Security issues for the IoT dealing with Mobile Payments and Secure Element for Objects

Pascal.Urien@Telecom-ParisTech.fr

CoFounder of the Ethertrust Company

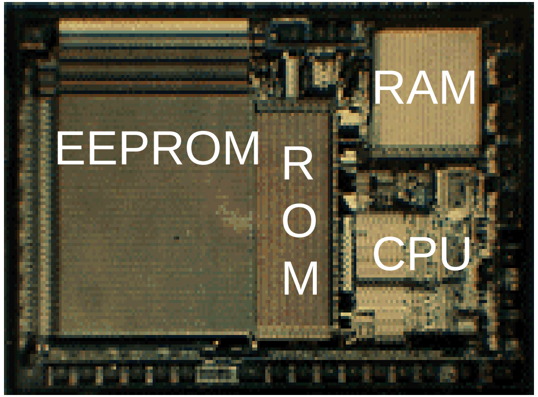
Venezia, Italy, October 9th 2016

Part One: Mobile Payments

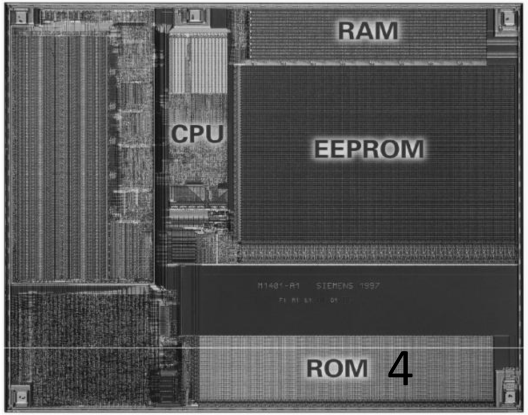
Background

Smartcard Genesis

- 1980, First BO’ French bank card, from CP8
- 1988, SIM card specification
- 1990, First ISO7816 standards
- 1991, First SIM devices
- 1995, First EMV standards
- 1997, First Javacard
  - The javacard is a subset of the java language
  - Patent US 6,308,317
- 1998, JCOP (IBM JC/OP)
- 1999, Global Platform (GP)
- 2002, First USIM cards



1988, the 21 (BO’) chip



Siemens (SIM) chip, 1997

What is a Secure Element ?

A Secure Element (SE) is a Secure Microcontroller, equipped with host interfaces such as ISO7816, SPI or I^{2}C .

OS JAVACARD JCOP

GP (Global Platform)

ROM 160 KB

EEPROM 72 KB

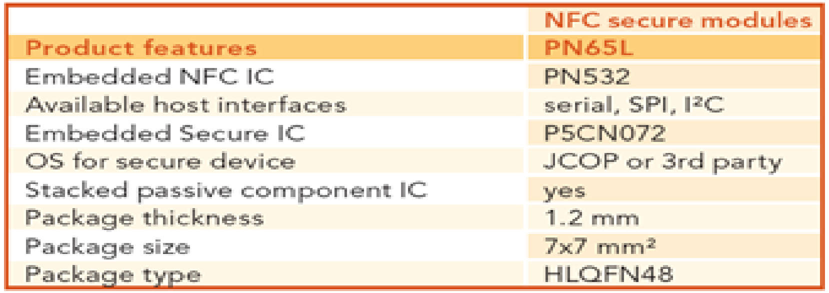
RAM 4KB

Crypto-processor

3xDES, AES, RSA, ECC

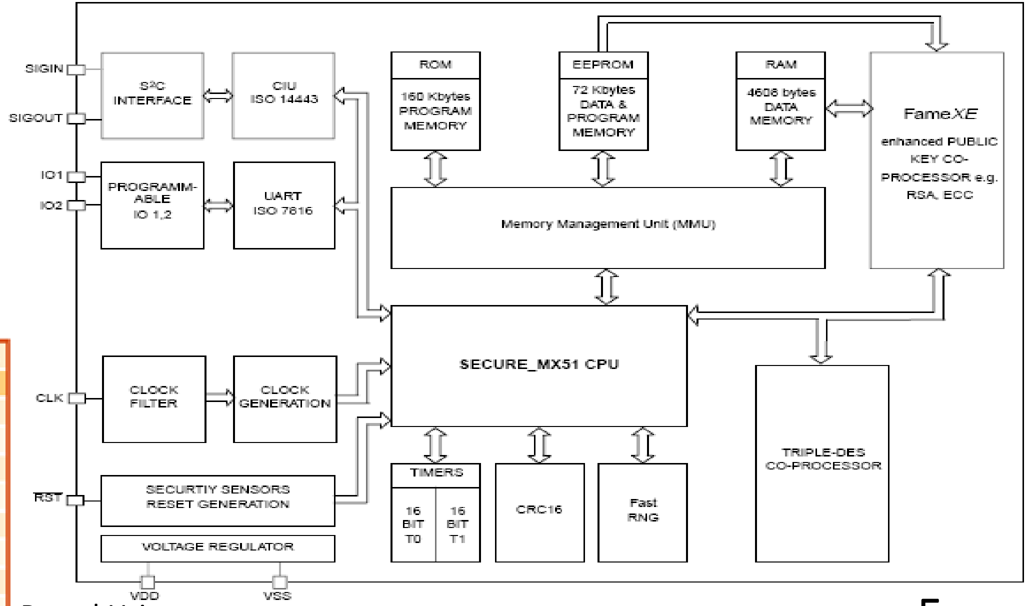
Certification CC (Common Criteria) EAL5+

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

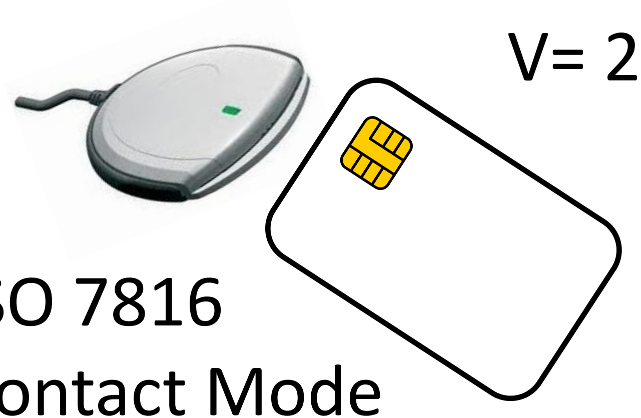


NFC Genesis

- 1994, Mifare 1K
  - In 2011 Mifare chips represent 70% of the transport market.
- 2001, ISO 14443 Standards (13,56 MHz)
  - Type A (Mifare)
  - Type B
  - Type F (Felica)
- 2004, NFC Forum
  - Mifare (NXP), ISO14443A, ISO14443B, Felica (Sony)
  - Three functional modes :
    - Reader/Writer, Card Emulation, Peer to Peer
- NFC controllers realize NFC modes

From ISO 7816 to ISO 14443

- The basic idea of Wi-Fi design was Wireless Ethernet.
- The basic idea of ISO 14443 design was Wireless (ISO 7816) Smartcard.
  - **Contrary to IEEE 802.11 there is no security features at the radio frame level.**



ISO 7816

Contact Mode

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

H = 5 A/m

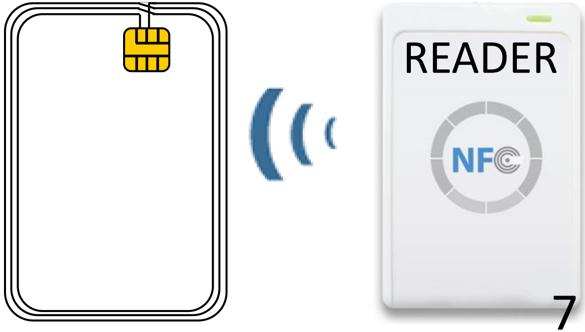
fc= 13,56 MHz

$S = 40.10^{-4}$

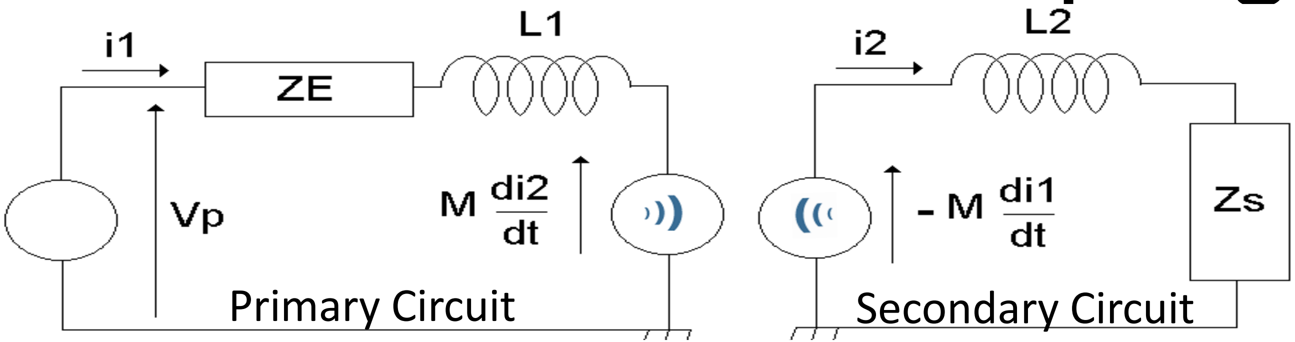
V= 2,2V

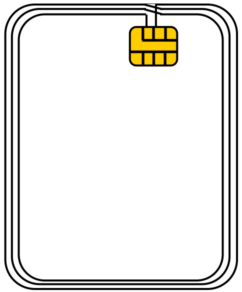
ISO 14443

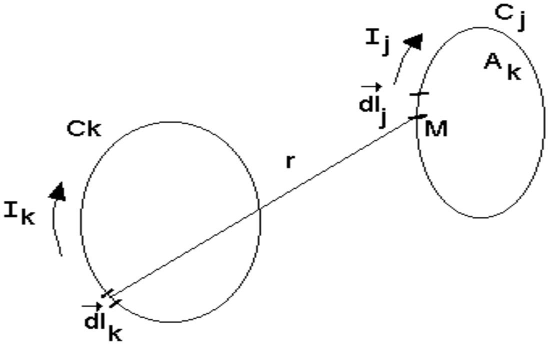
Contactless Mode



About Inductive Coupling







Neumann formula

$$\boxed{\phi_{jk} = M_{jk}\, I_k = \frac{\mu_0\, I_k}{4\pi} \oint_{C_j} \vec{dl}_j \oint_{C_k} \vec{dl}_k}$$

$$\boxed{M_{jk} = M_{kj}}$$

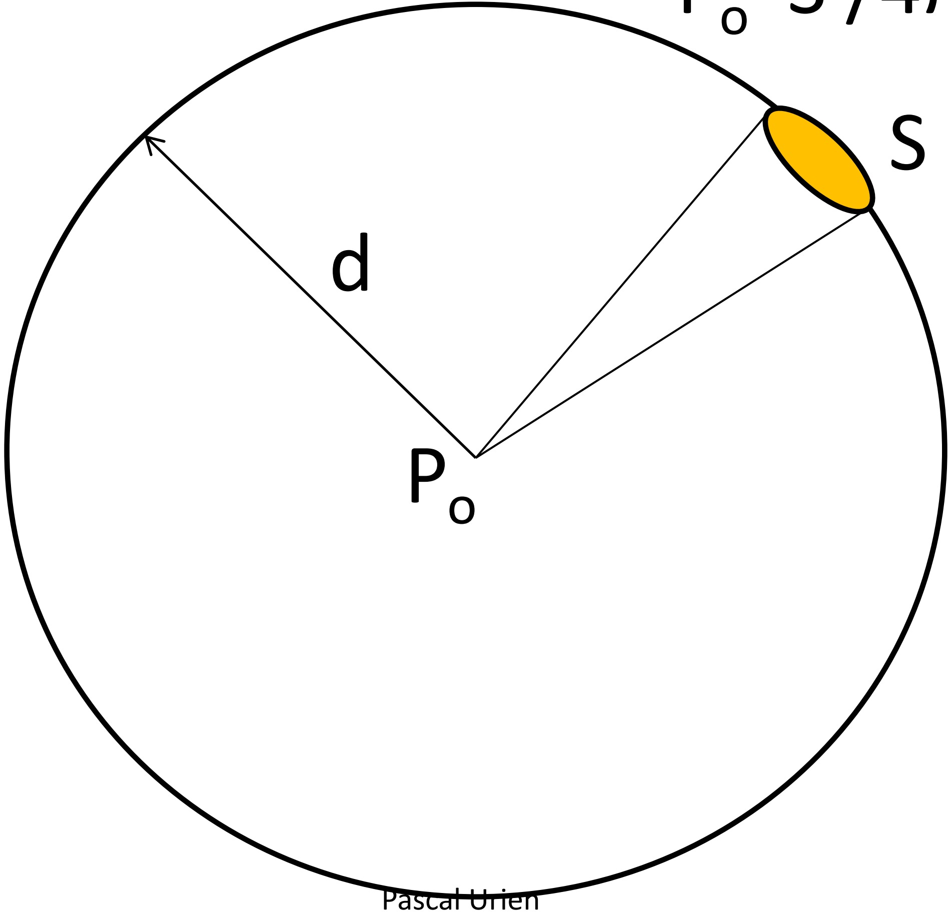
$$\begin{vmatrix} \Phi1 \\ \\ \Phi2 \end{vmatrix} = \begin{vmatrix} L1 & M \\ \\ L2 & M \end{vmatrix} \begin{vmatrix} i1 \\ \\ i2 \end{vmatrix}$$

No energy propagation, implies vicinity proof.

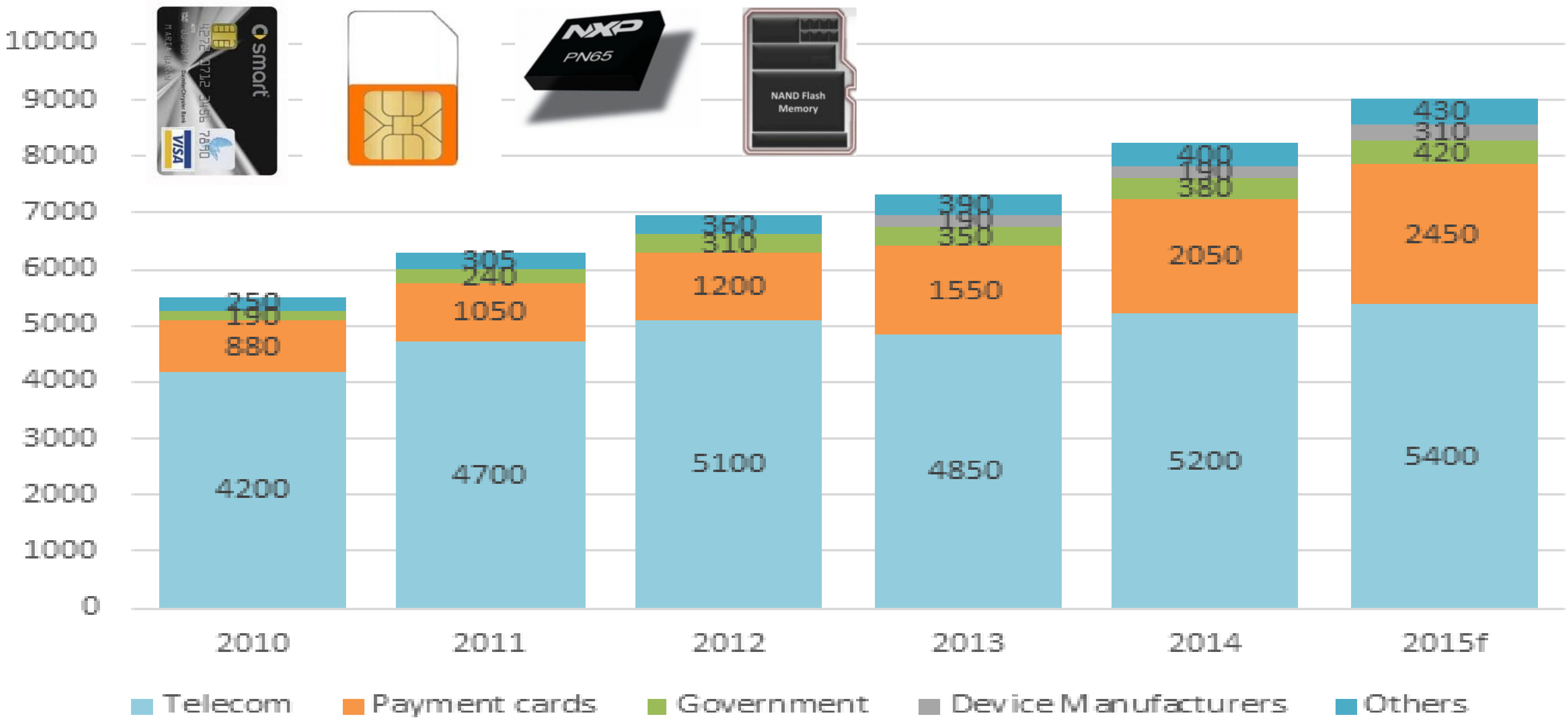
The Energy is conservative, i.e.
The Energy delivered by the primary circuit P_{PRI} = i1 . (M ω i2),
is equal to the energy consumed by the secondary circuit P_{SEC} = i2 . (M ω i1).

