persistant auquel peut accéder l'application.
- Grâce au Content Provider, d'autres applications peuvent lire ou modifier les données (si cette opération est autorisée).
- Par exemple, le système Android fournit un Content Provider qui gère les contacts de l'utilisateur.
- Par conséquent, une application munie des autorisations nécessaires peut accéder a ce Content Provider.

Broadcast Receiver

- Un Broadcast Receiver est un composant qui interagit avec les messages broadcast du système
- Les broadcast système par exemple, une indication d'extinction de l'écran, une alerte de batterie faible, ou la capture d'une photo.
- Les applications peuvent également produire des messages de broadcast, notifiant par exemple la disponibilité de données.
- Un Broadcast Receiver ne gère pas d'interface utilisateur, mais peut cependant afficher une barre de statut.
- Typiquement un Broadcast Receiver démarre un service pour assurer le traitement d'un évènement.

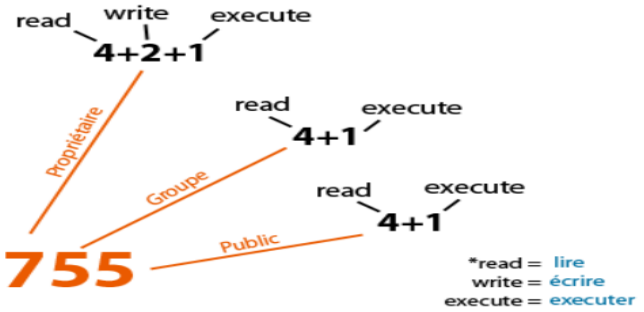
http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

Format APK Pour une Application Android



Application et Système de Fichiers

- Code
  - /system/app/app.apk
- Données
  - /data/data/app_package/databases/DataBase.db
  - /data/data/app_package/files
  - /data/data/app_package/lib

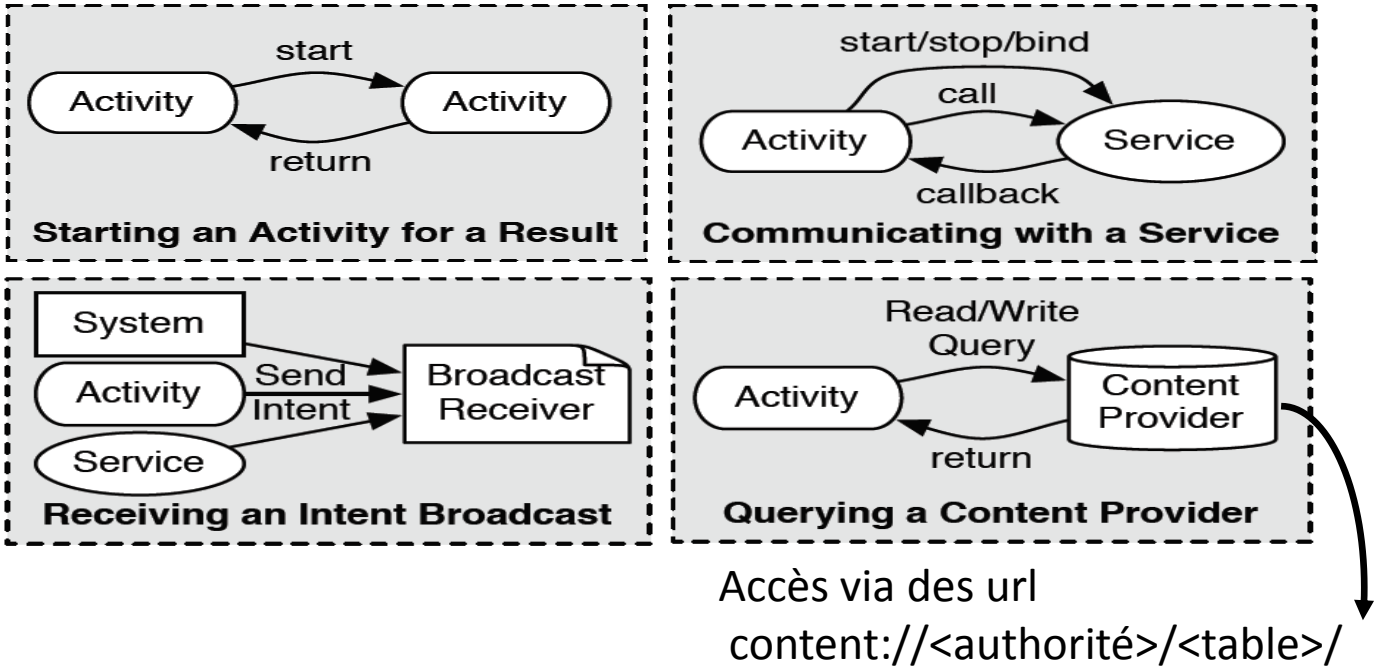


Android: Intent

- Les composants activités, services, et broadcast receivers, sont activés par un message asynchrone dénommé Intent.

- Les messages Intent sont échangées via une architecture logicielle nommée Inter-Process-Communication (IPC)

Interaction entre composants



Accès via des url
content://<authorité>/<table>/

Construction du Noyau Linux

- Le noyau est construit à l'aide d'un outil nommé GIT
  - On peut trouver un exemple d'archive à https://github.com/CyanogenMod/samsung-kernel-crespo
- L'image du noyau se nomme zImage
- Exemple de commandes:
  - http://devjlanza.wordpress.com/2011/09/02/android-nexus-s-custom-kernel/
  - $ cd ~/mydroid
  - $ git clone git://android.git.kernel.org/kernel/samsung.git
  - $ cd samsung
  - $ export ARCH=arm
  - $ export CROSS_COMPILE=<path_to_mydroid>/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
  - $ export CCOMPILER=<path_to_mydroid>/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-
  - $ make ARCH=arm clean
  - $ make ARCH=arm herring_defconfig
  - $ make ARCH=arm menuconfig
  - $ make -j2 ARCH=arm

samsung\arch\arm\boot\zImage

Construction du build Android (ROM)

- L'image (ROM) est construite à l'aide de l'outil REPO
  - http://source.android.com/source/index.html
- Exemple
  - $ mkdir ~/bin
  - $ PATH=~/bin:$PATH
  - $ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
  - $ chmod a+x ~/bin/repo
  - $ mkdir WORKING_DIRECTORY
  - $ cd WORKING_DIRECTORY
  - $ repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1
  - $ repo sync
  - $ source build/envsetup.sh
  - lunch crespo
  - $ make -j4
  - Le noyau (zImage) se trouve en \device\samsung\crespo\kernel

- Trois images sont générées dans out/target/product/crespo
  - boot.img, recovery.img, system.img
- Diverses applications sont générées dans out/host/linux-x86/bin
  - adb, emulator, fastboot, mkbootimg

Binaires Propriétaires pour le Nexus S

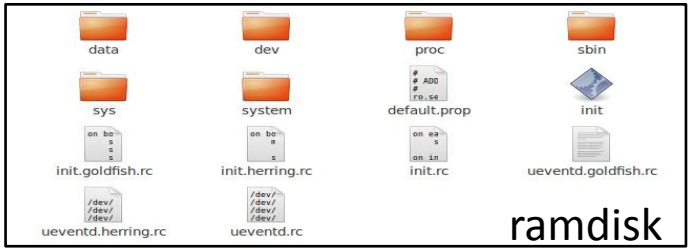
Hardware Component	Company
Orientation sensor	AKM
WiFi, Bluetooth, GPS	Broadcom
Graphics	Imagination
NFC	NXP
GSM	Samsung

Nexus S: boot.img & recovery.img

- Le mobile android comporte plusieurs “devices”, dont on obtient la structure (via adb shell) à l’aide de la commande (pour un Nexus S)
  - #cat /proc/mtd

```
dev: size erasesize name
mtd0: 00040000 00020000 "misc"
mtd1: 00500000 00020000 "recovery"
mtd2: 00280000 00020000 "boot"
mtd3: 04380000 00020000 "system"
mtd4: 04380000 00020000 "cache"
mtd5: 04ac0000 00020000 "userdata“
```

- Les partitions recovery et boot sont localisées en /dev/mtd/mtd1 et /dev/mtd/mtd2
- Pour un Nexus S une copie de mtd1 est stockée en /system/recovery.img.
  - A chaque coupure d’alimentation le contenu de /system/recovery.img est transféré dans /dev/mtd/mtd1
- La format de boot.img et recovery.img comporte
  - En préfixe de 2 Ko (header)
  - Une image noyau compressée (gzip kernel)
  - Un ramdisk, l’image d’un petit système de gestion de fichiers
  - Un suffixe optionnel
- En particulier le fichier default.prop contient la ligne ro.secure=1 qui permet à ADB d’être root.



http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack%2C_Edit%2C_and_Re-Pack_Boot_Images

Nexus S: Détails des partitions

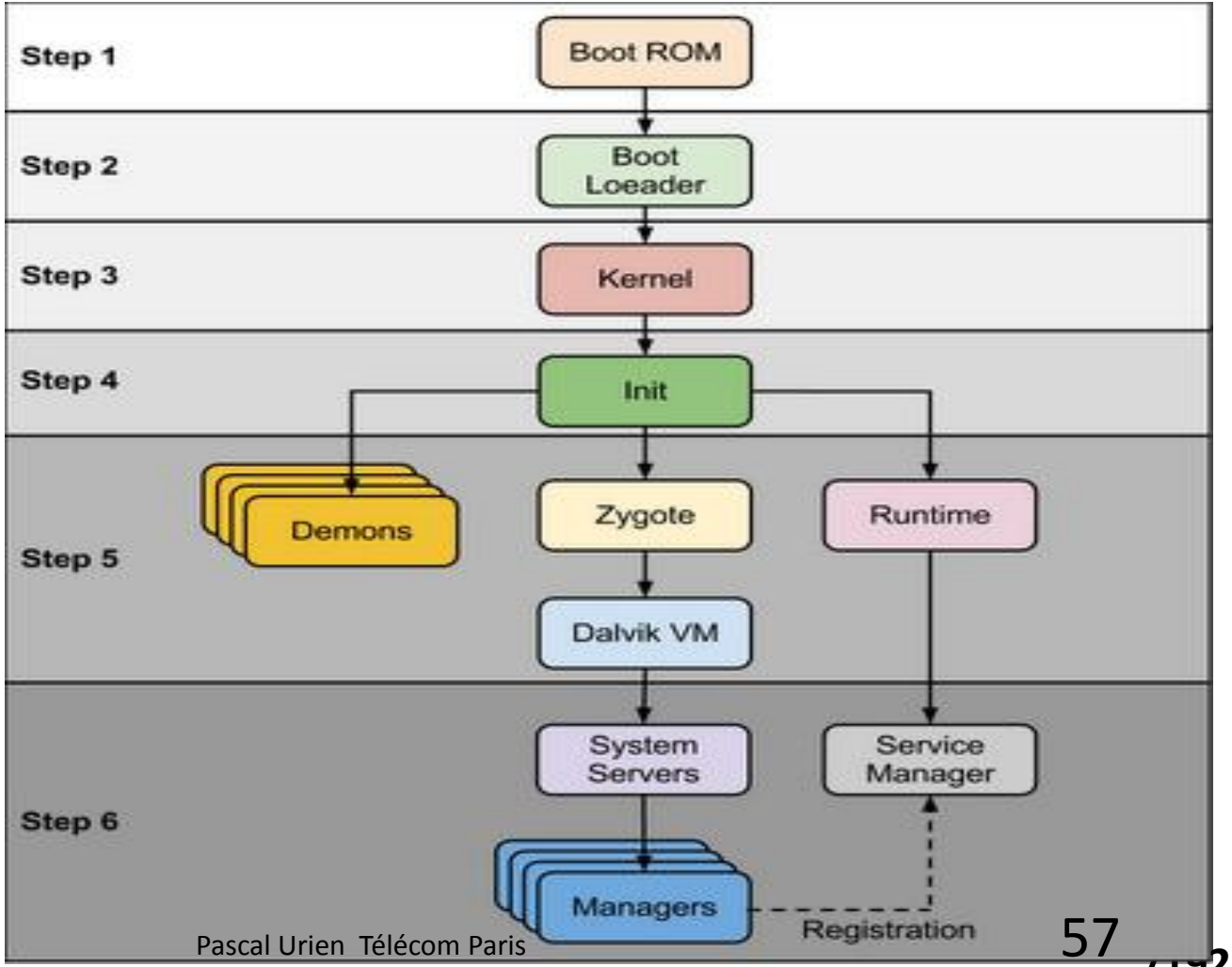
- /boot
  - C'est la partition de boot. Elle comporte le noyau Android et le ramdisk. Sa présence est nécessaire au démarrage du système Android. En cas de destruction de la partition recovery, une nouvelle ROM (incluant une partition boot) doit être installée avant la mise hors tension du système.
- /system
  - Cette partition contient l'OS Android, c'est-à-dire les blocs logiciels autres que le noyau et le ramdisk. La destruction de cette partition implique un démarrage en mode recovery, pour l'installation d'une nouvelle ROM.
- /recovery
  - C'est une partition de boot alternative, qui assure le démarrage du système à des fins de maintenance ou de mise à jour.
- /data
  - Cette partition contient des données utilisateur, telles que contacts, sms, paramètres, et applications Android installées. Son contenu est effacé lors d'une opération de retour à la configuration usine associée à une ROM (*factory reset*).
- /cache
  - Cette partition stocke des données fréquemment consultées (cache de navigateur WEB). Sa destruction n'a pas d'impact sur le bon fonctionnement du système.
- /misc
  - Cette partition contient divers paramètres de configuration sous forme d'indications activé/désactivé, relativement au réseau, au contexte USB ou des options matérielles. Sa destruction ou l'altération de son contenu entraine des disfonctionnements du terminal Android

Le format update.zip

- Un fichier update.zip comporte les éléments suivants
  - Une image d'un système de fichiers
  - Un répertoire \META-INF\com\google\android qui stocke un ficher updater-script
    - copy_dir, format, delete, delete_recursive, set_perm, set_perm_recursive, show_progress, symilink
  - Une signature
    - java -jar signapk.jar certificate.pem key.pk8 myupdate.zip update.zip
    - java -classpath testsign.jar testsign update.zip update-signed.zip
- Références
  - http://fokke.org/site/content/howto-create-android-updatezip-package
  - http://www.londatiga.net/it/how-to-create-android-update-zip-package/
  - http://www.synfulgeek.com/main/index.php/articles/76-scratchpad-documenting-edify-commands-for-android-updater-scritps-based-off-of-kernel-source-code

```
META-INF/
   +- MANIFEST.MF
   +- CERT.SF
   +- CERT.RSA
   +- com/
     +- google/
          +- android/
               +- update-script
               +- update-binary
               +- updater-script
system/
  +- etc/
     +- sysctl.conf
     +- security/
          +- cacerts.bks
```

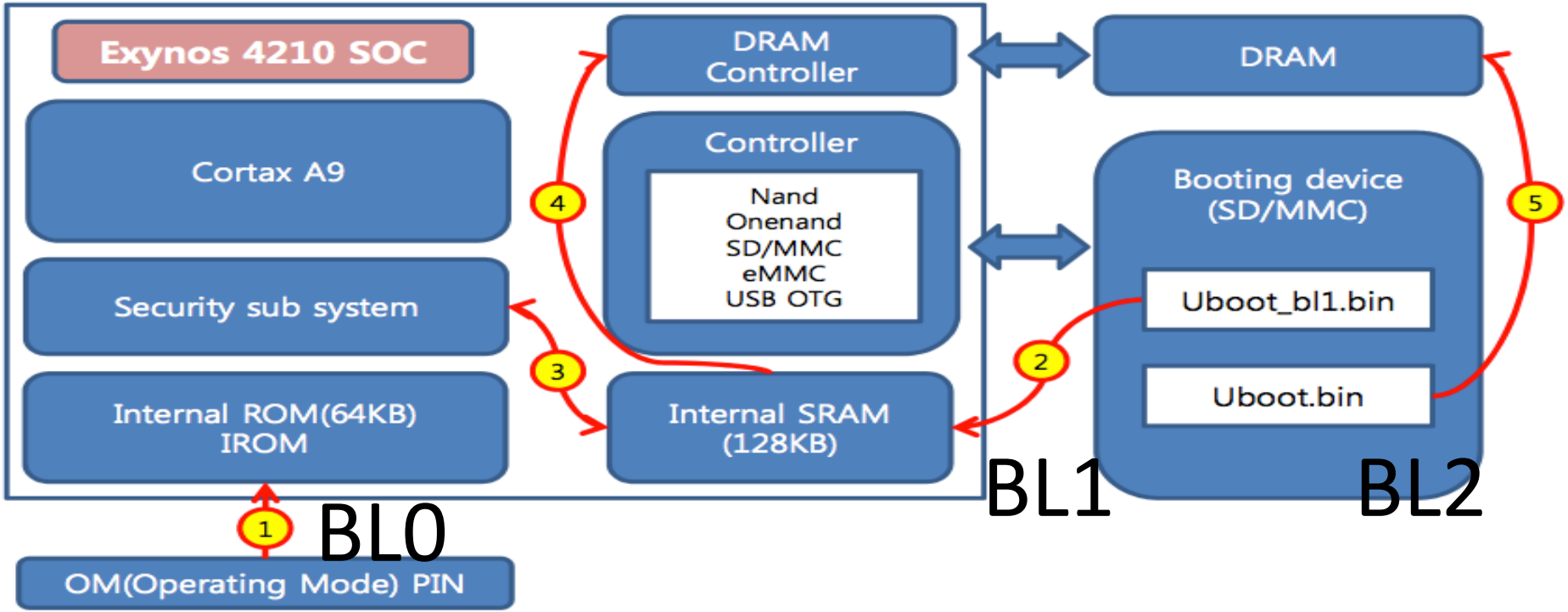
Android: Boot



Exemple de Séquence de Boot

- Trois niveaux
  - Niveau dit BL0. Une ROM interne initialise le hardware et cherche un *device* contenant un code de boot de niveau 1
  - Niveau dit BL1. Le code BL1 est chargé dans une RAM interne, exécuté, puis le code de niveau 2 est chargé dans la SDRAM
  - Niveau dit BL2. Le code BL2 est chargé (par exemple uboot.bin)

http://javigon.com/2012/08/24/from-poweron-to-android-the-boot-sequence/


Exynos 4210 SOC
Cortax A9
Security sub system
Internal ROM(64KB)
IROM
DRAM
Controller
Controller
Nand
Onenand
SD/MMC
eMMC
USB OTG
Internal SRAM
(128KB)
DRAM
Booting device
(SD/MMC)
Uboot_bl1.bin
Uboot.bin
1
2
3
4
5
BL0
BL1
BL2
OM(Operating Mode) PIN

Android: Boot 1/2

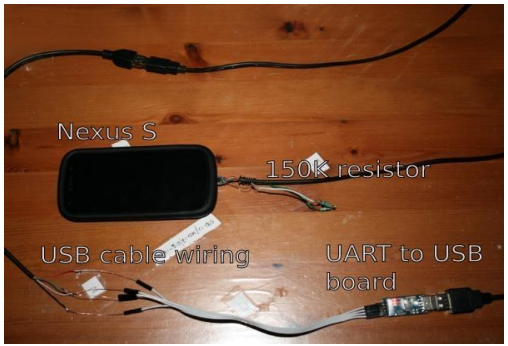
- Step 1 : Power On and System Startup
  - Lors de la mise sous tension le système exécute un code de démarrage, puis il le charge en RAM et démarre la procédure de *Bootloader*
- Step 2 : Bootloader
  - Bootloader est un petit programme exécuté avant le chargement du système d'exploitation Android; il est dédié à une carte mère particulière. Le fabricant du terminal peut utiliser un bootloader du marché tel que redboot, **uboot**, qi bootloader, ou une version propriétaire. Ce logiciel n'est pas un élément d'Android, il permet aux opérateurs de brider certaines fonctionnalités du système.
  - L'éxécution de Bootloader est réalisée en deux étapes (BL1, BL2), la détection de la mémoire RAM et le chargement de logiciels, puis la configuration de divers paramètres (réseau, mémoires, …).
  - Le code du Bootloader Android est stocké en <AndroidSource>\bootable\bootloader\legacy\usbloaderlegacy; il contient deux fichiers importants:
    - 1. init.s - Initializes stacks, zeros the BSS segments, call _main() in main.c
    - 2. main.c - Initializes hardware (clocks, board, keypad, console), creates Linux tags
- Step 3: Kernel
  - Le noyau Android démarre d'une manière similaire aux versions de Linux pour les PCs. A la fin de la phase d'initialisation du système, il recherche le fichier **"init"** et démarre le premier processus.
- Step 4: init process
  - Init est le premier processus (root processus), il réalise le montage des répertoires tels que /sys, /dev, /proc et éxécute le script init.rc
  - Le code source du processus init est à : <android source>/system/core/init
  - Le fichier init.rc se trouve en: <android source>/system/core/rootdir/init.rc
  - Le fichier readme.txt est localisé en: <android source>/system/core/init/readme.txt

Android Boot: 2/2

- Step 5: Zygote and Dalvik
  - Zygote est un processus dédié à la gestion de machine virtuelle DALVIK, qui pré-charge et initialise les bibliothèques de classe.
  - Zygote loading process
    - 1. Load ZygoteInit class, Source Code :<Android Source> /frameworks/base/core/java/com/android/internal/os/ZygoteInit.java
    - 2. registerZygoteSocket() - Registers a server socket for zygote command connections
    - 3. preloadClasses() - “preloaded-classes” is simple text file contains list of classes that need to be preloaded, you can find “preloaded-classes” file at <Android Source>/frameworks/base
    - 4. preloadResources() - preloadReaources means native themes and layouts, everything that include android.R file will be load using this method.
  - A cette étape une animation apparait sur l’écran du terminal.
- Step 6: System Service or Services
  - L’entité runtime demande à Zygote de démarrer les serveurs du système (system servers). Le code de ces serveurs est écrit en langage natif ou en langage java. Ces serveurs réalisent tous les services du système
  - Les codes sources se trouvent dans la classe ZygoteInit et la méthode “startSystemServer”
- Step 7 : Boot Completed
  - Le boot du système s’achève lorsque tous les services sont chargés en mémoire et sont activés

Console Série

- Certains mobiles Android possèdent une interface série, utilisable durant la séquence de boot
- http://redmine.replicant.us/projects/replicant/wiki/SamsungSerial
  - It is possible to setup a serial console on the Nexus S. It will show:
    - the 1st bootloader output
    - the 2nd bootloader output
    - the 2nd bootloader #2 output
    - the fiq debugger
    - (the kernel output if enabled)



Mémoires de Stockage

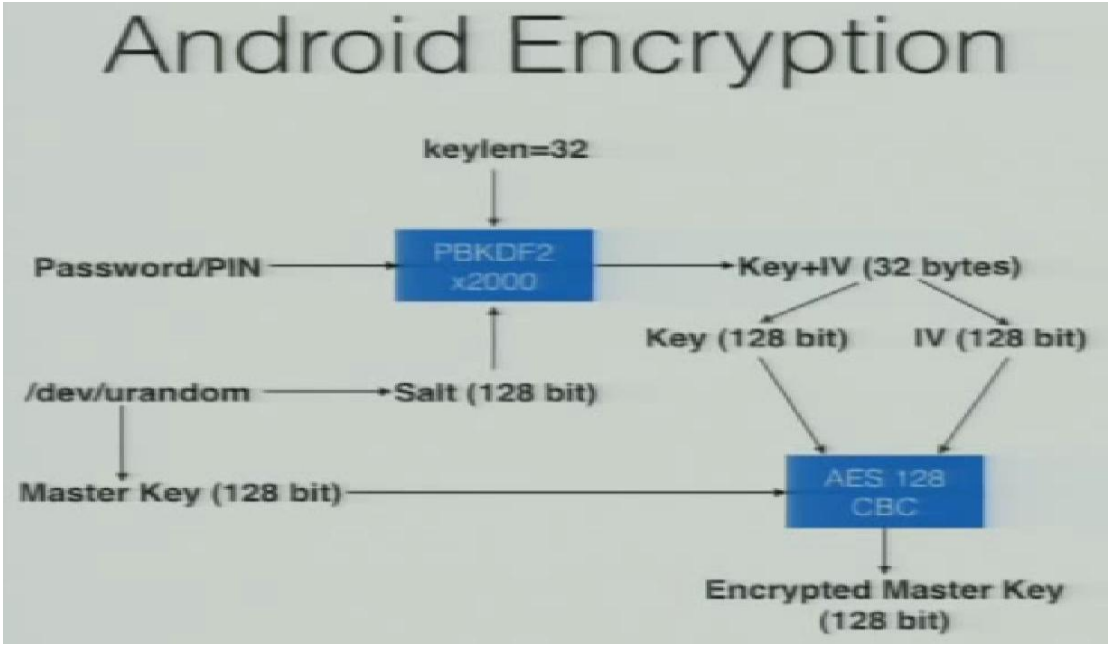
- Un système Android comporte deux types de mémoire de stockage
  - Mémoire Interne, de l'ordre de 1Go. Depuis Android 3.0 cette mémoire peut être chiffrée (plus exactement la partition des données utilisateur). Le système de fichier associé est de type UNIX, avec une politique d'accès DAC.
  - Mémoire Externe, souvent appelée SDCard. Cette mémoire peut être chiffrée pour certains modèles. Le système de fichiers associé (NTFS…) ne supporte cependant pas le DAC UNIX.

Chiffrement du système de fichiers

- Les données utilisateurs (stockées dans la partition données utilisateur) peuvent être chiffrées selon le procédé dmcrypt (importés de linux)
  - Chiffrement AES en mode CBC, avec IV
  - La clé maitre est nommé DEK
  - La clé maitre est chiffrée avec une clé SEK, liée au mot de passe
- Chaque secteur (512 octets) est chiffré en mode AES128_cbc avec
  - IV= ESSIV (Encrypted Salt Sector Initialization Vector)
    - SALT=sha256(DEK)
    - ESSIV=AES256_ecb(key=SALT,sector_number)
  - AES128 key= DEK

https://github.com/viaforensics/android-encryption/blob/master/decrypt.py

Détails des clés de chiffrement



http://www.realhacker.net/videos/how-to-gain-access-to-android-user-data?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+realhacker%2FKXef+%28Real+Hacker+Network%29

https://www.defcon.org/images/defcon-20/dc-20-presentations/Cannon/DEFCON-20-Cannon-Into-The-Droid.pdf

Rooting & Recovery

- Rooting
  - La procédure de "rooting" d'un système Android dépend du terminal qui l'héberge et met à profit des bugs logiciels de sécurité.
  - Le code binaire de la procédure *su* est copié dans un répertoire tel que /system/xbin/su, et muni des permissions nécessaires à l'aide de la commande *chmod*.
  - De nombreux documents disponibles sur Internet décrivent les opérations nécessaires au "rooting" d'un terminal Android.
- Custom Recovery
  - Lorsque le *bootloader* Android est en mode déverrouillé (*unlocked*) il est possible d'installer une partition recovery (revovery.img) non standard (*Custom Revovery*).
  - *ClockWorkMod Recovery* est par exemple fréquemment utilisé pour le "rooting " des systèmes Android.
  - La partition *recovery* n'est pas détruite lors de l'installation d'une nouvelle ROM.

https://sites.google.com/site/tomsgt123/adb-fastboot/understanding-android

Procédure de *rooting*

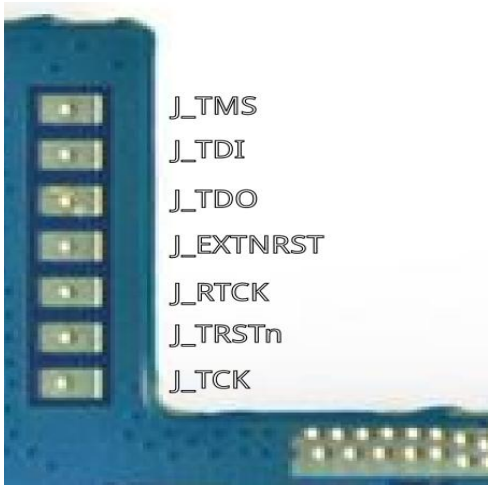
Chargement /Exécution
d'un logiciel malveillant

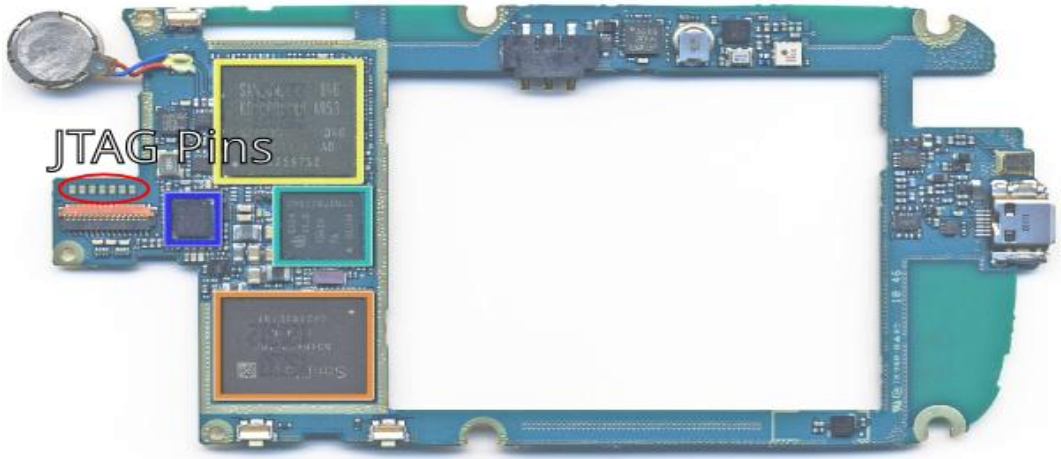
Exploit permettant d'accéder
aux privilèges root

Chargement/Exécution de
l'utilitaire "su"

Accès de la mémoire de stockage par interface JTAG

- De nombreux mobiles intègrent une interface JTAG qui permet de lire la mémoire NAND





https://sites.google.com/site/tomsgt123/adb-fastboot/understanding-android

Bootloader

- Le bootloader permet de modifier la ROM d'un système Android
  - En particulier recovery.img qui gère les mises à jour du système. Le bootloader doit être déverrouillé pour cette opération.
  - Certains mobiles interdisent le déverrouillage du bootloader
  - Lors du déverrouillage du bootloader les données utilisateur sont effacées

Le mode fastboot

- Lors de la mise en tension du mobile un combinaison de touches (ex. power_on volume_up) permet de passer en mode fastboot
- La commande “adb reboot bootloader” provoque également le passage en fastboot
- Par défaut le bootloader est verrouillé (locked). Pour verrouiller / déverrouiller le bootloader
  - fastboot oem unlock
  - fastboot oem lock
- Pour flasher un mobile
  - fastboot flashall –w (en mode fasboot)

L'utilitaire Fastboot

```
C:\vaio\android\platform-tools>fastboot
usage: fastboot [ <option> ] <command>

commands:
  update <filename>                        reflash device from update.zip
  flashall                                 flash boot + recovery + system
  flash <partition> [ <filename> ]         write a file to a flash partition
  erase <partition>                        erase a flash partition
  getvar <variable>                        display a bootloader variable
  boot <kernel> [ <ramdisk> ]              download and boot kernel
  flash:raw boot <kernel> [ <ramdisk> ]    create bootimage and flash it
  devices                                  list all connected devices
  continue                                 continue with autoboot
  reboot                                   reboot device normally
  reboot-bootloader                        reboot device into bootloader
  help                                     show this help message

options:
  -w                                       erase userdata and cache
  -s <serial number>                       specify device serial number
  -p <product>                             specify product name
  -c <cmdline>                             override kernel commandline
  -i <vendor id>                           specify a custom USB vendor id
  -b <base_addr>                           specify a custom kernel base address
  -n <page size>                           specify the nand page size. default: 2048
```

ADB : Android Debug Bridge

```
device commands:
  adb push <local> <remote>    - copy file/dir to device
  adb pull <remote> [<local>]  - copy file/dir from device
  adb sync [ <directory> ]     - copy host->device only if changed
                                 (-l means list but don't copy)
                                 (see 'adb help all')
  adb shell                    - run remote shell interactively
  adb shell <command>          - run remote shell command
  adb emu <command>            - run emulator console command
  adb logcat [ <filter-spec> ] - View device log
  adb forward <local> <remote> - forward socket connections
                                 forward specs are one of:
                                   tcp:<port>
                                   localabstract:<unix domain socket name>
                                   localreserved:<unix domain socket name>
                                   localfilesystem:<unix domain socket name>
                                   dev:<character device name>
                                   jdwp:<process pid> (remote only)
  adb jdwp                     - list PIDs of processes hosting a JDWP transport
  adb install [-l] [-r] [-s] <file> - push this package file to the device and install it
                                 ('-l' means forward-lock the app)
                                 ('-r' means reinstall the app, keeping its data)
                                 ('-s' means install on SD card instead of internal
                                       storage)
  adb uninstall [-k] <package> - remove this app package from the device
                                 ('-k' means keep the data and cache directories)
  adb bugreport                - return all information from the device
                                 that should be included in a bug report.
```

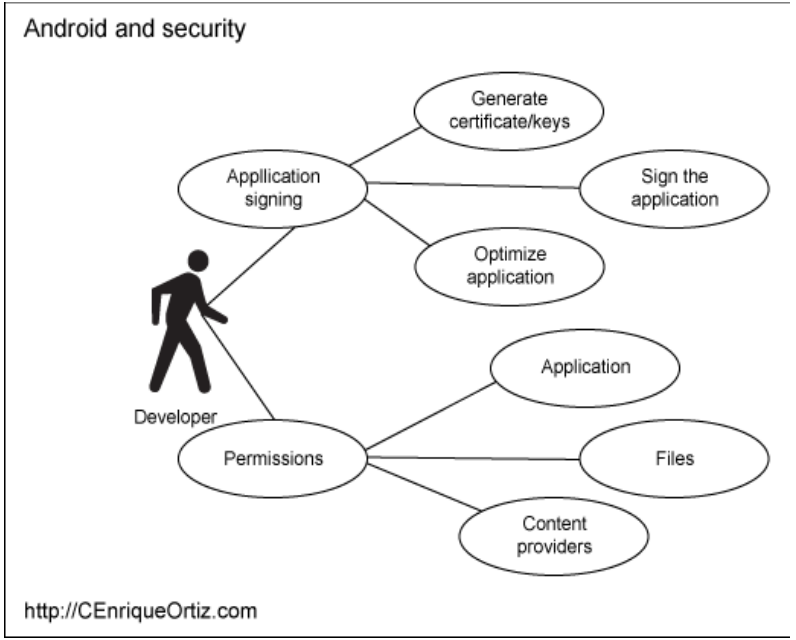
```
  adb backup [-f <file>] [-apk|-noapk] [-shared|-noshared] [-all] [-system|-nosystem] [<packages...>]

  adb restore <file>           - restore device contents from the <file> backup archive

  adb help                     - show this help message
  adb version                  - show version num

scripting:
  adb wait-for-device          - block until device is online
  adb start-server             - ensure that there is a server running
  adb kill-server              - kill the server if it is running
  adb get-state                - prints: offline | bootloader | device
  adb get-serialno             - prints: <serial-number>
  adb status-window            - continuously print device status for a specified device
  adb remount                  - remounts the /system partition on the device read-write
  adb reboot [bootloader|recovery] - reboots the device, optionally into the bootloader or recovery
  adb reboot-bootloader program
         - reboots the device into the bootloader
  adb root                     - restarts the adbd daemon with root permissions
  adb usb                      - restarts the adbd daemon listening on USB
  adb tcpip <port>             - restarts the adbd daemon listening on TCP on the specified port
```

Android: Sécurité

- La sécurité android repose sur quatres piliers
  - Les Sandboxes Unix, associés aux processus
  - Les permissions
  - La signature des applications
  - Le chiffrement des fichiers

Understanding security on Android
Enhance application security with sandboxes, application signing, and permissions
C. Enrique Ortiz, Developer and author, About Mobility Weblog

Tutorial: Building Secure Android Applications
http://siis.cse.psu.edu/slides/android-sec-tutorial.pdf

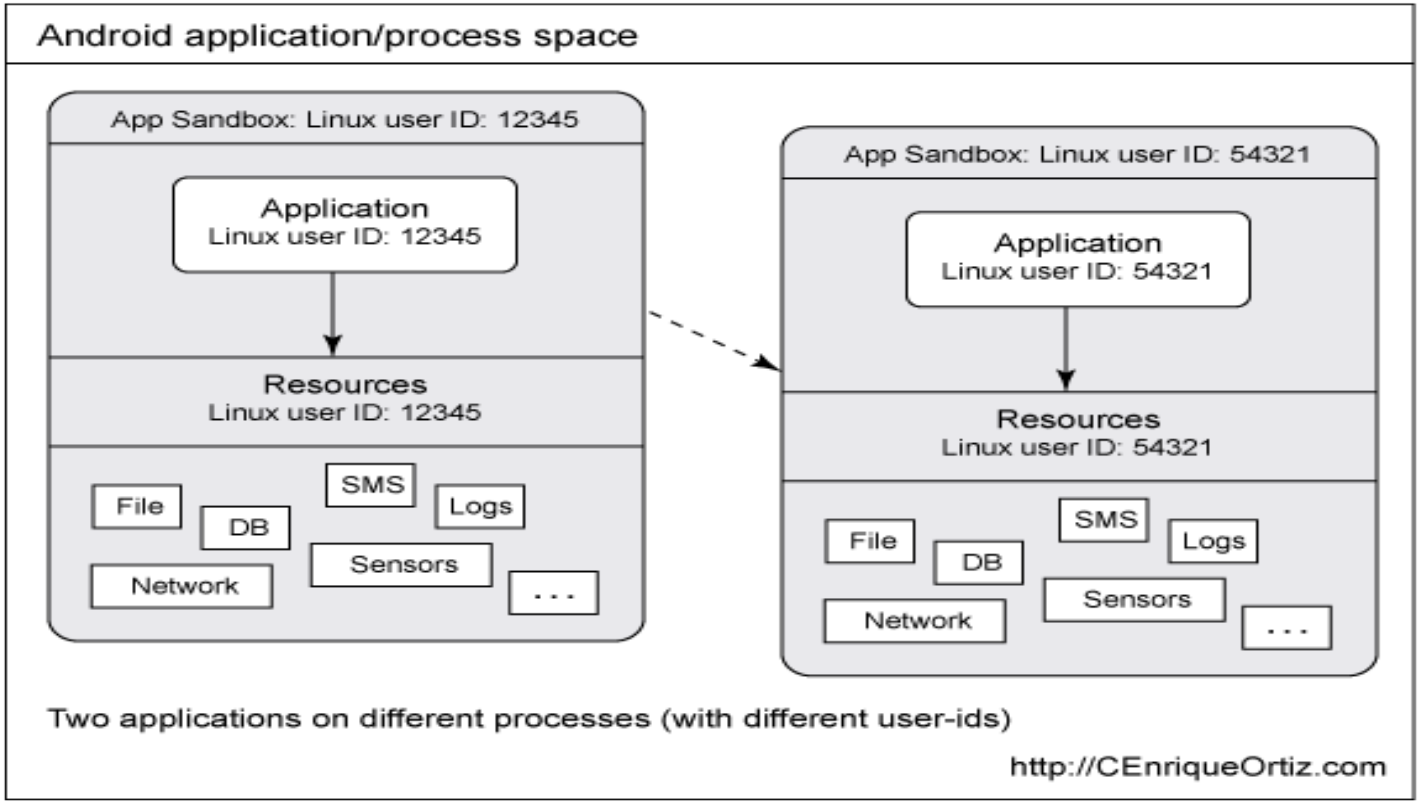


Application et User-Id

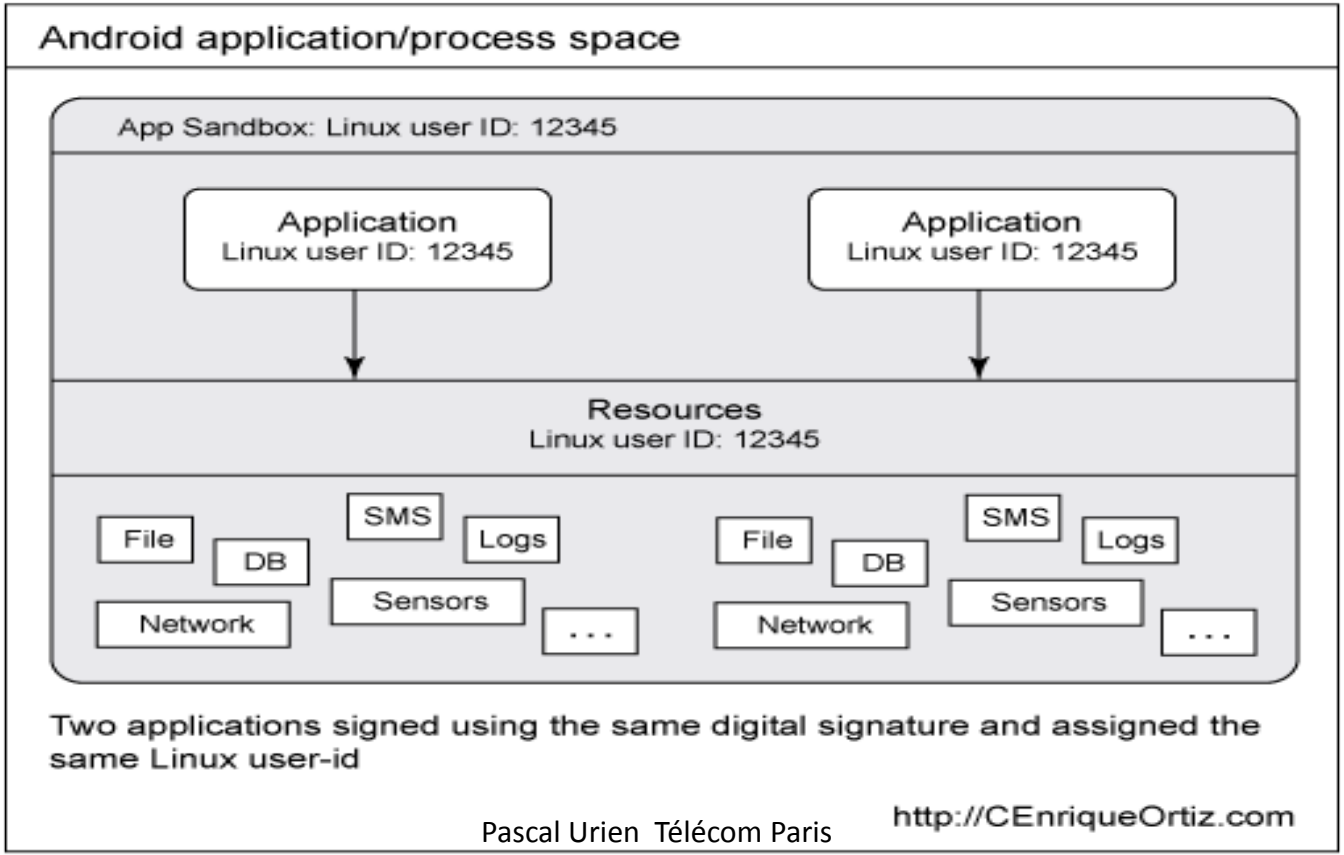
- Pour le noyau Linux associé à Android, une application possède un user-id, un id de groupe et un id de groupe secondaire.
  - Chaque application possède des droits d'accès en lecture, écriture et exécution.
  - C'est une politique de contrôle d'accès discrétionnaire (DAC).
- Le système Unix possède un compte root dont user-id est 0, et qui possède tous les droits.
- Chaque Application possède sa propre instance de Dalvik Virtual Machine (DVM)
- Deux applications signées par une même clé privée peuvent (optionnellement) utiliser un même UserID et donc partager un Sandbox identique.

David Ehringer, The_Dalvik_Virtual_Machine.pdf

Sandbox et Application User-id



Partage de processus



Permissions et Filtrage des Intents

- Le fichier Manifest.xml définit la structure d'une application, c'est-à-dire la liste de ses composants
  - Il contient les demandes de permission d'accès aux ressources du mobile
  - Il décrit les Intents traitées par l'application
- C'est le gestionnaire de la politique de sécurité de l'application

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="16" android:versionName="1.1.12" package="org.chemlab.dealdroidapp"
    android:sharedUserId="androware.test">

    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="4"/>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:process="androware.test">

        <!-- The app's UI is just a preferences page right now -->
        <activity android:name=".Preferences" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

Signature des Applications

- Chaque application (.apk) doit être signée
- Sous Eclipse on utilise les outils
  - Keytool pour la génération de clés RSA dans un keystore
  - Jarsigner pour la signature d'une application
- Un certificat peut être auto-signé

Business Model des applications Android

- Le modèle des applications Android "gratuites" est la collecte d'information destinée à la diffusion de publicités ciblées.
- La collecte d'information et son exportation implique l'autorisation par l'utilisateur d'un certain nombre de permissions.
- Cependant des techniques de collecte dites "Zéro Permission" sont possibles.

Exemple d’attaque 1

Code PIN de verrouillage sous Android

Le Code PIN de Verrouillage de Terminal

Code PIN : 1234

Vérification :

Le hash du code PIN est stocké dans le fichier /data/system/password.key

```
public byte[] passwordToHash(String password) {
if (password == null) { return null; }

String algo = null; byte[] hashed = null;
try {
byte[] saltedPassword = (password + getSalt()).getBytes();
byte[] sha1 = MessageDigest.getInstance(algo =
                                                    "SHA-1").digest(saltedPassword);
byte[] md5 = MessageDigest.getInstance(algo =
                                                     "MD5").digest(saltedPassword);
hashed = (toHex(sha1) + toHex(md5)).getBytes();
} catch (NoSuchAlgorithmException e) {
Log.w(TAG, "Failed to encode string because of missing algorithm: " + algo); }
return hashed; }
```

https://github.com/android/platform_frameworks_base/blob/master/core/java/com/android/internal/widget/LockPatternUtils.java

Le paramètre salt

```
private String getSalt() {
long salt = getLong(LOCK_PASSWORD_SALT_KEY, 0);
if (salt == 0) {
try {
salt = SecureRandom.getInstance("SHA1PRNG").nextLong();
setLong(LOCK_PASSWORD_SALT_KEY, salt);
Log.v(TAG, "Initialized lock password salt");
} catch (NoSuchAlgorithmException e) {
// Throw an exception rather than storing a password we'll never be able to recover
throw new IllegalStateException("Couldn't get SecureRandom number", e);
}
}
return Long.toHexString(salt);
}
```

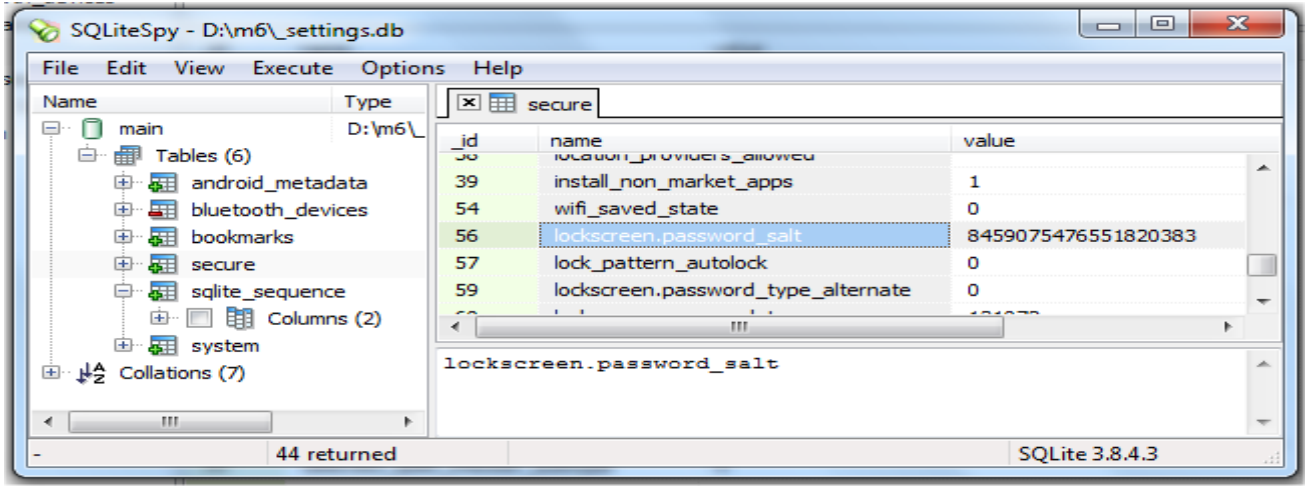
le salt reste constant et ne change qu'avec une nouvelle installation

Le salt est stocké dans le fichier /data/data/com.android.provider.settings/databases/settings.db

Par la suite l'attaque force brute est triviale

Exemple

- password.key
  - 3C04F2A6E1D8989CE4A7D69795F2C76A5C3DFB8A40 1E2A5C3C385D28229635ADFE054CBA
- settings.db



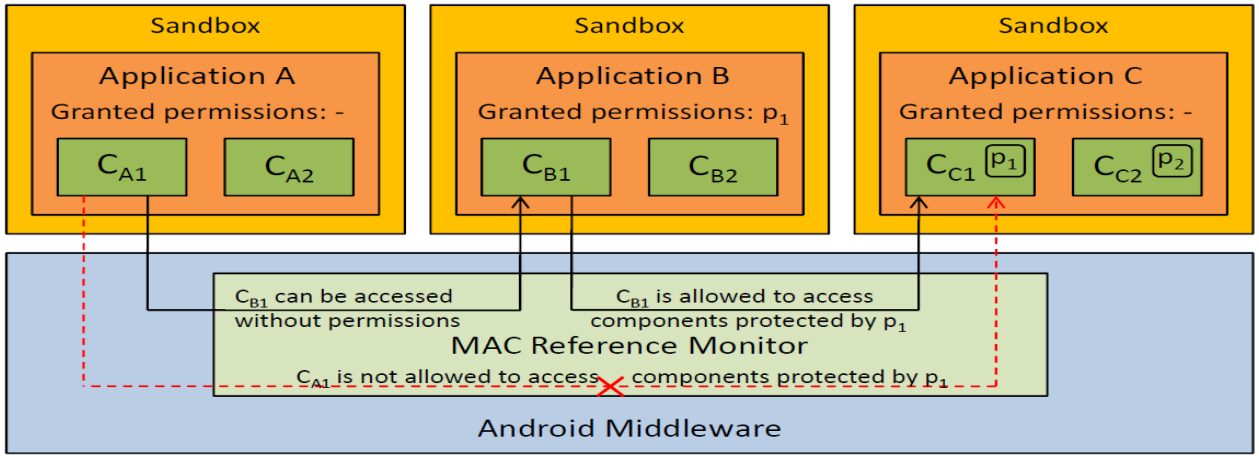
Exemple d’attaque 2

Escalade de privilèges

"Privilege Escalation Attacks on Android"
Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Marcel Winandy

Escalade de privilèges

- Les privilèges d'un composant d'une application sont contrôlés par le processus associé (au runtime)
- Les privilèges du composant appelant un composant publique ne sont pas contrôlés par le processus de l'appelé.
- L'escalade de privilège exploite ce défaut



Partage de processus

- Deux applications partagent le même processus, si elles utilisent
  - le même utilisateur Linux (défini à l'installation et précisé dans l'AndroidManifest avec sharedUser = « … »)
  - le même nom du processus (également définissable dans l'AndroidManifest avec dans la partie « application » android :process = « … »)
  - Le même certificat

- Ces possibilités sont offertes aux développeurs qui souhaitent réaliser des plugins ou des extensions d'une application

- Le partage de processus et l'escalade des privilèges permettent par exemple à une Application sans permission d'envoyer des SMS via une Application autorisée.

Exemple d’attaque 3

Zero Permission Android Application

Attaques Zéro Permission

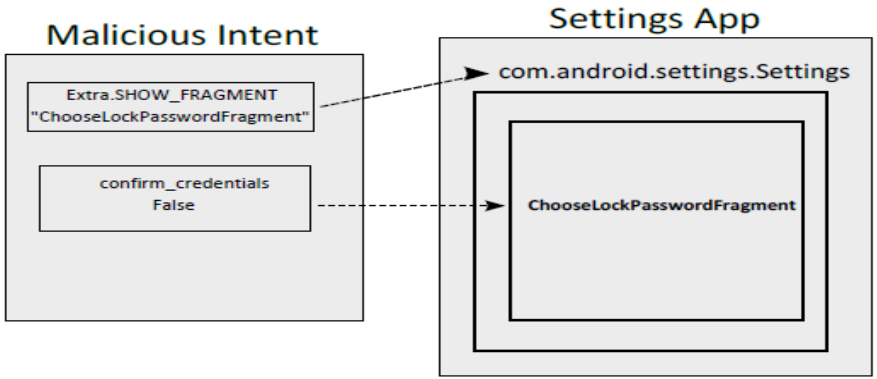
- La liste des applications et fichiers stockés dans la sdcard peut être collectée sans permission dans le répertoire /sdcard/.
  - Android ne gère pas de politique de sécurité pour les données stockées sur une sdcard externe
- Le fichier /data/system/packages.list contient la liste des applications installées, et peut être lu sans permission
  - La lecture de certain fichiers utilisés par les applications était possible en 2012
- Sans la permission PHONE_STATE il est impossible d'obtenir les paramètres IMEI ou IMSI
  - Cependant il est possible de récupérer les identifiants de l'opérateur et du fournisseur SIM.
- Sans la permission INTERNET il est possible de lancer un browser via l'INTENT URI_ACTION_VIEW, et de transférer de l'information via les paramètres de l'URI.

Paul Brodeur "Zero-permission android applications" (2012)
http://www.leviathansecurity.com/blog/zero-permission-android-applications/

Zero Permission Android Applications -Attacks and Defenses,
Vee]asha Moonsamy, Lym Batten, 2012

Exemple d'attaque 4

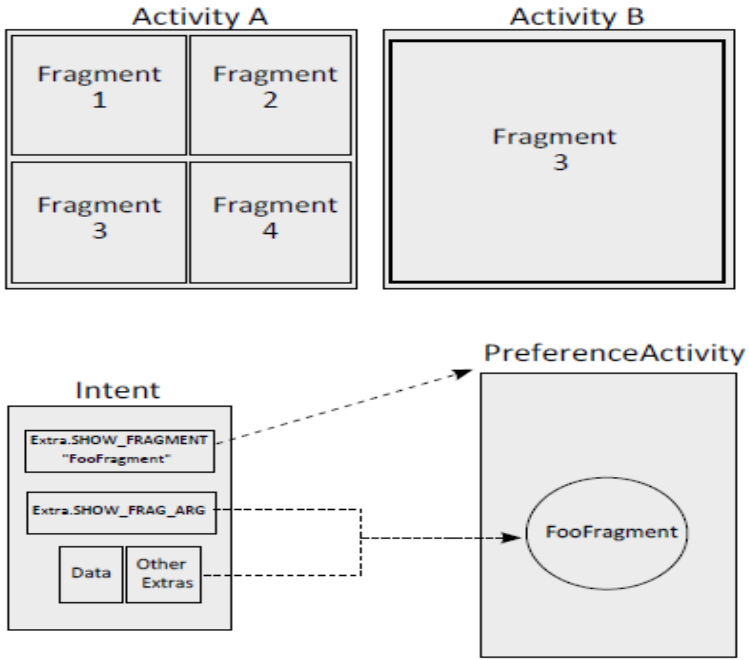
Attaques par fragmentation sur Android 4.3 (2013)



Roee Hay, IBM Security Systems, "ANDROID COLLAPSES INTO FRAGMENTS"
http://securityintelligence.com/wp-content/uploads/2013/12/android-collapses-into-fragments.pdf

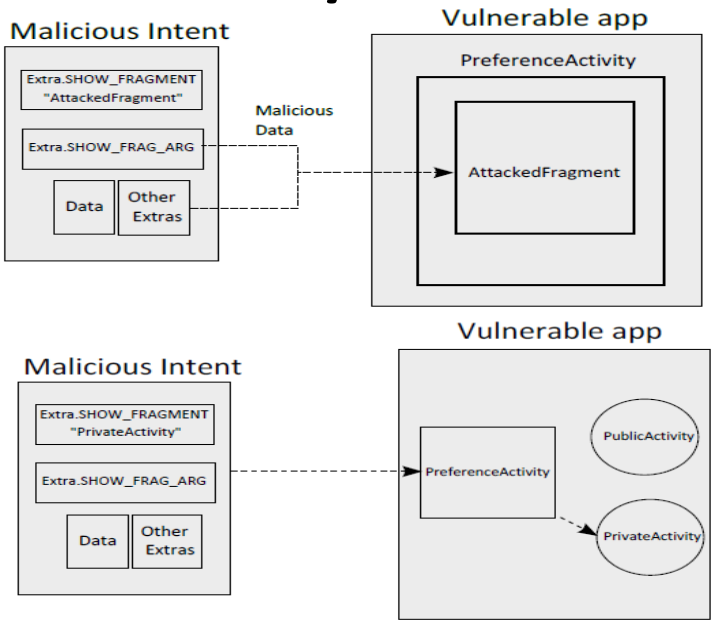
Attaques par fragmentation 1/3

- Une activité peur être construite à l'aide de Fragments
  - Un fragment est associé à une interface utilisateur (UI)
  - android.app.Fragment.class
- La classe *android.preference.PreferenceActivity* gère un choix de préférences
  - L'activité PreferenceActivity intègre dynamiquement des Fragments
  - Deux paramètres extrait de l'Intent entrant sont utilisés pour l'intégration des fragments
    - PreferenceActivity.EXTRA_SHOW_FRAGMENT(':android:show_fragment')
      - Le nom (fname) du fragment
    - PreferenceActivity.EXTRA_SHOW_FRAGMENT_ARGUMENTS(':android:show_fragment_arguments').
      - Le Bundle (attribut utilisé lors de la création associé au fragment.



Attaque par Fragmentation 2/3

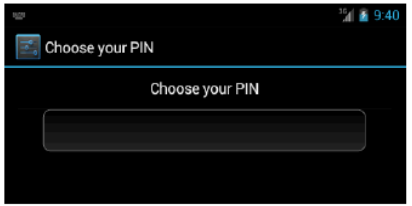
```
577 public static Fragment instantiate(Context
context,String fname,Bundle args)
{
578 try {
579 Class<?> clazz = sClassMap.get(fname) ;
580 if ( clazz == null ) {
582 clazz = context.getClassLoader().loadClass(fname);
583 sClassMap.put(fname, clazz) ;
584 }
585 Fragment f = (Fragment) clazz.newInstance();
586 if (args != null) {
587 args.setClassLoader(f.getClass().getClassLoader());
588 f.mArguments = args ;
589 }
590 return f ;
591  . . .
604 }
```

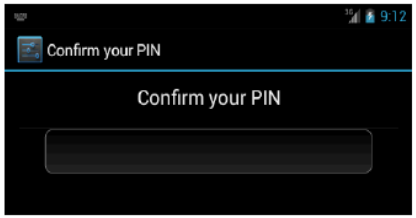


- Toutes les applications qui dérivent de la classe PreferenceActivity sont vulnérables à une attaque par injection de fragments
  - Par l'usage de fragments malicieux (attaque par constructeurs, les éléments statiques du fragments sont instanciés avant la génération d'une exception)
  - Par l'usage de Bundle malicieux

Attaque par Fragmentation 3/3

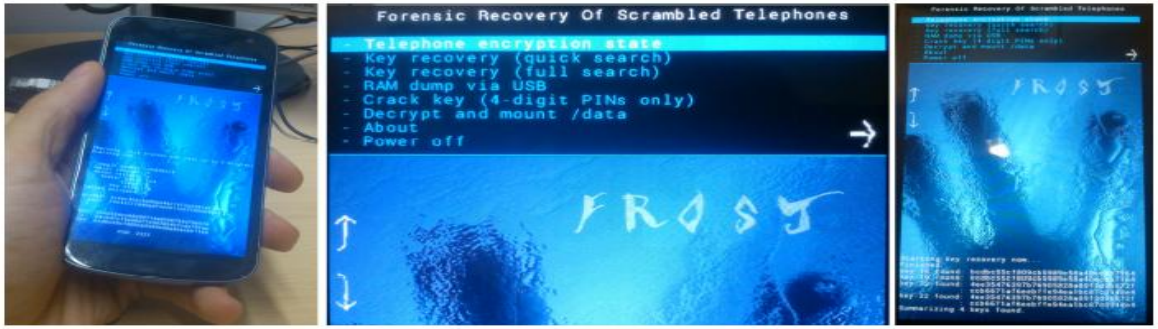
- L'application com.android.settings.Settings dérive de PreferenceActivity. Elle utilise un fragment pour modifier un mot de passe, en deux étapes, confirmation de l'ancien mot de passe puis définition d'une nouvelle valeur.
- Ce fragment (ChooseLockPassword$ChooseLockPasswordFragment) est instancié à partir de la classe ChooseLockPassword associée à un fichier manifest qui comporte les définitions suivantes:
  - <activity
  - android:name="ChooseLockPassword"
  - android:exported="false" ... />
- L'attaque consiste à instancier com.android.settings.Settings à l'aide d'un Intent qui annule la phase de confirmation du mot de passe.






```
Intent i = new Intent();
i.setFlags (Intent.FLAGACTIVITYCLEARTASK) ;
i.setClassName ("com.android.settings", "com.android.settings.Settings") ;
i.putExtra(":android:showfragment" ,
"com.android.settings.ChooseLockPassword$ChooseLockPasswordFragment") ;
i.putExtra("confirmcredentials",false);
startActivity(i) ;
```

Exemple d’attaque 5
Attaques DRAM diverses



Tilo Muller, Michael Spreitzenbarth, and Felix C. Freiling, "Frost, Forensic Recovery of Scrambled Telephones", October 2012
http://www1.cs.fau.de/filepool/projects/frost/frost.pdf

Dimitris Apostolopoulos, Giannis Marinakis, Christoforos Ntantogian, Christos Xenakis, "Discovering authentication credentials in volatile memory of Android mobile devices", 2012
http://cgi.di.uoa.gr/~xenakis/Published/49-I3E-2013/2013-I3E-AMNX.pdf

Stockage de données sensibles dans la DRAM

- A l'aide de noyau UNIX intégrant un logiciel de copie de la DRAM, il a été démontré que la DRAM stocke de nombreuses données sensibles (credentials) telles que login et mot de passe.

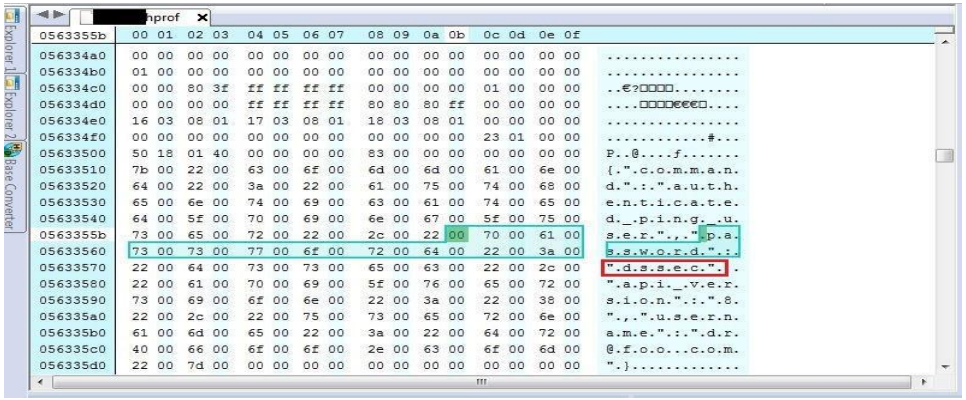


Table 1. Numerical results for each application category

	Usernames	Passwords
m-banking	4/5	5/5
e-shopping/financial	3/3	2/3
Social networks/communications	5/5	5/5
Password managers/encryption	-	17/17
Total	12/13	29/30

Christoforos Ntantogian, Dimitris Apostolopoulos, Giannis Marinakis, Christos Xenakis, "Evaluating the privacy of Android mobile applications under forensic analysis« , 2014

Dimitris Apostolopoulos, Giannis Marinakis, Christoforos Ntantogian, Christos Xenakis, "Discovering authentication credentials in volatile memory of Android mobile devices", 2012

Lecture de la DRAM (FROST)

- Des chercheurs ont observé que le refroidissement d'un mobile vers -15°C permettait de réaliser un "cold boot reset", en retirant brièvement la batterie
  - Si le bootloader est déverrouillé une image recovery dédiée est chargée, elle permet de lire dans la mémoire DRAM la clé de chiffrement de la RAM-FLASH
    - Une attaque force brute est possible pour déduire le mot de passe de démarrage
  - Si le bootloader est verrouillé, diverses informations sensibles (login, mot de passe) sont capturées dans la DRAM (NB: le déverrouillage du bootloader implique l'effacement des données utilisateur stockées dans la RAM-FLASH).

```
adb> insmod frost.ko fullsearch=0 ; dmesg

key-32: 4ee35476397b76905828a89f3d9b872f
      : ccb6671af6eebffe94ea1bc87c0948e4
key-32: 4ee35476397b76905828a89f3d9b872f
      : ccb6671af6eebffe94ea1bc87c0948e4
key-16: bcdbc55cf809cb5989e58a40ecbb7164
key-16: bcdbc55cf809cb5989e58a40ecbb7164

Summarizing 4 keys found.
```

Fig. 2: AES keys recovered by the FROST LKM.

```
adb> ./crackpin

  magic: D0B5B1C4
 encdek: 3c4ac402c6095ed46cf4f1e2281a1f3e
   salt: 19043211840adfde95110c7f99263d6c

>>  KEK: 2165534cc66099714a8226753d70b576
>>   IV: 05cb47cf3a98d77e563bb4cfcde791aa
>>  DEK: bcdbc55cf809cb5989e58a40ecbb7164
>>  PIN: [2323]
```

Fig. 3: Key and PIN recovered by brutefore.