Propagation

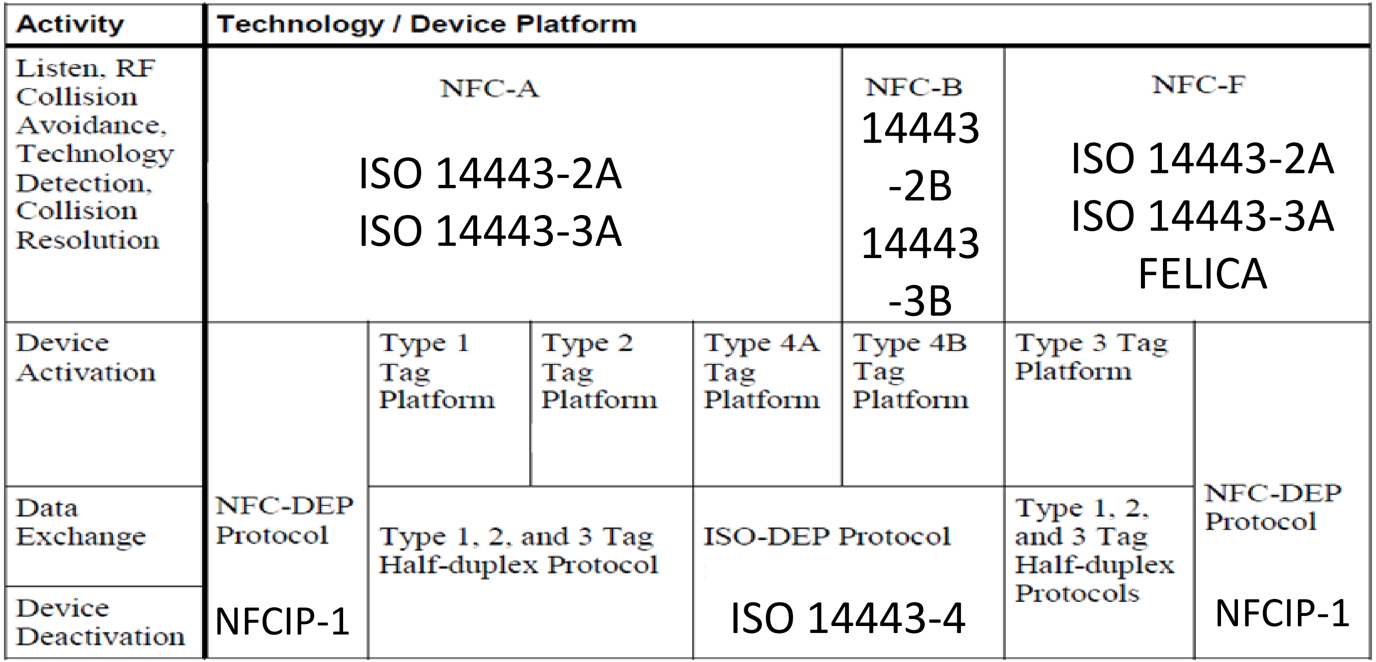
$P_o\ S / 4\pi d^2\ e^{-d/\lambda}$



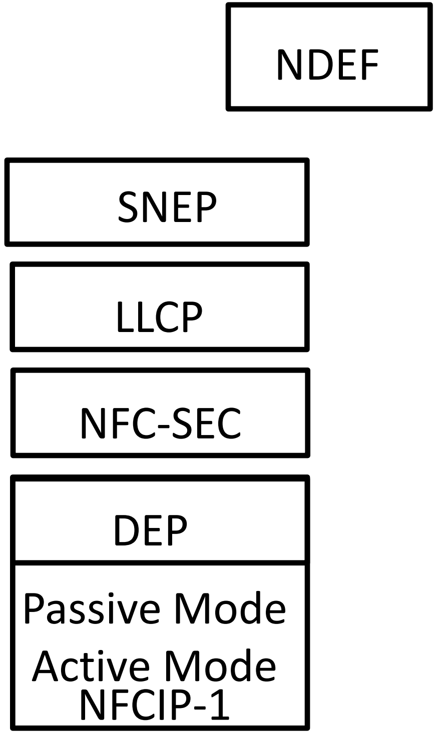
Secure Elements Market



NFC Standards Overview



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	NFC-DEP Protocol NFCIP-1	Type 1 Tag Platform	Type 2 Tag Platform	Type 4A Tag Platform	Type 4B Tag Platform	Type 3 Tag Platform	NFC-DEP Protocol NFCIP-1
Data Exchange		Type 1, 2, and 3 Tag Half-duplex Protocol		ISO-DEP Protocol ISO 14443-4		Type 1, 2, and 3 Tag Half-duplex Protocols	
Device Deactivation							



NDEF

SNEP

LLCP

NFC-SEC

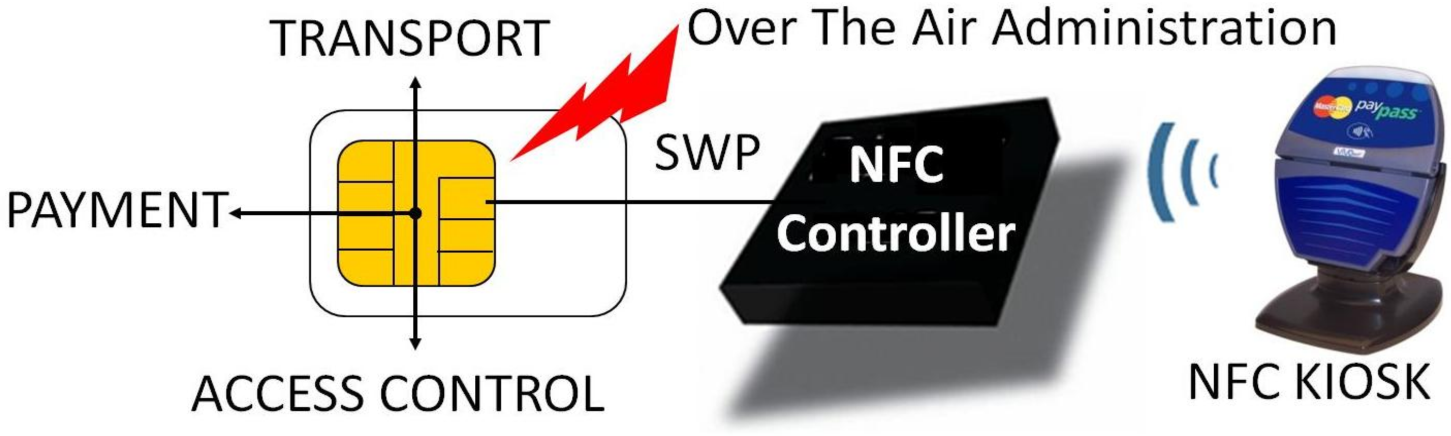
DEP

Passive Mode
Active Mode
NFCIP-1

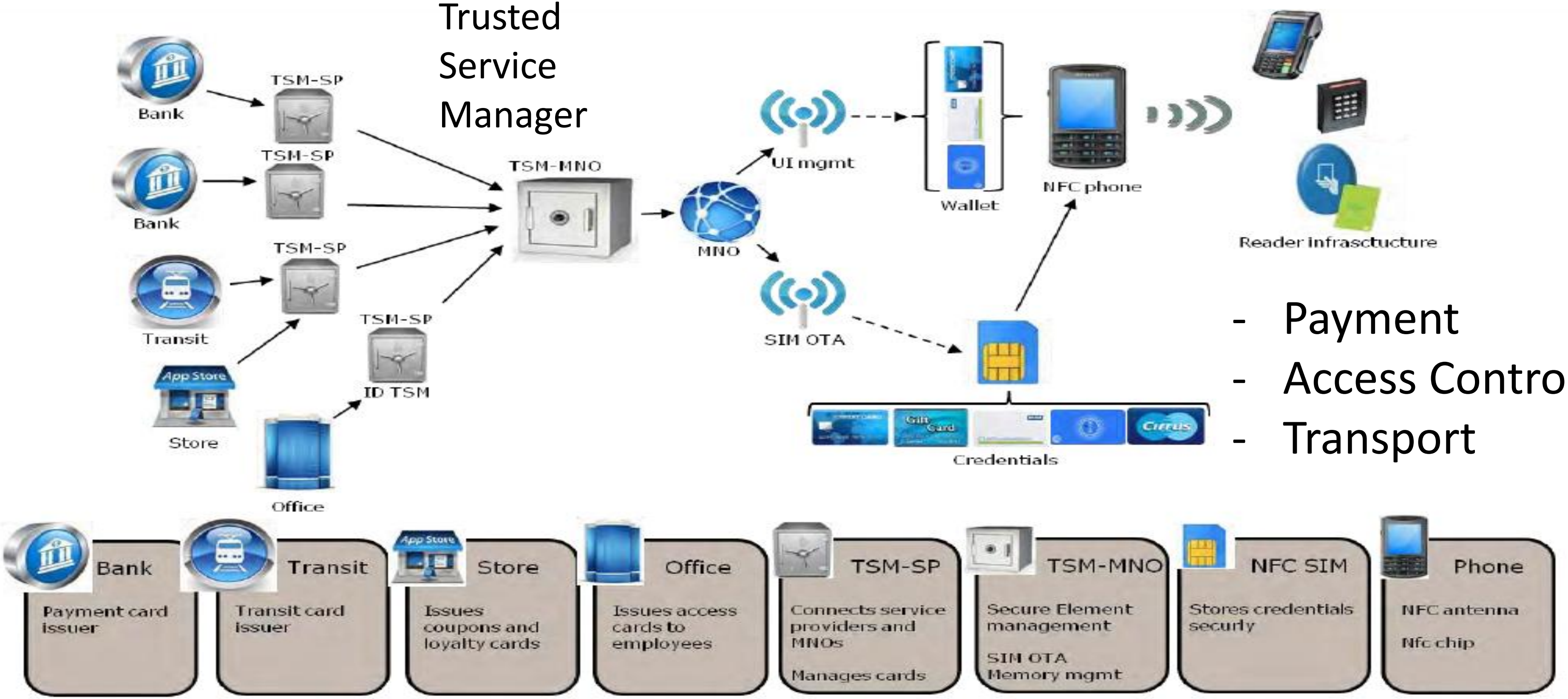
*ISO/IEC_18092 standard and NFCIP-1 standards are similar

DEP: Data Exchange Protocol

SIM-Centric Legacy Paradigm



HID NFC White Paper: SIM Centric Services



Trusted
Service
Manager

- Payment
- Access Control
- Transport

Bank: Payment card issuer

Transit: Transit card issuer

Store: Issues coupons and loyalty cards

Office: Issues access cards to employees

TSM-SP: Connects service providers and MNOs; Manages cards

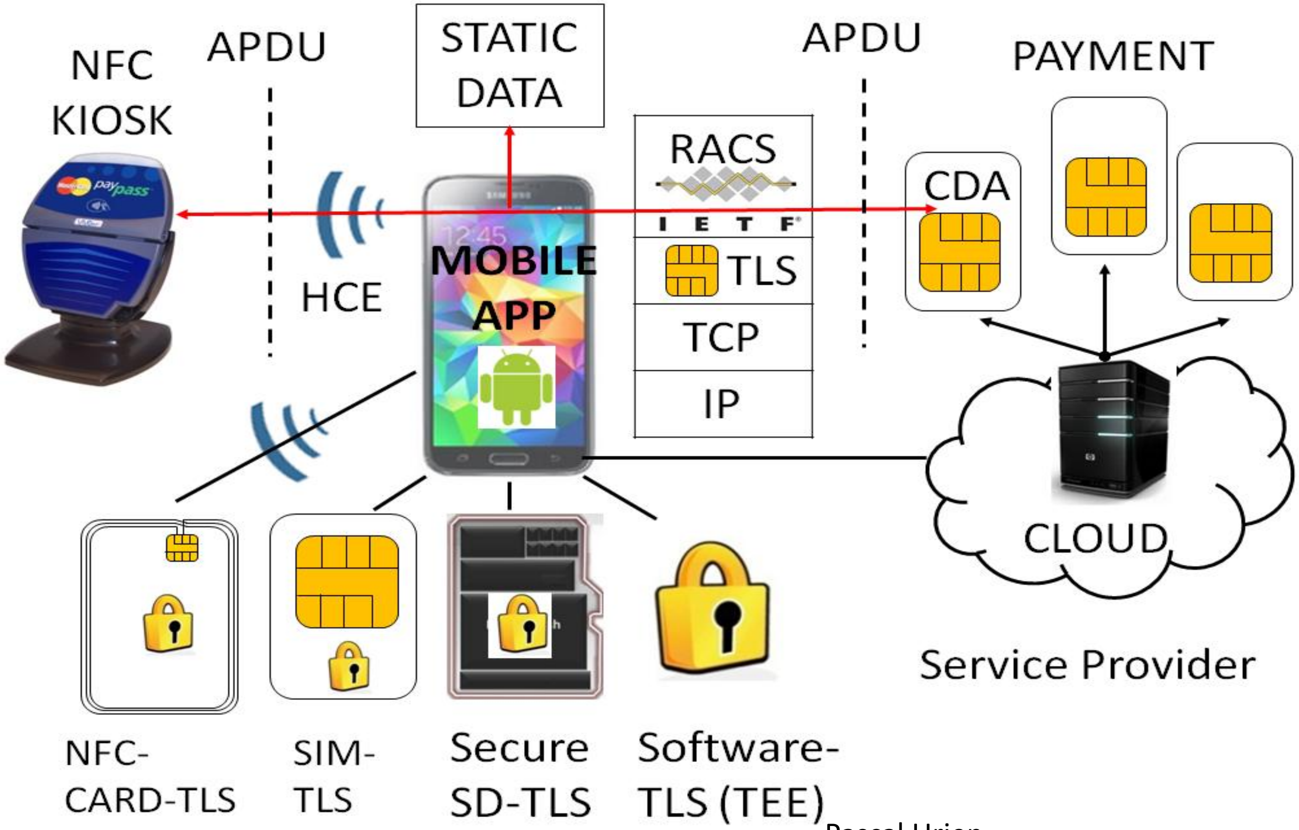
TSM-MNO: Secure Element management; SIM OTA; Memory mgmt

NFC SIM: Stores credentials securly

Phone: NFC antenna; Nfc chip

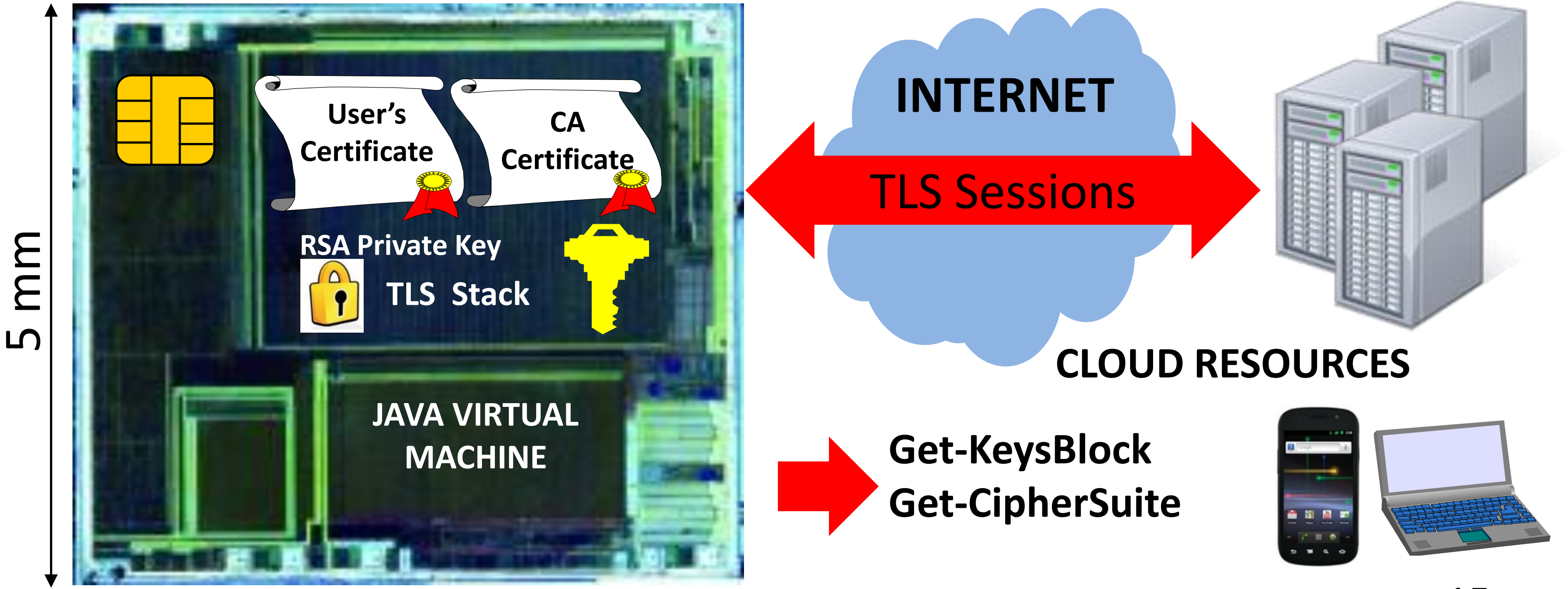
NFC ecosystem with the Secure Element in the SIM and one MNO

Cloud of Secure Elements



- Remote use of Secure Elements hosted in the cloud through secure TLS channel

About TLS Stack for Secure Element

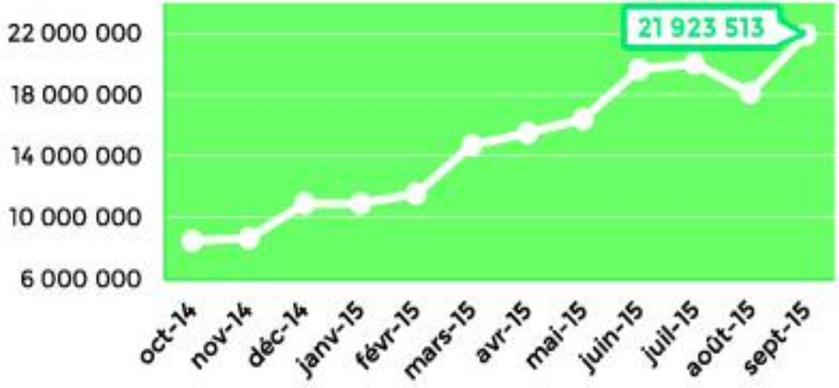


About Mobile Payments

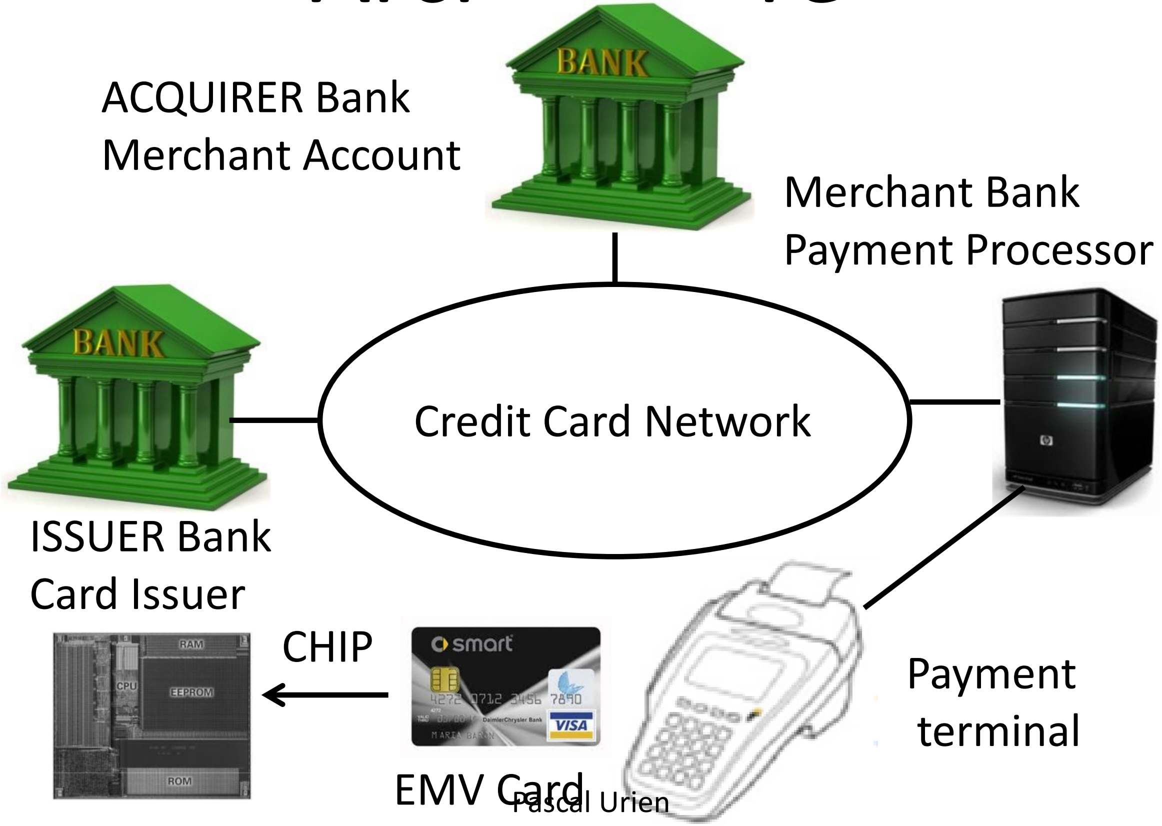
US payment cards market
2011: 21 trillion $

Some Figures

- According to the French national bank ("Banque de France"), the France gross domestic product (GDP) was about 1900 billion € in 2013.
- The global amount of financial transactions was about 27 000 billion €.
- 1,7% of these operations were performed with bank cards (leading to about 450 billion €).
- Nine billion of card transactions were performed in 2013, with an average value of 50€.
- The number of payment cards in France was about 86 million, more than the population.
- In France in 2015
  - About 0,025 billion of NFC payment operations
  - 10 € in average
  - 0,25 billion €
  - 0,05 % of payment card transactions



About the EMV Payment Four Corner Architecture



ACQUIRER Bank
Merchant Account

Merchant Bank
Payment Processor

Credit Card Network

ISSUER Bank
Card Issuer

CHIP

EMV Card

Payment
terminal

A typical EMV transaction comprises five steps

- 1) Selection of the PPSE (*Proximity Payment Systems Environment*) application, which gives the list of embedded payment EMV applications identified by their AID.
- 2) Selection of an EMV payment application.
- 3) Reading of the application capacities, thanks to the GPO (*Get Processing Options*) command, which also returned the structure of embedded information organized according to a records/files scheme.
- 4) Reading of records and files via the *ReadRecord* command. Certificates are checked and a DDA procedure may be used as non cloning proof.
- 5) Generation of payment cryptograms, triggered by *GenerateAC* or CDA commands.

Legacy EMV

- According to iso7816 standards EMV applications are identified by AID (*Application IDentifier*) attributes, 16 bytes at the most.
- An EMV application embeds an index of a certification authority (such as VISA or MasterCard), an issuer certificate signed by the CA, and an ICC (*integrated circuit card*) certificate delivered (and signed) by the issuer.
- The ICC certificate authenticates most of information stored within the EMV application (PAN, bearer's name, validity dates...), encoded according to the ASN.1 syntax.
- An ICC private RSA key is available and used for non cloning proof, thanks to a dedicated command called DDA (*Dynamic Data Authentication*), in which a 32 bits random is encrypted by the ICC private key.

Legacy EMV

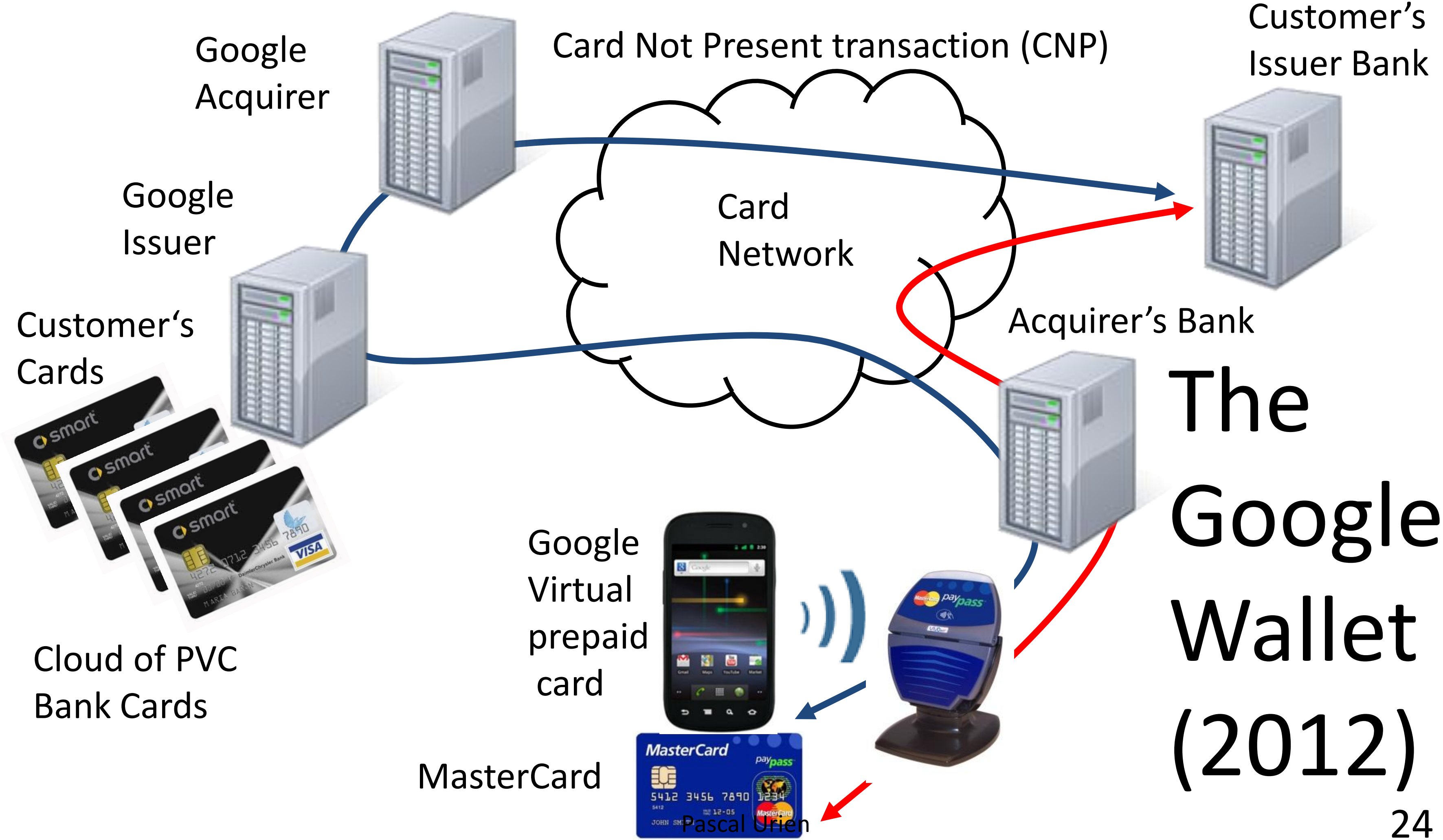
- Financial transactions are associated with cryptogam generation based on symmetric 3xDES cryptographic algorithm.
- One or two dedicated commands (*GenerateAC*) are required by a payment operation, whose input parameters include, among others, the amount and the date.
- DDA and *GenerateAC* may be combined in a single procedure called CDA (*Combined Dynamic Authentication*).

EMV ISO7816 main commands

EMV Iso7816 request	Binary (hexadecimal) Encoding CLA INS P1 P2 P3
SELECT AID	00 A4 04 00 P3=AID-length AID
GetProcessingOptions	80 A8 00 00 P3=parameters-length
ReadRecord	00 B2 P1 P2 00 P1=record number (P2-4)/8 = file number (FSI)
GenerateAC	80 AE P1 00 P3=parameters length P1= type of cryptogram

PayPass Mag Stripe (PMS)

- PMS is an adaptation of EMV standards to magnetic stripe.
- It generates a dynamic Card Validation Code (named CVC3).
- A PayPass transaction comprises the five following operations:
  - 1) Selection of the PPSE application.
  - 2) Selection of the PayPass application.
  - 3) Issuance of the GPO command.
  - 4) Reading of the record one, file one, which contains the track1 and track 2 equivalent data
  - 5) Issuance of the Compute Cryptographic Checksum (CCC) iso7816 request, including an unpredictable number. The PayPass application returns the CVC3 value.
- Contrary to EMV the PMS profile does not embed certificates or RSA private key. Thanks to CVC3 it is compatible with legacy magnetic card networks.



Google Acquirer

Card Not Present transaction (CNP)

Customer's Issuer Bank

Google Issuer

Card Network

Customer's Cards

Acquirer's Bank

Cloud of PVC Bank Cards

Google Virtual prepaid card

MasterCard

The Google Wallet 2 (2012)

Google PrePaid Card Transaction

```
// SELECT 2PAY.SYS.DDF01
>> 00A404000E325041592E5359532E4444463031
<< 6F2C840E325041592E5359532E4444463031A51ABF0C1761154F10A000000004
1010AA54303200FF01FFFF8701019000
```

```
6F File Control Information (FCI) Template
    84 Dedicated File (DF) Name
          325041592E5359532E4444463031
    A5 File Control Information (FCI) Proprietary Template
          BF0C File Control Information (FCI) Issuer Discretionary Data
                    61 Application Template
                              4F Application Identifier (AID) – card
                                        A0000000041010AA54303200FF01FFFF
                              87 Application Priority Indicator
                                        01
```

Google PrePaid Card Transaction

```
// Select MasterCard Google Prepaid Card
>> 00A4040010A0000000041010AA54303200FF01FFFF
<<
    6F208410A0000000041010AA54303200FF01FFFFA50C500A4D617374657243617
    2649000

6F File Control Information (FCI) Template
    84 Dedicated File (DF) Name
        A0000000041010AA54303200FF01FFFF
    A5 File Control Information (FCI) Proprietary Template
        50 Application Label
                MasterCard
```

Google PrePaid Card Transaction

// Get Processing Options
>> 80A8000002830
<< 770A 8202 **0000** 9404 **08010100** 9 000
AIP=0000 AFI= 08010100

// Reader Record one File one
>> 00B2010C00
<< 706A9F6C0200019F62060000000000389F63060000000003C64 **5629** 235343330
393939393039393739393939395E202F5E3137313131303130303130303030303030
303030309F6401049F650200389F660203C6 **9F6B13** 5430 **999909979999** D **1711** 1
01 0010000000000F 9F670104 9000

Track 1 data

Track 2 data

PAN= **999909979999**

Validity Date= **1711**

// COMPUTE Cryptographic Checksum (CVC3)
>> 802A8E8004 00000080
<< 770F**9F6102** **0038** **9F6002** **0038** **9F3602** **0012** 9000

CVC3 Track 2   CVC3 Track 1   ATC

Pascal Urien

VISA MSD

- VISA VCPS (*Visa Contactless Payment Specification*) MSD (Magnetic Stripe Data), is an adaptation of EMV standards to magnetic stripe for contactless payments.
- It generates a dynamic Card Verification Value (dCVV, a three digits code) based on a 3xDES (112 bits) secret key.
- A VISA MSD transaction comprises the four following operations:
  - 1) Selection of the PPSE application.
  - 2) Selection of the VISA MSD application.
  - 3) Sending of the GPO command with payment attributes (amount, date...).
  - 4) Reading of the record one, file one, which contains the track 2 equivalent data. This file includes a dCVV computed after the previous GPO.
- Contrary to EMV the VISA MSD profile does not embedded certificate or RSA private key. Thanks to dCVV it is compatible with legacy magnetic card networks.

Apple Pay

```
Select PPSE
00A404000E  325041592E5359532E4444463031
6F 23 […] 9000
6F File Control Information (FCI) Template
      84 Dedicated File (DF) Name
                325041592E5359532E4444463031
      A5 File Control Information (FCI) Proprietary Template
                BF0C File Control Information (FCI) Issuer Discretionary Data
                                61 Application Template
                                                4F Application Identifier (AID) – card
                                                                A0000000031010
                                                87 Application Priority Indicator
                                                                01
```

Apple Pay

Select VISA MSD

00A4040007 **A0000000031010**

6F 39 […] 9000

```
6F File Control Information (FCI) Template
      84 Dedicated File (DF) Name
                A0000000031010
      A5 File Control Information (FCI) Proprietary Template
                9F38 Processing Options Data Object List (PDOL)
       9F6604 9F0206 9F0306 9F1A02 9505 5F2A02 9A03 9C01 9F3704 9F4E14
                BF0C File Control Information (FCI) Issuer Discretionary Data
                                9F4D Log Entry
                                                1401
                                9F5A Unknown tag
                                                1108400840
```

Apple Pay

GPO

```
80 A8 00 00 37  83 35 [...]
83 35
          86 00 00 80
          00 00 00 00 00 01
          00 00 00 00 00 00
          04 80
          00 00 00 00 00
          04 80
          14 08 18
          01
          4A 94 57 1A
          00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

80 06 00 80  08 01 01 00  9000
```

AIP = 0080 , MSD mode

AFL = 08010100,   one record one file

Apple Pay

```
//Read Record 1 file 1
00 B2 01 0C  00

701A  57 13  40 71 23 13 11 22 33 44 D2 00 32 01 00 00 05 09  00 02 5F 5F 20
      02 20 2F 90 00
```

70 EMV Proprietary Template

DAN

dCVV

57 Track 2 Equivalent Data

40712313112234 4D 2003 201 0 0000 **509** 00025 F

5F20 Cardholder Name

/

EMV

```
//Select PPSE
00A404000E325041592E5359532E444446303100

6F23840E325041592E5359532E4444463031A511BF0C0E610C4F07A00000000
410108701019000

6F File Control Information (FCI) Template
	84 Dedicated File (DF) Name
		325041592E5359532E4444463031
	A5 File Control Information (FCI) Proprietary Template
		BF0C File Control Information (FCI) Issuer Discretionary Data
			61 Application Template
				4F Application Identifier (AID) – card
					A0000000041010
				87 Application Priority Indicator
					01
```

EMV

```
// Select Master Card
>> 00A4040007A000000004101000
<< 6F388407A0000000041010A52D500A4D6173746572436172648701015F2D0266
729F1101019F120A4D617374657263617264BF0C059F4D020B0A9000
```

EMV

```
6F File Control Information (FCI) Template
        84 Dedicated File (DF) Name
                A0000000041010
        A5 File Control Information (FCI) Proprietary Template
                50 Application Label
                        M a s t e r C a r d
                87 Application Priority Indicator
                        01
                5F2D Language Preference
                        f r
                9F11 Issuer Code Table Index
                        01
                9F12 Application Preferred Name
                        M a s t e r c a r d
                BF0C File Control Information (FCI) Issuer Discretionary Data
                        9F4D Log Entry
                                0B0A
```

EMV

GPO

```
80A8000002830000
7716 8202 1980 9410 08010100100101011801020020010100 9000
```

77 Response Message Template Format 2

- 82 Application Interchange Profile
- 94 Application File Locator (AFL)

82 (AIP - Application Interchange Profile)

- 1000 (Byte 1 Bit 5) Cardholder verification is supported
- 0800 (Byte 1 Bit 4) Terminal risk management is to be performed
- 0100 (Byte 1 Bit 1) CDA supported
- 0080 (Byte 2 Bit 8) EMV and Magstripe Modes Supported

94 (AFL - Application File Locator)

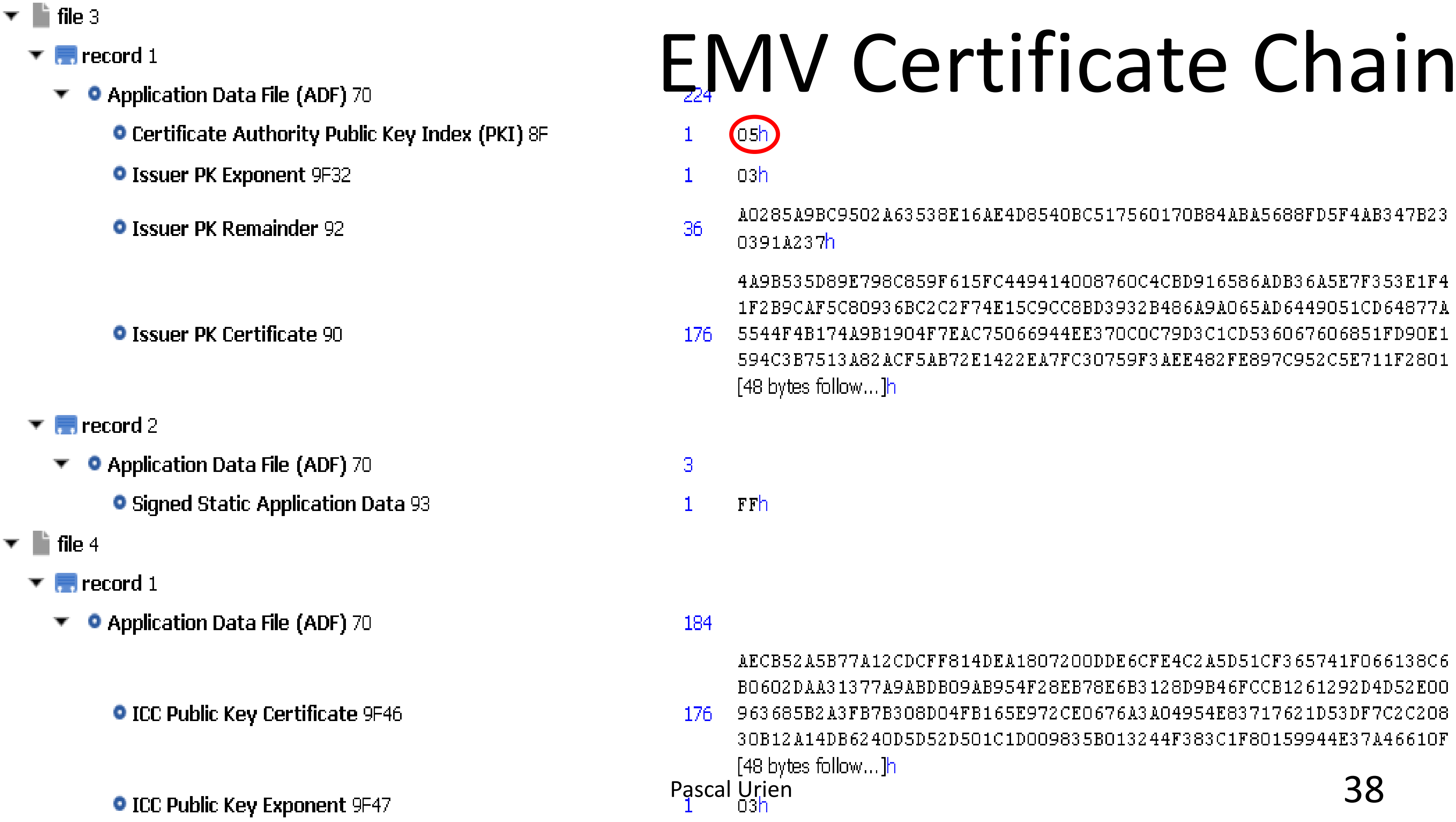
List of records that should be read by the terminal.
Each record is identified by the pair (SFI - short file indicator, record number)
SFI 1 record 1, SFI 2 record 1, SFI 3 records 1-2, SFI 4 record 1

EMV Records and Files Reading

P1=record number, (P2-4)/8 = file number (SFI)

```
// read record 1, file 1
00 B2 01 0C    00

// read record 1, file 2
00 B2 01 14    00

// read record 1, file 3
00 B2 01 1C    00

// Read record 2, file 1
00 B2 02 1C    00

// Read record 1 file 4
00 B2 01 24    00
```

EMV Certificate Chain



```
▼ file 3
  ▼ record 1
    ▼ Application Data File (ADF) 70                      224
        Certificate Authority Public Key Index (PKI) 8F   1     05h
        Issuer PK Exponent 9F32                           1     03h
        Issuer PK Remainder 92                            36    A0285A9BC9502A63538E16AE4D8540BC51756017OB84ABA5688FD5F4AB347B23
                                                                0391A237h
                                                                4A9B535D89E798C859F615FC449414008760C4CBD916586ADB36A5E7F353E1F4
                                                                1F2B9CAF5C80936BC2C2F74E15C9CC8BD3932B486A9A065AD6449051CD64877A
        Issuer PK Certificate 90                          176   5544F4B174A9B1904F7EAC75066944EE370C0C79D3C1CD5360676O6851FD90E1
                                                                594C3B7513A82ACF5AB72E1422EA7FC30759F3AEE482FE897C952C5E711F2801
                                                                [48 bytes follow...]h
  ▼ record 2
    ▼ Application Data File (ADF) 70                      3
        Signed Static Application Data 93                 1     FFh
▼ file 4
  ▼ record 1
    ▼ Application Data File (ADF) 70                      184
                                                                AECB52A5B77A12CDCFF814DEA180720ODDE6CFE4C2A5D51CF365741F066138C6
                                                                B0602DAA31377A9ABDB09AB954F28EB78E6B3128D9B46FCCB1261292D4D52E00
        ICC Public Key Certificate 9F46                   176   963685B2A3FB7B308D04FB165E972CE0676A3A04954E83717621D53DF7C2C208
                                                                30B12A14DB6240D5D52D501C1D009835B013244F383C1F80159944E37A46610F
                                                                [48 bytes follow...]h
        ICC Public Key Exponent 9F47                      1     03h
```

```
// P1= Generate TC (01xx) + CDA signature Request (xxx1)= 50
80AE50002B 000000006290 000000000000 250 0000000000 0978 150610 00 90B4
E0D2 25 0000 0000000000000000 1F0302 00
```

EMV

000000000690	Amount
000000000000	Cashback
250	Country Code
0000000000	Terminal Verfication Result
0978	Currency code
150610	Transaction Date
00	Transaction Type
90B4E0D2	Unpredictable Number
25	Terminal type
0000	Data Authentication Code
0000000000000000	ICC Dynamic Number
1F0302	Cardholder Verification Method
00	LE

```
//CDOL1 tag 8C
9F0206 9F0306 9F1A02 9505 5F2A02 9A03 9C01 9F3704 9F3501 9F4502 9F4C08 9F3403
```

EMV CDA

```
7781 91

9F27 01 80                    Cryptogram Information Data
9F36 02 001F                  Application Transaction Counter
9F4B 70                       Signed Dynamic Application Data
   9E92DE44738A7C5533D5E29A7A6D230A
   0E2123F3EE1DCD83C868551D4F01C1D2
   4979BBAA978F95589731C1CA73DA77DD
   80E3B49D7B0CEA3B4CFE711D021DA8F9
   4BE408C44EF614EB5F150FDDFE6DA8C8
   920E041F8401E3DE0D313EB15DC7C6C9
   DCD0279F4EF450D39F8CA12361065124

9F10 12                       Issuer Application Data (optionnal)
  0F10
   A04003223000000000000000000000FF

9000
```

EMV CDA

6a

05	Signed Data Format
01	Hash Algorithm Indicator
26	ICC Dynamic Data Length (LDD)
08	ICC Dynamic Number Length
a1 bb 29 ce d6 89 95 7c	ICC Dynamic Number
80	Cryptogram Information Data
ec e9 3c d4 a0 80 34 c8	**TC**

09 a1 86 bd eb 56 60 ba 15 b2 b2 8d 9f 1c b2 e4 74 a6 8d 8c
Transaction Data Hash Code

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb Padding

154fb6d9d774f5e6f7eef512b557eaf754c9c8f3bc Signature

EMV CDA

```
signature= 154fb6d9d774f5e6f7eef512b557eaf754c9c8f3bc =sha1
 {       05012608a1bb29ced689957c80
      ||  ece93cd4a08034c8
      ||  09a186bdeb5660ba15b2b28d9f1cb2e474a68d8c
      ||  49 x bb (49 = 0x70 - 0x26 - 25)
      ||  90B4E0D2 } // Unpredictable Number
```

Transaction Data Hash code = hash of

-The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command

- The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal in the first GENERATE AC command.