Exemple d'attaque 6
Obtenir la clé de chiffrement de la RAM-FLASH (2012)

https://www.defcon.org/images/defcon-20/dc-20-presentations/Cannon/DEFCON-20-Cannon-Into-The-Droid.pdf

Obtenir la clé de chiffrement RAM

- Les informations de chiffrement (Salt, Encrypted Master Key) sont stockées dans un footer sous diverses formes (fin de partition, fichier partition)
- A partir du tuple(Salt, Encrypted Master Key) il est possible de conduire une attaque force brute pour obtenir le mot de passe
  - SEK_128 || IV_128 = 2000 x PBKDF2(keylen=32, Password, Salt)
  - Encrypted_Master_Key = AES_128_CBC(IV_128, SEK_128)
- Le mot de passe de chiffrement et celui du verrouillage du clavier sont généralement identiques. Un PIN de 4 chiffres rend l'attaque force brute triviale.

```
struct crypt_mnt_ftr {
    __le32 magic;
    __le16 major_version;
    __le16 minor_version;
    __le32 ftr_size;
    __le32 flags;
    __le32 keysize;
    __le32 spare1;
    __le64 fs_size;
    __le32 failed_decrypt_count;
    unsigned char crypto_type_name[MAX_CRYPTO_TYPE_NAME_LEN];
};
```

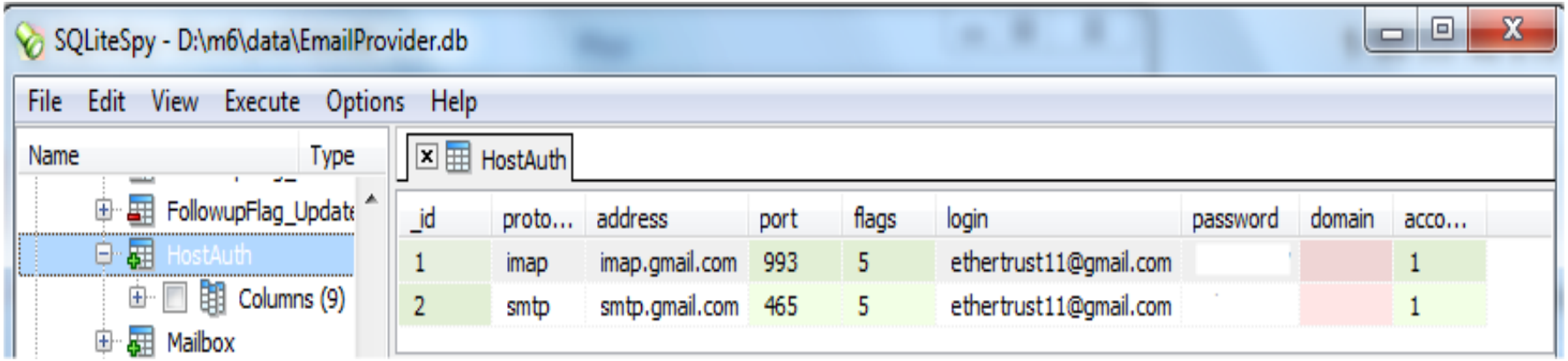
```
~ # mkdir /efs
mkdir /efs
~ # mount -t yaffs2 /dev/block/mtdblock6 /efs
mount -t yaffs2 /dev/block/mtdblock6 /efs
~ # ls /efs
ls /efs
bluetooth       lost+found      nv_data.bin     userdata_footer
imei            nv.log          nv_data.bin.md5
```

https://www.defcon.org/images/defcon-20/dc-20-presentations/Cannon/DEFCON-20-Cannon-Into-The-Droid.pdf

Exemple d’Attaque 7
Divers Sécurité des Applications

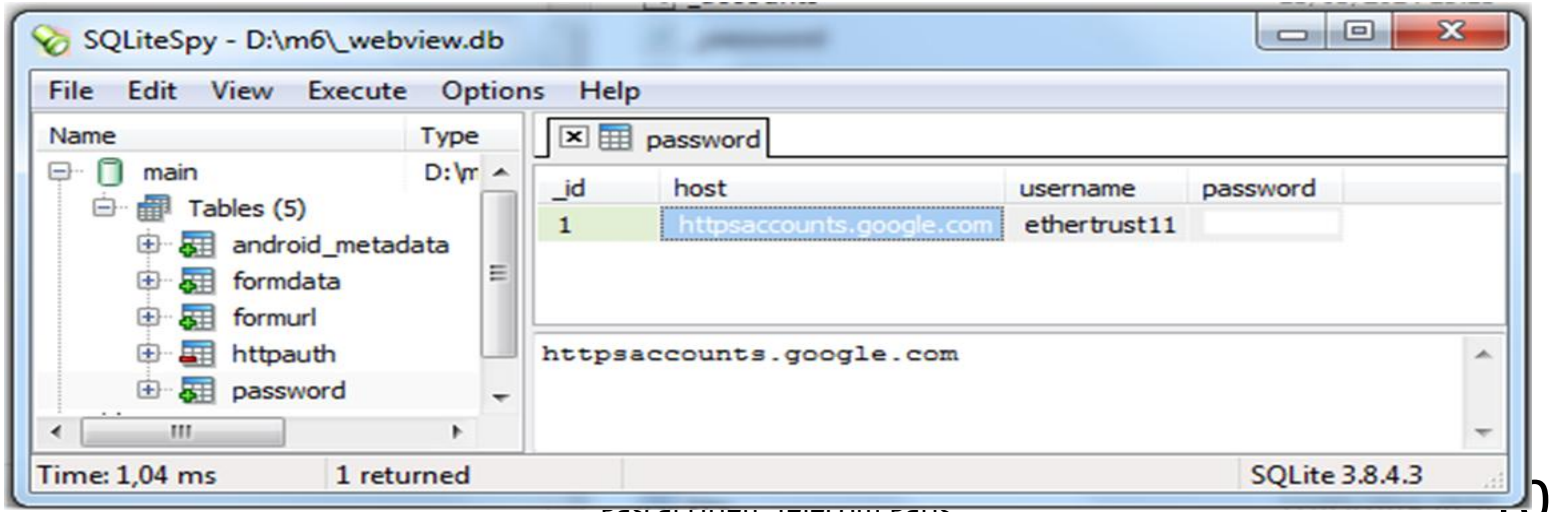
eMail

- /data/data/com.android.email/databases/EmailProvider.db
- Ce fichier mémorise les logins et mots de passe de serveurs de courrier



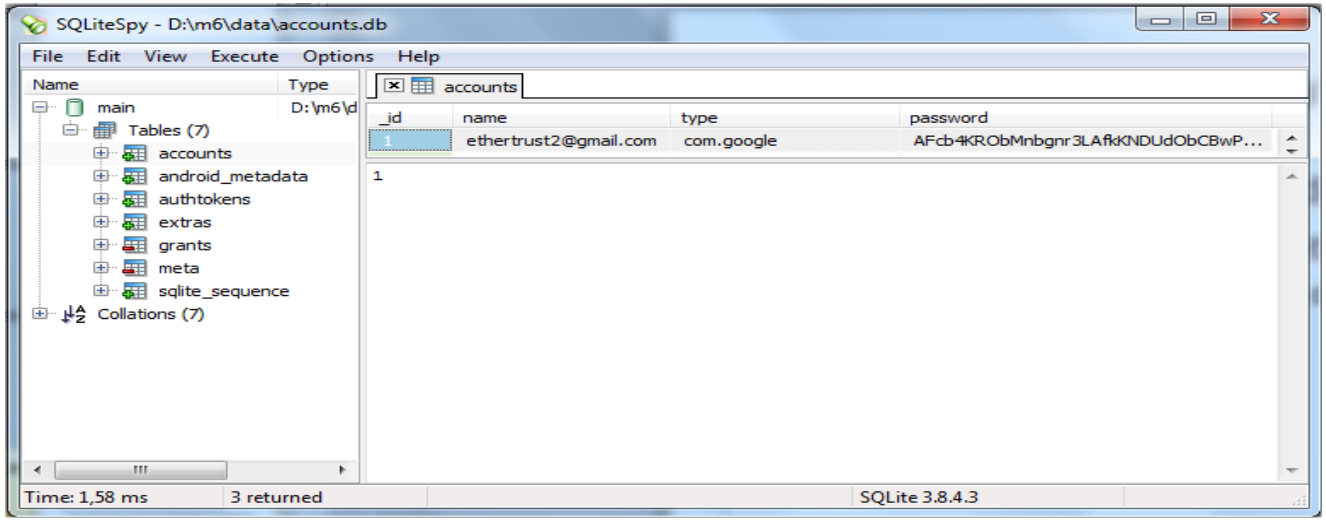
Browser

- /data/data/com.android.browser/databases/webview.db
- Ce fichier stocke les logins et mots de passe des sites web visités



Configuration

- /data/system/accounts.db
- Ce fichier stocke des credentials divers
  - Sur certaines distributions des logins et mots de passes sont en clair

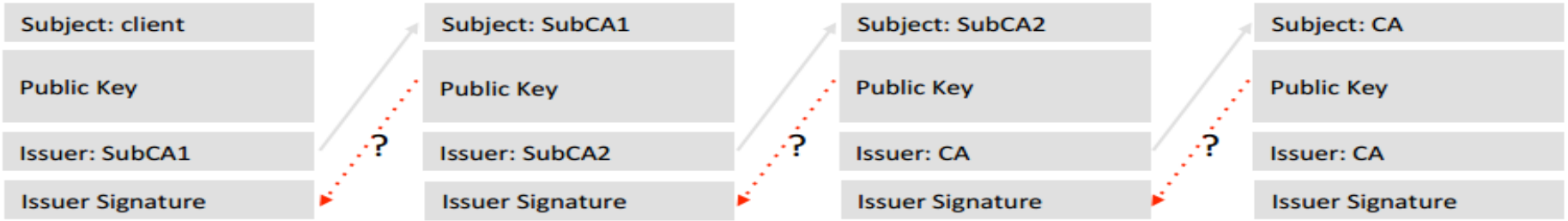


Exemple d’attaque 8
Contourner la signature des applications

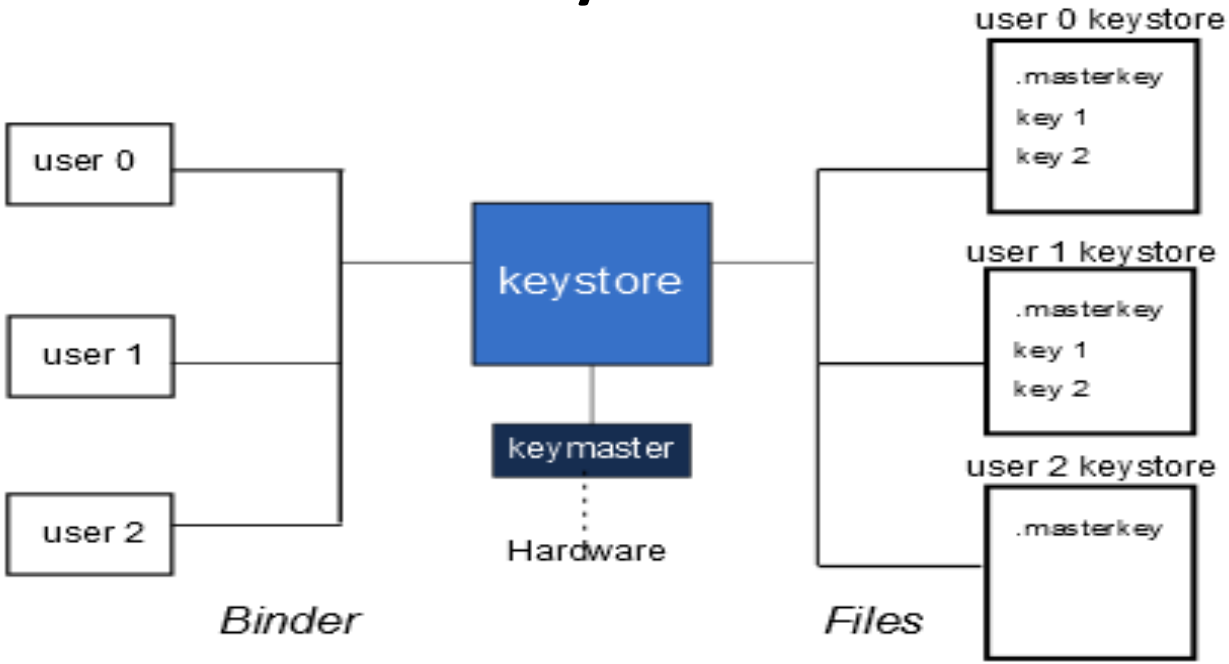
https://www.blackhat.com/docs/us-14/materials/us-14-Forristal-Android-FakeID-Vulnerability-Walkthrough.pdf

FakeID

- En 2014 Jeff Forristal directeur technique de la société Bluebox a publié lors de la conférence Blackhat 2014 une attaque Android nommé "Android Fake ID".
- Le système d'exploitation utilise des chaines de certificats dont la racine est un certificat auto-signé pour la vérification des signatures.
- Cependant la signature des certificats à l'intérieur de la chaine n'est pas vérifiée.



Exemple d'attaque 9
Le keystore



Au sujet du Keystore

- Le système d'exploitation Android supporte un service dénommé *keystore* dont le but est de fournir un environnement *"protégé"* pour le stockage de clés et l'exécution d'algorithmes cryptographiques.
- Le service *keystore* est définit dans le fichier init.rc.
- Le logiciel est stocké dans le répertoire /system/bin/keystore et les données sous /data/misc/keystore.
- Le *keystore* est compatible avec les algorithmes asymétriques RSA et ECC (courbes elliptiques), et l'algorithme symétrique AES.
  - Il intègre des procédures de génération clés (RSA, ECC), des signatures basées sur RSA ou ECC, des algorithmes de chiffrement utilisant AES ou RSA, et des procédures d'intégrité HMAC basées sur SHA1 et SHA2 (SHA256, SHA384, SHA512).
- Le modèle de sécurité du *keystore* repose sur le stockage des clés et l'exécution des procédures cryptographiques selon les mécanismes DAC UNIX.
  - Le service *keystore* réalise l'isolation logique des clés utilisées par différentes applications, identifiées par les UIDs intégrés aux noms de fichier. Il utilise un fournisseur de service cryptographique (le *keymaster*) qui gère optionnellement des ressources hardware, par exemple un TEE.
- Dans le cas le plus courant d'un mono utilisateur Android, les informations associées au service sont des fichiers logés dans le répertoire /data/misc/keystore/user_0 dont le propriétaire est l'utilisateur *keystore*.

Fichiers du Keystore

- Le *keystore* loge deux types de fichiers:
  - le *.masterkey* (unique)
  - et les blobs (*Binary Large Objects*) qui sont chiffrés ou en clair.
- Le nom d'un fichier *blob* est construit à partir de deux ou trois éléments.
- Il débute par un préfix (nombre entier suivi d'un caractère "_ ", par exemple 1000_) qui indique le UID (*User IDentifier*) du processus ayant créé l'élément d'information.
  - Il comporte de manière optionnelle un identifiant (en majuscule) de la procédure cryptographique associée (par exemple USRPKEY ou USRCERT), et enfin un suffixe (identifiant) définit par l'application. Soit en résumé un nom sous la forme:

UID_PROCEDURE_identifier ou *UID_identifier*

Chiffrement des fichiers du Keystore

- Certain fichiers du *keystore* sont encodés selon une structure blob, qui comporte un entête (metada), la valeur initiale IV utilisée pour le chiffrement AES-CBC avec la MasterKey, et la concaténation de l'empreinte MD5 des données et de leur valeur, soit :

key-blob =
metadata || { AES-CBC(MasterKey, IV=iv, MD5(M) || M) }

- Avec M= LenData || Data || Padding, dont la longueur est un multiple de 16 octets

Attaque d'intégrité du Keystore

- La méthode de chiffrement des blobs, visant à garantir l'intégrité des données chiffrées (*Authenticated Encryption* - AE), et généralement nommée *MAC then Encrypt*, n'est pas sure d'un point de vue cryptographique.
- Dans sa thèse* Mohamed Sabt a démontré une attaque permettant de modifier le contenu d'un blob sans connaitre la MasterKey.
- Cette dernière considère un message M1 tel que M1= MD5(M2) || M2, avec M2 = LenData2 || Data2 || Padding2 (la taille de M2 étant un multiple de 16 octets,soit 16.k octets)
- Soit le cryptogramme AES-CBC(MasterKey, IV=iv, MD5(M1) || M1) = E_1 || E_2 || E_3 || E_{k+1} dont la longueur est de 16.(k+1) octets.
- On montre facilement que la suite E_2 || E_3 || E_{k+1} dont la taille est de 16.k octets est la valeur chiffrée de :
  - AES-CBC(MasterKey, IV=E_1, MD5(M2) || M2)
- En pratique cette attaque permet de modifier les valeurs chiffrés des blobs sans connaitre la *MasterKey*, mais nécessite des privilèges *root*.

* Thèse de Mr Mohamed SABT, "Outsmarting Smartphones - Trust Based on Provable Security and Hardware Primitives in Smartphones Architectures", soutenue le 13 décembre 2016

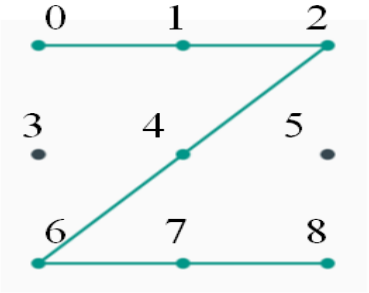
Fichier .keymaster

- Certain fichiers du *keystore*, usuellement désignés *key_blob* sont chiffrés par une clé unique nommée *MasterKey* (un nombre aléatoire de 128 bits), selon le mode cryptographique AES-CBC.
- La valeur chiffrée du *MasterKey* (à l'aide de l'algorithme AES), est stockée dans le fichier .masterkey.
  - La clé de chiffrement (*passwordAesKey*) est calculée à partir d'un mot de passe *password* et d'un nombre aléatoire *msalt*, (stocké en fin du fichier *.masterkey*) grâce à la procédure PBKDF2 (définie par la RFC 2898) selon la relation:

passwordAesKey =
PKCS5_PBKDF2_HMAC_SHA1(password, msalt, 8192, 16),

- qui produit 16 octets (la clé *passwordAesKey*) en 8192 itérations

Pattern PIN



- Un code *pattern* est un mot de passe représenté sous forme graphique. C'est un ensemble de points voisins, placés sur une grille carrée, identifiés par un index (numéro de ligne + 3 x numéro de colonne) compris entre zéro et 8.
- Par exemple le geste Z est associé au tableau d'octets 0,1,2,4,6,7,8. Le nombre maximum de combinaisons est de l'ordre de 400,000 soit une entropie de 19 bits.
- L'empreinte SHA1 de la suite d'octets associée au *pattern* est stockée dans le fichier :

/data/system/gesture.key = SHA1(gesture={0||1||2||4||6||7||8})

- L'extraction du fichier gesture.key (en mode *root*), permet une attaque par force brute d'un *pattern*.

Structure d'un Blob

```
struct __attribute__((packed)) blob {
    uint8_t version;
    uint8_t type;     // 1=Generic, 2=MasterKey 3=KeyPair
    uint8_t reserved;  // 0=Clear, 1=Cipher
    uint8_t info; // Length of msalt
    uint8_t vector[AES_BLOCK_SIZE]; // IV
    uint8_t encrypted[0]; // Marks offset to encrypted data.
    uint8_t digest[MD5_DIGEST_LENGTH];
    uint8_t digested[0]; // Marks offset to digested data.
    int32_t length; // in network byte order when encrypted
    uint8_t value[VALUE_SIZE + AES_BLOCK_SIZE];
};
```

Extrait du code source keystore.cpp

Le fichier comporte 84 octets. Le 4ième octet indique la taille du msalt (16 octets), dont la valeur est située à la fin du fichier. La structure du fichier *.masterkey* est la suivant e

version-1B || type-1B || res-1B || info-1B || IV-16B || CipherText-48B || msalt-16B

Le CipherText avant chiffrement comporte le MD5 de la longueur de la clé (4 octets) concaténé à la *MasterKey* (16 octets) et à 12 octets nuls de padding soit en résumé:

MD5(16B) || Len(4B) || MasterKey(16B) || padding(12B) , c'est à dire 48 octets

```
02 02 01 10                                      // entête (4 octets)
71 B0 E1 35 13 F1 AB F3 5A 6D 56 B8 23 E3 4A 12  // IV (16 octets)
06 B1 97 6F 3E 44 16 40 46 A1 1D 8E 44 38 3A A1  // 48 octets chiffrés
5F E9 7F CA 06 D1 0F 9F 9C E4 39 DC 0D 00 7C B6
DC 3A C9 E1 B5 7F E6 8F 56 64 0A F7 8D 6F FC AD
3C 44 41 9E 23 10 4F 7C 30 72 D5 B3 1B 0D D2 7C
```

Le fichier .master key

Le fichier déchiffré est le suivant :

```
7E FE 82 36 3C D3 CE 3C 46 0E C4 31 20 EE 77 8E  // MD5         (16 octets)
00 00 00 10                                      // longueur=16 (4 octets)
12 3F 06 2C 4E 18 9A34 04 43 72 14 4B 5D 61 A8   // MasterKey   (16 octets)
00 00 00 00 00 00 00 00 00 00 00 00              // Padding     (12 octets)
```

Un fichier du *keystore* nommé *10052_aeskey,* stocke une valeur secrète de 32 octets. Il possède la structure suivante :

version-1B || type-1B || res-1B || info-1B || IV-16B || CipherText-64B

```
02 01 01 00                                     // en tête (4 octets)
CD 63 C9 F9 42 0F 38 58 24 83 32 AB AE 7B B6 34 / IV (16 octets)
C4 94 A6 86 B1 F8 FA 6D 1D 64 21 F1 B7 C1 49 B4 // 64 octets chiffrés
2A 33 84 ED B0 C0 2F 77 54 3E 14 68 08 57 6E 4A
9C 8E F9 C2 9E CF 02 37 BE 2A 7D E4 84 4B CB 6B
BF EF 5E 2B 49 6D 84 48 51 02 1D 67 C3 EF 5E E8
```

Exemple d'un fichier Blob chiffré

Le CipherText avant chiffrement comporte le MD5 de la longueur de la clé (Len=4 octets) concaténé la valeur secrète de octets et à 4 octets nuls de padding, soit en résumé :

MD5(16B) || Len(4B) || AesKey(32B) || padding(12B) , c'est à dire 64 octets

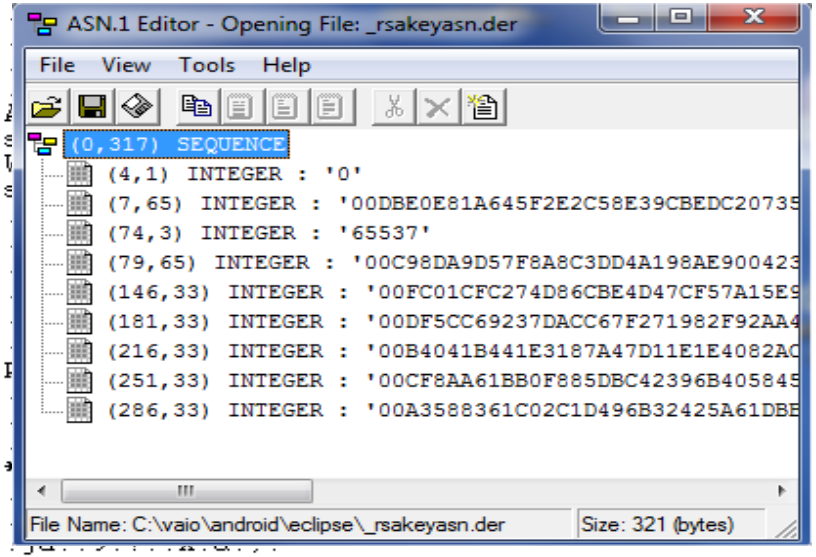
```
FD 11 33 3B 88 7C E9 85 D0 B3 FD D0 67 F8 F3 E8   // MD5
00 00 00 20                                       // longueur=32(4 octets)
30 4C 86 7D D9 E0 E4 D2 5B 79 22 78 8E 52 2C EC   // clé (16 octets)
CD B3 93 9E 68 7A A9 FA C6 25 01 3C E6 A9 7B 61   // clé (16 octets)
00 00 00 00 00 00 00 00 00 00 00 00               // padding (12 octets)
```

Exemple d’un fichier de clés asymétriques

- Un fichier 10052_USRPKEY_rsakey stocke une paire de clé asymétrique
- Il possède la structure suivante:
- version-1B || type-1B || res-1B || info-1B || Opaque-32B || Len-4B || ImportPK#8-16B || KeyJavaPkcs#8
- Les quatre premiers octets codent l'entête d'un blob KEY-PAIR, les 32 suivants sont opaques.
  - La longueur existante du reste du fichier (321 octets) est codée sur 4 octets.
  - On reconnait un entête d'importation de clés du *keymaster* (16 octets) qui débute par les 4 caractères (*Magic Number*) "PK#8" et se termine par la longueur (codée sur 4 octets) du bloc PK#8.
- Les clés publiques et privées sont stockées en clair au format dit *Java pkcs#8*, selon la syntaxe ASN.1.

Structure d'un fichier blob KEY-PAIR

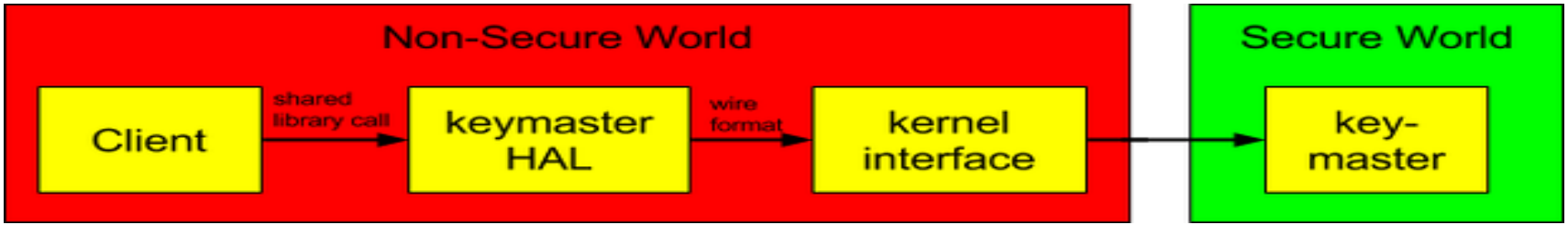
```
00000000  02 03 00 00 A0 03 B1 B7  D8 03 B1 B7 00 04 B1 B7
00000010  02 01 01 00 BF EF 5E 2B  49 6D 84 48 51 02 1D 67
00000020  C3 EF 5E E8 00 00 01 51  50 4B 23 38 00 00 00 06
00000030  00 00 00 00 00 00 01 41  30 82 01 3D 02 01 00 02
00000040  41 00 DB E0 E8 1A 64 5F  2E 2C 58 E3 9C BE DC 20
00000050  73 5F C2 B1 DD FC C2 0E  C3 89 B3 F0 0A 42 BB D4
00000060  57 81 8F 68 C0 E6 DC 14  70 BB 5B 8E 7C BA 57 59
00000070  73 50 40 B5 A9 44 F6 75  3B 24 E2 15 81 0A 97 1C
00000080  B6 79 02 03 01 00 01 02  41 00 C9 8D A9 D5 7F 8A
00000090  8C 3D D4 A1 98 AE 90 04  23 1E 75 CF 2B E9 25 4D
000000a0  F9 4A 82 2A 18 8D C1 B5  CD 1B 5F C3 2D F0 E1 03
000000b0  A1 BA CA 4F 92 66 55 09  EF D3 2F C5 E4 36 0D D0
000000c0  D9 C2 D2 40 64 DE E1 C7  9C 81 02 21 00 FC 01 CF
000000d0  C2 74 D8 6C BE 4D 47 CF  57 A1 5E 95 4E BB E7 82
000000e0  70 20 B4 FA 02 A1 3D 4A  26 6F CA 72 31 02 21 00
000000f0  DF 5C C6 92 37 DA CC 67  F2 71 98 2F 92 AA 42 A1
00000100  F7 EC CF E5 61 B2 B4 86  15 6A 96 2F B7 98 6E C9
00000110  02 21 00 B4 04 1B 44 1E  31 87 A4 7D 11 E1 E4 08
00000120  2A C7 5F 41 CD 03 99 53  53 D4 0F D2 DC 34 3E B6
00000130  B6 97 D1 02 21 00 CF 8A  A6 1B B0 F8 85 DB C4 23
00000140  96 B4 05 84 5C 7B A1 21  92 9E E1 A3 AA 27 B8 78
00000150  9C 6A 64 1E A8 39 02 21  00 A3 58 83 61 C0 2C 1D
00000160  49 6B 32 42 5A 61 DB B9  6F EF AD D4 E1 F4 AA ED
00000170  ED 69 5E 19 0D E1 47 7A  C1 00 00 00 00 00 00 00
00000180  00 00 00 00
```

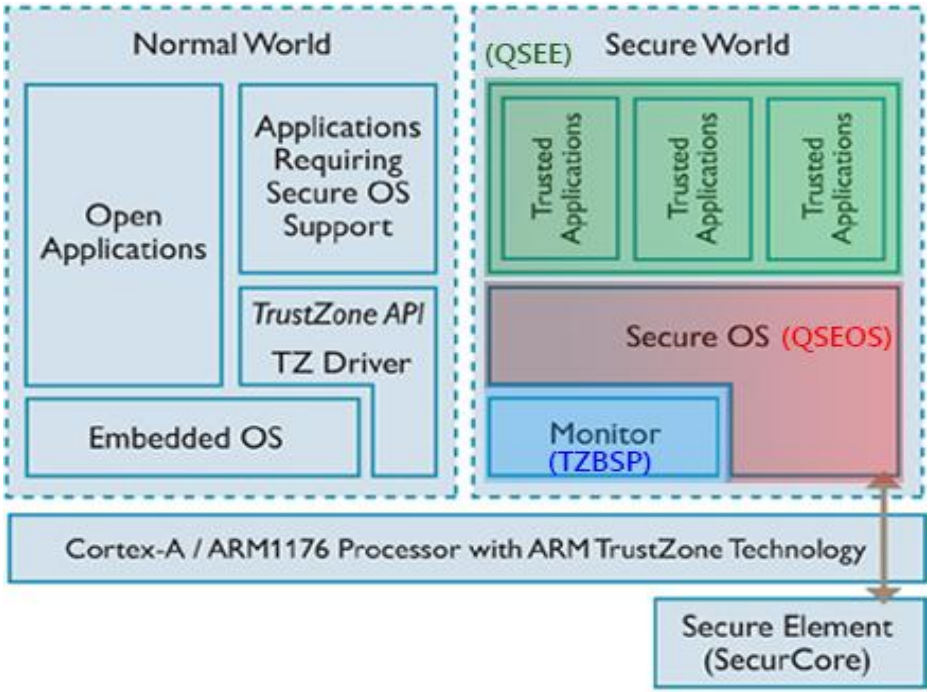


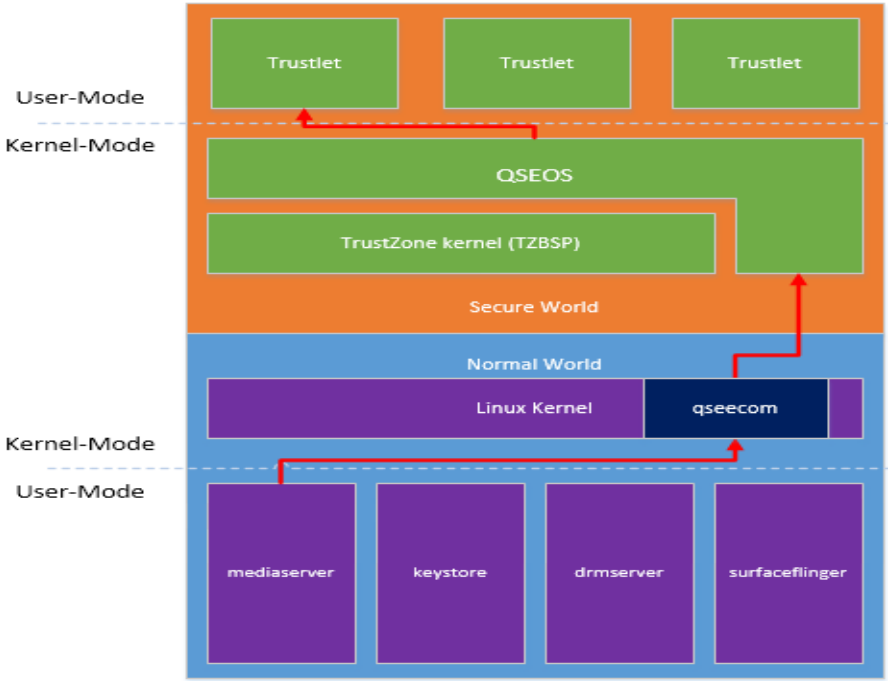
Hardware-Backed Keystore

- Le composant *keystore* utilise un module nommée *keymaster* qui délivre des services cryptographiques (génération de clés...).
- Par défaut ces derniers sont réalisés de manière logicielles et utilisent la libraire openssl (dans une partie des codes sources sont décrit par les fichiers *softkeymaster/module.cpp*, *softkeymaster/keymaster_openssl.cpp*).
- Un module *keymaster* dit *Hardware Abstraction Layer* (HAL) fournit des ressources cryptographiques de nature hardware, typiquement délivrées par un TEE (*Trusted Execution Environment*).

keymaster_qcom.cpp




Normal World
Open Applications
Applications Requiring Secure OS Support
TrustZone API
TZ Driver
Embedded OS
(QSEE) Secure World
Trusted Applications
Trusted Applications
Trusted Applications
Secure OS (QSEOS)
Monitor (TZBSP)
Cortex-A / ARM1176 Processor with ARM TrustZone Technology
Secure Element (SecurCore)


User-Mode
Kernel-Mode
Trustlet
Trustlet
Trustlet
QSEOS
TrustZone kernel (TZBSP)
Secure World
Normal World
Linux Kernel
qseecom
Kernel-Mode
User-Mode
mediaserver
keystore
drmserver
surfaceflinger

QSEE Qualcomm

- Le QSEE (Qualcomm's Secure Execution Environment) est un exemple de TEE intégré à des mobiles.
- Il dispose de deux applications (trustlets) principales :
  - Un trustlet implémentant un keystore hardware, c'est à dire fournissant des ressources cryptographiques
  - Un trustlet réalisant une application de DRM(*Digital Right Management*) proposée par la solution *Widevine*, une société de Google.
- La référence* analyse le chargement des trustlets stockés sur le mobile et exécutés par le TEE.
  - Elle constate que les fichiers *trustlets*, conformes au classique format ELF(*Executable and Linkable Format*) ne sont pas chiffrés, mais authentifiés par un chaine de certificats.
  - Cette opportunité permet l'analyse des codes sources et l'extraction des failles.