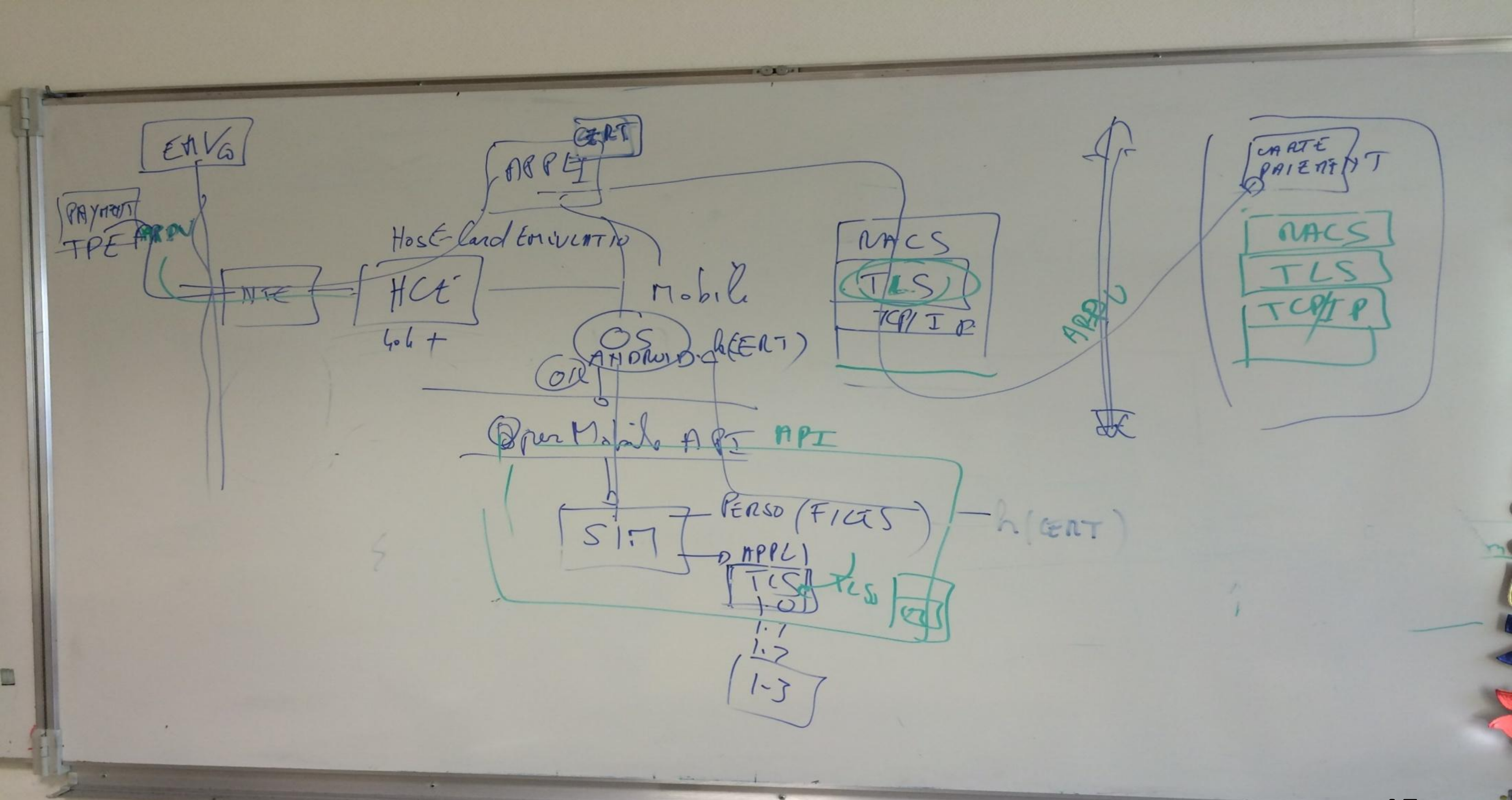
- The tags, lengths, and values of the data elements returned by the ICC in the response to the GENERATE AC command in the order they are returned, with the exception of the Signed Dynamic Application Data.

EMV CDA

Transaction Data Hash code= sha1 (
0000000006290000000000000250000000000000978150610009 0B4E0D2220000000
00000000000001F0302

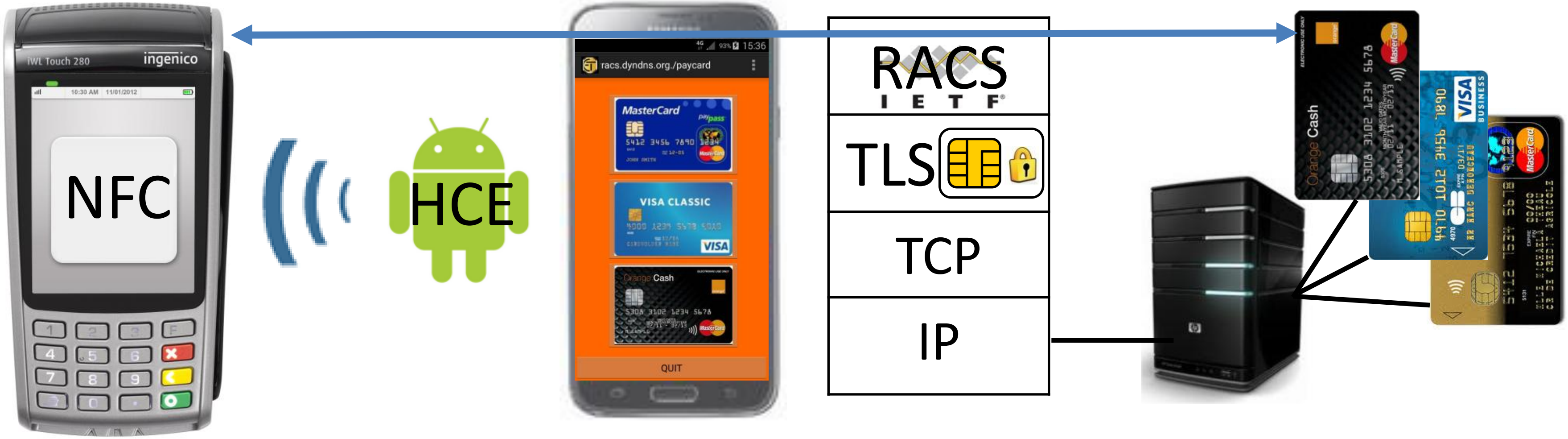
```
            ||  9F27 01 80
            ||  9F36 02 001F
            ||  9F10 12 0F10A04003223000000000000000000000FF )
     = 09 a1 86 bd eb 56 60 ba 15 b2 b2 8d 9f 1c b2 e4 74 a6 8d 8c
```

The SIMulation Project


EMVCo
CERT
APPLI
PAYMENT
TPE
APDU
Host Card Emulation
NFC
HCE
4G +
Mobile
OS
ANDROID
OR
h(CERT)
Open Mobile API
API
SIM
PERSO (FIGS)
h(CERT)
APPLI
TLS
1.0
1.1
1.2
1.3
TLS
MACS
TLS
TCP/IP
APDU
CARTE PAIEMENT
MACS
TLS
TCP/IP

Scope

- Remote use of Secure Elements hosted in the Cloud through secure TLS channel



NFC

HCE

RACS

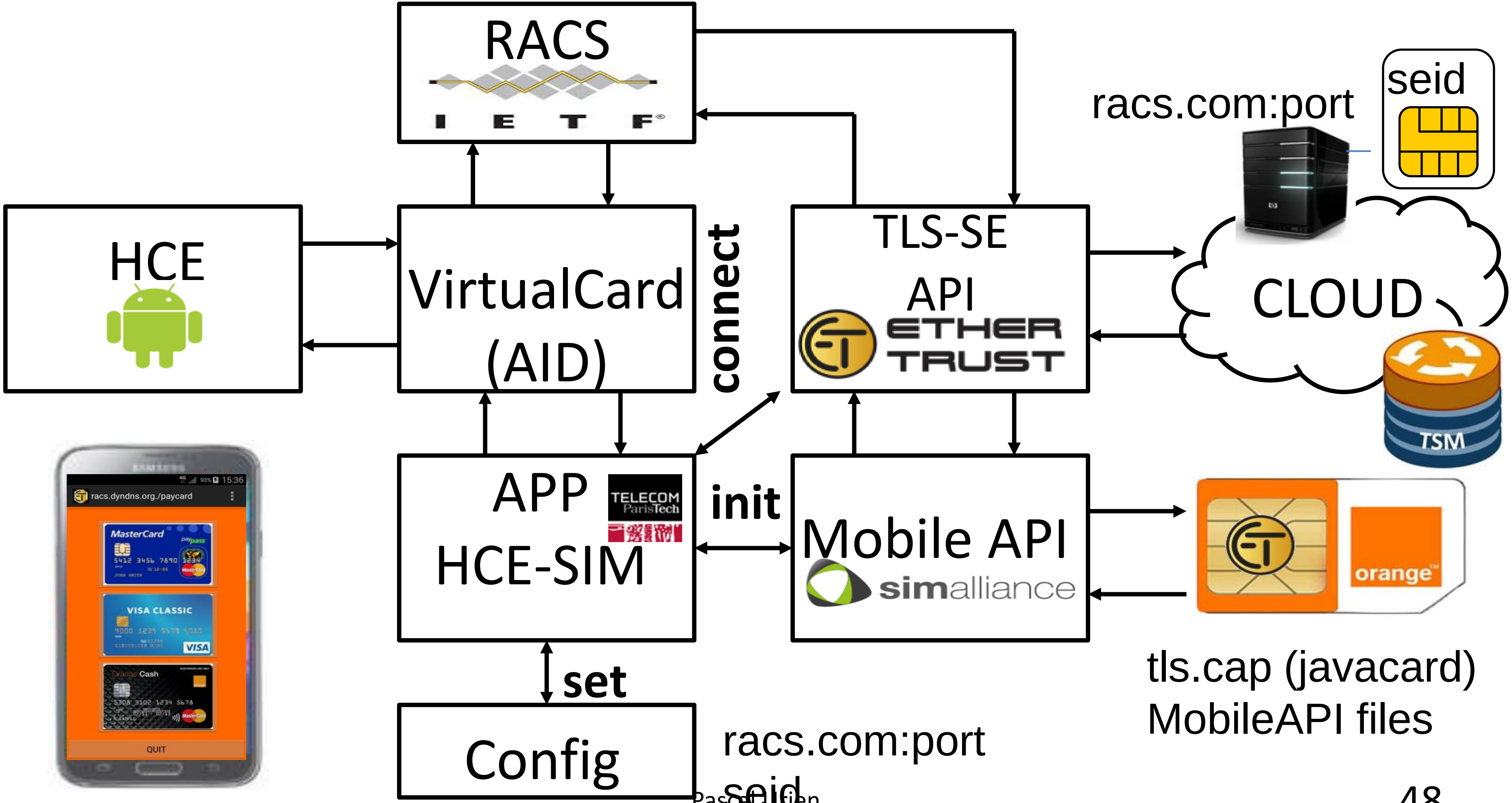
TLS

TCP

IP

Architecture

- Four Components
  - Legacy payment terminal
  - Android Mobile
    - Host Card Emulation
    - Mobile API for SIM interface
  - SIM card
    - Delivering a TLS stack
  - Payment servers
    - Built over RACS server and legacy payment card



RACS

IETF

HCE

VirtualCard (AID)

connect

TLS-SE API

ETHER TRUST

CLOUD

racs.com:port

seid

TSM

APP HCE-SIM

TELECOM ParisTech

init

Mobile API

simalliance

orange

set

Config

racs.com:port
seid

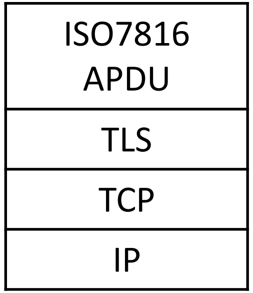
tls.cap (javacard)
MobileAPI files

RACS

- The idea is to put secure elements in the Cloud
- RACS works over TLS
- Splitting between Access Control (authorization) and Services
  - The risks (Fraud…) are managed in two separate plans (access and service)
  - Remote resources are monitored
- Giving an identifier to a secure element in the cloud
  - A WEB of Secure Elements
  - RACS://Server.com:Port/SEID

Introducing RACS

- The RACS protocol is in the perspective of these former experiments.
- It has been designed for efficient and secure remote use of secure elements via the internet.
- It also provides smartcard readers virtualization, and therefore facilitates secure elements deployment in environments dealing with virtual machines and cloud computing.



ISO7816 APDU

TLS

TCP

IP

RACS Uniform Resource Identifier

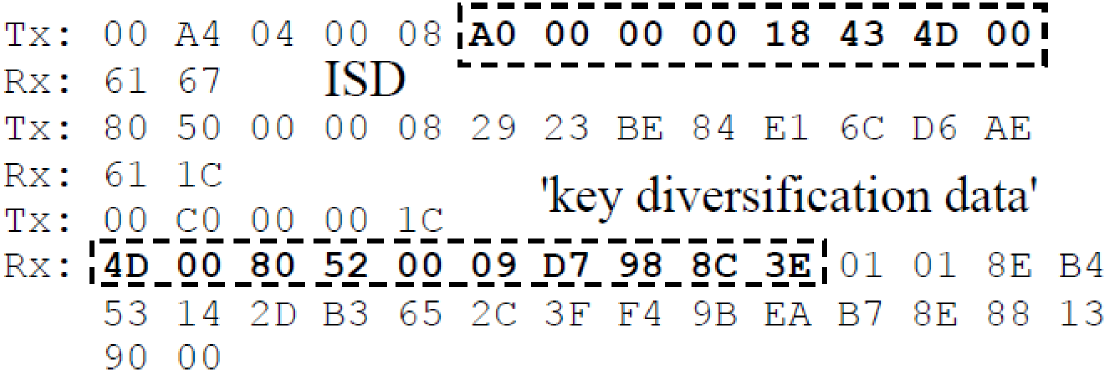
- RACS setups a secure (TLS) connection with a remote server, it collects the list of hosted secure elements, and thereafter it powers on a secure element, resets the device and exchanges APDU requests and responses.
- RACS defines an URI (*Uniform Resource Identifier*), such as
    - **ServerName:Port/SEID**
- It comprises the server name or IP address, the TCP port and a SE identifier (the SEID).
- Therefore it creates a concept somewhat similar to a WEB of secure elements, or a WEB of cryptographic procedures.

A PKI Infrastructure

- In order to perform strong mutual authentication both RACS client and server are equipped with X509 certificates, dealing with asymmetric cryptography (RSA or elliptic curves).
- The client SUBJECT attribute, more precisely the *Common Name* (CN) field of this attribute identifies a legitimate client, and is associated within the server to an index UID, the user identifier.

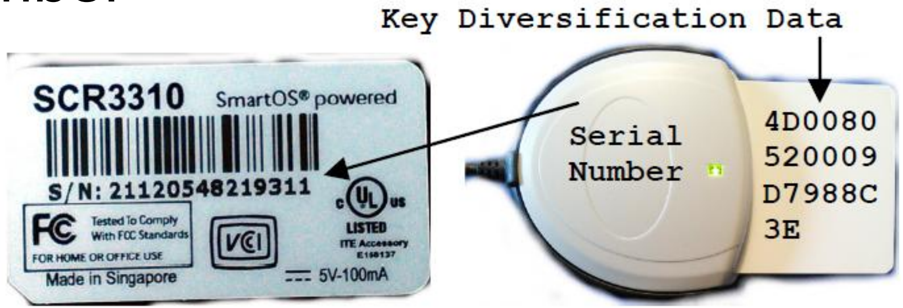
How to allocate SEID

Key Diversification Data



```
Tx: 00 A4 04 00 08 A0 00 00 00 18 43 4D 00
Rx: 61 67       ISD
Tx: 80 50 00 00 08 29 23 BE 84 E1 6C D6 AE
Rx: 61 1C
Tx: 00 C0 00 00 1C          'key diversification data'
Rx: 4D 00 80 52 00 09 D7 98 8C 3E 01 01 8E B4
    53 14 2D B3 65 2C 3F F4 9B EA B7 8E 88 13
    90 00
```

Reader Serial Number



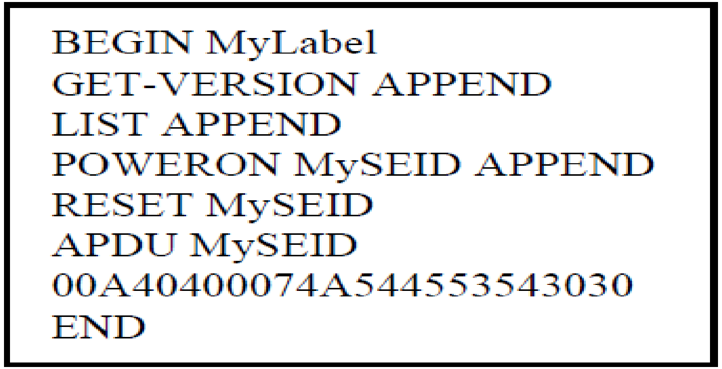
SIM-Server SlotID

725

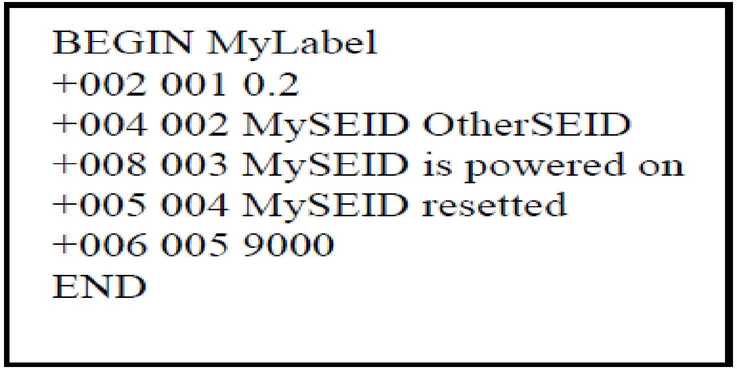
RACS Commands

Command	SEID	Comment
BEGIN	no	First request command
END	no	Last Request command
GET-VERSION	no	Return current version
SET-VERSION	no	Set current version
ECHO	no	Request the server to perform an echo
LIST	no	Return the list of authorized secure elements
SHUTDOWN	yes	Shutdown a secure element
POWERON	yes	Power on a secure element
RESET	yes	Reset a secure element
APDU	yes	Perform an ISO7816 request