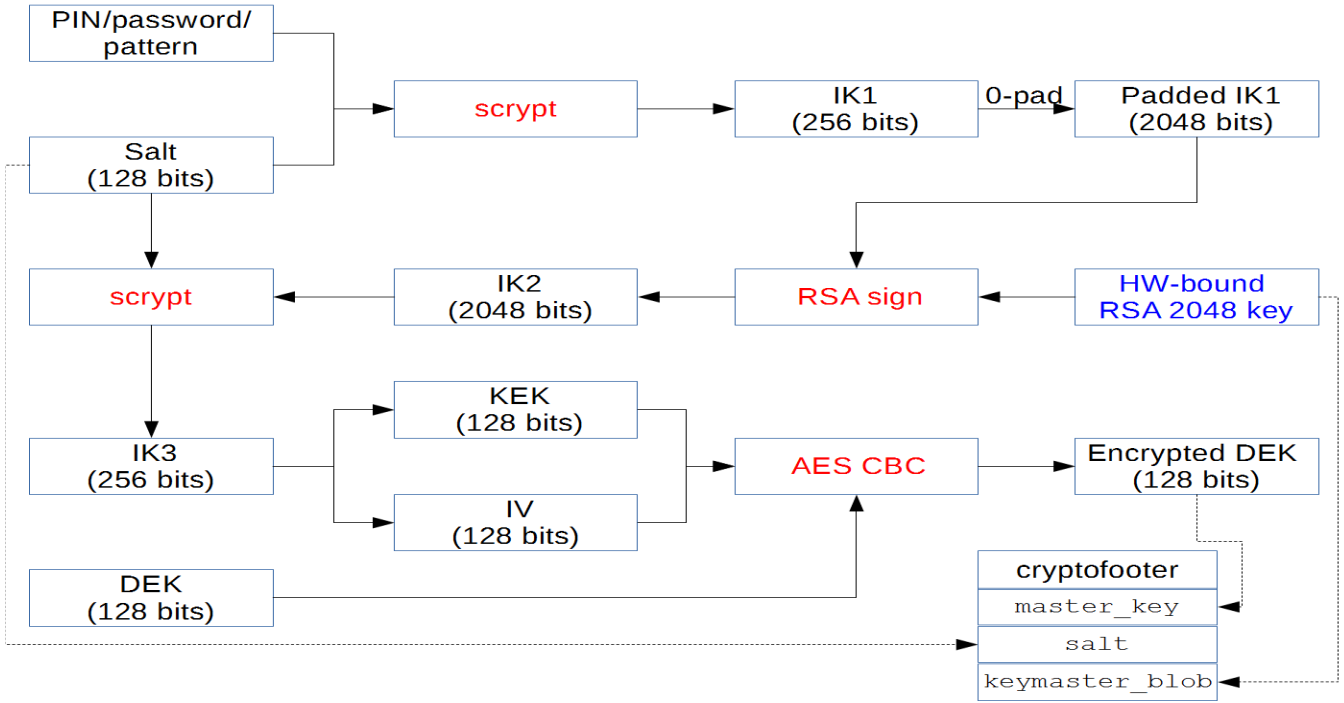
*http://bits-please.blogspot.fr/2016/04/exploring-qualcomms-secure-execution.html

Key Blob

- Le trustlet *keystore* réalise quatre classes de service cryptographique: la génération de clés asymétriques, la signature, la vérification de signature, et l'importation de clés.
- Un composant *qseecom* gère l'échange de messages et les transferts mémoires entre le monde normal et le monde sécurisé.
- Les clés asymétriques sont échangées à l'aide de *blob* stockés dans le *keystore*.
- Dans un key blob, les informations sont authentifiées par une empreinte HMAC, la valeur de l'exposant privé est chiffrée; les clés utilisées pour la protection du blob sont calculées par le TEE.

```
struct  qcom_km_key_blob {
  uint32_t magic_num;
  uint32_t version_num;
  uint8_t  modulus[KM_KEY_SIZE_MAX];
  uint32_t modulus_size;
  uint8_t  public_exponent[KM_KEY_SIZE_MAX];
  uint32_t public_exponent_size;
  uint8_t  iv[KM_IV_LENGTH];
  uint8_t
encrypted_private_exponent[KM_KEY_SIZE_MAX];
  uint32_t encrypted_private_exponent_size;
  uint8_t  hmac[KM_HMAC_LENGTH];
};
```

Chiffrement des fichiers Android

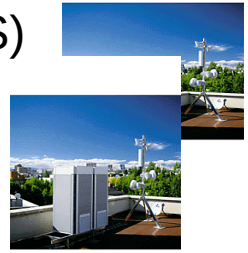


https://nelenkov.blogspot.fr/2014/10/revisiting-android-disk-encryption.html

Sécurité du GSM

Architecture

Base Station (BTS)

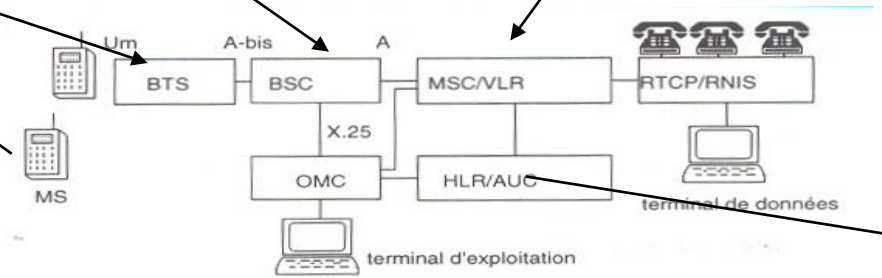


Base Station Controler (BSC)

MSC+VLR

RTC

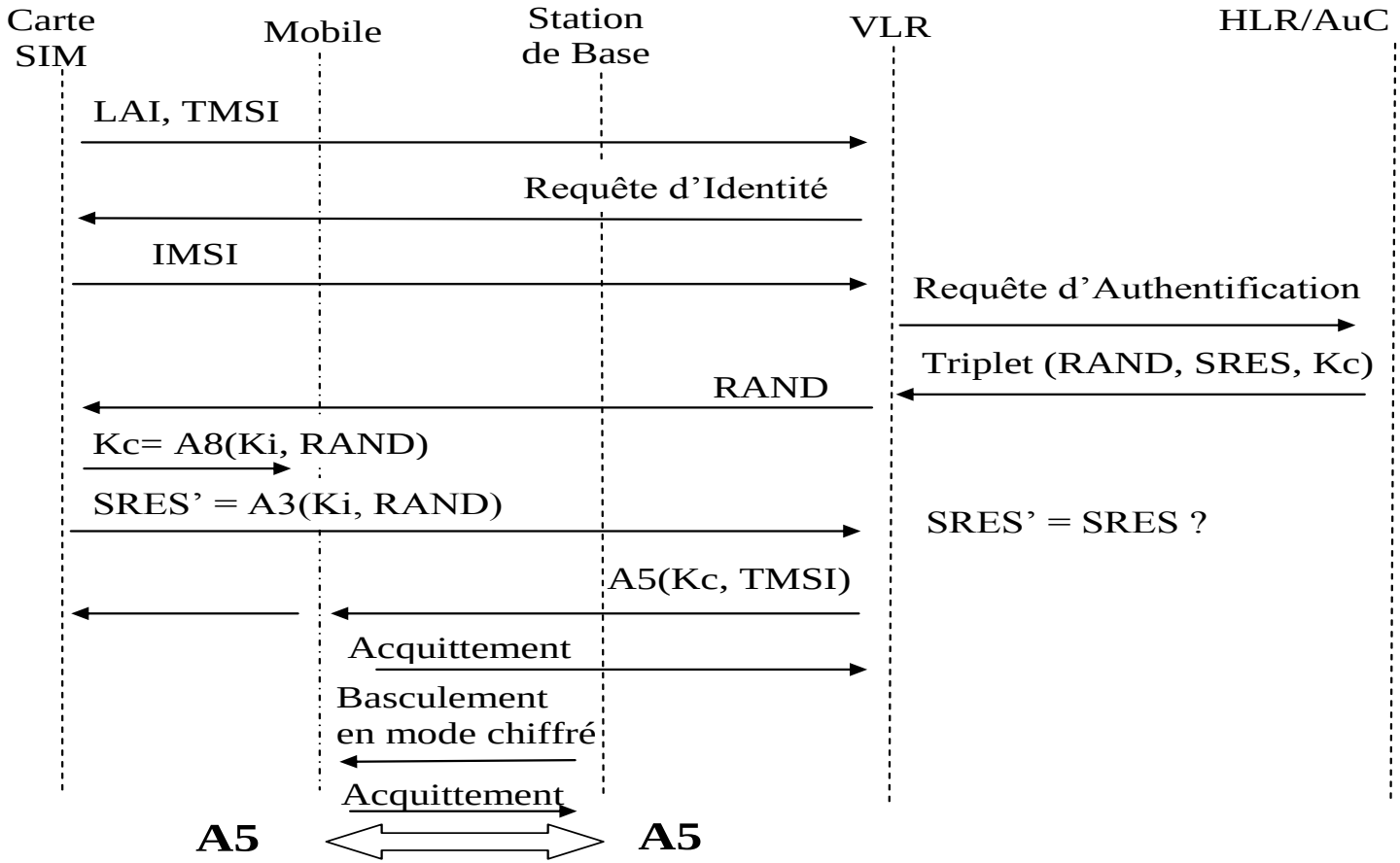
HLR+AuC



terminaux d'abonnés	sous-système radio	sous-système réseau	réseau téléphonique commuté public

Principes

- Mécanisme de type provisionning
  - Vecteurs d'authentification (triplet du GSM)
  - RAND (64 bits), SRES (32 bits), Kc (64 bits, dont 10 à zéro)
- Algorithmes
  - Clé Ki de 128 bits
  - A3Ki(RAND), calcul de la signature SRES
  - A8Ki(RAND), calcul de Kc
  - A3/A8 est en fait un algorithme unique, le COMP-128
    - COMP128-1, craqué en 1998, 2^{19} vecteurs
    - COMP128-2, version améliorée de COMP128-2
    - COMP 128-3, basé sur AES
  - A5(Kc), chiffrement de paquets données (voix)
    - Mode bloc de 112 bits
    - A5/1, version forte, craquée en 99
    - A5/2, version faible, craquée en 99
    - A5/3, nouvelle version


Carte SIM
Mobile
Station de Base
VLR
HLR/AuC
LAI, TMSI
Requête d'Identité
IMSI
Requête d'Authentification
Triplet (RAND, SRES, Kc)
RAND
Kc= A8(Ki, RAND)
SRES' = A3(Ki, RAND)
SRES' = SRES ?
A5(Kc, TMSI)
Acquittement
Basculement en mode chiffré
Acquittement
A5
A5

Contrôle d'accès du GSM 1/2

- La norme GSM 03.20 décrit de manière concise les principes de mise en œuvre des éléments de sécurité dans un environnent GSM .Une cellule ou un ensemble de cellules sont identifiées par une étiquette LAI (Location Area Identity).
- Un abonné dispose d'un couple de valeurs LAI-TMSI stocké dans son module SIM . Le mobile transmet au VLR les indications LAI et TMSI ; ce dernier tente de retrouver l'IMSI identifiant de manière univoque un abonné. En cas d'échec de cette opération, il délivre au mobile une requête d'identification qui récupère en clair l'IMSI mémorisé dans la carte SIM.
- A ce stade le VLR connaît l'IMSI de l'abonné. Il transmet au HLR (qui est généralement regroupé avec le bloc AuC) une demande d'authentification. Si le compte utilisateur est valide dans la base de donnée de l'opérateur, l'AuC produit une suite de valeurs connue sous l'appellation « triplet du GSM », trois nombres notés RAND, SRES et Kc.

Contrôle d'accès GSM 2/2

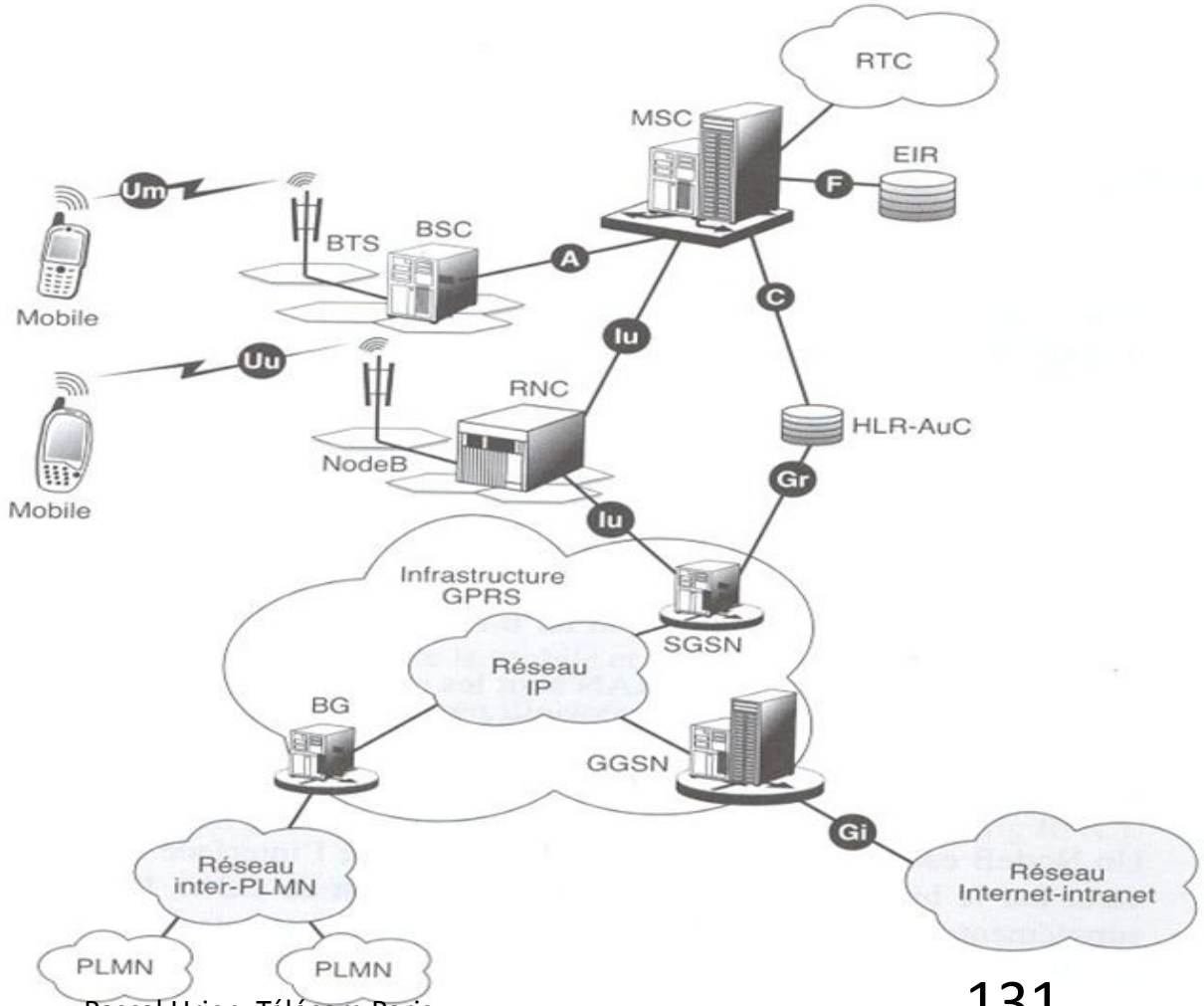
- RAND est un nombre aléatoire de 16 octets, SRES (Signed RESponse, mot à mot réponse signée) est calculé par l'algorithme A3 associé à la clé Ki, soit SRES= A3(Ki, RAND).
- Kc est la clé utilisée pour le chiffrement des communications, elle est déduite de l'algorithme A8 associée à la clé Ki, soit Kc = A8(Ki, RAND).
- Ce mécanisme est une caractéristique originale du GSM ; en effet le VLR obtient de la part du HLR un ou plusieurs triplets d'authentification. Le VLR transmet au mobile un défi RAND. La carte SIM exécute alors la fonction A3 dont le résultat (SRES'=A3(Ki, RAND)) est renvoyé au HLR qui vérifie alors l'égalité entre SRES et SRES'.
- Le VLR choisit un nouveau TMSI, réalise son chiffrement avec l'algorithme A5 muni de la clé Kc, puis transmet ce paramètre au mobile qui effectue l'opération de déchiffrement.
- C'est la station de base dont dépend le mobile, qui décide du basculement en mode chiffré (basé sur l'algorithme A5 muni de la clé Kc) des communications.

Eléments d'identifications

- Mobile Equipment (ME)
  - IMEI, International Mobile Equipment Identity
- Subscriber Identity Module (SIM)
  - Ki – Subscriber Authentication Key
    - RUN_GSM_ALGO
  - IMSI – International Mobile Subscriber Identity
    - DF_GSM/EF_IMSI
  - TMSI – Temporary Mobile Subscriber Identity
  - PIN – Personal Identity Number protecting a SIM
  - LAI – location area identity
    - DF_GSM/EF_LOCI

Sécurité de l’UMTS

Architecture

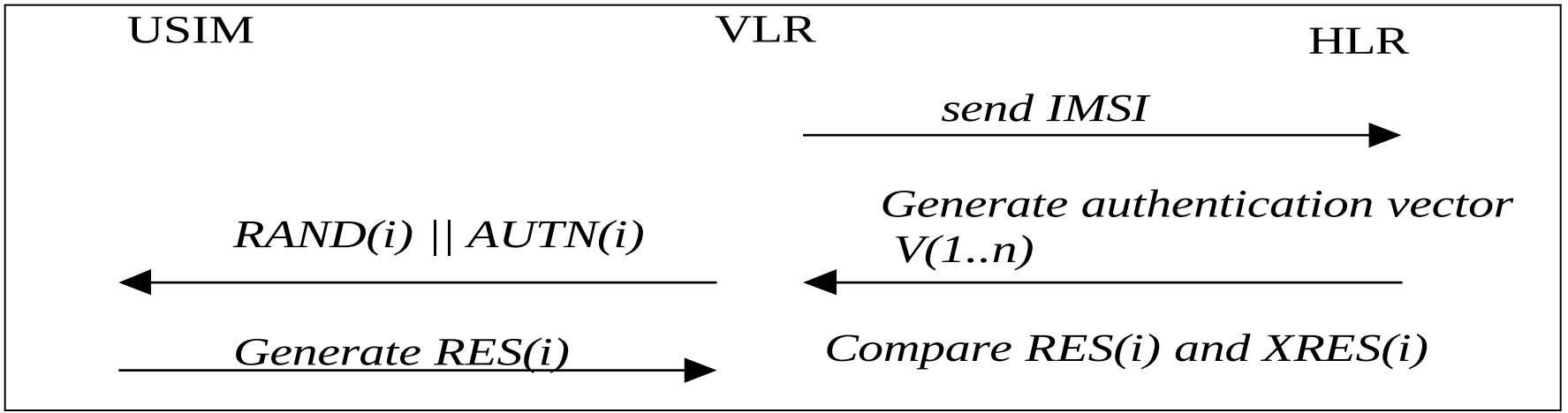


X	Interface X
AuC	(AUthentication Center)
BG	(Border Gateway)
BSC	(Base Station Controller)
BTS	(Base Transceiver Station)
EIR	(Equipment Identity Register)
GGSN	(Gateway GPRS Support Node)

GMSC	(Gateway Mobile-services Switching
HLR	(Home Location Register)
MSC	(Mobile-services Switching Center)
PLMN	(Public Land Mobile Network)
RNC	(Radio Network Controller)
RTC	(réseau téléphonique commuté)

Authentification UMTS - Principes

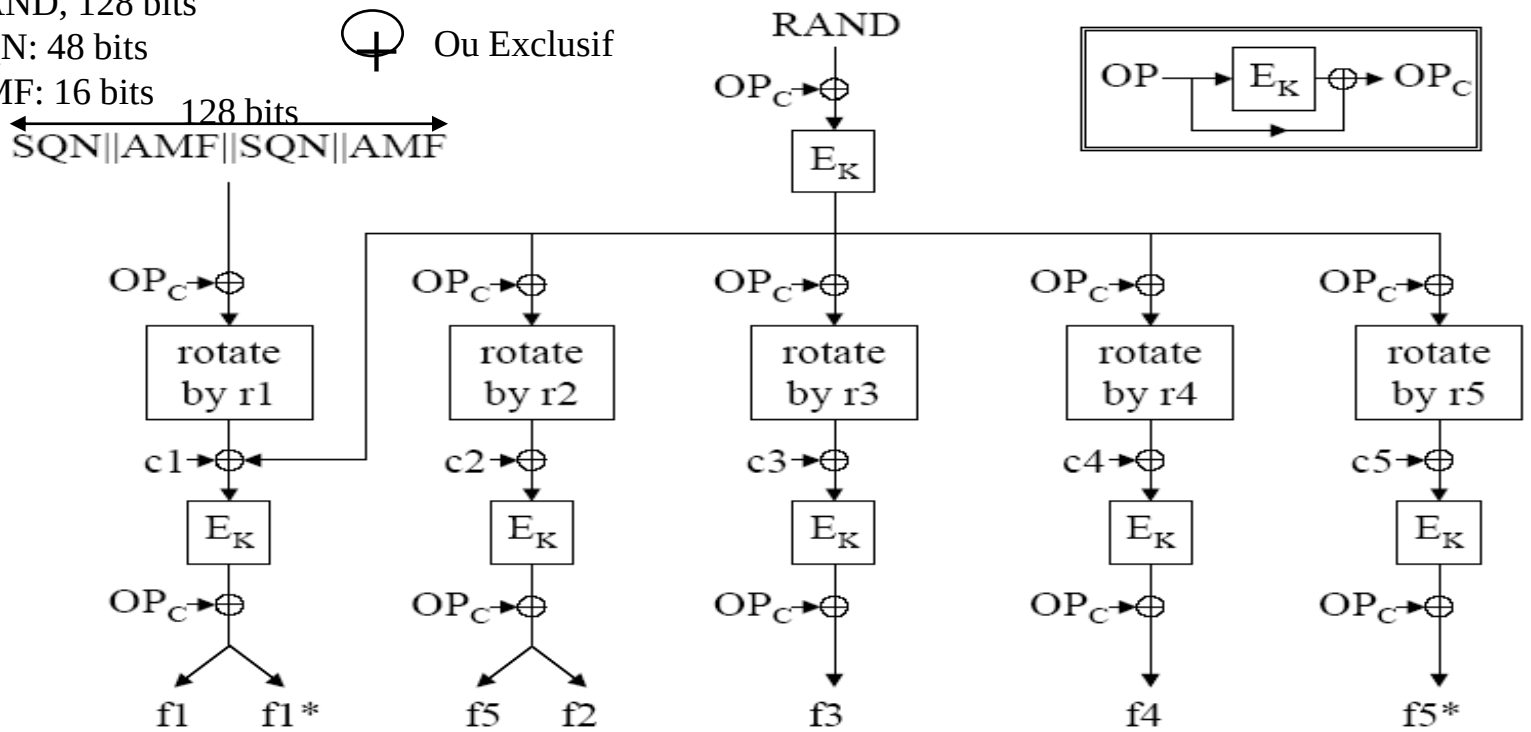
- Mutuelle Authentification
  - Authentication and Key Agreement (AKA)
  - Cipher key (CK) and Integrity key (IK)



Authentification UMTS - MILENAGE

RAND, 128 bits
SQN: 48 bits
AMF: 16 bits

Ou Exclusif

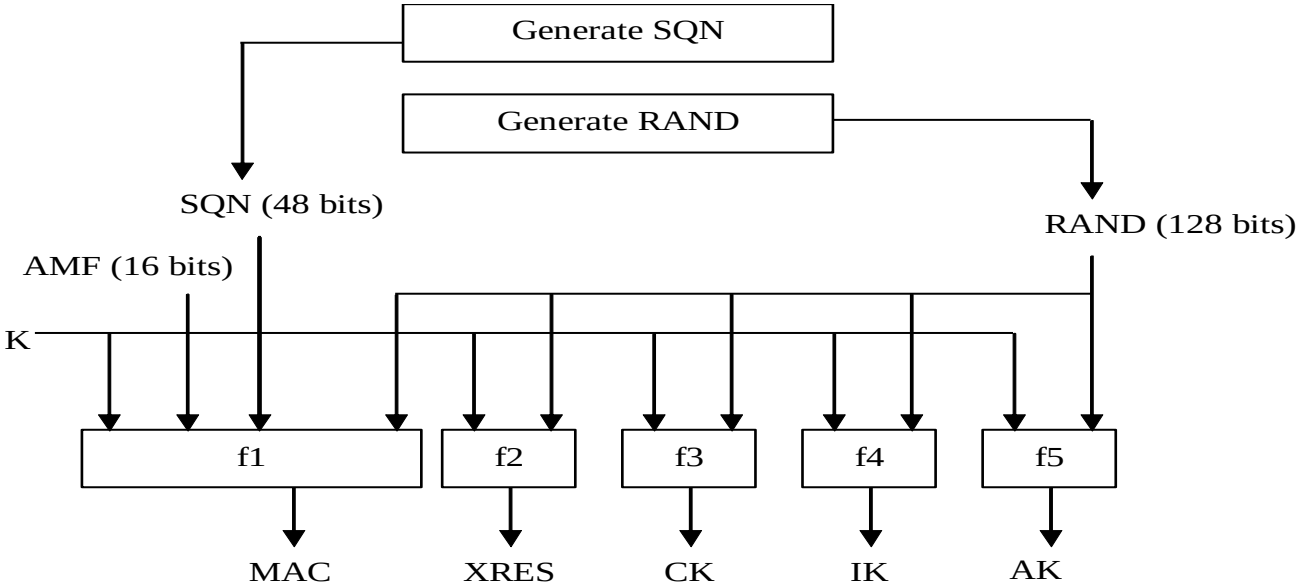


ri= 64,0,32,64,96 (rotation gauche)