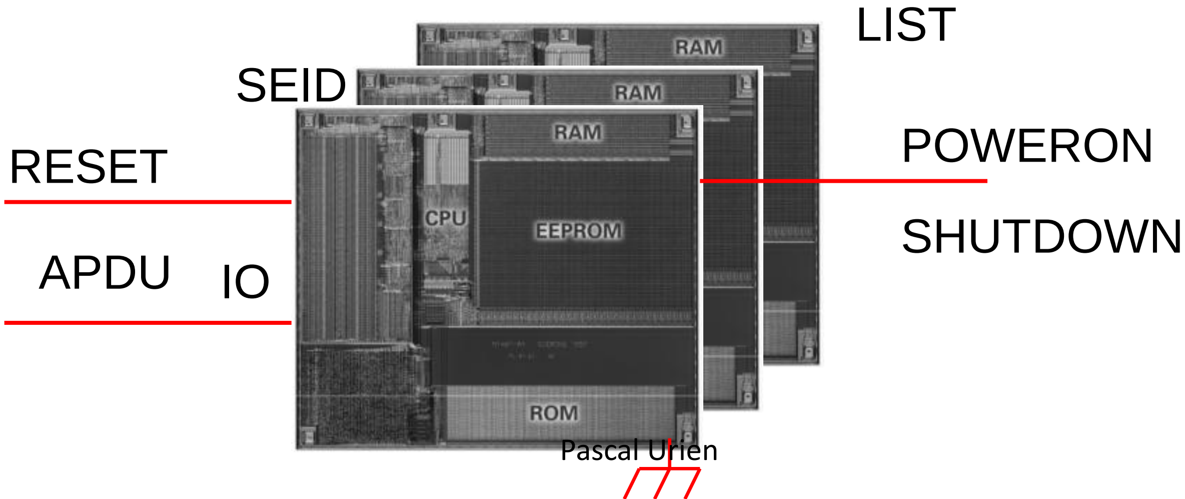
RACS Requests and Responses



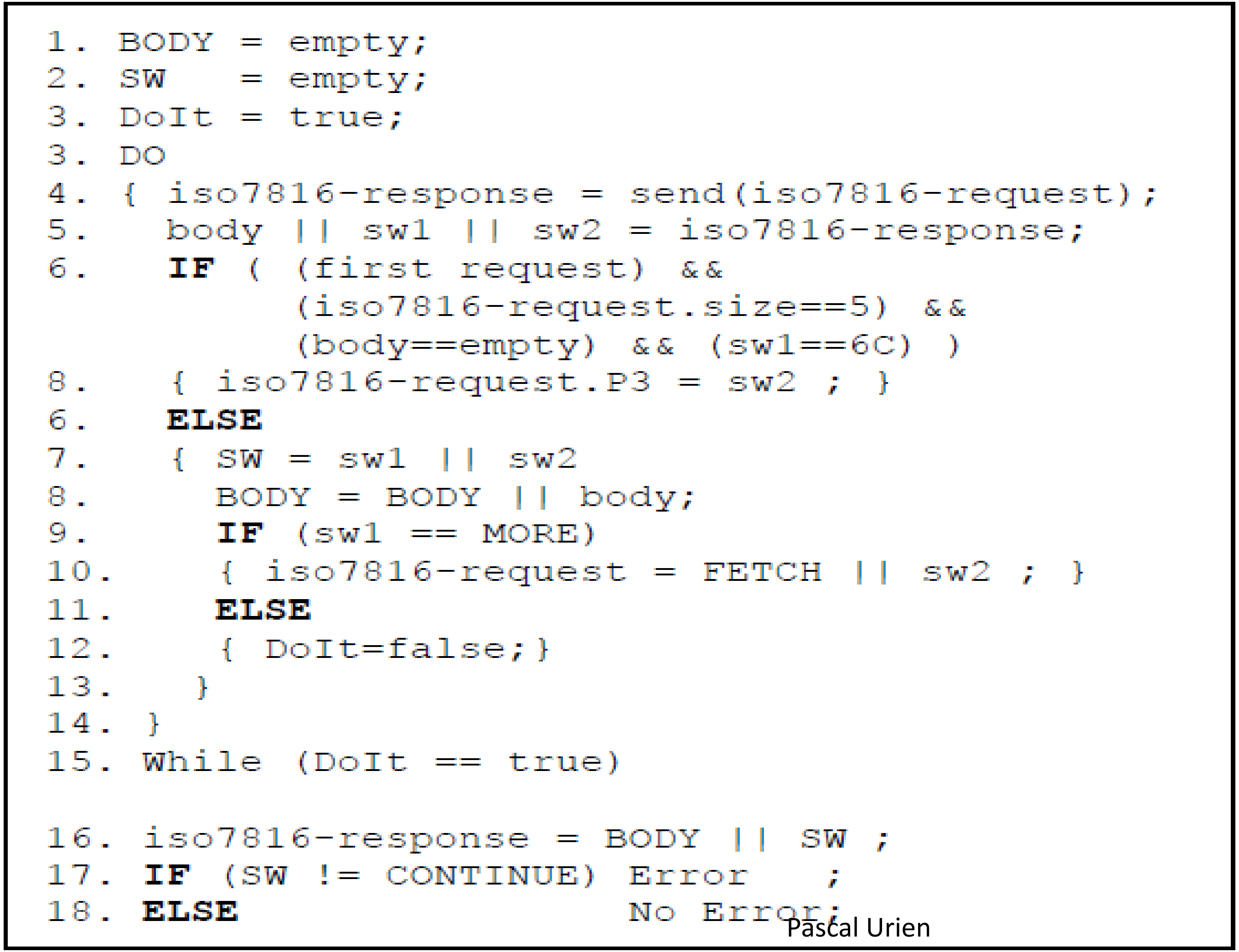
```
BEGIN MyLabel
GET-VERSION APPEND
LIST APPEND
POWERON MySEID APPEND
RESET MySEID
APDU MySEID
00A40400074A544553543030
END
```



```
BEGIN MyLabel
+002 001 0.2
+004 002 MySEID OtherSEID
+008 003 MySEID is powered on
+005 004 MySEID resetted
+006 005 9000
END
```



APDU Command



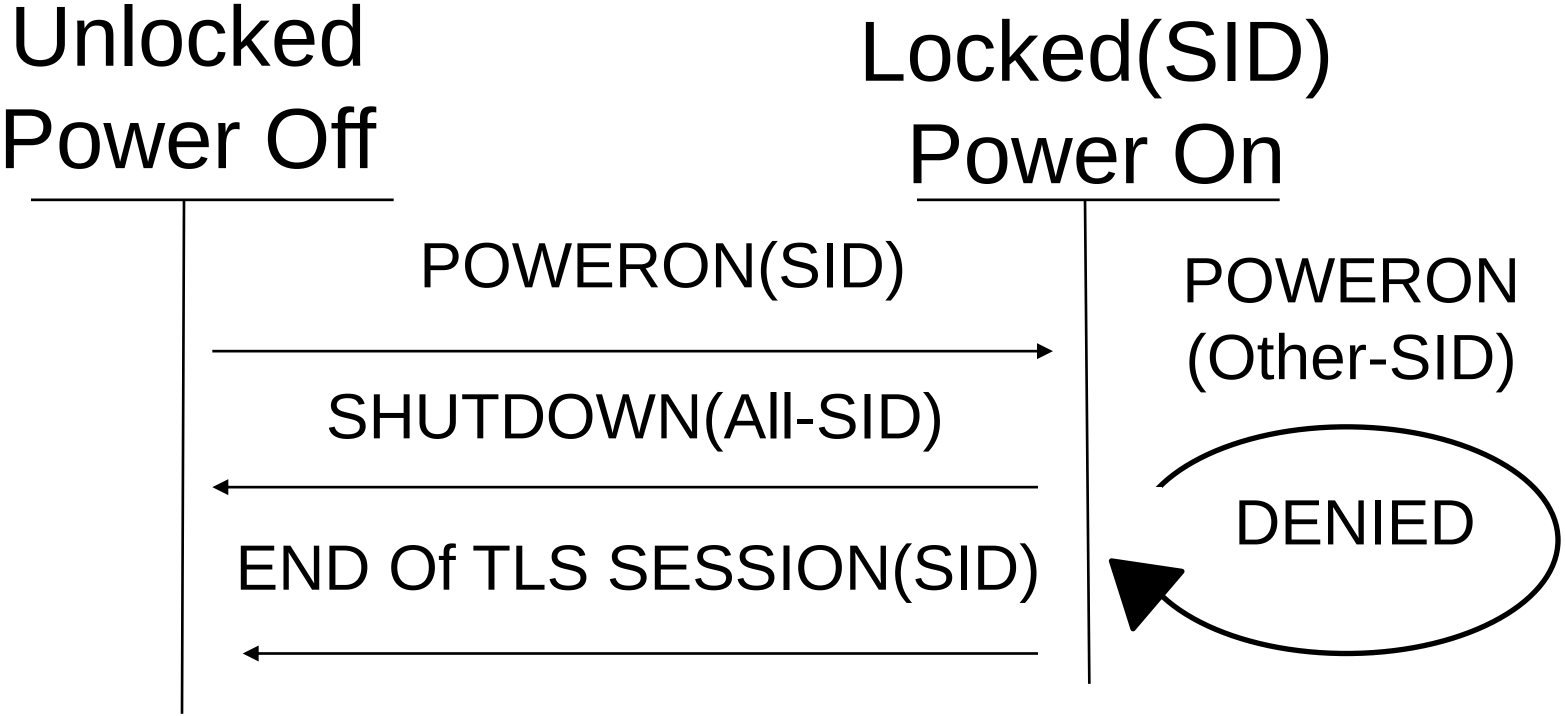
```
BODY = empty;
SW   = empty;
DoIt = true;
DO
{ iso7816-response = send(iso7816-request);
  body || sw1 || sw2 = iso7816-response;
  IF ( (first request) &&
       (iso7816-request.size==5) &&
       (body==empty) && (sw1==6C) )
  { iso7816-request.P3 = sw2 ; }
  ELSE
  { SW = sw1 || sw2
    BODY = BODY || body;
    IF (sw1 == MORE)
    { iso7816-request = FETCH || sw2 ; }
    ELSE
    { DoIt=false;}
   }
}
While (DoIt == true)

iso7816-response = BODY || SW ;
IF (SW != CONTINUE) Error    ;
ELSE                No Error;
```

- No Script !

Security Policy

- Only users equipped with valid certificates successfully establish TLS sessions.
- A user identifier (UID) is derived from the certificate Common Name (CN) attribute.
- A TLS session is identified by a unique identifier (the SID).
- Every secure element has two states, *unlocked* and *locked*.
  - The SHUTDOWN command forces the *unlocked* state; the POWERON command switches the SE state from *locked* to *unlocked*.
  - In the *locked* state the SE may be only used by the SID that previously locked it.
- At the end of a TLS session all used SEs are unpowered and *unlocked*.

SEID Locking



Unlocked
Power Off

Locked(SID)
Power On

POWERON(SID)

SHUTDOWN(All-SID)

END Of TLS SESSION(SID)

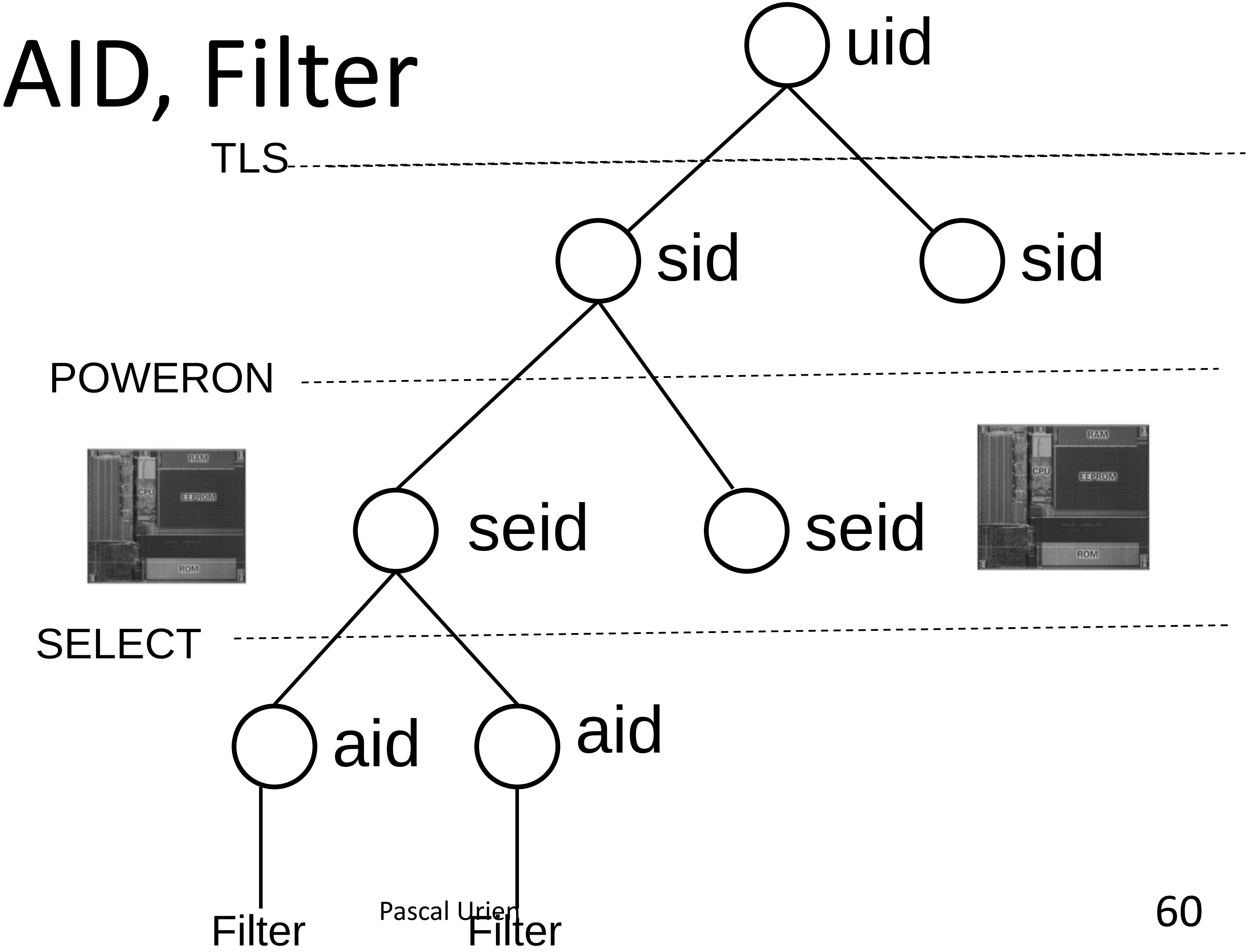
POWERON
(Other-SID)

DENIED

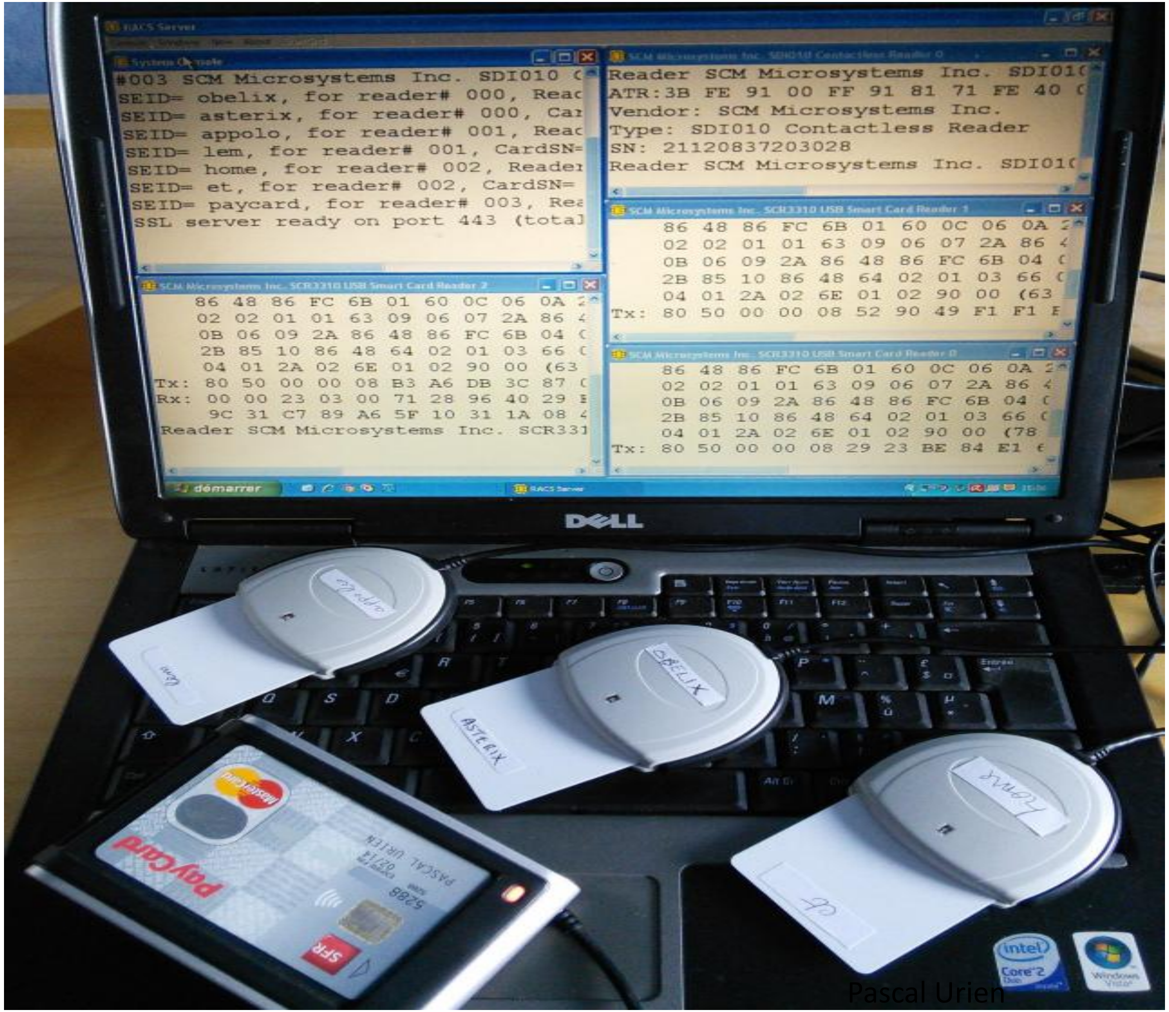
Access Control

- The server manages two kinds of table:
  - The *Users-Table* stores for each CN a list of authorized SEIDs.
  - Each SEID is linked to a *SEID-Table* storing for every AID (embedded application) a list of authorized CNs.

UID, SID, AID, Filter



Experimental Platform


RACS Server
System Console
SCM Microsystems Inc. SDI010 Contactless Reader 0
SCM Microsystems Inc. SCR3310 USB Smart Card Reader 1
SCM Microsystems Inc. SCR3310 USB Smart Card Reader 2
SCM Microsystems Inc. SCR3310 USB Smart Card Reader 0
démarrer
DELL
appolo
OBELIX
ASTERIX
Lem
et
payCard
SFR
PASCAL URIEN
intel Core 2
Windows Vista

```
#003 SCM Microsystems Inc. SDI010 C
SEID= obelix, for reader# 000, Reac
SEID= asterix, for reader# 000, Car
SEID= appolo, for reader# 001, Reac
SEID= lem, for reader# 001, CardSN=
SEID= home, for reader# 002, Reader
SEID= et, for reader# 002, CardSN=
SEID= paycard, for reader# 003, Rea
SSL server ready on port 443 (total
```

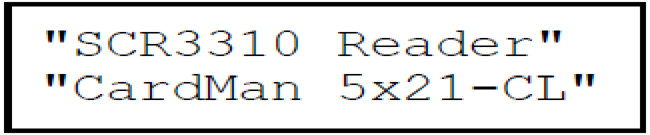
```
Reader SCM Microsystems Inc. SDI010
ATR:3B FE 91 00 FF 91 81 71 FE 40 C
Vendor: SCM Microsystems Inc.
Type: SDI010 Contactless Reader
SN: 21120837203028
Reader SCM Microsystems Inc. SDI010
```

```
     86 48 86 FC 6B 01 60 0C 06 0A 2
     02 02 01 01 63 09 06 07 2A 86 4
     0B 06 09 2A 86 48 86 FC 6B 04 C
     2B 85 10 86 48 64 02 01 03 66 C
     04 01 2A 02 6E 01 02 90 00 (63
Tx:  80 50 00 00 08 52 90 49 F1 F1 E
```

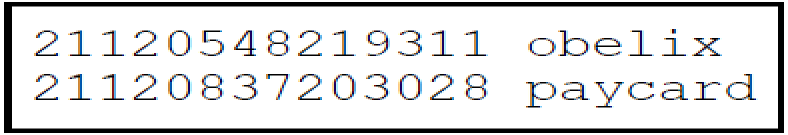
```
     86 48 86 FC 6B 01 60 0C 06 0A 2
     02 02 01 01 63 09 06 07 2A 86 4
     0B 06 09 2A 86 48 86 FC 6B 04 C
     2B 85 10 86 48 64 02 01 03 66 C
     04 01 2A 02 6E 01 02 90 00 (63
Tx:  80 50 00 00 08 B3 A6 DB 3C 87 C
Rx:  00 00 23 03 00 71 28 96 40 29 E
     9C 31 C7 89 A6 5F 10 31 1A 08 4
Reader SCM Microsystems Inc. SCR331
```

```
     86 48 86 FC 6B 01 60 0C 06 0A 2
     02 02 01 01 63 09 06 07 2A 86 4
     0B 06 09 2A 86 48 86 FC 6B 04 C
     2B 85 10 86 48 64 02 01 03 66 C
     04 01 2A 02 6E 01 02 90 00 (78
Tx:  80 50 00 00 08 29 23 BE 84 E1 6
```

SEID files



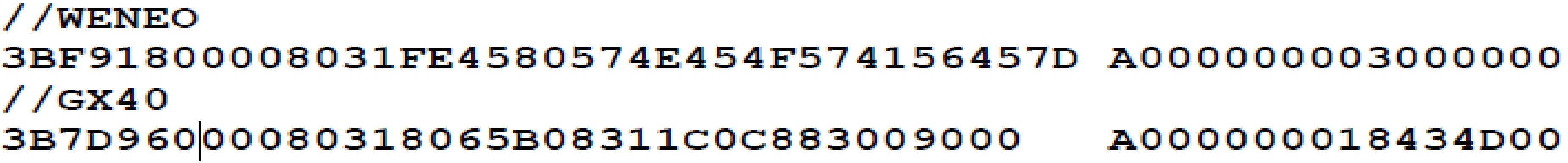
```
"SCR3310 Reader"
"CardMan 5x21-CL"
```



```
21120548219311 obelix
21120837203028 paycard
```

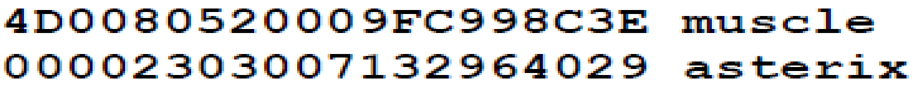
Reader.txt file

ReaderSN.txt file



```
//WENEO
3BF91800008031FE4580574E454F574156457D A000000003000000
//GX40
3B7D96000080318065B08311C0C883009000   A000000018434D00
```

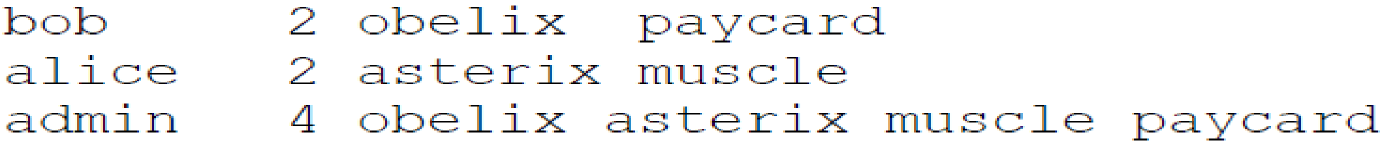
ATR.txt File



```
4D0080520009FC998C3E muscle
00002303007132964029 asterix
```

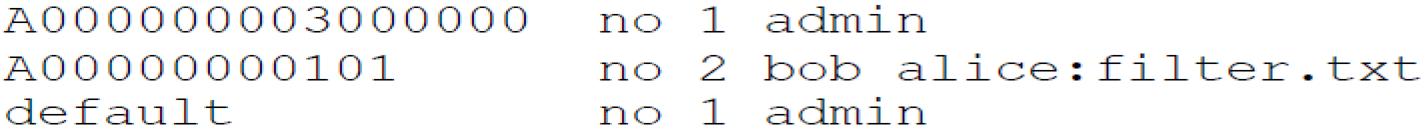
CardSN.txt File

Access Control Files



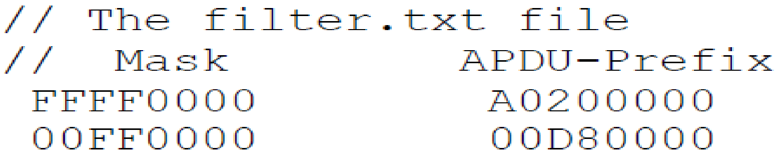
```
bob      2 obelix  paycard
alice    2 asterix muscle
admin    4 obelix asterix muscle paycard
```

Users.txt file



```
A000000003000000  no 1 admin
A00000000101      no 2 bob alice:filter.txt
default           no 1 admin
```

SEID.txt file



```
// The filter.txt file
//  Mask          APDU-Prefix
 FFFF0000          A0200000
 00FF0000          00D80000
```

Filter.txt file

The Open MobileAPI

- The API defines a generic framework for the access to Secure Elements in a mobile environment. It is based on four main objects.
- The ***SEService*** is the abstract representation of all SEs that are available for applications running in the mobile phone.
  - SEService seService = new SEService(this,this)
  - public void serviceConnected(SEService service)
  - seService.shutdown()
- The ***Reader*** is the logical interface with a Secure Element. It is an abstraction from electronics devices which are needed for contact (ISO 7816) and contactless (ISO 14443) smartcards.
  - Reader[] readers = seService.getReaders()
- The ***Session*** is opened and closed with a Reader. It establishes the logical path with the SE managed by the Reader.
  - Session session = readers[0].openSession()
  - session.close() or readers[0].closeSessions()
- The ***Channel*** is associated with an application running in the SE and identified by an ID (the AID= Application IDentifier)
  - Channel channel = session.openLogicalChannel(aid)
  - byte[] response channel.transmit(byte[] command)
  - channel.close()

OpenMobileAPI: The SIM File System

```
MF (3F00)
|-EF-DIR (2F00) -->  reference to DF-PKCS#15
|
|-DF-PKCS Access Control Main File #15 (7F50)
   |-ODF (5031)               -->   reference to DODF
   |-DODF (5207)              -->   reference to EF-ACMain
   |-EF-ACMain (4200)         -->   reference to EF-ACRules
   |-EF-ACRules (4300)        -->   reference to EF-ACConditions
   |-EF-ACConditions1 (4310)
   |-EF-ACConditions2 (4311)
   |-EF-ACConditions3 (4312)
```

EF-ACRules

```
30 10
  A0 08 // aid
    04 06
      A0 00 00 01 51 01 // Application Identifier (AID)
        30 04
          04 02
            43 10 // EF-ACCondition  File
30 10 A0 08 04 06 A0 00 00 01 51 02 30 04 04 02 43 11
30 10 A0 08 04 06 A0 00 00 01 51 03 30 04 04 02 43 11
30 08
  82 00 // other
    30 04
      04 02
        43 12 // file
FF FF FF 90 00
```

No access to any application

```
Tx: 00 A4 00 04 02 43 10 // Select AC-Conditions1
Rx: 61 20
Tx: 00 C0 00 00 12
Rx: 62 1E 82 02 41 21 83 02 43 10 A5 06 C0 01 00 DE 01 00 61 0E
Tx: 00 B0 00 00 00 // Read AC-Conditions1 - empty file, no access to any application
Rx: 6C 1E
Tx: 00 B0 00 00 1E
Rx: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF 90 00
```

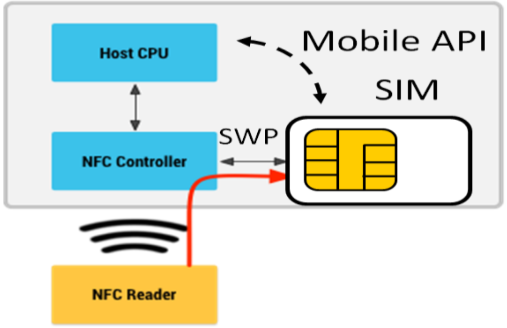
Access to a single application

```
Tx: 00 A4 00 04 02 43 11 // Select AC-Conditions2
Rx: 61 20
Tx: 00 C0 00 00 20
Rx: 62 1E 82 02 41 21 83 02 43 11 A5 06 C0 01 00 DE 01 00 8A 01
05 8B 03 6F 06 02 80 02 00 1E 88 00 90 00
Tx: 00 B0 00 00 00 // Read AC-Conditions2,
Rx: 6C 1E
Tx: 00 B0 00 00 1E
Rx: 30 16
       04 14
         11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
// CertHash
      FF FF FF FF FF FF 90 00
```

Access by any application

```
Tx: 00 A4 00 04 02 43 12   // Select AC-Conditions3
Rx: 61 20
Tx: 00 C0 00 00 20
Rx: 62 1E 82 02 41 21 83 02 43 12 A5 06 C0 01 00 DE 01 00 8A 01 05 8B
03 6F 06 02 80 02 00 1E 88 00 90 00
Tx: 00 B0 00 00 00  // Read AC-Conditions3,  access by any application
Rx: 6C 1E
Tx: 00 B0 00 00 1E
Rx: 30 00  // empty condition entry,
    FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF  FF FF FF FF FF FF FF FF FF FF
    90 00
```

Host Card Emulation



Host CPU

Mobile API

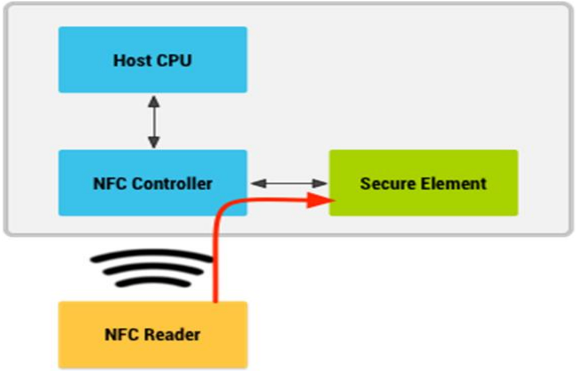
SIM

SWP

NFC Controller

NFC Reader

LEGACY



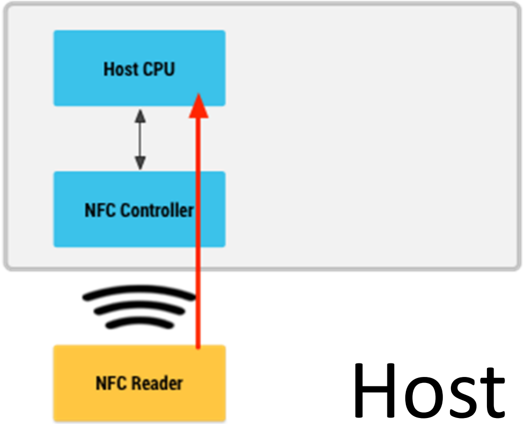
Host CPU

NFC Controller

Secure Element

NFC Reader

Google Nexus S



Host CPU

NFC Controller

NFC Reader

Host Card Emulation

HCE Service

```
<service
  android:name=".MyHostApduService"
   android:exported="true"
   android:permission="android.permission.BIND_NFC_SERVICE" >
  <intent-filter>
 <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
 </intent-filter>
 <meta-data
   android:name="android.nfc.cardemulation.host_apdu_service"
   android:resource="@xml/apduservice" />
 </service>
```

HCE Service

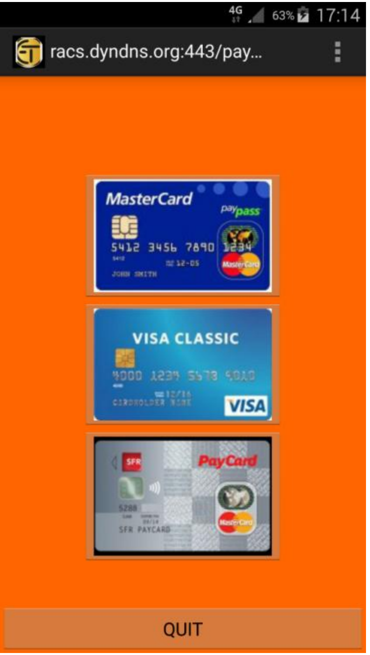
```
<host-apdu-service
  xmlns:android= "http://schemas.android.com/apk/res/android"
  android:description="@string/servicedesc"
  android:requireDeviceUnlock="false" >
<aid-group
   android:category="other"
   android:description="@string/aiddescription" >
   <aid-filter android:name= "325041592E5359532E4444463031" />
   <aid-filter android:name= "a000000041010aa54303200ff01ffff" />
</aid-group>
</host-apdu-service>
```

The HCE service implements two methods for NFC communication:

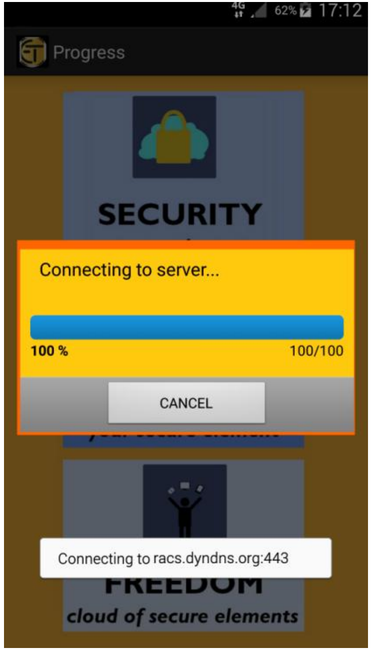
- public byte[] processCommandApdu(byte[] apdu, Bundle extras).
- public void sendResponseApdu(byte[] responseAPDU).

User’s Experience

Selection of a bank card



Connection to server

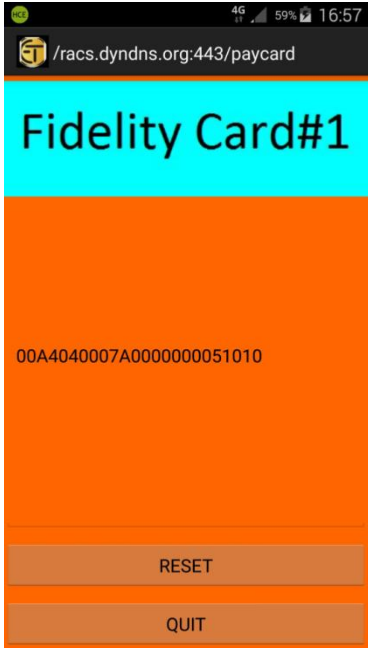


Connecting to server...

Connecting to racs.dyndns.org:443

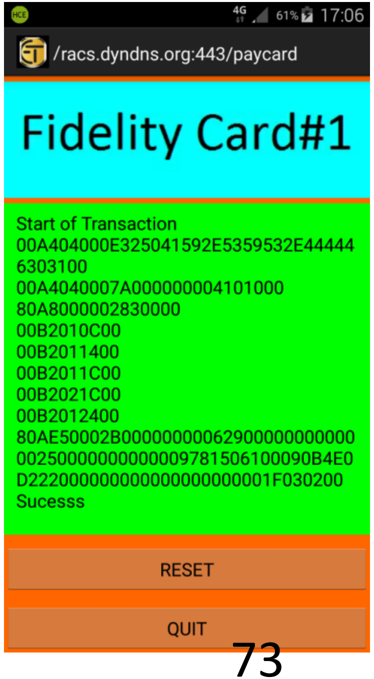
Ready for payment

Fidelity Card Reading



```
00A4040007A0000000051010
```

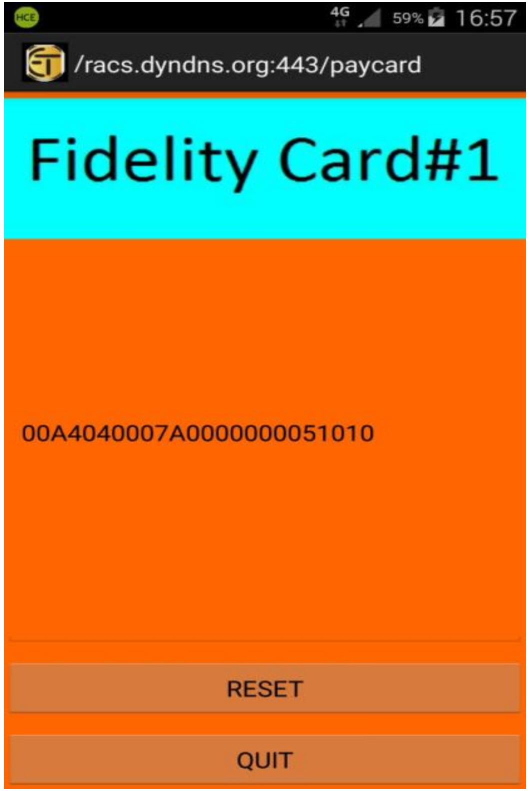
Payment Transaction



```
Start of Transaction
00A404000E325041592E5359532E44444
6303100
00A4040007A000000004101000
80A8000002830000
00B2010C00
00B2011400
00B2011C00
00B2021C00
00B2012400
80AE50002B00000000006290000000000
002500000000000009781506100090B4E0
D22200000000000000000001F030200
Sucesss
```

About Virtual Fidelity Cards

- A virtual fidelity card is associated with an application AID registered with the payment application.
- The merchant terminal selects this virtual card (via the dedicated iso7816 SELECT command) before the transaction.
- The returned information includes a card number to which the payment would be bound.



/racs.dyndns.org:443/paycard

Fidelity Card#1

00A4040007A0000000051010

Pascal Urien

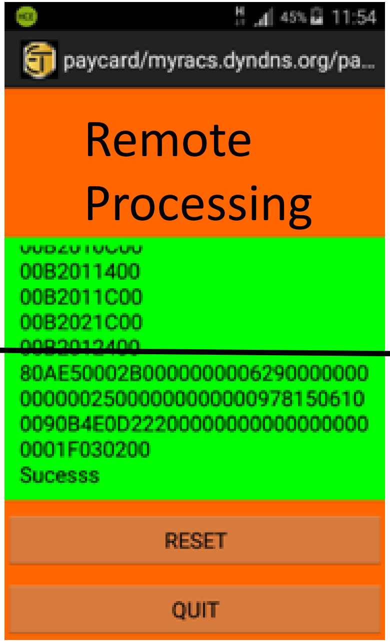
Legacy Timing

- A legacy contactless transaction consumes about 400ms, and requires 8 ISO7816 requests, which are detailed below:
  - Selection of the PPSE application.
  - Selection of the NFC payment application.
  - Issuance of the GPO command.
  - Four *ReadRecord* commands used for collecting four files located in two records.
  - One GenerateAC request, realizing a CDA operation.
- About 50% of the transaction time (200 ms) is consumed by the CDA computing.