ci= 0,1,2,4,8

OP: clé OPérateur (128 bits) Ek: AES + clé 128bits

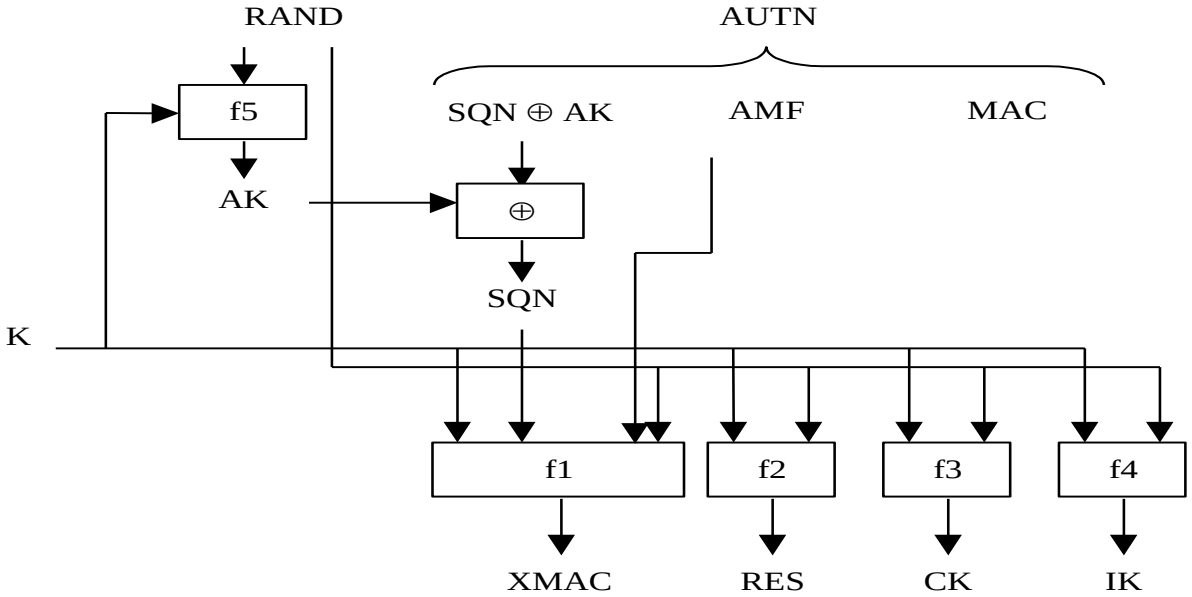
Authentication Vector - AV



AUTN := SQN ⊕ AK || AMF || MAC

AV := RAND || XRES || CK || IK || AUTN

Authentification UMTS - USIM



Verify MAC = XMAC

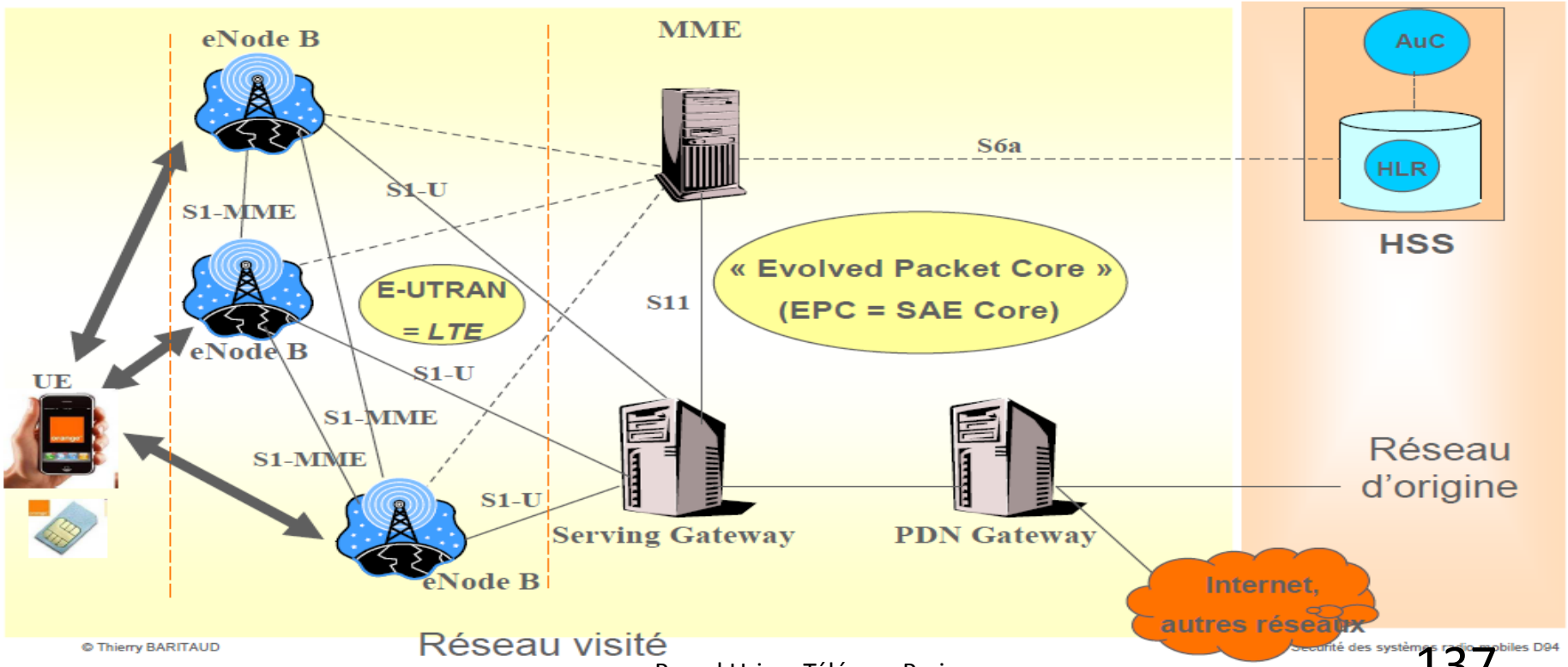
Verify that SQN is in the correct range

LTE

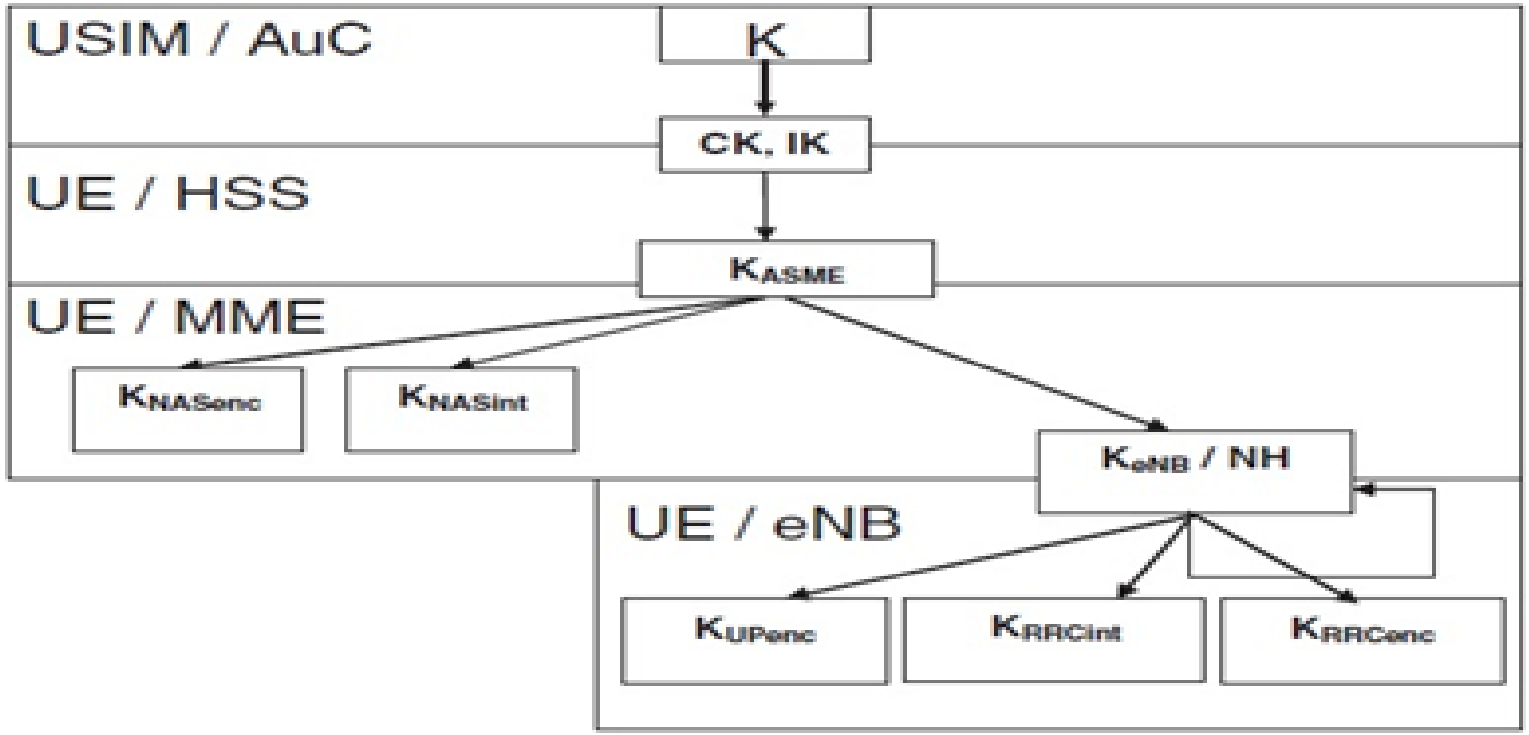
Architecture du LTE

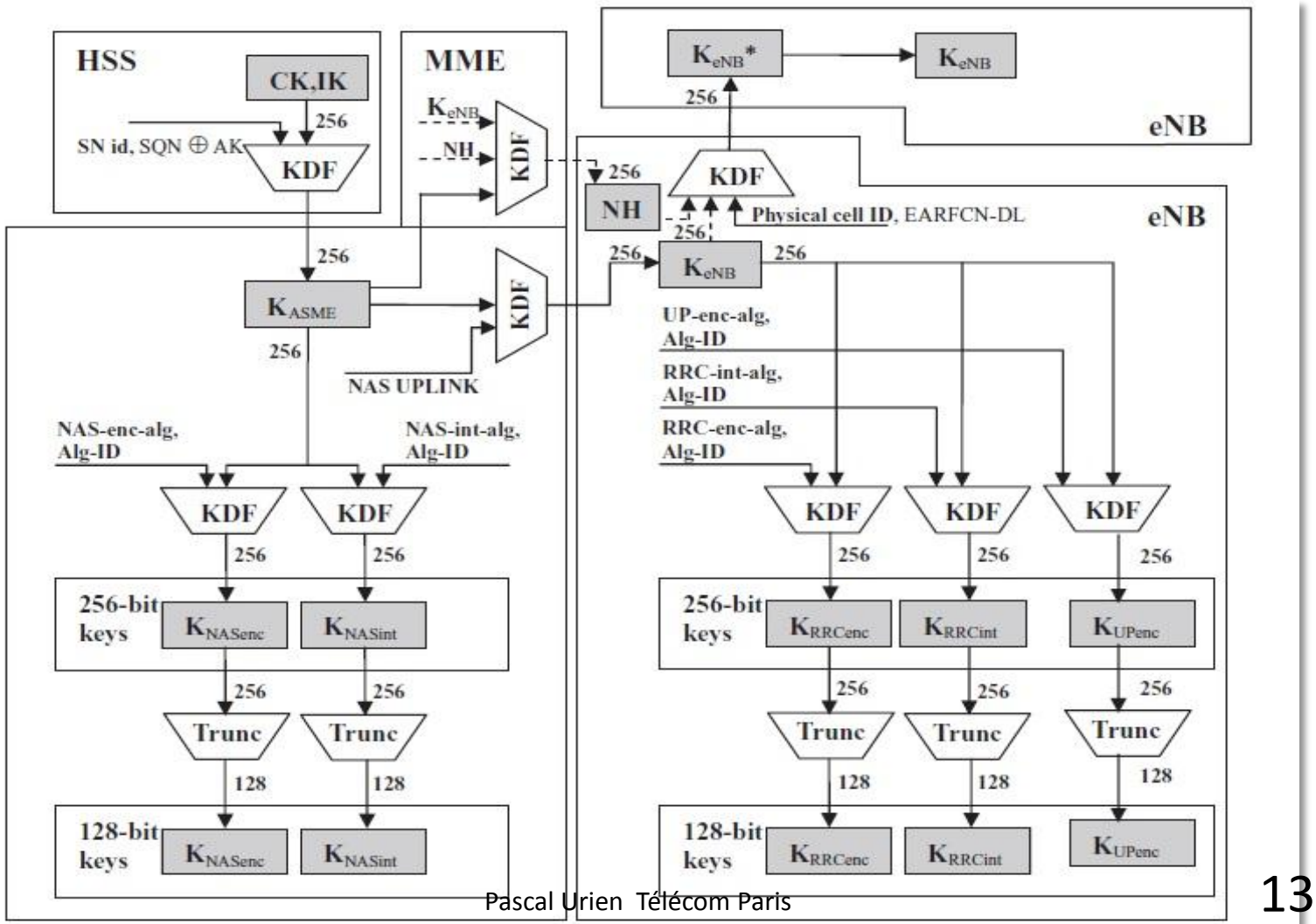
HSS : Home Subscriber Server MME : Mobile Management Entity UE: User Equipment
PDNG: Packet Data Network Gateway
S1-MME (ou X2): signaling between eNode B and MME
S1-U: (traffic) user plane between eNB and serving gateways

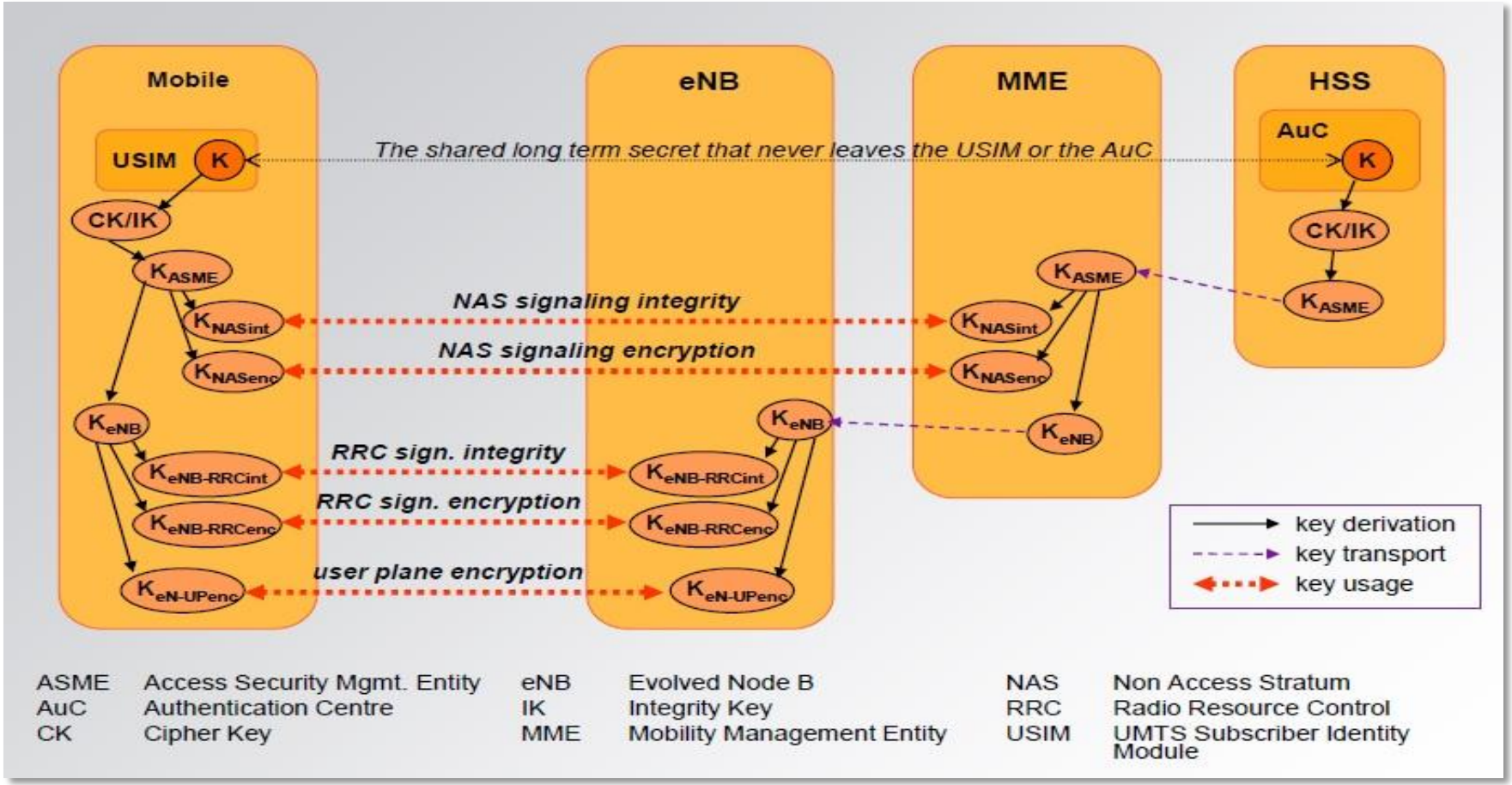
3GPP TS 36-300
3GPP TS 23.401



LTE: Hiérarchie des clés




HSS
CK,IK
256
SN id, SQN ⊕ AK
KDF
MME
K_{eNB}
NH
KDF
256
K_{ASME}
256
NAS UPLINK
KDF
NAS-enc-alg, Alg-ID
NAS-int-alg, Alg-ID
KDF
KDF
256
256
256-bit keys
K_{NASenc}
K_{NASint}
256
256
Trunc
Trunc
128
128
128-bit keys
K_{NASenc}
K_{NASint}
K_{eNB}*
K_{eNB}
256
eNB
KDF
NH
256
Physical cell ID, EARFCN-DL
eNB
256
K_{eNB}
256
UP-enc-alg, Alg-ID
RRC-int-alg, Alg-ID
RRC-enc-alg, Alg-ID
KDF
KDF
KDF
256
256
256
256-bit keys
K_{RRCenc}
K_{RRCint}
K_{UPenc}
256
256
256
Trunc
Trunc
Trunc
128
128
128
128-bit keys
K_{RRCenc}
K_{RRCint}
K_{UPenc}

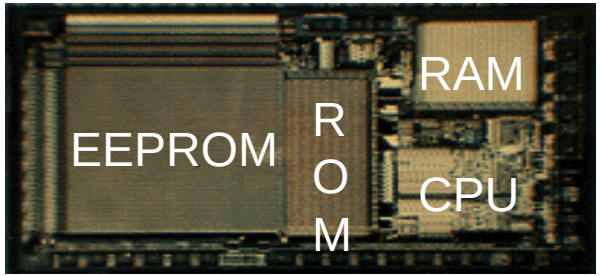


https://www.troopers.de/wp-content/uploads/2011/10/TR12_TelcoSecDay_Schneider_LTE.pdf

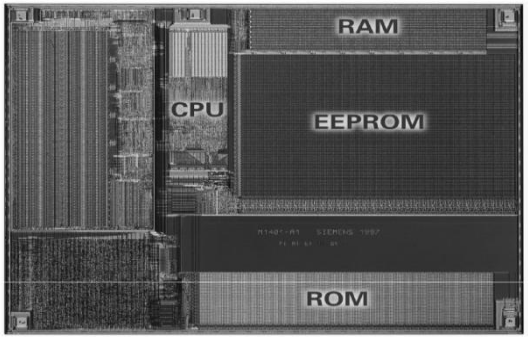
Cartes à Puce & Secure Elements

Quelques dates

- 1974, Brevet de R.Moreno
- 1977, Brevet de M.Ugon
- 1987, Première norme ISO 7816
- 1988, Spécification de la carte SIM
- 1995, Attaque DPA Paul Kocher
- 1996, Première norme EMV
- 1997, Brevet Java Card, US 6,308,317
- JCOP (IBM JC/OP), 1998
- 1999, Global Platform (GP)
- 2002, dotnet smart card, Hiveminded



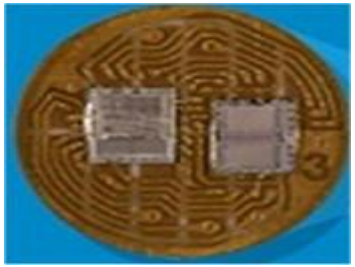
1988, the 21 (BO') chip



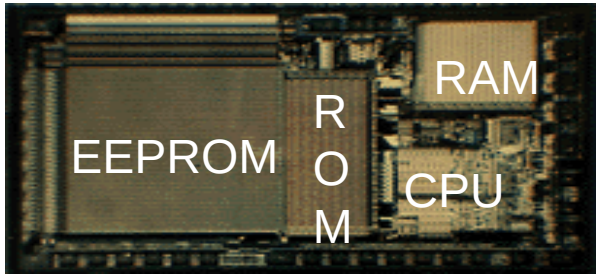
Siemens (SIM) chip, 1997

Le SPOM

- Mars 1979, CII-Honeywell Bull et Motorola,
  - Deux puces: une mémoire 2716 EPROM et un microprocesseur 8 bits 3870.
- Octobre 1981 puce monolithique CII-Honeywell Bull et Motorola
  - SPOM, Self Programmable One chip Microcomputer



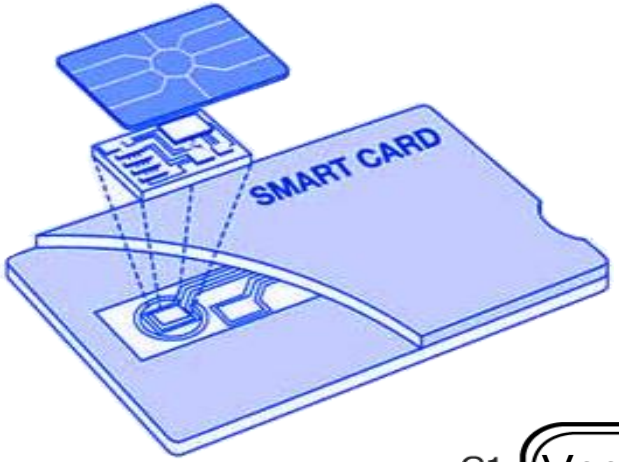
1979, carte hybride
à deux puces

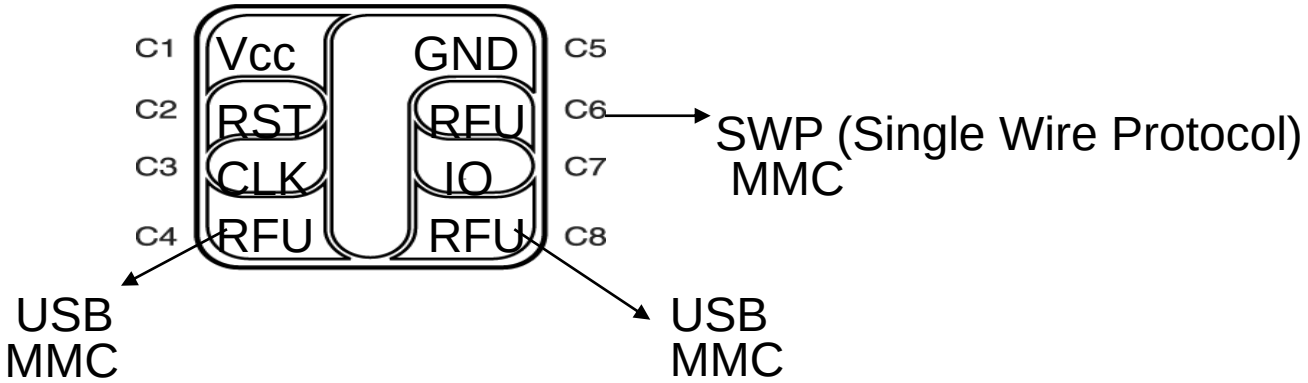


1988, le chip 21
avec une mémoire EEPROM

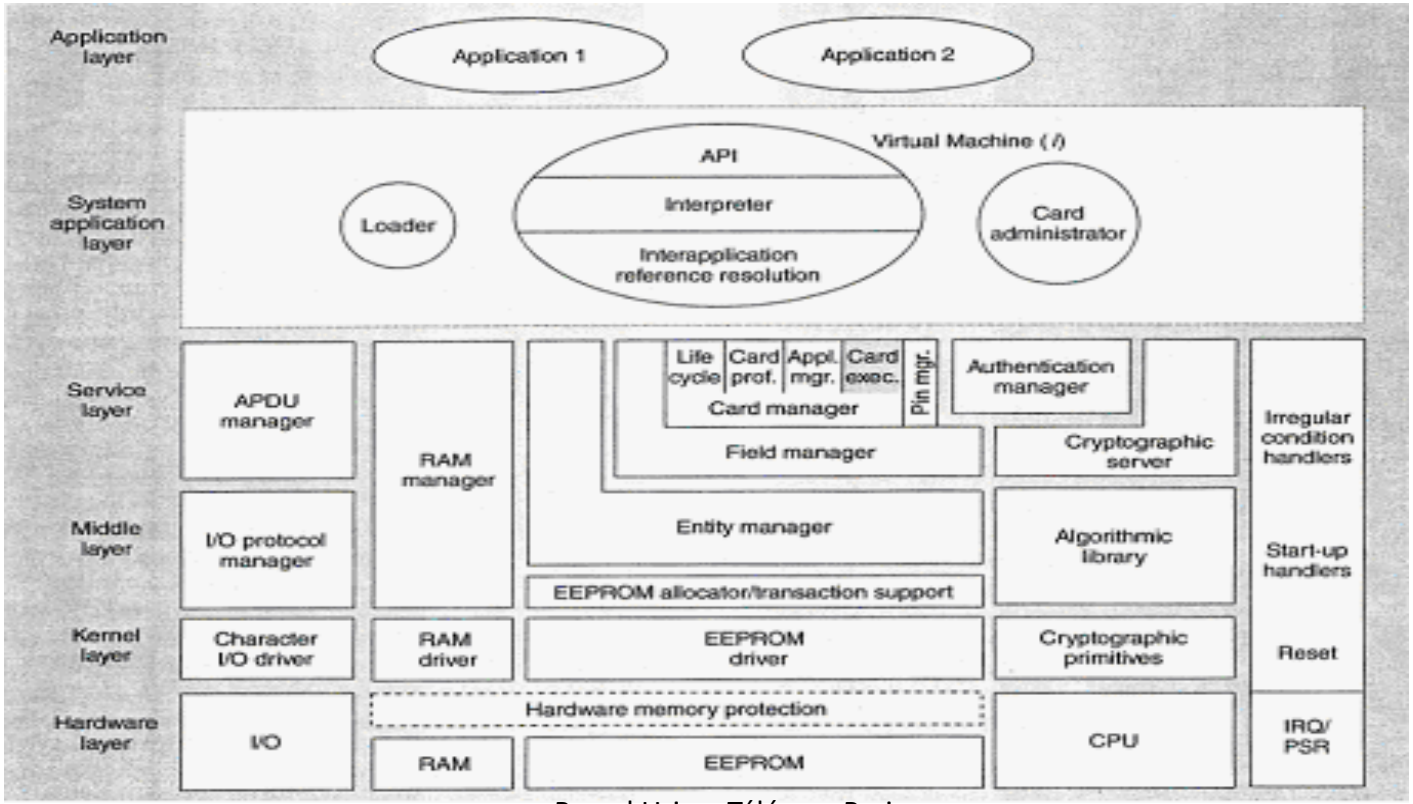
1981, chip SPOM1 en NMOS 3.5 µm
(42000 transistors sur 19.5mm²).

Aperçu technologique

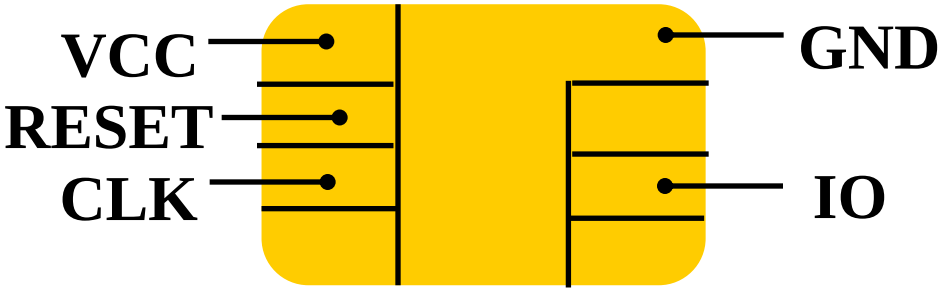


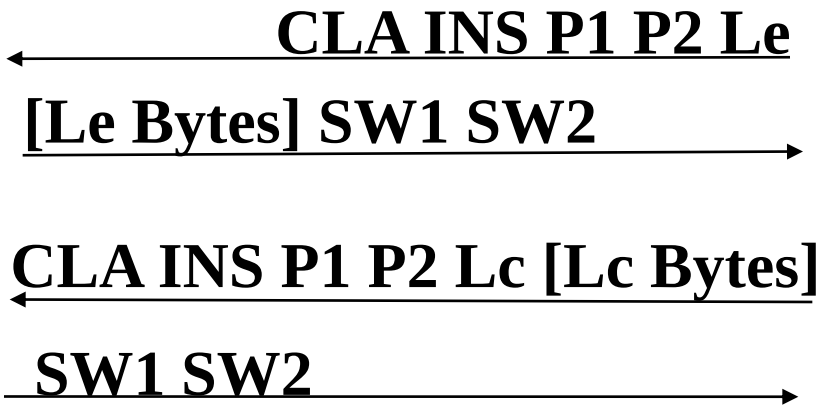


Exemple de Système d'Exploitation



- ***Guillou,L.C, Ugon, M, Quisquater,J.J "Smartcard: a Standardized Security Device Dedicated to Public Cryptology", 1992.***
  - *"*What a smartcard does*. The five operations of a smartcard are 1-input data, 2- output data, 3- read data from non volatile memory (NVM), 4- write or erase data in NVM, 5- compute a cryptographic function."*





Commandes de base ISO7816-4

Lecture LE bytes

CLA INS P1 P2 Le

[Le Bytes] SW1 SW2

Ecriture Lc bytes

CLA INS P1 P2 Lc [Lc Bytes]

SW1 SW2

Y=F(x)

1- Ecriture xx bytes

CLA INS P1 P2 xx [xx Bytes]

SW1=61 SW2=yy

2- Lecture yy bytes

CLA INS=C0 P1=0 P2=0 P3=yy

[yy bytes] SW1 SW2

Le Paiement Sécurisé

HSBC
MONEY GALLERY

*British Museum

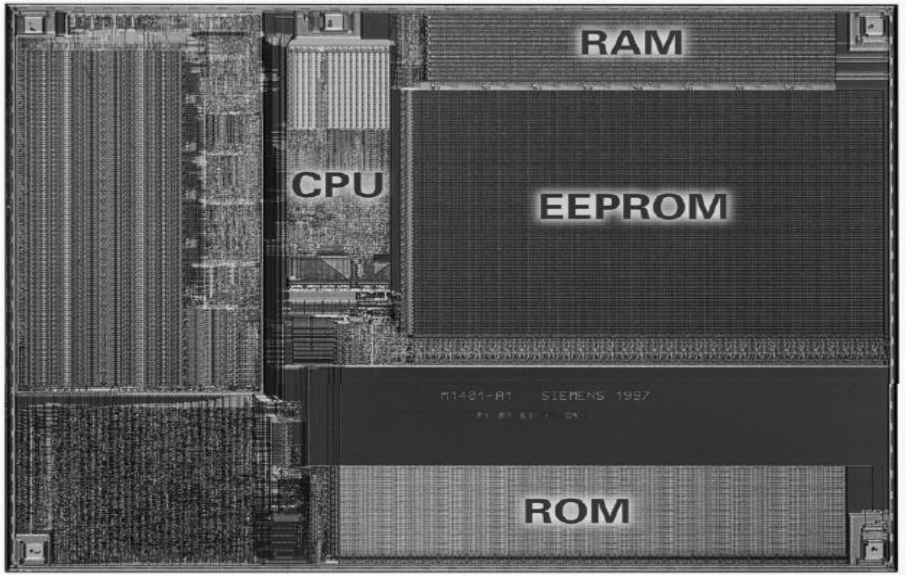
MONEY TODAY

The development of monetary technology continues today. A bank card permits the account holder to make payments by direct transfer and withdraw money from cash machines. Coins and notes now compete with a new generation of 'smart cards'. These contain microchips which store electronic cash to provide a fast, convenient way of paying.

Case 17, Section 4
Actual size 85 x 54mm

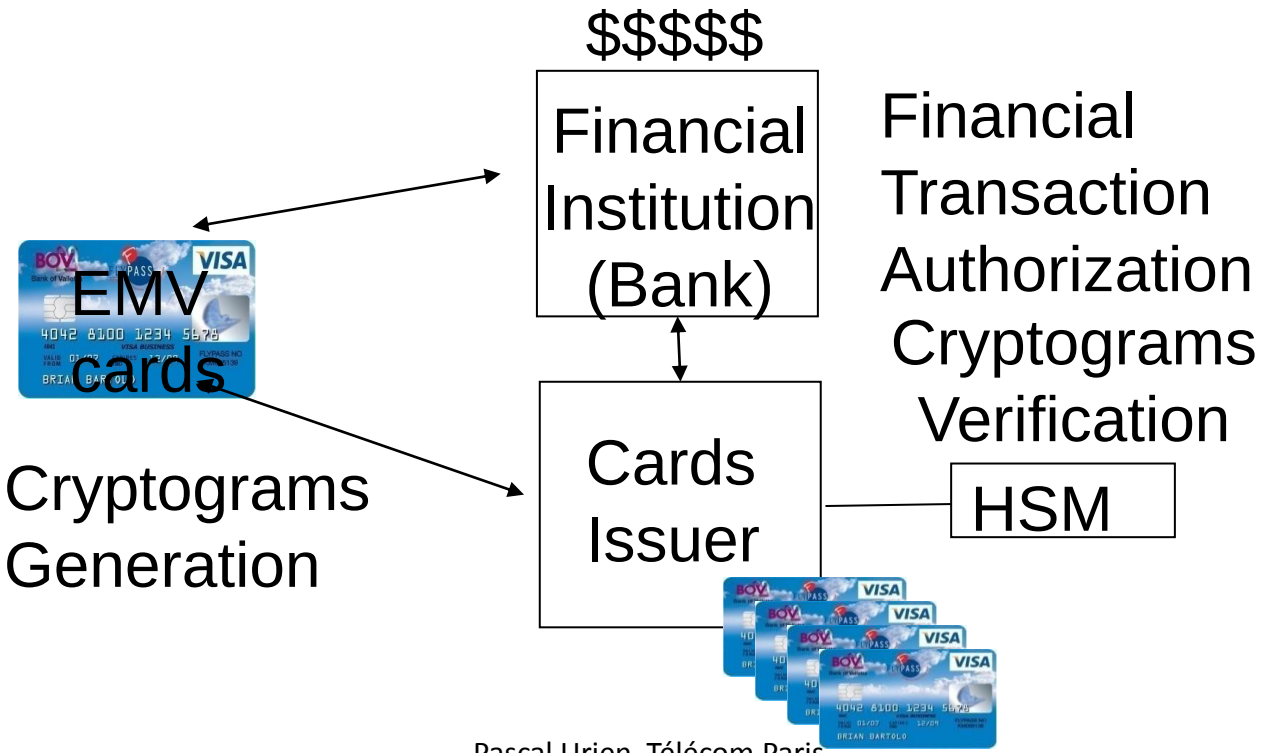
La carte SIM

- From the report of SIMEG#1 in January 1988
  - "A SIM is the physically secured module which contains the IMSI, an authentication algorithm, the authentication key and other (security related) information and functions. The basic function of the SIM is to authenticate the subscriber identity in order to prevent misuse of the MS (Mobile Station) and the network."