Transparent Mode

TERMINAL

APDU



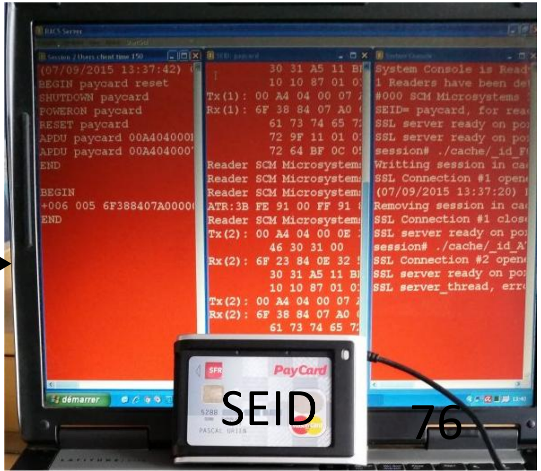
Remote Processing

```
00B2010C00
00B2011400
00B2011C00
00B2021C00
00B2012400
80AE50002B00000000006290000000
00000025000000000000978150610
0090B4E0D222000000000000000000
0001F030200
Sucesss
```

RACS Script

```
BEGIN APDUScript
APDU
END
```

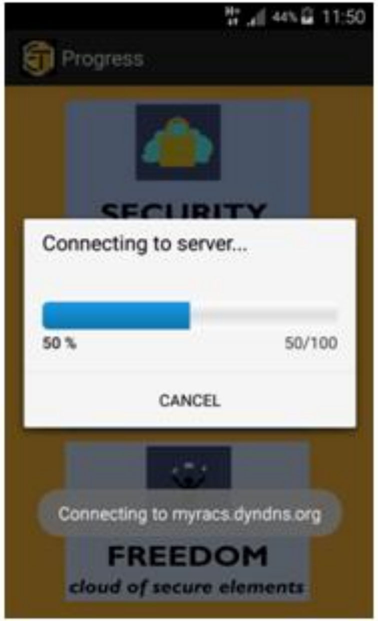
RACS Server



```
(07/09/2015 13:37:42) (
BEGIN paycard reset
SHUTDOWN paycard
POWERON paycard
RESET paycard
APDU paycard 00A404000
APDU paycard 00A404000
END

BEGIN
+006 005 6F388407A0000
END
```

```
          30 31 A5 11 BF
          10 10 87 01 01
Tx(1): 00 A4 04 00 07
Rx(1): 6F 38 84 07 A0
          61 73 74 65 72
          72 9F 11 01 01
          72 64 BF 0C 05
Reader SCM Microsystems
Reader SCM Microsystems
Reader SCM Microsystems
ATR:3B FE 91 00 FF 91
Reader SCM Microsystems
Tx(2): 00 A4 04 00 0E
          46 30 31 00
Rx(2): 6F 23 84 0E 32
          30 31 A5 11 BF
          10 10 87 01 01
Tx(2): 00 A4 04 00 07
Rx(2): 6F 38 84 07 A0
          61 73 74 65 72
```

```
System Console is Read
1 Readers have been de
#000 SCM Microsystems
SEID= paycard, for rea
SSL server ready on po
SSL server ready on po
session# ./cache/_id_F
Writting session in ca
SSL Connection #1 open
(07/09/2015 13:37:20)
Removing session in ca
SSL Connection #1 close
SSL server ready on po
session# ./cache/_id_A
SSL Connection #2 open
SSL server ready on po
SSL server_thread, err
```



RACS Script executed after the connection to the server

```
BEGIN ResetScript
SHUTDOWN SEID
POWERON SEID
END
```

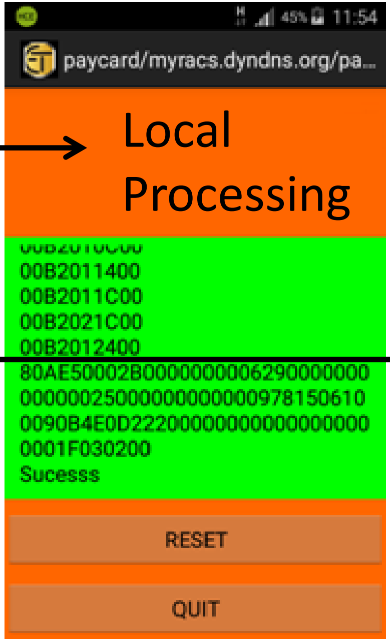
Transparent Mode

- In the transparent mode every iso7816 request is forwarded to the server
- What leads to an extra time cost of about 250ms (in average) per APDU
- Total duration is about 8x250+400= 2400ms.

TERMINAL

SELECT PPSE
SELECT AID
GPO
READ-RECORD(s)

GENERATE-AC



Local
Processing

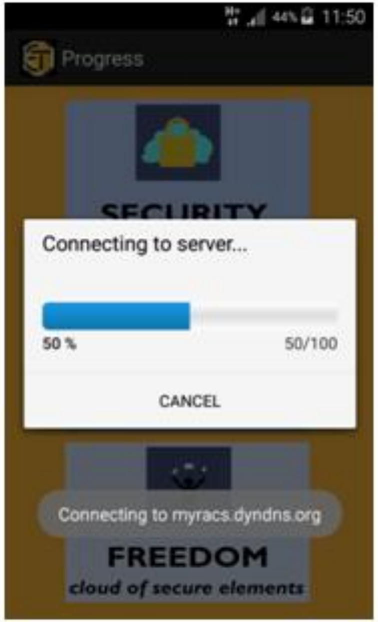
```
00B2010C00
00B2011400
00B2011C00
00B2021C00
00B2012400
80AE50002B000000000006290000000
0000002500000000000000978150610
0090B4E0D2220000000000000000000
0001F030200
Sucesss
```

RACS Script executed for cryptographic (CDA) computing

```
BEGIN CdaScript
APDU [GPO]
APDU [GENERATE-AC]
END
```

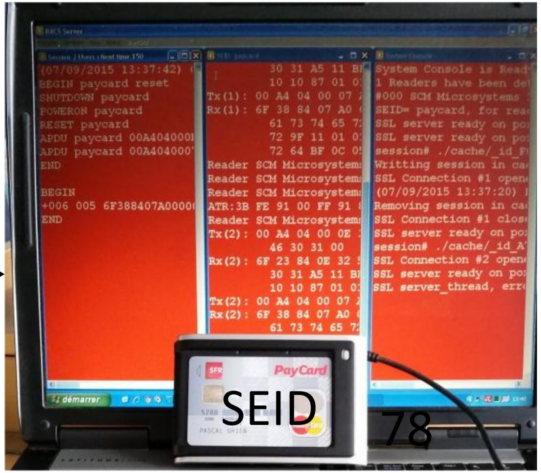
RACS Server

Cache Mode



RACS Script executed after the connection to the server

```
BEGIN ResetScript
SHUTDOWN SEID
POWERON SEID
SELECT SEID AID
END
```



```
(07/09/2015 13:37:42) (
BEGIN paycard reset
SHUTDOWN paycard
POWERON paycard
RESET paycard
APDU paycard 00A404000E
APDU paycard 00A4040007
END

BEGIN
+006 005 6F388407A00000
END
```

```
          30 31 A5 11 BF
          10 10 87 01 01
Tx(1): 00 A4 04 00 07 A
Rx(1): 6F 38 84 07 A0 0
          61 73 74 65 72
          72 9F 11 01 01
          72 64 BF 0C 05
Reader SCM Microsystems
Reader SCM Microsystems
Reader SCM Microsystems
ATR:3B FE 91 00 FF 91 8
Reader SCM Microsystems
Tx(2): 00 A4 04 00 0E 3
          46 30 31 00
Rx(2): 6F 23 84 0E 32 5
          30 31 A5 11 BF
          10 10 87 01 01
Tx(2): 00 A4 04 00 07 A
Rx(2): 6F 38 84 07 A0 0
          61 73 74 65 72
```

```
System Console is Read
1 Readers have been de
#000 SCM Microsystems
SEID= paycard, for rea
SSL server ready on po
SSL server ready on po
session# ./cache/_id_F
Writting session in ca
SSL Connection #1 open
(07/09/2015 13:37:20) 
Removing session in ca
SSL Connection #1 clos
SSL server ready on po
session# ./cache/_id_A
SSL Connection #2 open
SSL server ready on po
SSL server_thread, err
```

Cache Mode

- The mobile application manages a cache; the seven first iso7816 requests, which return static information, are locally processed by the smartphone.
- Each operation needs about 30ms; therefore seven APDUs cost 210ms, which is nearly equivalent to the legacy transaction.
- The last request (*GenerateAC*) is forwarded to the remote server, which implies a delay ranging between 350 and 650 ms, according to the following repartition:
  - 200 ms are burnt by the remote CDA operation
  - 100-250 ms are spent by the platform components (mobile phone, server operating system and network components)
  - 50-200 ms are consumed by the latency of 3G/4G cellular network.
- Total: 560-860ms

Part II – Secure Elements For Object

About the Internet of Things (IoT)

- Pretz, K. (2013). “The Next Evolution of the Internet”

The Internet of Things (IoT) is a *network of connected things*.

Beyond The Horizon

- The IoT is the death of the Moore Law.
- Waldrop M. "More Than Moore", Nature February 2016 Vol 530
  - *The semiconductor industry will soon abandon its pursuit of Moore’s Law.*
- "Rebooting the IT Revolution: A Call to Action" (SIA/SRC), 2015
  - "*Security is projected to become an even bigger challenge in the future as the number of interconnected devices increases... In fact, the Internet of Things can be viewed as the largest and most poorly defended cyber attack surface conceived by mankind*"

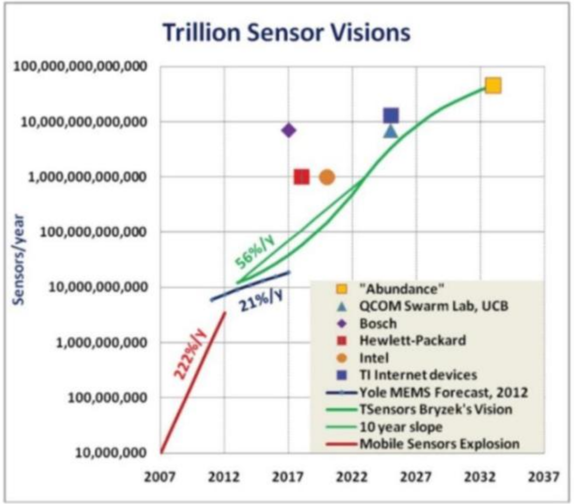
Trillion Sensors

*W= ½ Nq x V

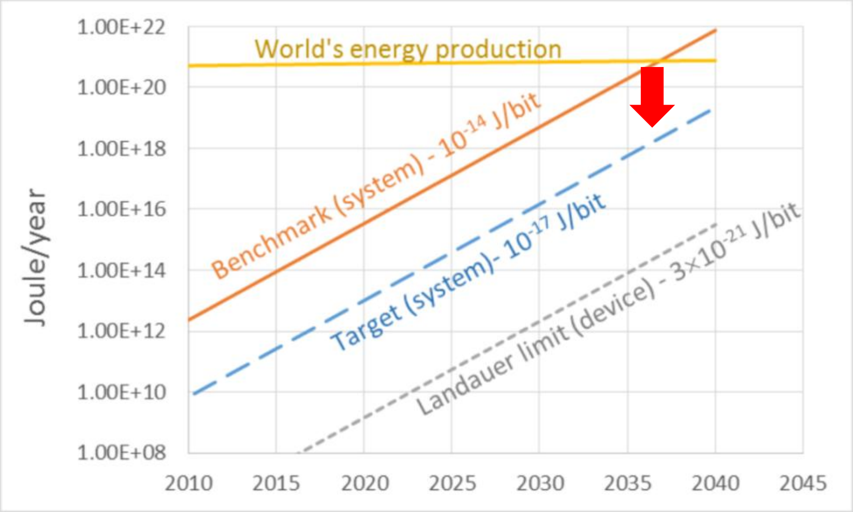
q = 1,6 10^{-19}

10^{-14} J == 125,000 electrons

- In current mainstream systems, the lower-edge system-level energy per one bit *transition is ~10^{-14} J, which is referred as the "benchmark".



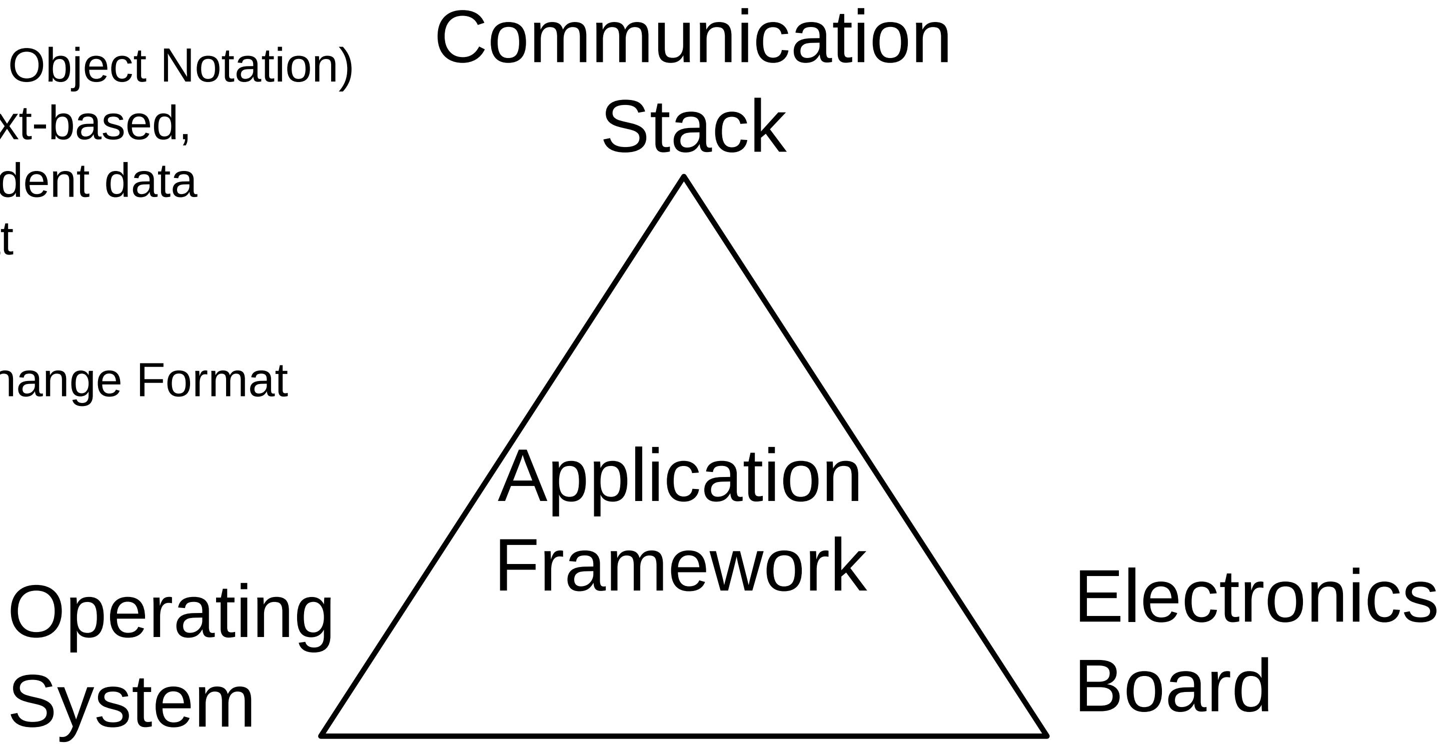
Towards Cyber Physical Systems (CPS)



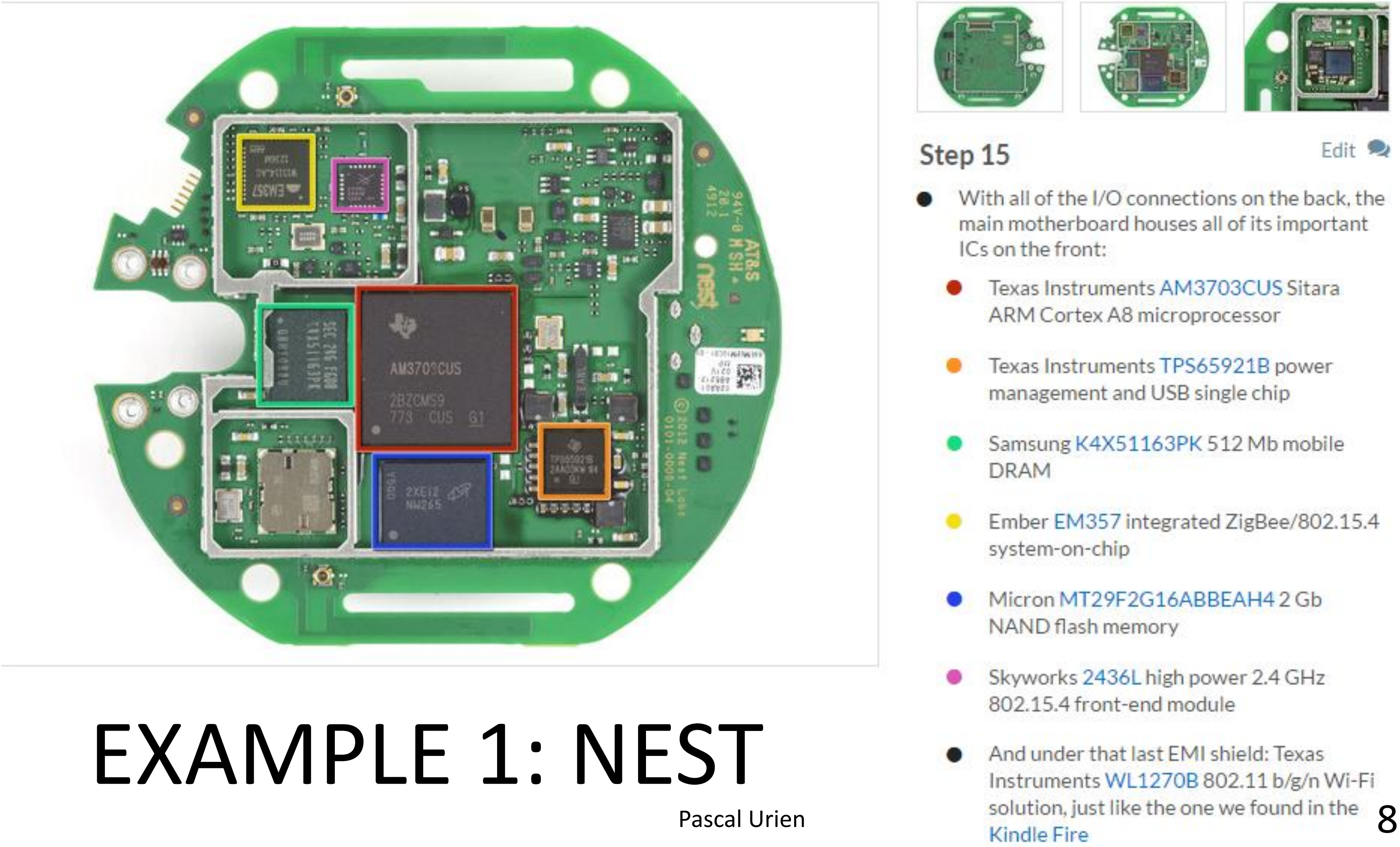
Internet Of Things

JSON (JavaScript Object Notation) is a lightweight, text-based, language-independent data interchange format

JSON Schema
JSON Data Interchange Format
REST protocol



EXAMPLE 1: NEST



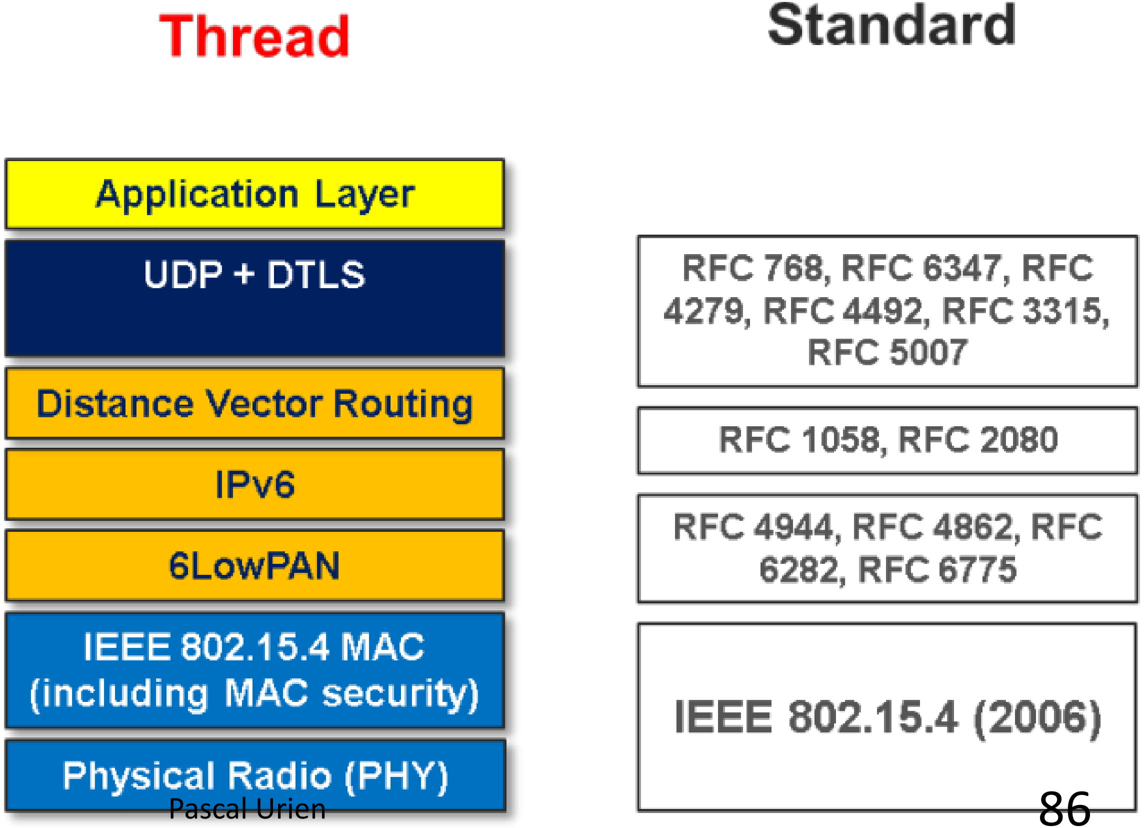
Step 15

- With all of the I/O connections on the back, the main motherboard houses all of its important ICs on the front:
  - 🟥 Texas Instruments AM3703CUS Sitara ARM Cortex A8 microprocessor
  - 🟧 Texas Instruments TPS65921B power management and USB single chip
  - 🟩 Samsung K4X51163PK 512 Mb mobile DRAM
  - 🟨 Ember EM357 integrated ZigBee/802.15.4 system-on-chip
  - 🟦 Micron MT29F2G16ABBEAH4 2 Gb NAND flash memory
  - 🟪 Skyworks 2436L high power 2.4 GHz 802.15.4 front-end module
  - ● And under that last EMI shield: Texas Instruments WL1270B 802.11 b/g/n Wi-Fi solution, just like the one we found in the Kindle Fire

https://www.threadgroup.org

DTLS + J-PAKE
Authentification

J-PAKE is a password-authenticated key exchange (PAKE) with “juggling” (hence the “J”).
It essentially uses elliptic curve Diffie-Hellmann for key agreement and Schnorr signatures as a NIZK (Non-Interactive Zero-Knowledge) proof mechanism