Siemens CHIP, 1997

Le modèle EMV : EuroCard, MasterCard, Visa

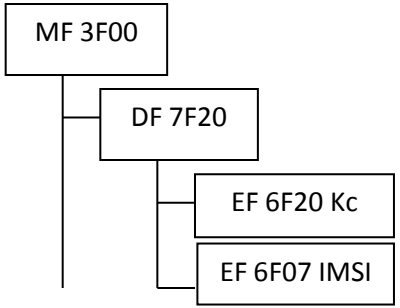


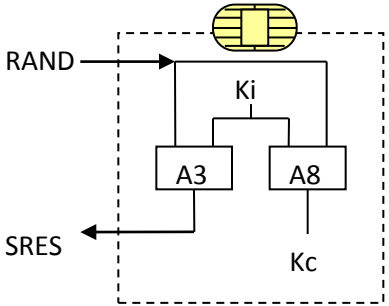
Exemple: ARQC

- >> 80AE8000 1D
  - 00 00 00 00 00 00, the transaction amount
  - 00 00 00 00 00 00, the cash back
  - 00 00, the national code of the payment terminal
  - 80 00 00 00 00, the terminal verification result
  - 00 00, the transaction currency code
  - 01 01 01, the transaction date
  - 00, the type of transaction
  - 12 34 56 78, a four bytes random value
- << 77 1E
  - 9F 27 01 80, Cryptogram Information Data
  - 9F 36 02 00 18, Application Transaction Counter (ATC)
  - 9F 26 08 80 29 D3 A0 BB 2A 5E 60, Application Cryptogram
  - 9F 10 07 06 7B 0A 03 A4 A0 00, Issuer Application data

La carte SIM

- L'information est organisée en répertoires et fichiers
- Quelques données
  - IMSI
  - Deux répertoires téléphoniques
  - Un fichier SMS
- Quelques procédures
  - RUN_GSM_ALGORITHM, calcule l'algorithme A3/A8





// Run_Gsm_Algorithm(RAND) **RAND**

>> A0 88 00 00 10 01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF

<< 9F 0C

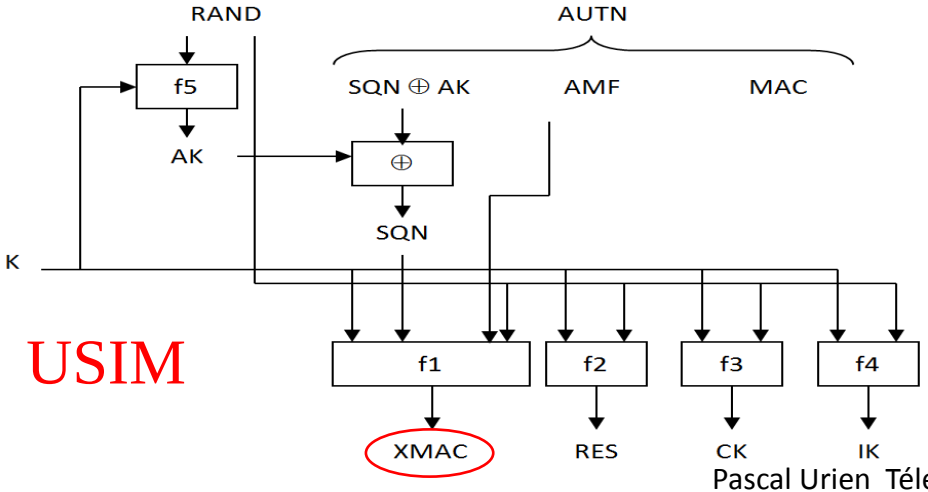
>> A0 C0 00 00 0C **KC**

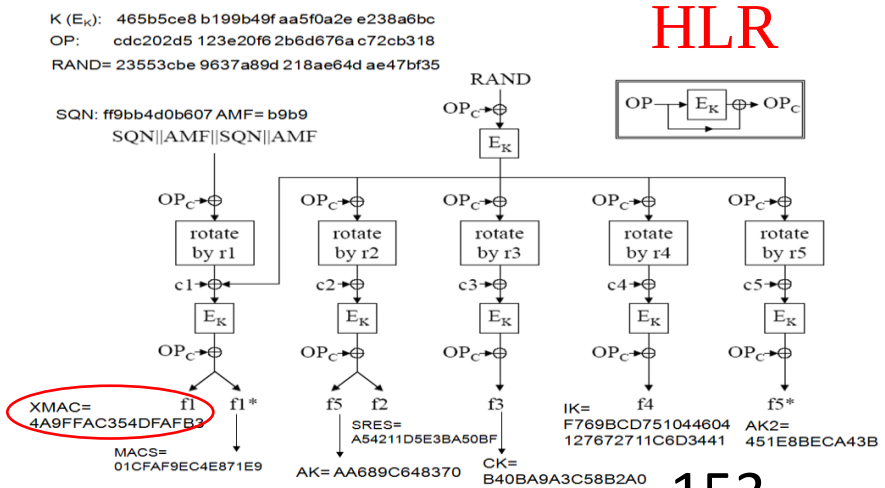
<< EE 67 7C 9D B8 DD F1 B1 DE 27 18 00 90 00

SRES

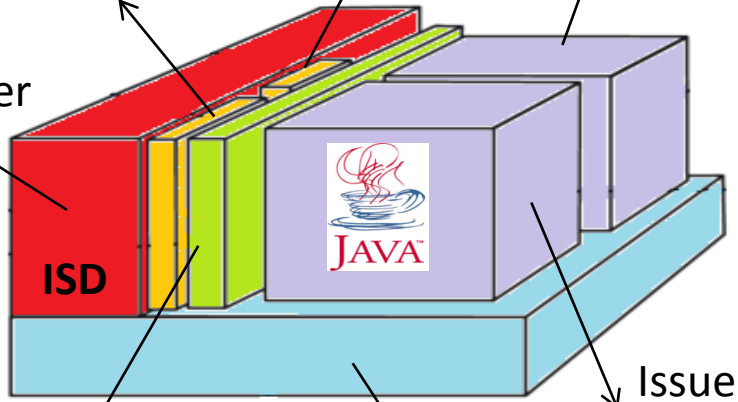
La carte USIM

- Le module UICC stocke au moins une application USIM
  - Le fichier EF_DIR contient la liste des applications USIM
- Au moins deux applications peuvent être activées simultanément (notion de canaux logiques
  - L'index de l'application est indiqué dans les deux derniers bits de l'octet CLA
- L'algorithme d'authentification AKA est réalisé par la commande AUTHENTICATE (INS=88) command
- Exemple
- >> 00 88 00 00 20 RAND || AUTN
- << DB 28 SRES || CK || IK 9000
  - AUTN:= SQN ⊕ AK || AMF || XMAC





Smartcard Administration: GP



Issuer Security Domain

Other Security Domain

Other Application

Card Manager

Global Platform API

Runtime Environment

Issuer Application

The VISA Model*

*EMV Card Personalization Specification Version 1.1 July 2007

KMC-ID (6B)

CSN (Chip Serial Number, 4B)

KMC (DES Master Key for Personalization Session Keys)

K_{ENC} K_{MAC} K_{DEK}

Select ISD

Mutual Authentication

SKU_{ENC} SKU_{MAC} SKU_{DEK}

Secure Channel

Application Management downloading - deletion

Near Field Communication

Petite Histoire du NFC

- 1994, Mifare 1K
  - En 2011 les chips Mifare représentent 70% du marché du transport.
- 2001, ISO 14443 Standards (13,56 Mhz)
  - Type A (Mifare)
  - Type B
  - Type F (Felica)
- 2004, NFC Forum
  - Mifare (NXP), ISO14443A, ISO14443B, Felica (Sony)
  - Trois modes fonctionnels
    - Reader/Writer, Card Emulation, Peer to Peer
- Les contrôleurs NFC réalisent le protocole NFC

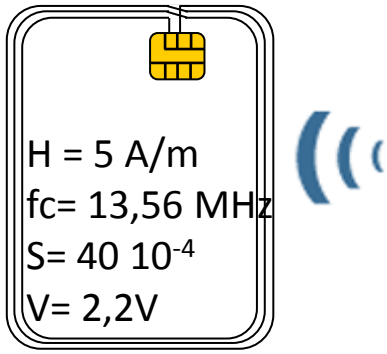
De l'ISO 7816 à ISO 14443

- L'idée de base du Wi-Fi était la définition d'un réseau Ethernet sans fil.
- L'idée de base de l'ISO 14443 était la définition de cartes à puce (ISO 7816) sans contact.
- **Contrairement à la norme IEEE 802.11 aucune sécurité n'est définie pour les protocoles radio ISO14443**

ISO 7816
Contact Mode

$$V = 2\pi f_c S \mu_o H$$

ISO 14443
Contactless Mode



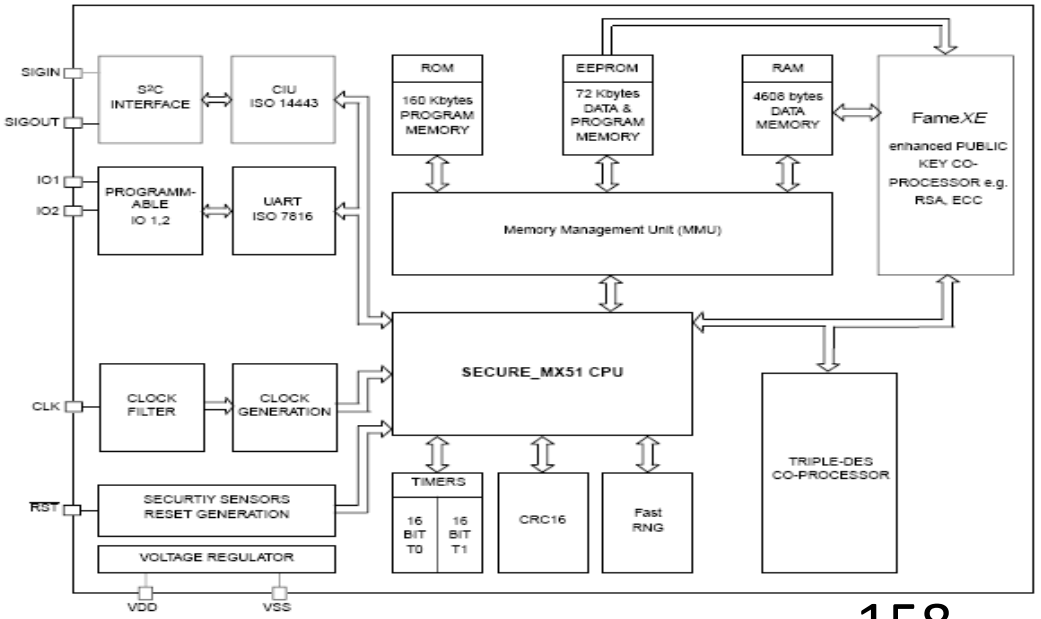
Definition d'un Secure Element.

Un Secure Element (SE) est un microcontrôleur sécurisé muni d'interfaces électriques et de communication, telles que ISO7816, SPI or I²C .

OS JAVACARD JCOP
GP (Global Platform)
ROM 160 KB
EEPROM 72 KB
RAM 4KB
Crypto-processor
3xDES, AES, RSA, ECC
Certification CC EAL5+
Security Certificates EMVCo

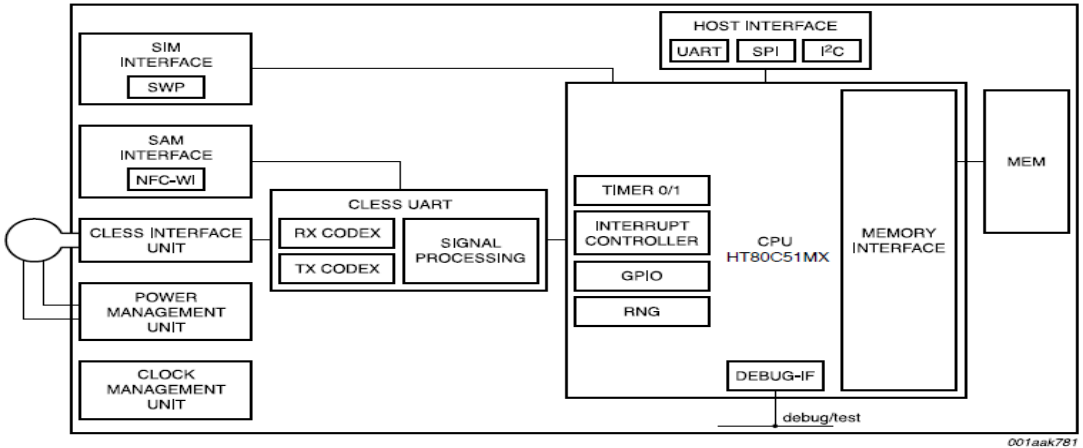
	NFC secure modules
Product features	**PN65L**
Embedded NFC IC	PN532
Available host interfaces	serial, SPI, I²C
Embedded Secure IC	P5CN072
OS for secure device	JCOP or 3rd party
Stacked passive component IC	yes
Package thickness	1.2 mm
Package size	7x7 mm²
Package type	HLQFN48

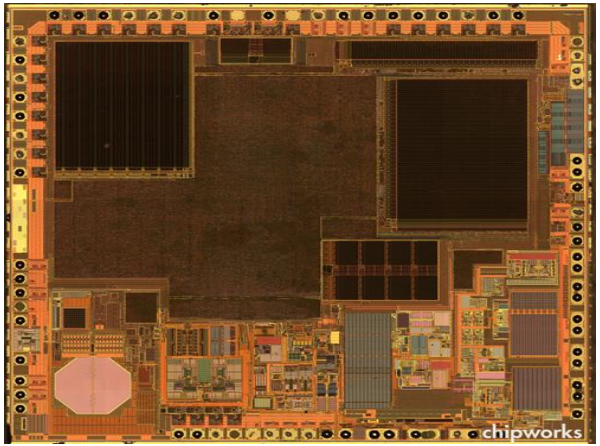
EXAMPLE: NXP PN532



NFC et Secure Elements

- Certain contrôleurs NFC embarquent un Secure Element.
- Dans ce cas le mode "card emulation" peut être géré par le Secure Element
- C'est le modèle Android 2.3





www.chipworks.com

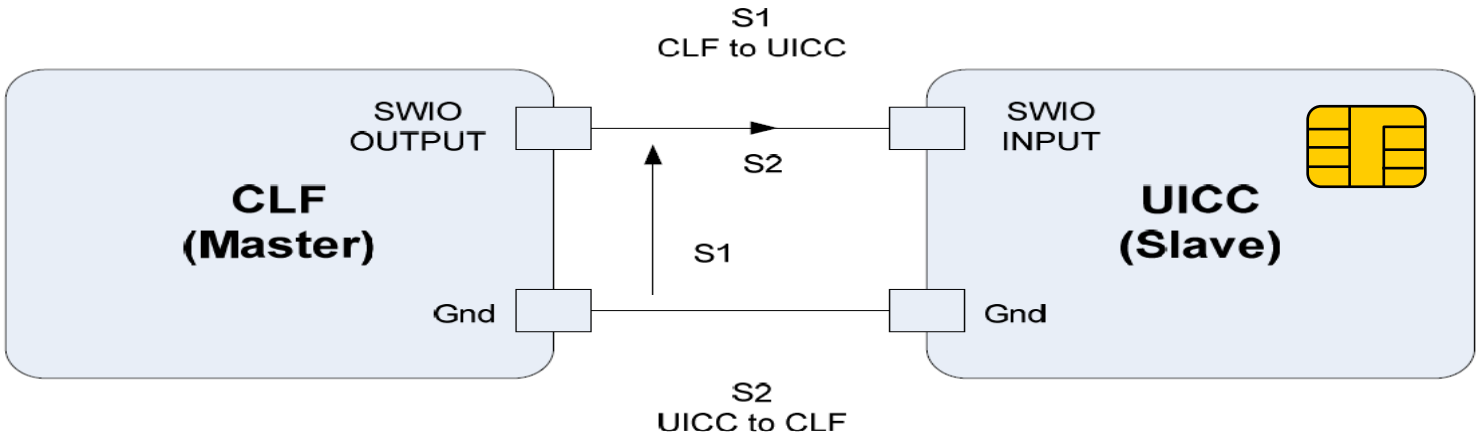
Reader/writer ISO 14443 –A-B, MIFARE, FeliCa®, NFC Forum tags, ISO 15693

Card Emulation ISO 14443 –A-B-B', MIFARE, FeliCa RF , SWP

RAM 5Ko, ROM 128 Ko, EEPROM 52 Ko

La carte SIM se transforme en device NFC: le Contactless Front-end (CLF)

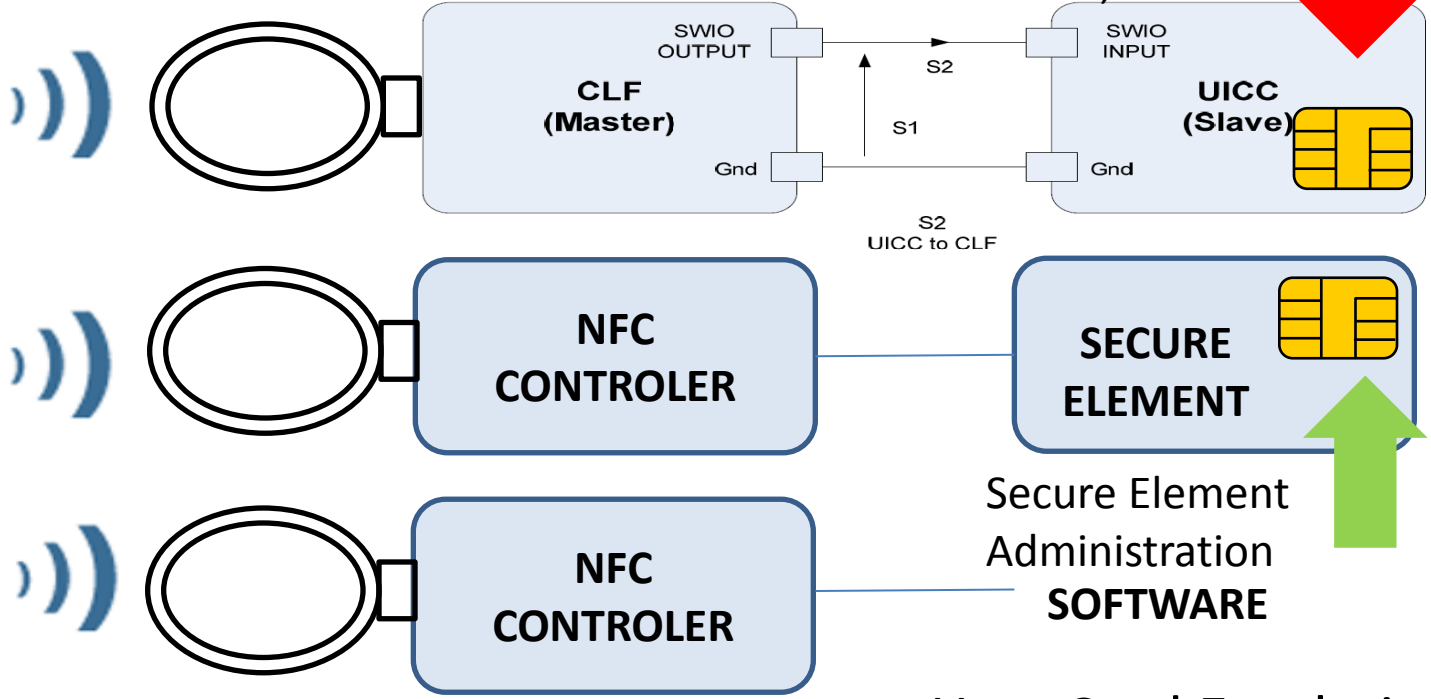
The ETSI TS 102 613 Standard



A simplified HDLC protocol: SHDLC

A physical Link: Single Wire Protocol (SWP)

NFC Reader/Writer & Card Emulation



Reader/Writer

Card Emulation

Host Card Emulation
Android 4.4

NFC P2P Mode

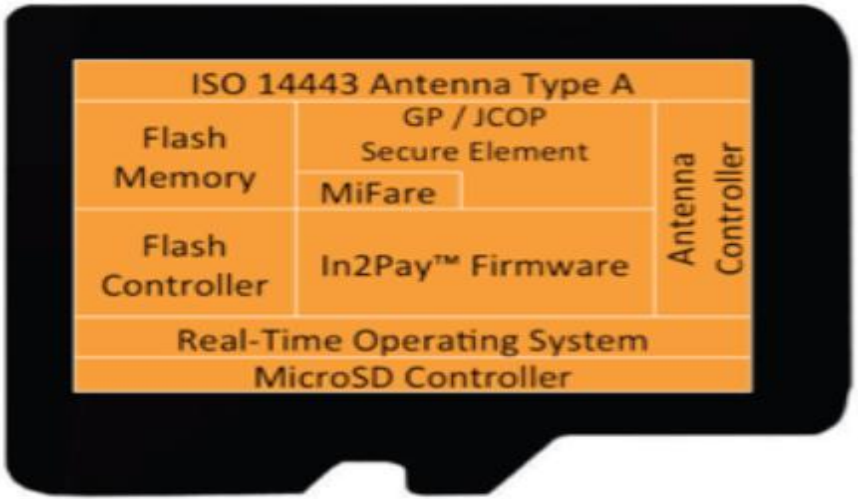
- **Android NDEF Push Protocol Specification**
  - **Version 1, 2011-02-22**

"The NDEF Push Protocol (NPP) is a simple protocol built on top of LLCP which is designed to push an NDEF message from one device to another."

Initiator Target

SD Card with NFC Controller

EEPROM 72 Ko

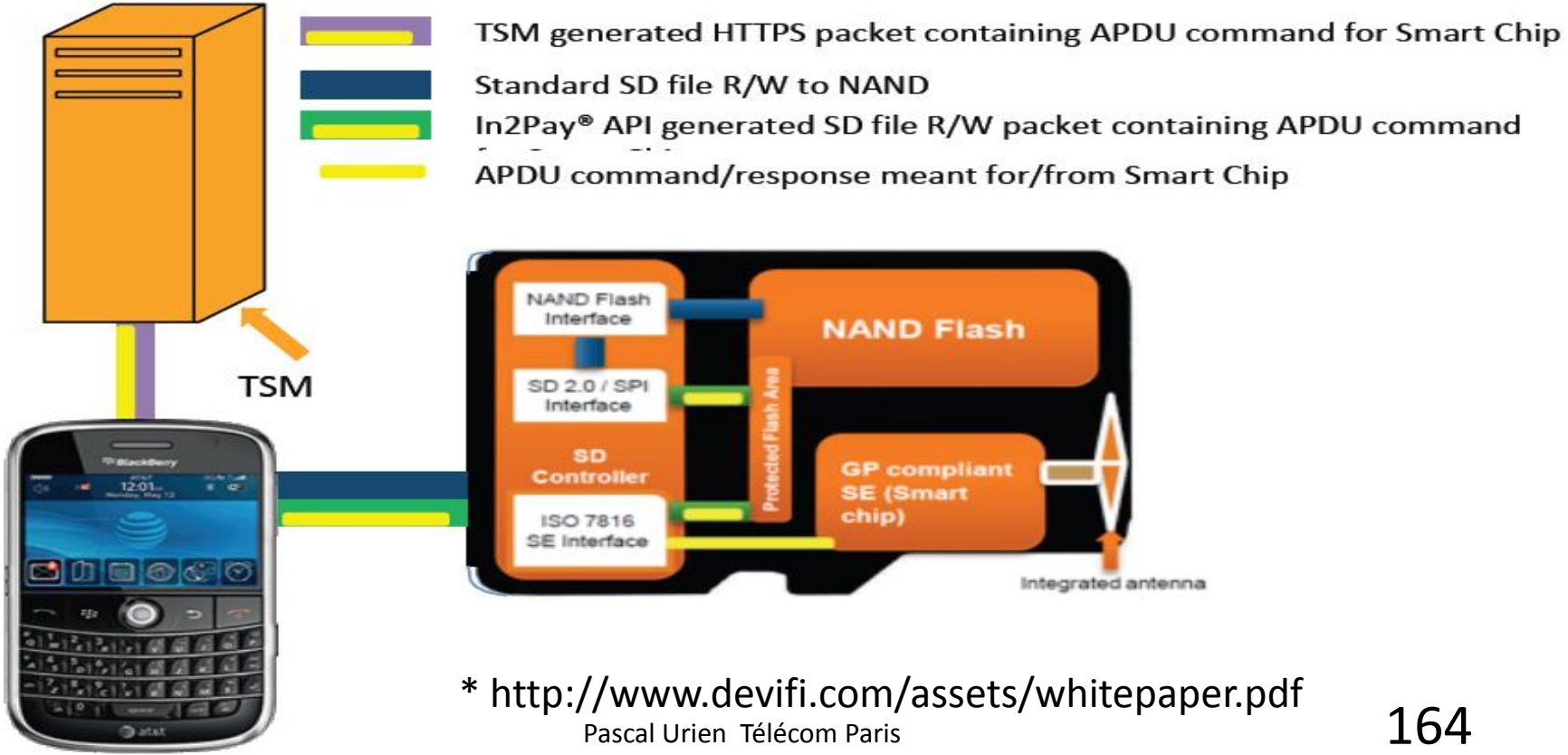


http://www.devicefidelity.com/

http://www.tyfone.com/

Secure Element	
ISO 7816, T=0, T=1 (communication speed, kbit/s	223.1
ISO 14443 T=CL (communication speed, kbit/s)	424
Available EEPROM Options kBytes	72
MiFare 1K	Yes
ROM (free for applets, up to kBytes)	65
APDU Buffer (RAM/bytes)	261
Java Card Version	2.2.1
Global Platform	2.1.1
SCP Secure Channel Protocol	SCP01, SCP02
DES/TDES (bit)	56/112/168
RSA (bit)	2432
SHA	SHA-1
Random Number Generator	Yes
RSA Key Generation	Yes
Logical Channels	1
VSDC 2.7.1	Yes
VMPA 1.3.1	Yes
MChip4	Yes
Mobile PayPass 0.91	Yes
Discover Zip	Yes

The In2Pay Administration Model*



* http://www.devifi.com/assets/whitepaper.pdf

Apercu
des standards NFC

Résumé des Standards NFC

Activity	Technology / Device Platform						
Listen, RF Collision Avoidance, Technology Detection, Collision Resolution	NFC-A ISO 14443-2A ISO 14443-3A				NFC-B 14443 -2B 14443 -3B	NFC-F ISO 14443-2A ISO 14443-3A FELICA	
Device Activation		Type 1 Tag Platform	Type 2 Tag Platform	Type 4A Tag Platform	Type 4B Tag Platform	Type 3 Tag Platform	
Data Exchange	NFC-DEP Protocol	Type 1, 2, and 3 Tag Half-duplex Protocol		ISO-DEP Protocol		Type 1, 2, and 3 Tag Half-duplex Protocols	NFC-DEP Protocol
Device Deactivation	NFCIP-1			ISO 14443-4			NFCIP-1

NDEF

SNEP

LLCP

Passive Mode
Active Mode
NFCIP-1

*ISO/IEC_18092 standard and NFCIP-1 standards are similar

DEP: Data Exchange Protocol (Supports Read/Write Operations for Tags)

ISO 14443
106 kbps
212 kbps
424 kbps
848 kbps

Standard	PCD to ICCC Reader to Card	PICC to PCD Card to Reader
ISO 14443-2A NFC-A	ASK 100% Modified Miller	Subcarrier fc/16 OOK Manchester
ISO 14443-2B NFC-B	ASK 10%, NRZ-L	Subcarrier fc/16 BPSK, NRZ-L

NFCIP-1
Passive
Mode

Bit Rate	Initiator	Target
106 kbps	ASK 100% Modified Miller	Subcarrier fc/16 OOK Manchester
212-424 kbps	ASK 8-30% OOK Manchester	ASK 8-30% OOK Manchester

NFCIP-1
Active
Mode

Bit Rate	Initiator	Target
106 kbps	ASK 100% Modified Miller	ASK 100%, Modified Miller
212-424 kbps	ASK 8-30 % OOK Manchester	ASK 8-30%, OOK Manchester

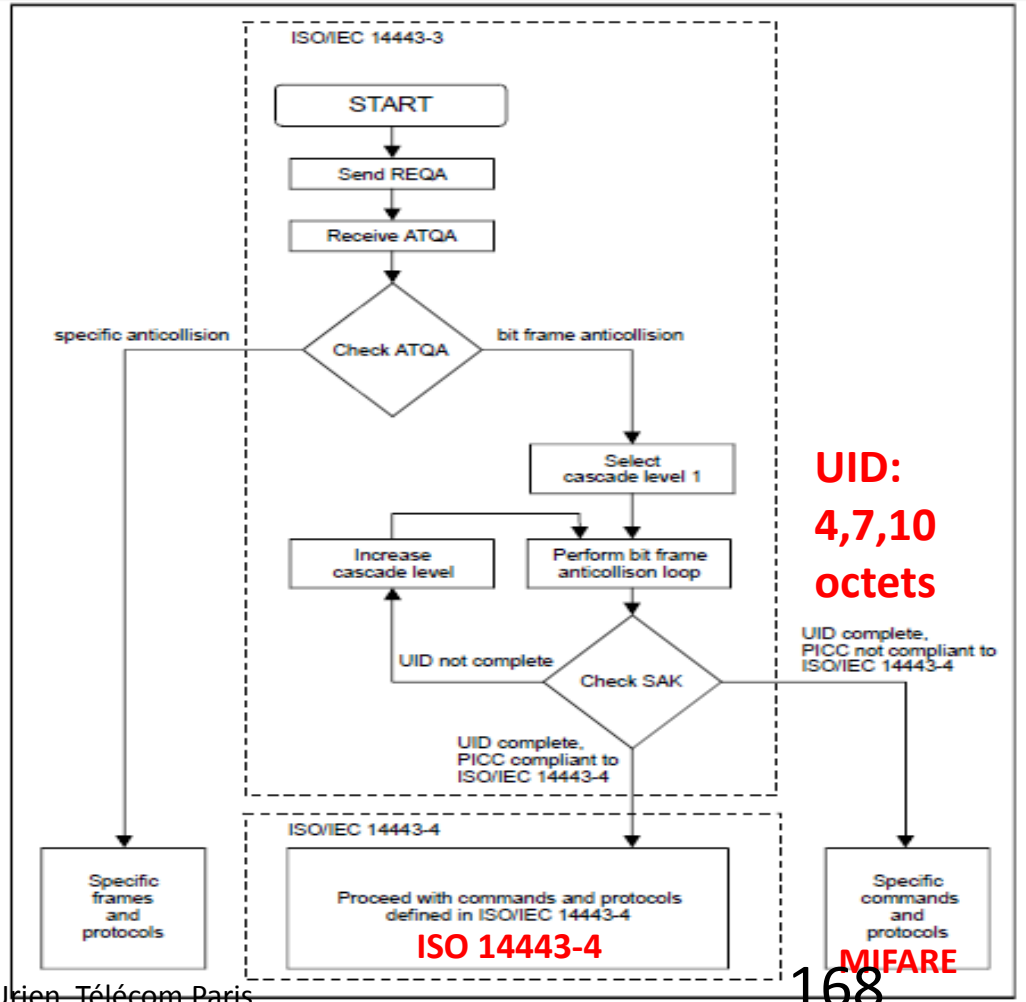
NFC Radio

ISO 14443-3A State Machine

- REQA: Request Command for Type A
- ATQA: Answer To Request of Type A
- SAK: Select AcKnowledge
- UID: Unique IDentifier

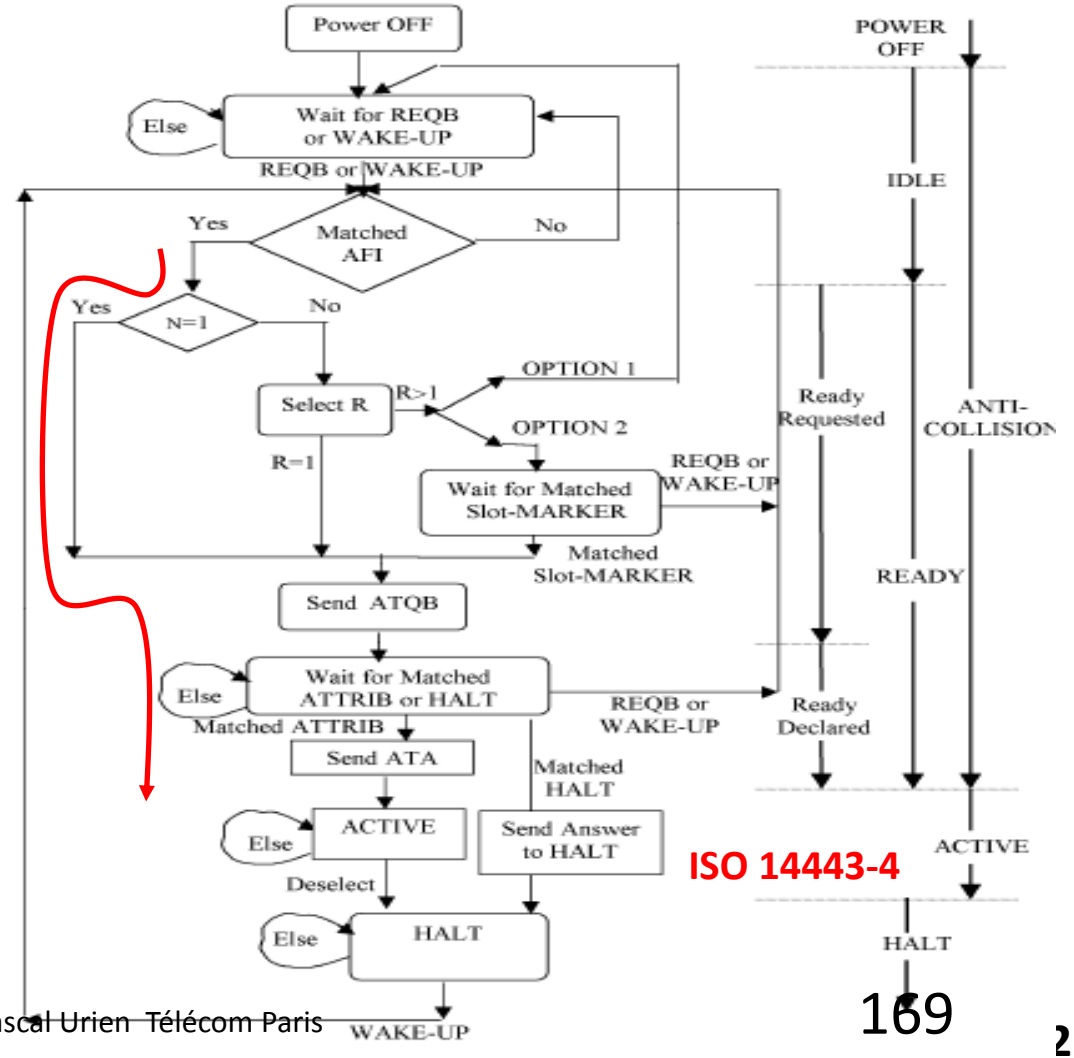
About Anti-collision

Several devices may be detected at the same time. For example the PN532 NFC controller can work simultaneously with two ISO14443 devices.



ISO 14443-3B State Machine

- AFI: Application Family Identifier (4 bytes). A card pre-selection criteria by application type.
- REQB: Request of Type B
- ATQB: Answer To Request of Type B
- ATA: Answer To ATTRIB



ISO 14443-4 Frames (T=CL)

- ISO 14443-4 frames transport ISO 7816-4 APDUs

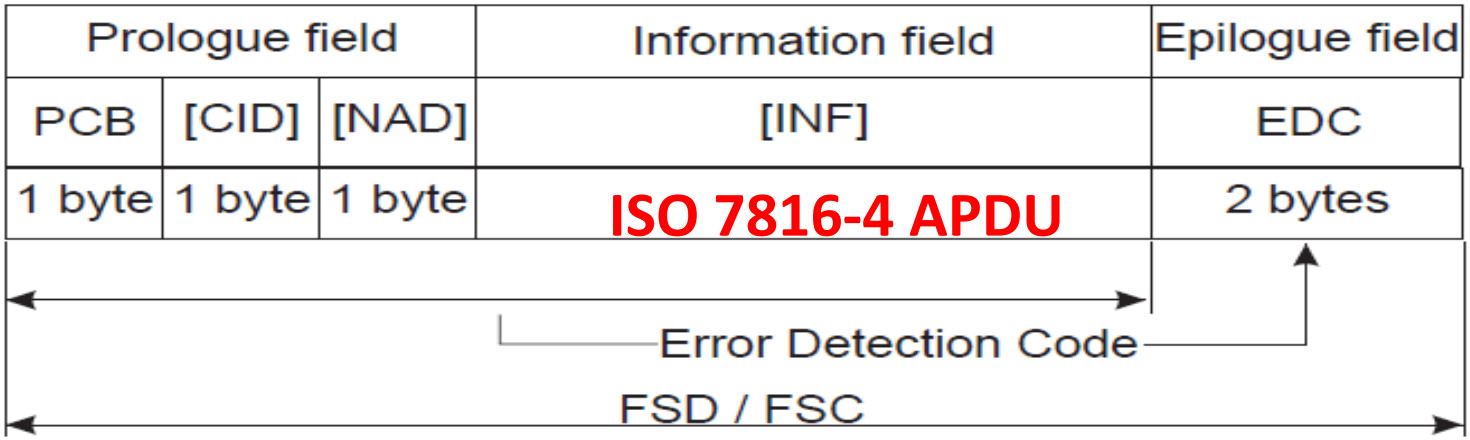


Figure 14 — Block format

ISO 14443-4

- RATS: Request for Answer To Select
- ATS: Answer To Select

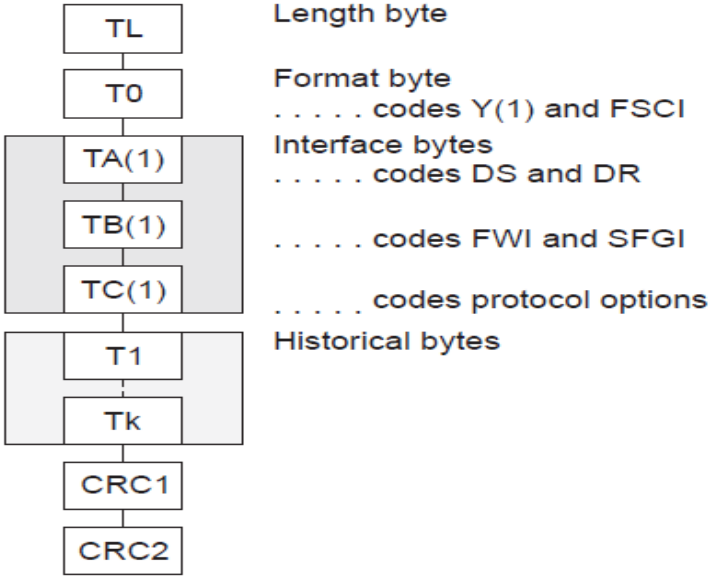


Figure 4 — Structure of the ATS

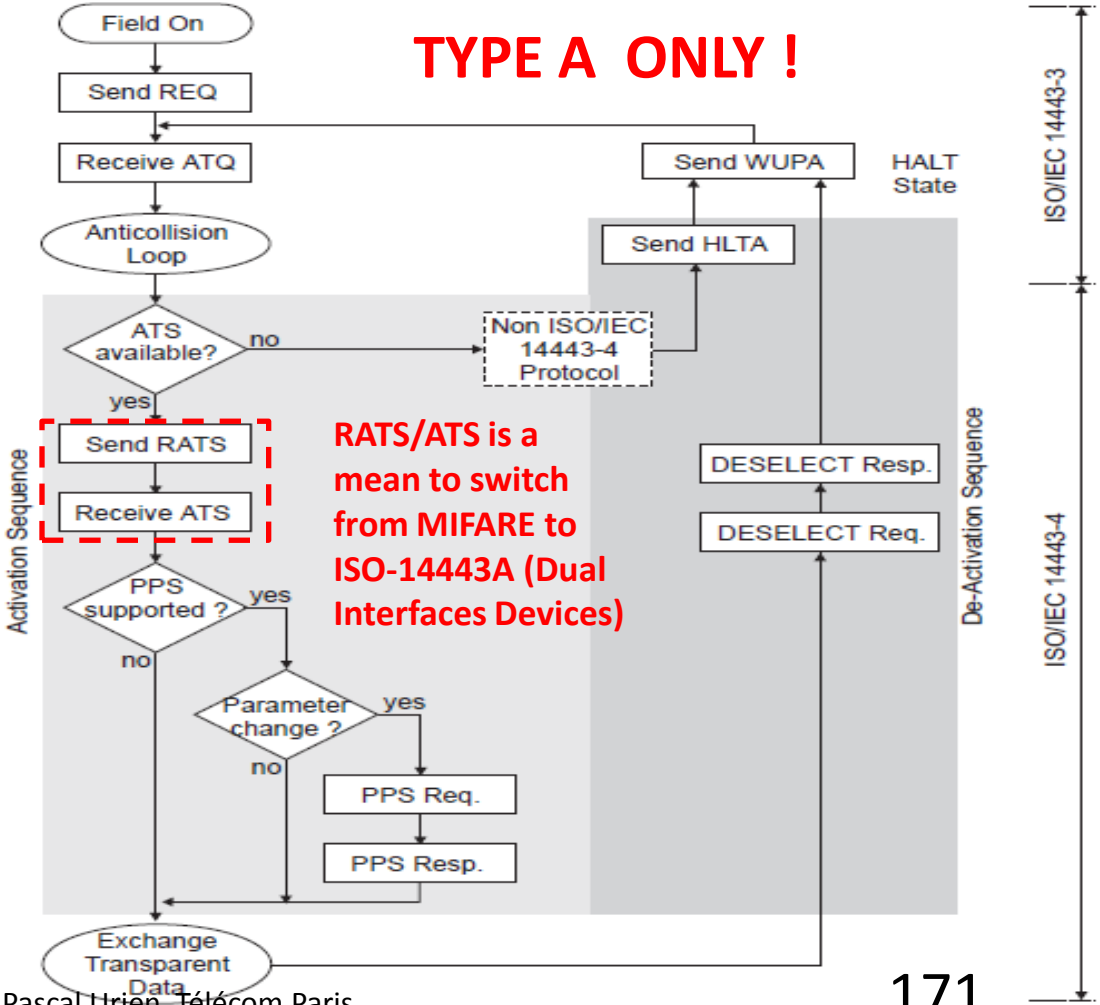
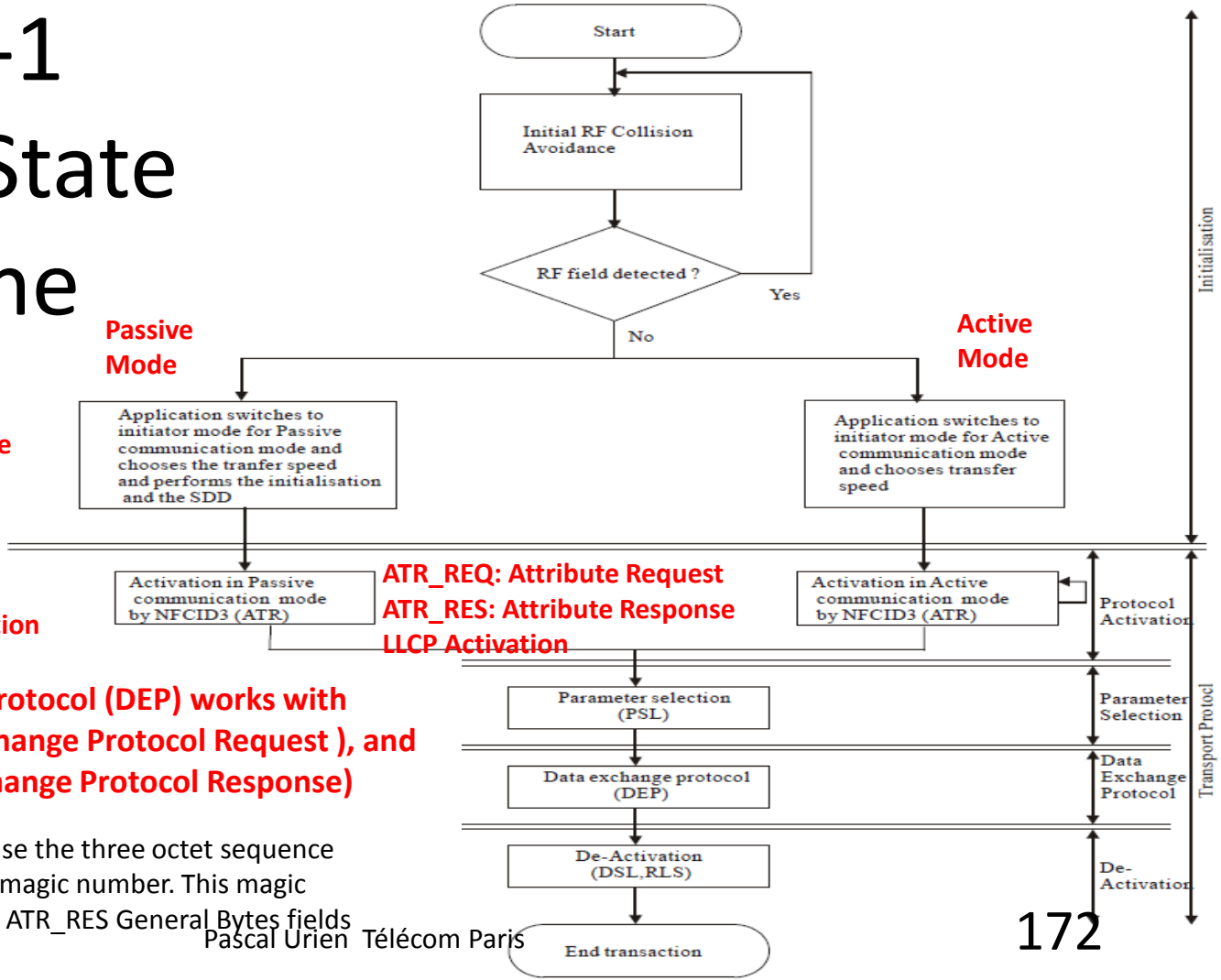


Figure 1 — Activation of a PICC Type A by a PCD

NFCIP-1 Initiator State Machine



SDD Single Device Detection

NFCID3: Random ID for transport protocol activation

ATR_REQ: Attribute Request
ATR_RES: Attribute Response
LLCP Activation

The Data Exchange Protocol (DEP) works with
- DEP_REQ (Data Exchange Protocol Request), and
- DEP_RES (Data Exchange Protocol Response)

The NFC-DEP MAC component SHALL use the three octet sequence "46h 66h 6Dh" as the NFC Forum LLCP magic number. This magic number is encoded into the ATR_REQ / ATR_RES General Bytes fields

LLCP: a Bridge to LAN Technologies

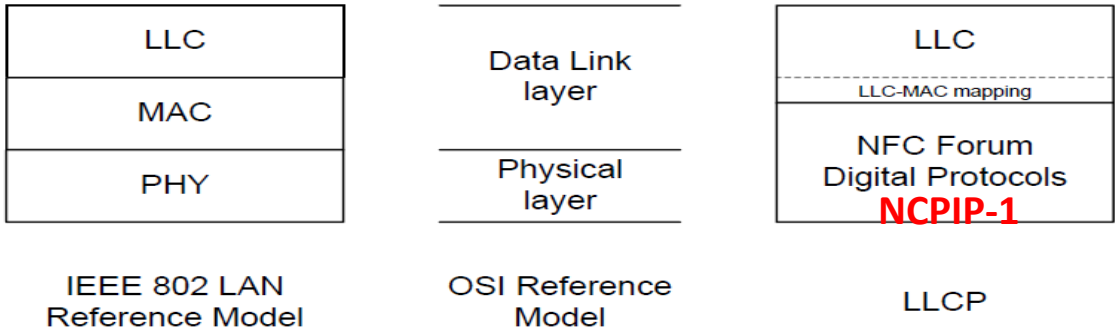


Figure 1: Relationship to OSI Reference Model

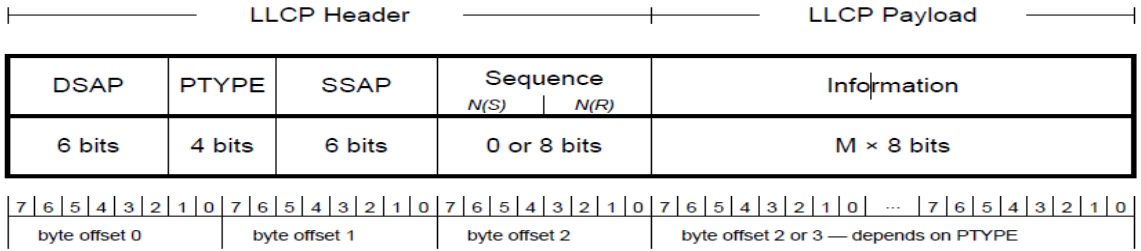


Table 3: PDU Type Values

PDU Type	PTYPE	Link Service Class
SYMM	0000	1, 2, 3
PAX	0001	1, 2, 3
AGF	0010	1, 2, 3
UI	0011	1, 3

PDU Type	PTYPE	Link Service Class
CONNECT	0100	2, 3
DISC	0101	1, 2, 3
CC	0110	2, 3
DM	0111	1, 2, 3
FRMR	1000	2, 3
SNL	1001	1, 2, 3
reserved	1010	
reserved	1011	
I	1100	2, 3
RR	1101	2, 3
RNR	1110	2, 3
reserved	1111	

NDEF: NFC Data Exchange Format

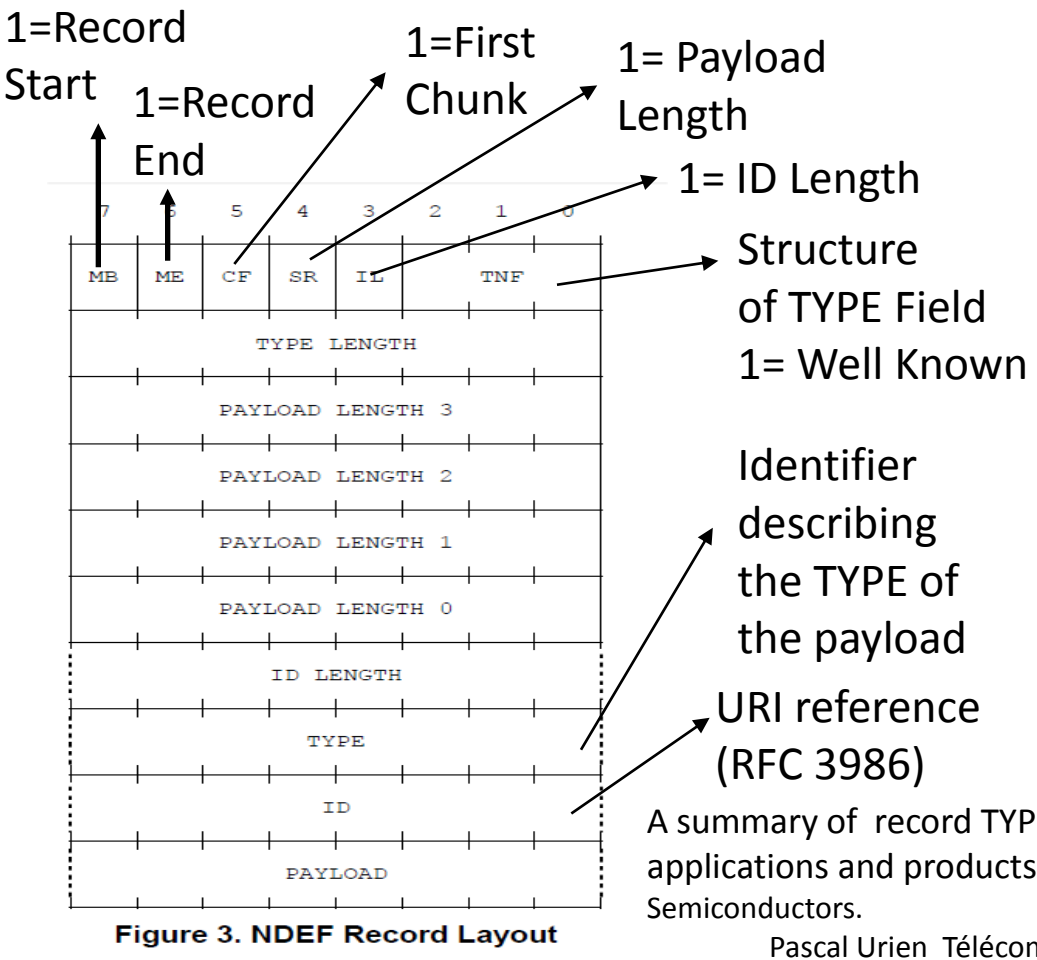


Figure 3. NDEF Record Layout

NDEF Record Example:
(NFC Text Record Type Definition)

D1: 1 1 0 1 0 001
01: Type Length
0A: Payload Length
54: Type= 'T', Text
02: ID= UTF8
65 6E: "EN"
53 61 6D 70 6C 65 20: "Sample "

A summary of record TYPE may be found in "NFC Tags A technical introduction, applications and products Rev. 1.3 - 1 December 2011 White paper", NXP Semiconductors.

NFC TAGs

NDEF Format for passive TAG

- **Type 1**
  - Based on ISO 14443-A
  - Innovision Topaz, Broadcom BCM20203
- **Type 2**
  - Similar to Type1
  - Based on ISO 144413-A
  - Compatible with NXP MIFARE Ultralight.
- **Type 3**
  - Similar to Type1
  - Based on the Japanese Industrial Standard (JIS) X 6319-4.
  - Compatible with Sony Felica
- **Type 4**
  - Similar to Type1
  - Based on ISO 14443-A
  - Compatible with standard ISO 14433-4 Smartcards
- **NXP-specific type tag**
  - Mifare Classic

LLCP NDEF services

- SNEP: Simple NDEF Exchange Protocol
- SNEP Requests and SNEP Responses
- LLC service access point address 4
- Service Name “urn:nfc:sn:snep”

Exemple de tag, Type2 Tag Mifare

Mifare Ultralight

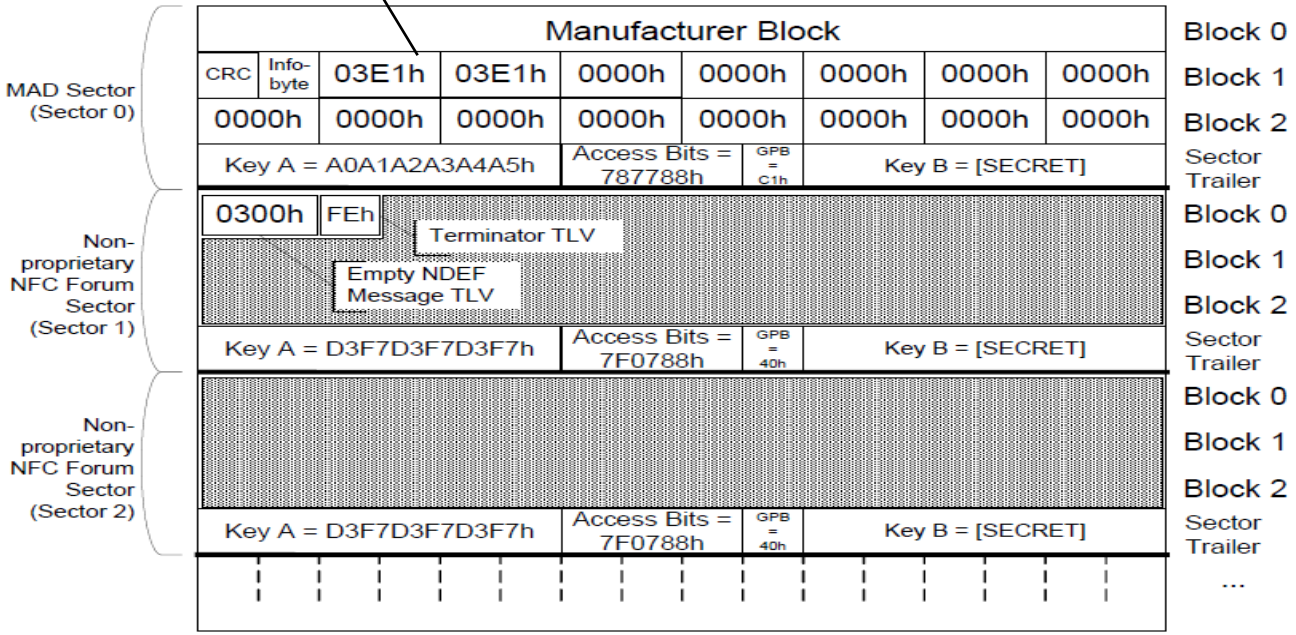
Block	Hex		ASCII
0	04D3	E0BF	.Óà¿
1	0277	1E80	.w.€
2	EB48	0000	ëH..
3	E110	0600	á...
4	030E	D101	..Ñ.
5	0B54	0265	.T.e
6	6E53	616D	nSam
7	706C	6520	ple
8	FE00	0000	þ...
9	0000	0000	....
10	0000	0000	....
11	0000	0000	....
12	0000	0000	....
13	0000	0000	....
14	0000	0000	....
15	0000	0000	....

Type2 Tag

Size 48 bytes

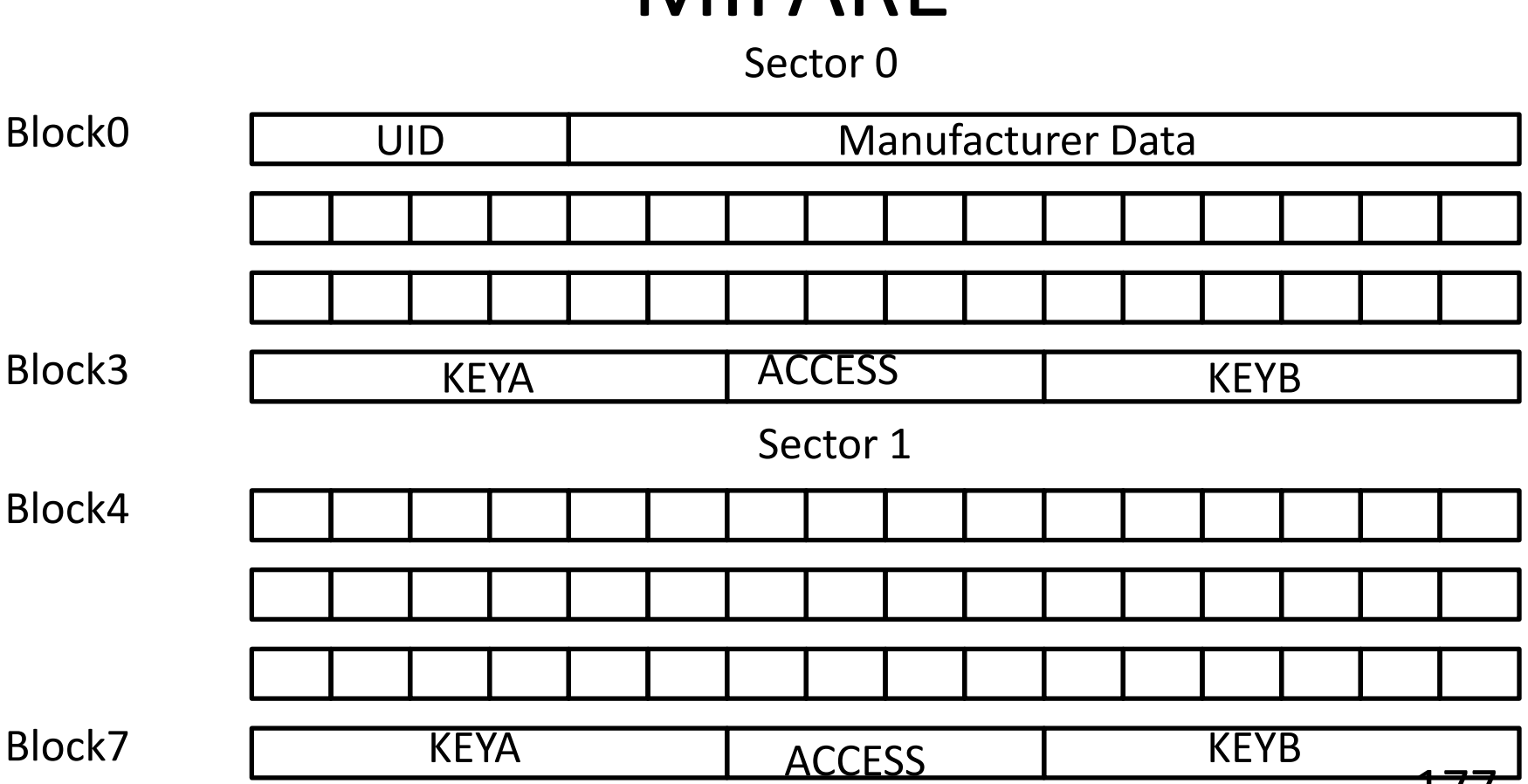
Mifare Classic

NDEF AID



Mifare Application Directory, MAD

MIFARE


Sector 0
Block0
UID
Manufacturer Data
Block3
KEYA
ACCESS
KEYB
Sector 1
Block4
Block7
KEYA
ACCESS
KEYB

Mifare UltraLight

OTP: One-Time Programmable (1=set)

LOCK bits

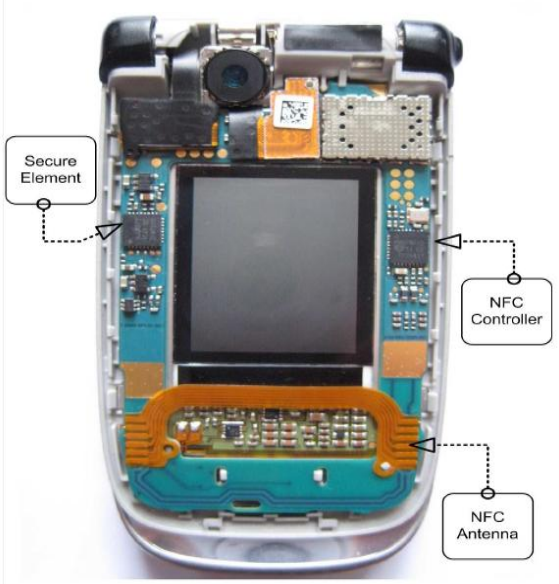
L7	L6	L5	L4	L OTP	BL 15-10	BL 9-4	BL OTP

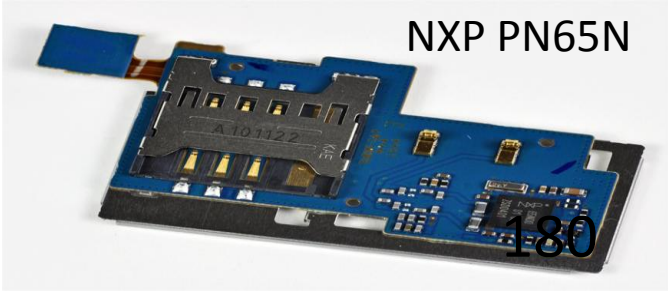
L15	L14	L13	L12	L11	L10	L9	L8

NFC et Smartphones

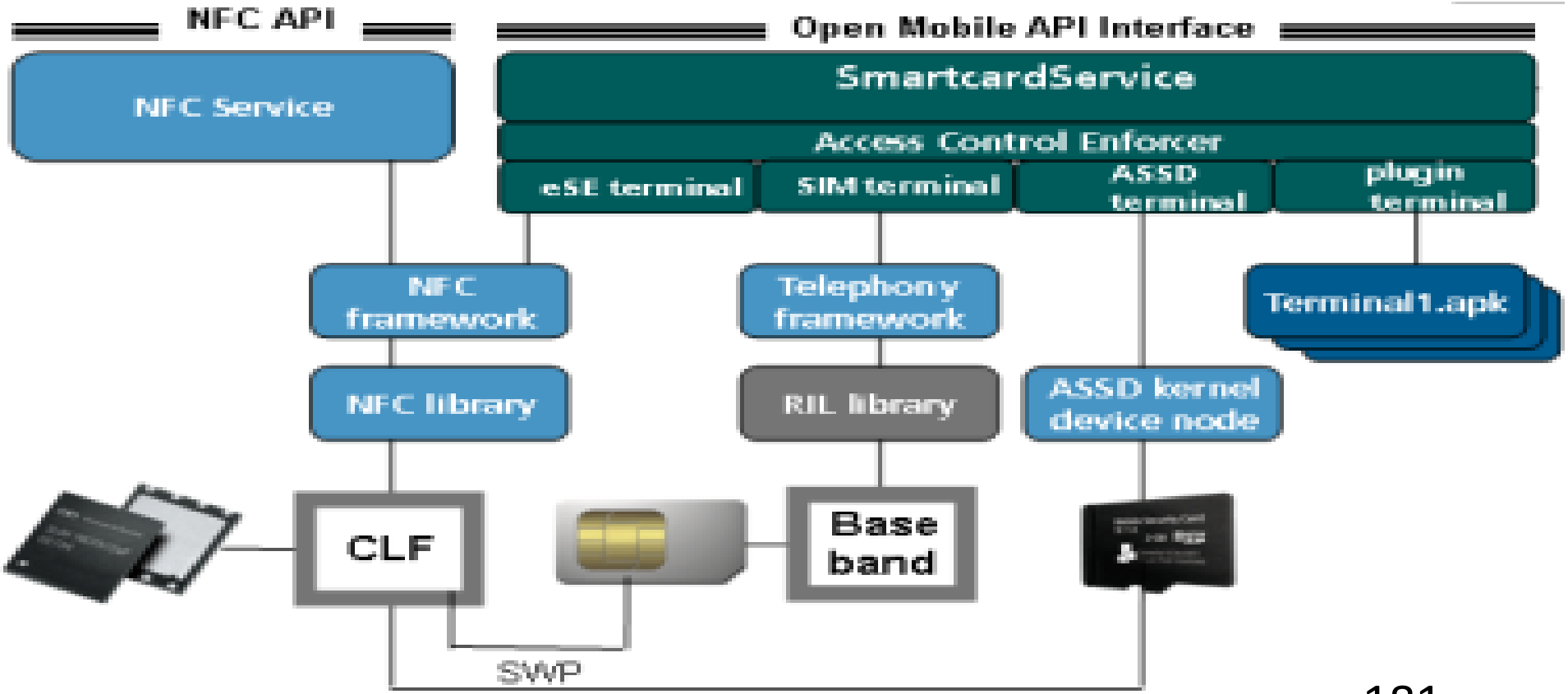
NFC et Smartphones

- Nokia
  - Card Emulation and SWP
  - NOKIA 6131
- Android 2.3 (Gingerbread), Android 4.0
  - Reader/Writer and P2P
  - Nexus S (v2.3) , Galaxy Nexus (v4.0), Galaxy S2(v2.3)
  - NXP NFC Controller PN65N
- Android 4.4 (KitKat)
  - Host Card Emulation (HCE)
  - Reader/Writer
  - P2P
- RIM JDE 7.0.0 (October 2011), BB10 (2013)
  - Reader/Writer and Card Emulation
  - JSR 177 (SIM Access)
  - Blackberry Bold 9900, 9930
  - INSIDE SecureRead NFC Controller
- BB10 (2013)
  - Reader/Write
  - Card Emulation
  - P2P
- IPHONE 6
  - Card Emulation
  - NXP NFC Controller

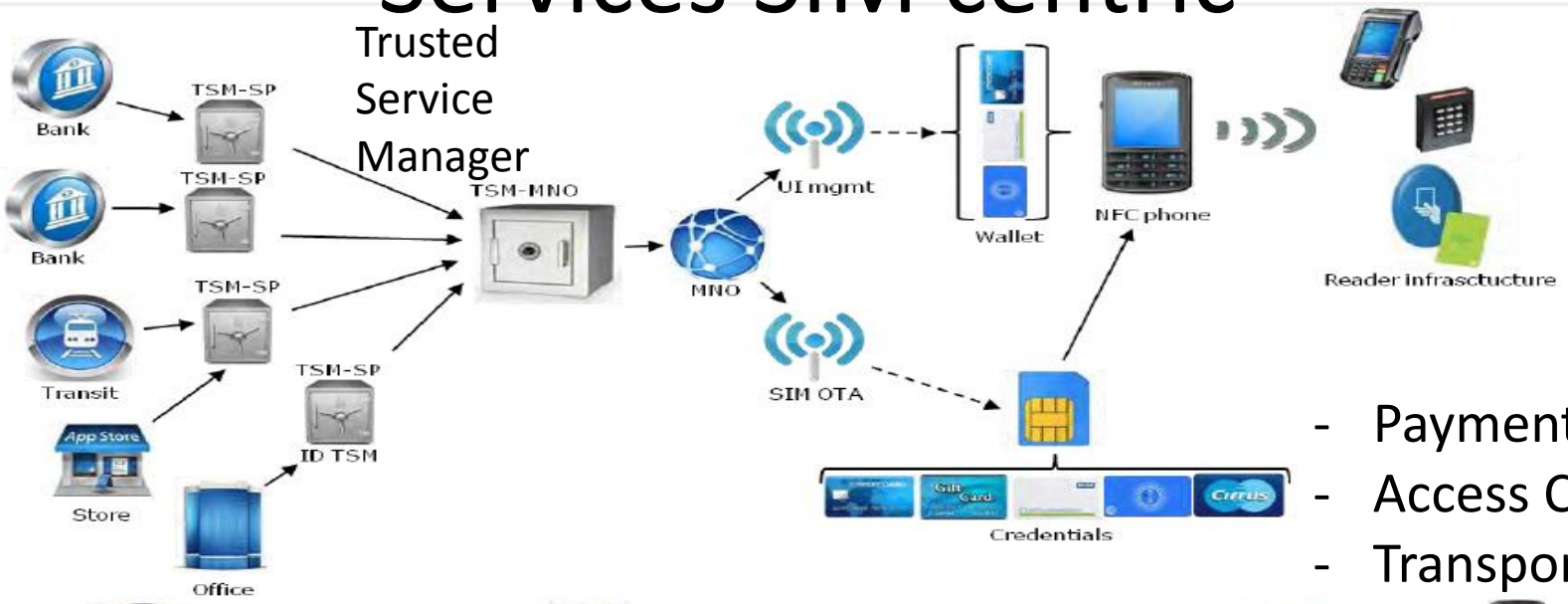




2011, Open Mobile API



HID NFC White Paper: Services SIM centric



NFC ecosystem with the Secure Element in the SIM and one MNO

La plateforme NFC Google

Reader/Writer

Google places

Card Emulation

Peer to Peer

Android Beams

NFC Tags

- EMV Magnetic Stripe Profile
- Cloud Storage

SNEP

SNEP, Android 4.x

NFC Initiator

NFC Target

ATR_REQ, NFC-MAGIC VERSION WKS LTO →

← **ATR_RES**, NFC-MAGIC VERSION WKS LTO

LLCP-SYMM [0000] →

← **LLCP-SYMM** [0000]

LLCP-SYMM [0000] →

← **CONNECT** [1120], DSAP=4, SSAP=32

CC [818402020078], DSAP=32, SSAP=4, MUI →

Information , SNEP PUT
[132000 10020000000F D1010B5402656E **6B657976616C7565**]
DSAP=4, SSAP=32, NS=0, NR=0, SNEP HEADER, NDEF RECORD, **keyvalue** ←

RR(1) [834401], SSAP=4, DSAP=32 →

Information, SNEP Success [830401 10810000000] ,
SSAP=4, DSAP=32, NS=0, NR=1 →

← **RR**(1) [13600], DSAP=4, SSAP=32

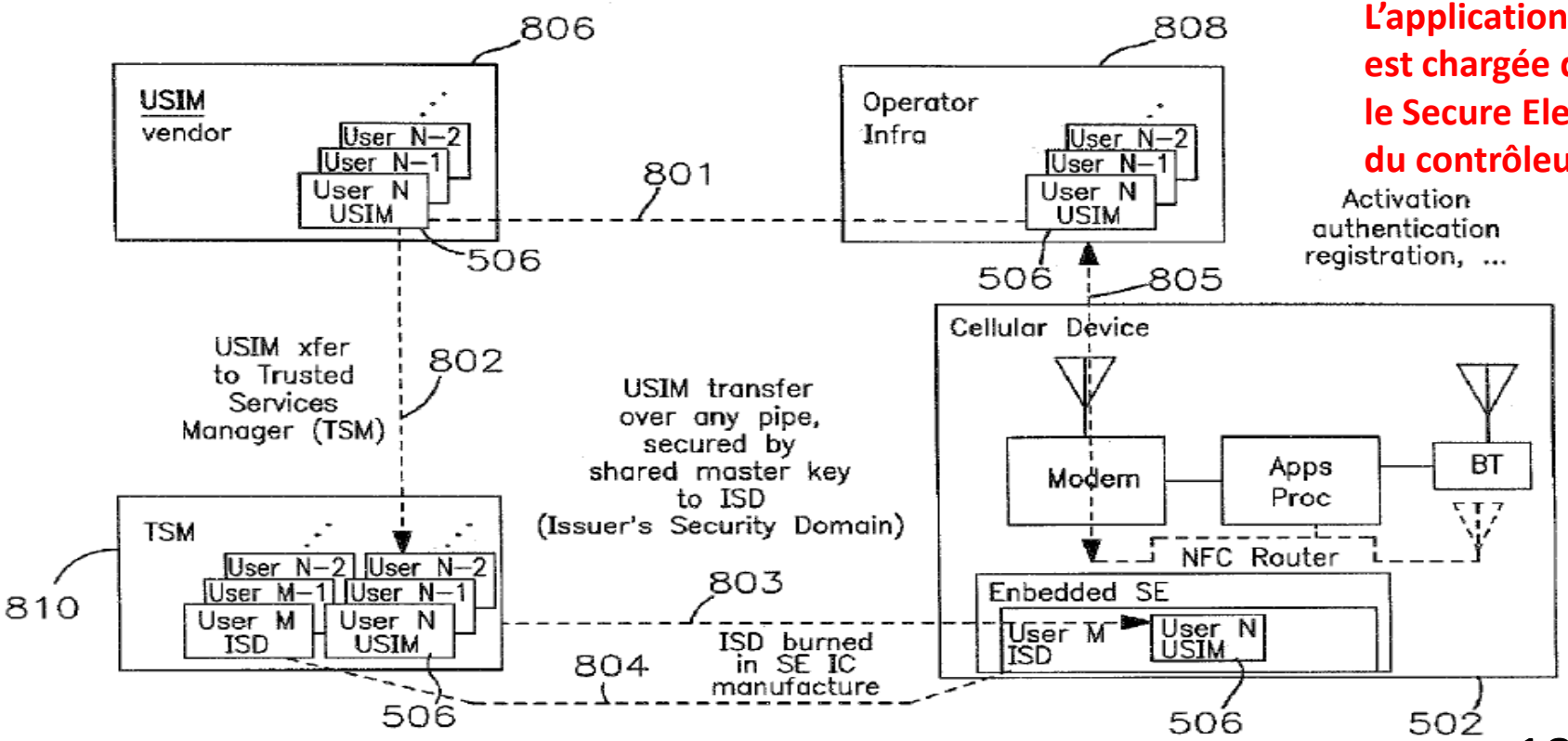
← **DISCONNECT** [1160] DSAP=4, SSAP=32

DM [C400], DSAP=16, SSAP=32 →

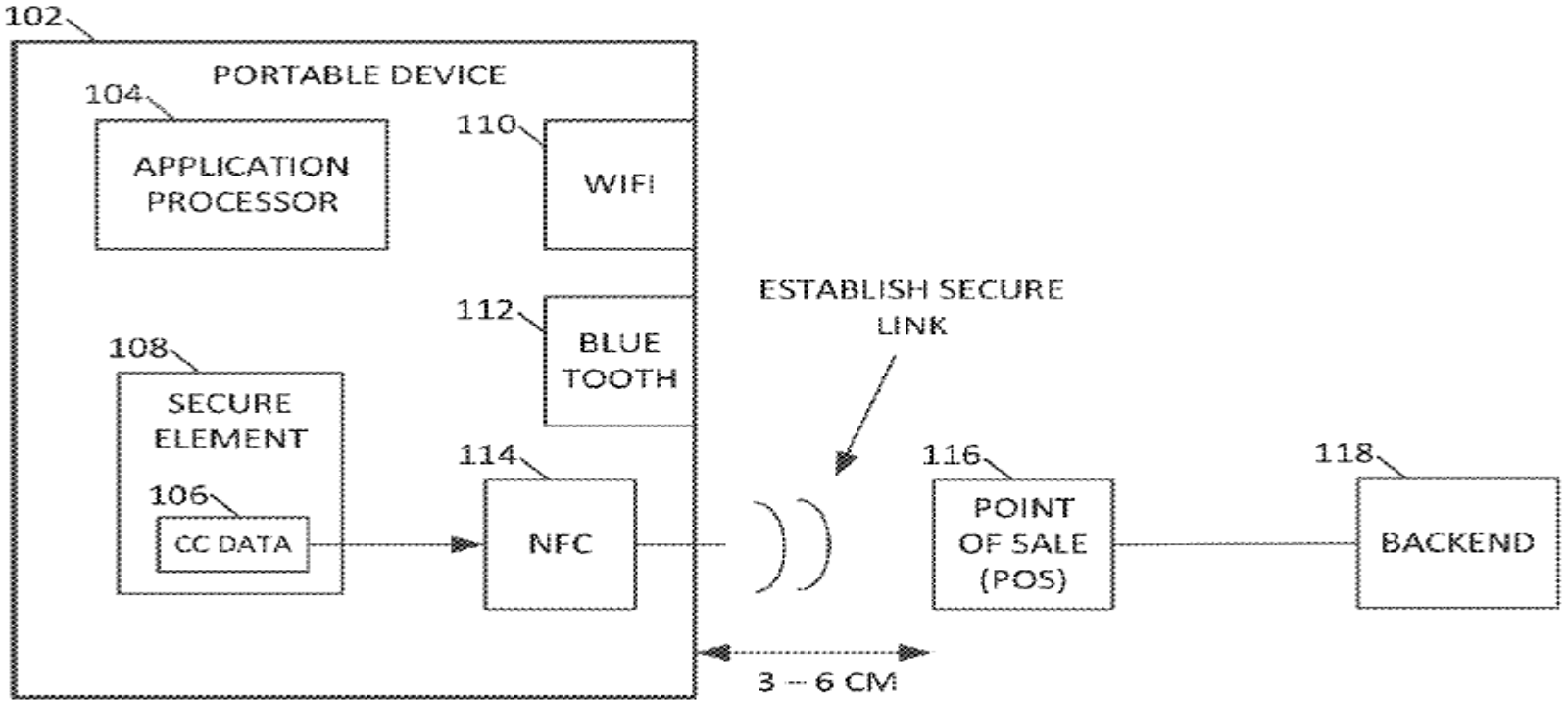
← **LLCP-SYMM** [0000]

Apple Patent, US 2011 0269423 A1

L'application USIM est chargée dans le Secure Element du contrôleur NFC

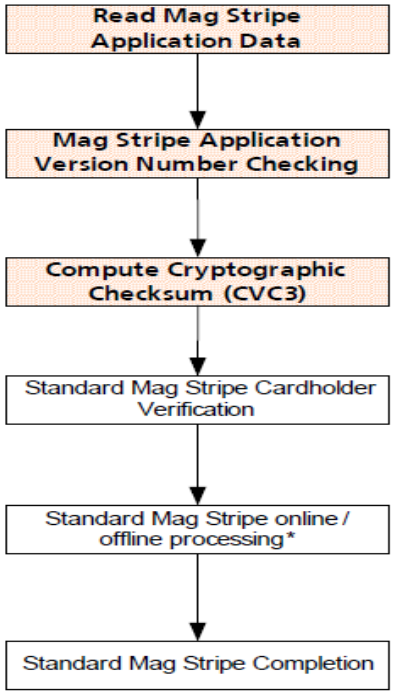


US Patent US 2014 0019367, Apple



Au Sujet des Paiements NFC

- Certain paiements NFC sont basés sur la spécification MasterCard PayPass
- Deux modes sont définis
  - Mag Stripe, un CVC3 de quatre digits (*Card Verification Value*) is calculé à partir d'un algorithme 3xDES de divers paramètres (PAN, ATC counter,…)
  - Contactless EMV
- Le Secure Element réalise les calculs cryptographiques et exécute l'application EMV.



* MasterCard® *PayPass™, M/Chip, Acquirer Implementation Requirements,* v.1-A4 6/06

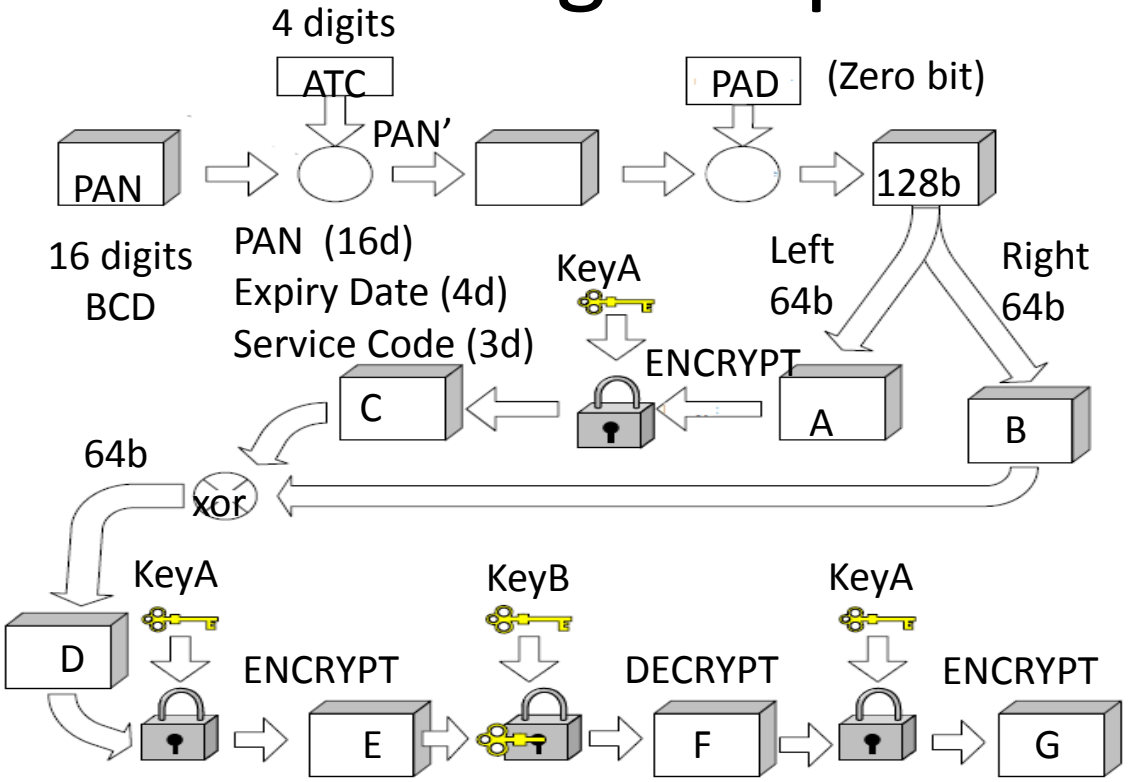
Details du profile EMV Mag. Stripe*

SELECT 2PAY.SYS.DDF01

SELECT Visa

GET PROCESSING OPTIONS

READ first record

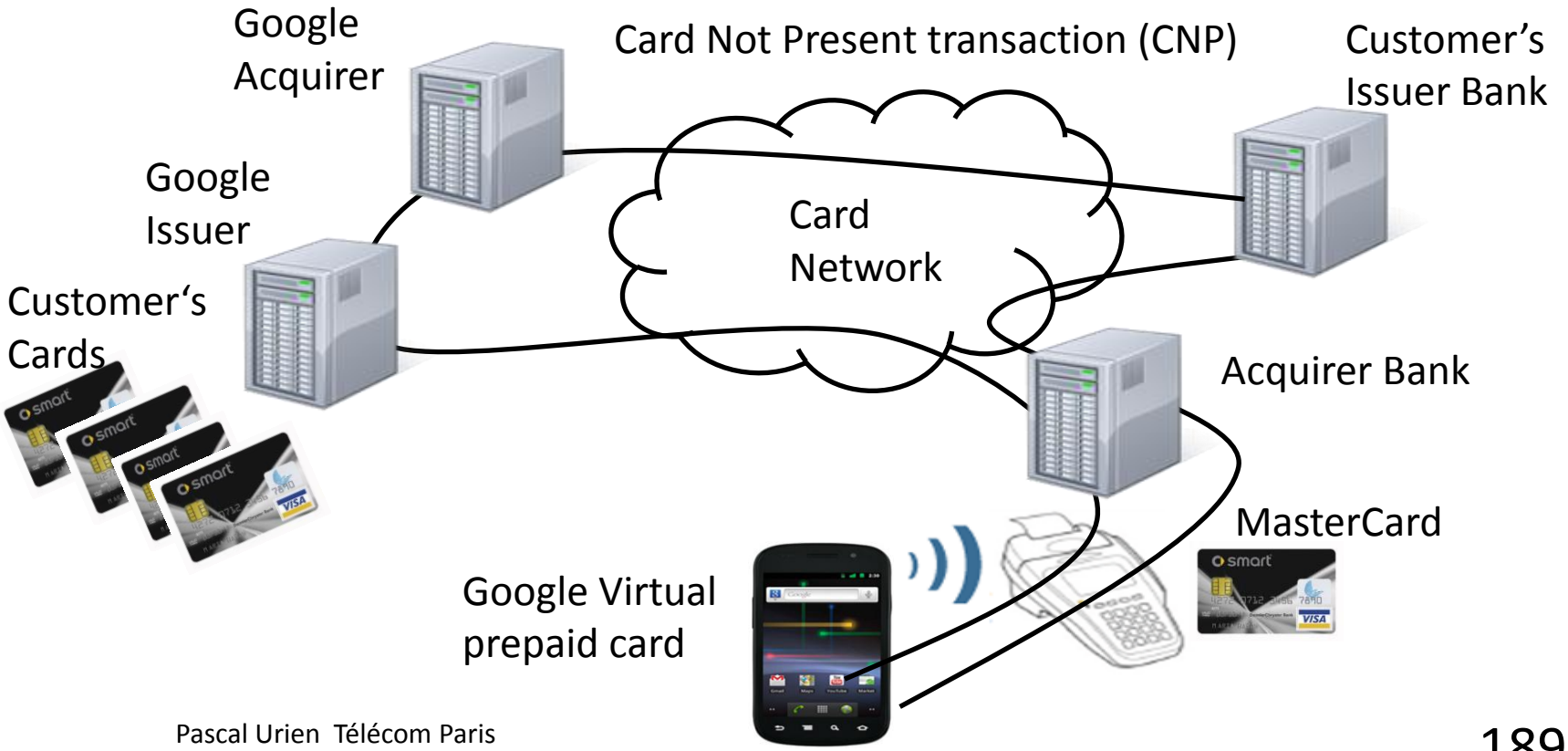


G=A23FB35FC89AE3A9
23358939AFBFCAEA
2335893905152040

CVC3=233

*Visa Contactless Payment Specification Version 2.0.2 July 2006

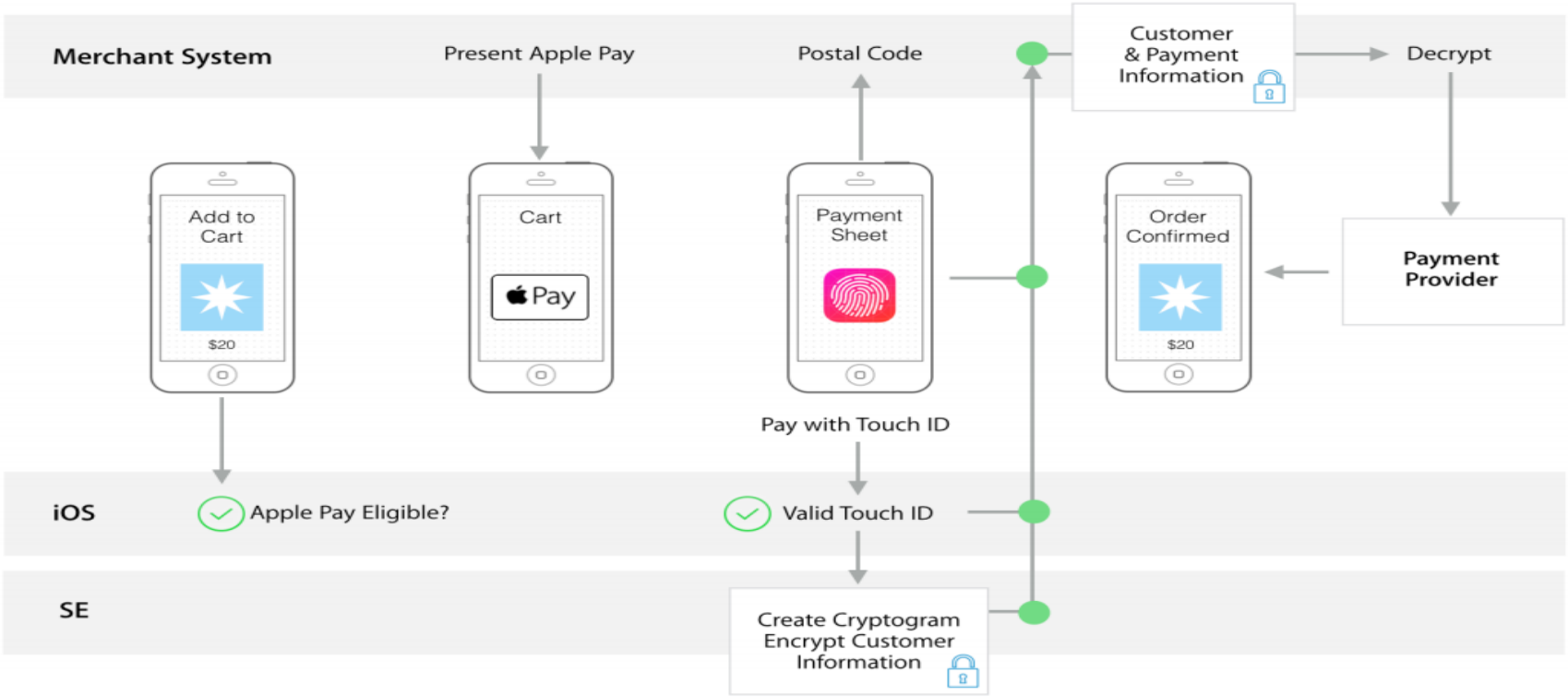
Le Google Wallet 2

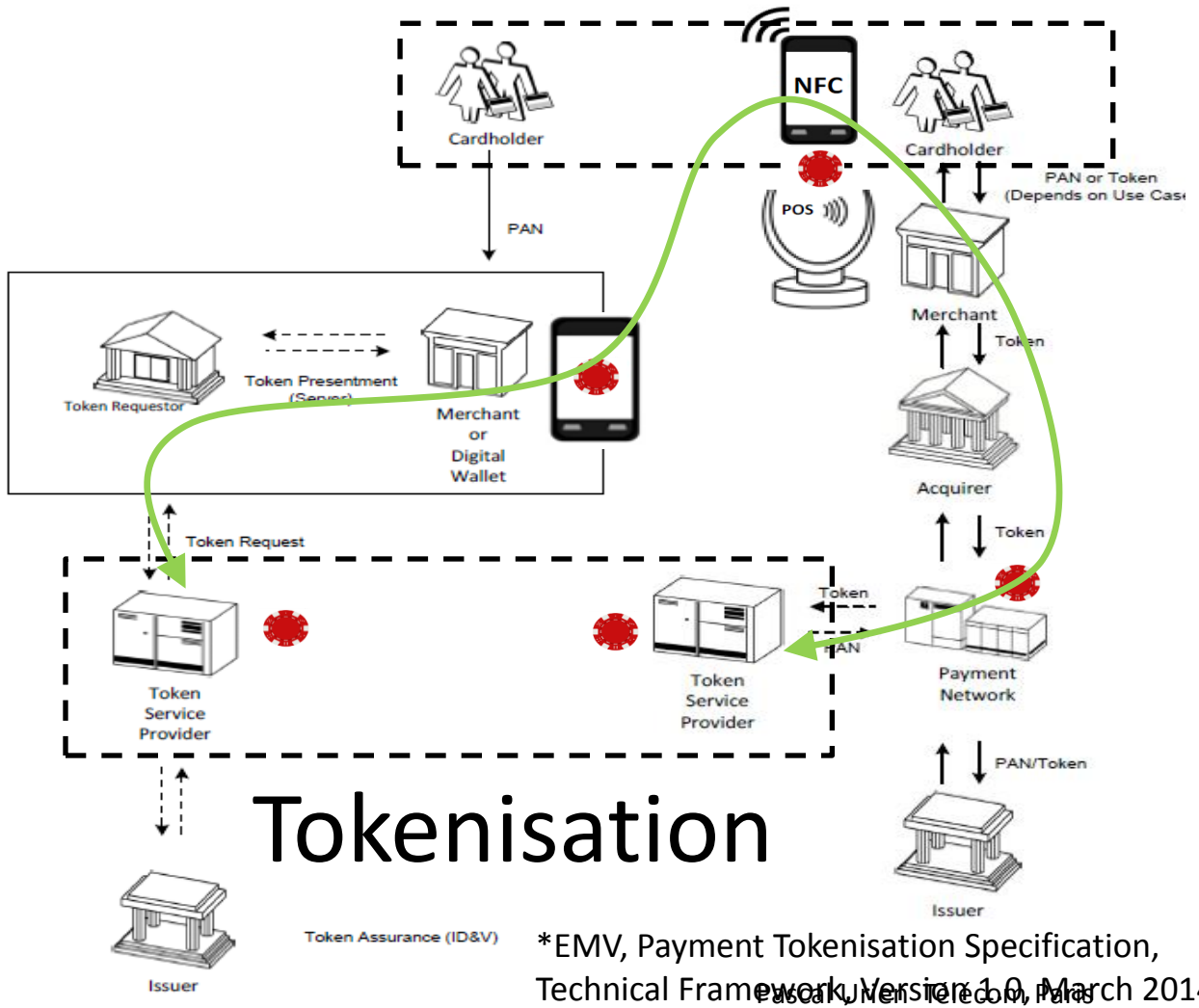


ApplePay

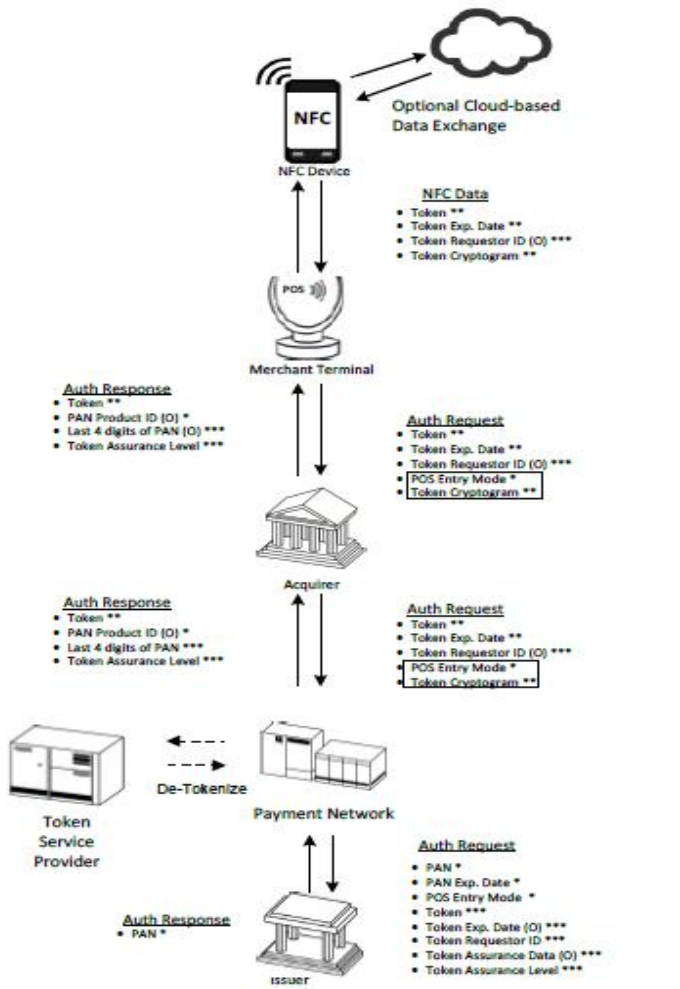
- L'iPhone 6 intègre un contrôleur NFC avec un Secure Element
  - Mode NFC Card Emulation uniquement
- Le service ApplePay est basé sur les standards de tokenisation de VISA, MasterCard, AmericanExpress
  - Les application de paiement sont provisionnées
  - ApplePay est un *Token-Requester*

ApplePay


Merchant System
Present Apple Pay
Postal Code
Customer & Payment Information
Decrypt
Add to Cart
$20
Cart
Pay
Payment Sheet
Order Confirmed
$20
Payment Provider
Pay with Touch ID
iOS
Apple Pay Eligible?
Valid Touch ID
SE
Create Cryptogram Encrypt Customer Information



Tokenisation



*EMV, Payment Tokenisation Specification, Technical Framework, Version 1.0, March 2014