Thread

Standard

Application Layer

UDP + DTLS

RFC 768, RFC 6347, RFC 4279, RFC 4492, RFC 3315, RFC 5007

Distance Vector Routing

RFC 1058, RFC 2080

IPv6

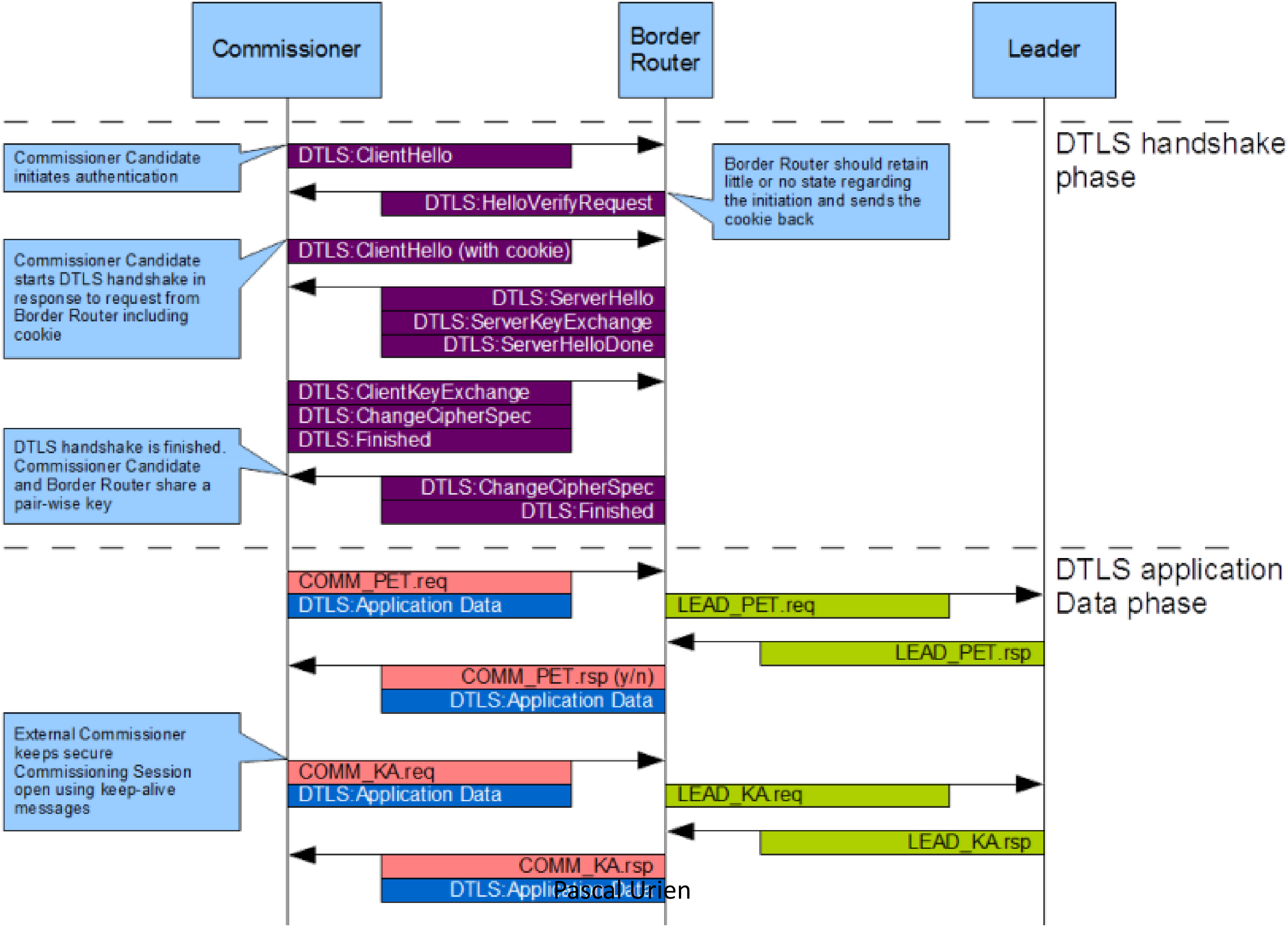
6LowPAN

RFC 4944, RFC 4862, RFC 6282, RFC 6775

IEEE 802.15.4 MAC (including MAC security)

Physical Radio (PHY)

IEEE 802.15.4 (2006)



Commissioner

Border Router

Leader

DTLS handshake phase

Commissioner Candidate initiates authentication

DTLS:ClientHello

DTLS:HelloVerifyRequest

Border Router should retain little or no state regarding the initiation and sends the cookie back

Commissioner Candidate starts DTLS handshake in response to request from Border Router including cookie

DTLS:ClientHello (with cookie)

DTLS:ServerHello
DTLS:ServerKeyExchange
DTLS:ServerHelloDone

DTLS:ClientKeyExchange
DTLS:ChangeCipherSpec
DTLS:Finished

DTLS handshake is finished. Commissioner Candidate and Border Router share a pair-wise key

DTLS:ChangeCipherSpec
DTLS:Finished

DTLS application Data phase

COMM_PET.req
DTLS:Application Data

LEAD_PET.req

LEAD_PET.rsp

COMM_PET.rsp (y/n)
DTLS:Application Data

External Commissioner keeps secure Commissioning Session open using keep-alive messages

COMM_KA.req
DTLS:Application Data

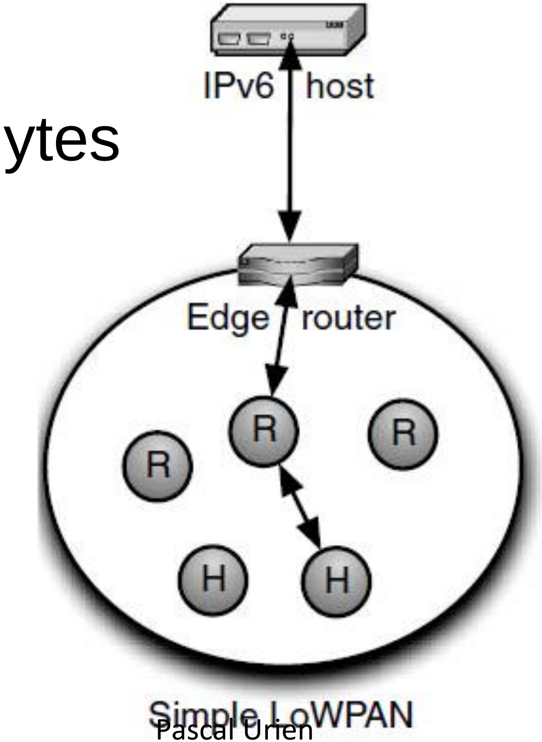
LEAD_KA.req

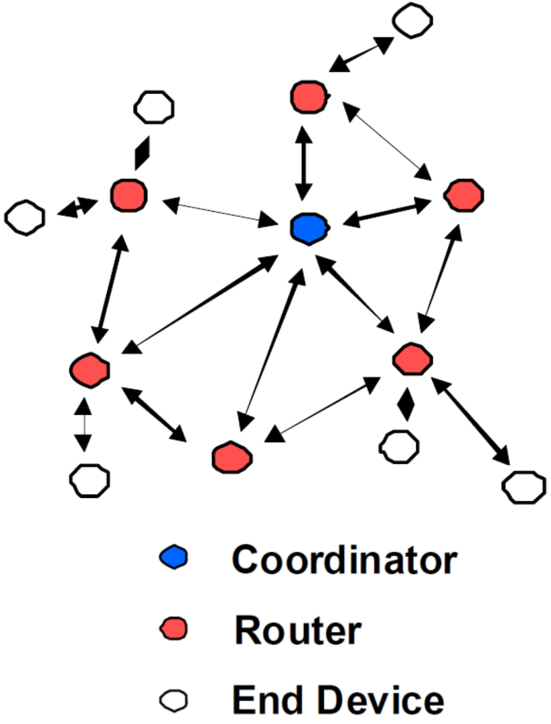
LEAD_KA.rsp

COMM_KA.rsp
DTLS:Application Data

6LoWPAN deals with IPv6 and Mesh networks

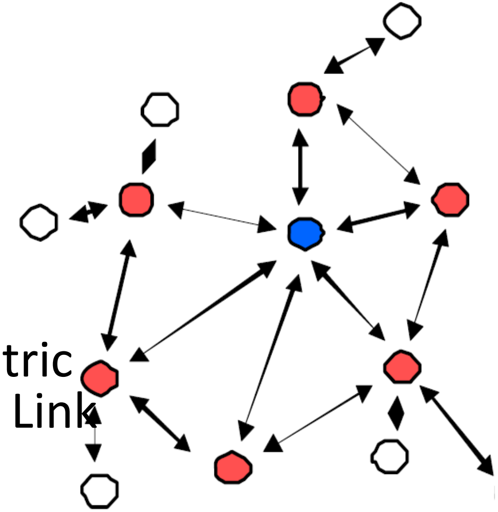
IEEE 802.15.4
MAC Frame Size 127 Bytes
IpV6 header 40 Bytes
TCP header 20 Bytes



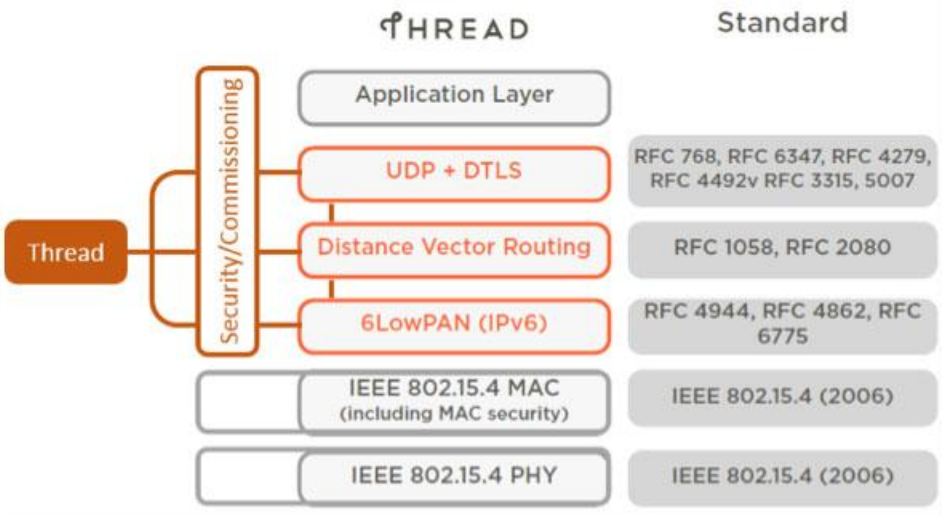


IEEE 802.15.4

- Coordinator is assumed to be the Trust Center (TC) and provides
  - Cryptographic key establishment
  - Key transport
  - Frame protection
  - Device management
- Cryptographic Keys
  - **Master** , basis for long term security used for symmetric key establishment. It is used to keep confidential the Link Keys exchange between two nodes in the Key Establishment Procedure (SKKE).
  - **Link**, shared exclusively between two network peers for Unicast communication.
  - **Network**, used for broadcast communication security.



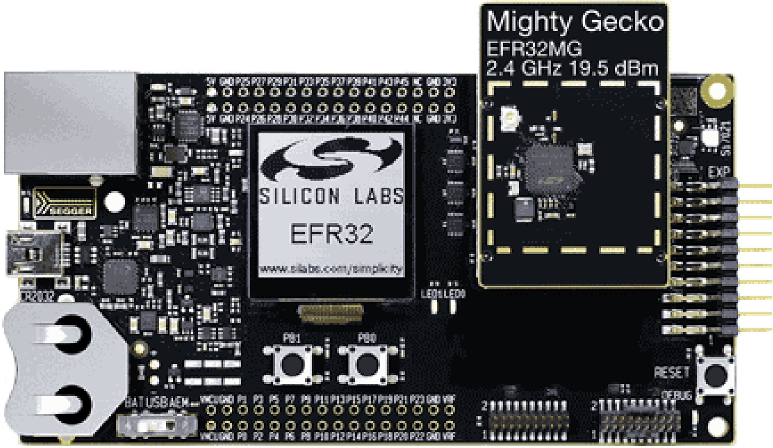
THREAD BOARD



THREAD	Standard
Application Layer	
UDP + DTLS	RFC 768, RFC 6347, RFC 4279, RFC 4492v RFC 3315, 5007
Distance Vector Routing	RFC 1058, RFC 2080
6LowPAN (IPv6)	RFC 4944, RFC 4862, RFC 6775
IEEE 802.15.4 MAC (including MAC security)	IEEE 802.15.4 (2006)
IEEE 802.15.4 PHY	IEEE 802.15.4 (2006)

SILICON LABS

http://www.silabs.com/



Pascal Urien

Example 2: Open Connectivity Foundation (OCF)

https://openconnectivity.org/

The **Open Connectivity Foundation (OCF)** is creating a specification and sponsoring an open source project to make this possible.
The OCF sponsors the IoTivity open source project which includes a reference implementation of our specification available under the Apache 2.0 license.

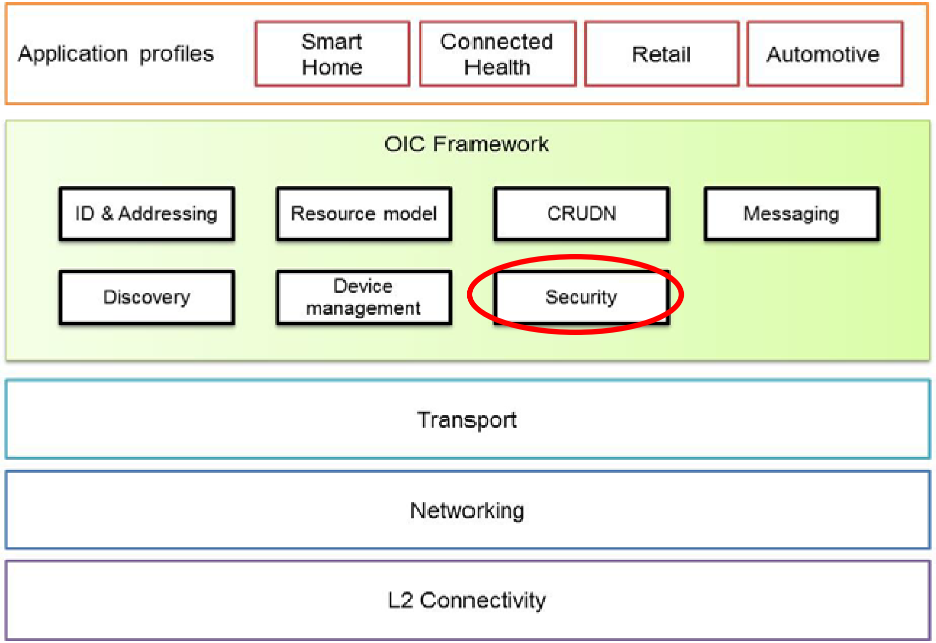


Figure 2: OIC functional block diagram

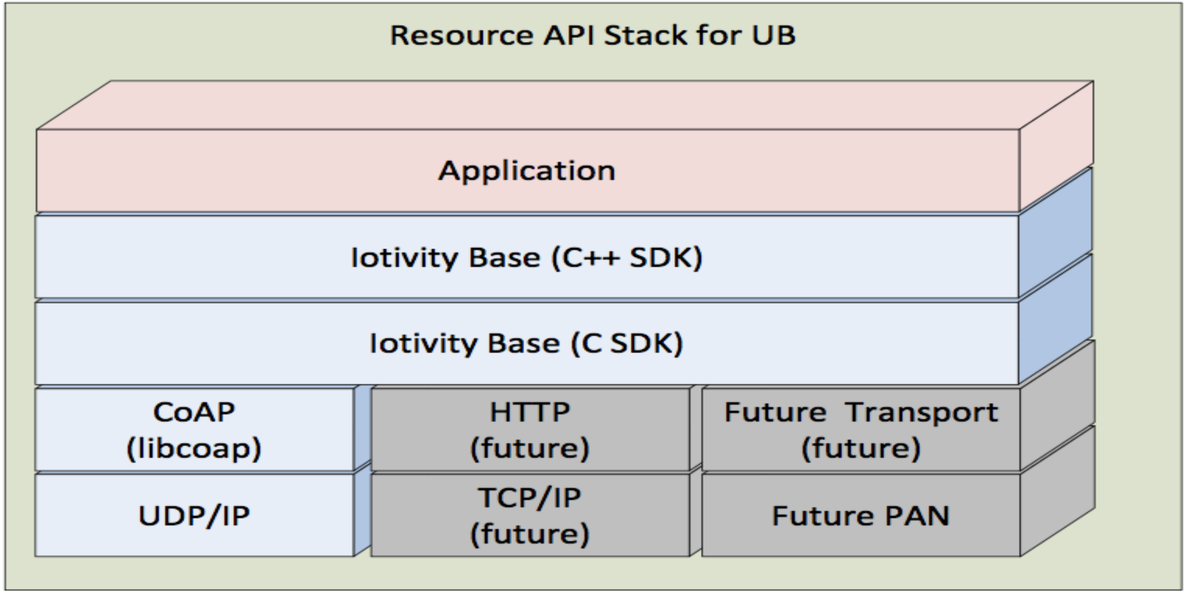
Create, Read, Update, Delete, Notify: CRUDN
Open Interconnect Consortium (OIC)

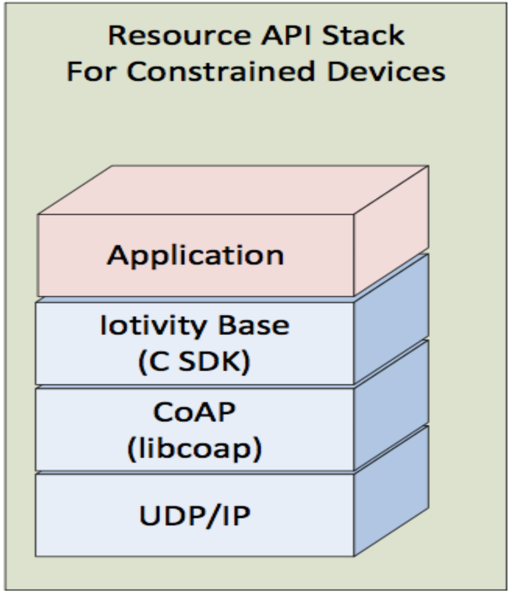
IOTIVITY

https://www.iotivity.org/

Unified Block (UB) stack

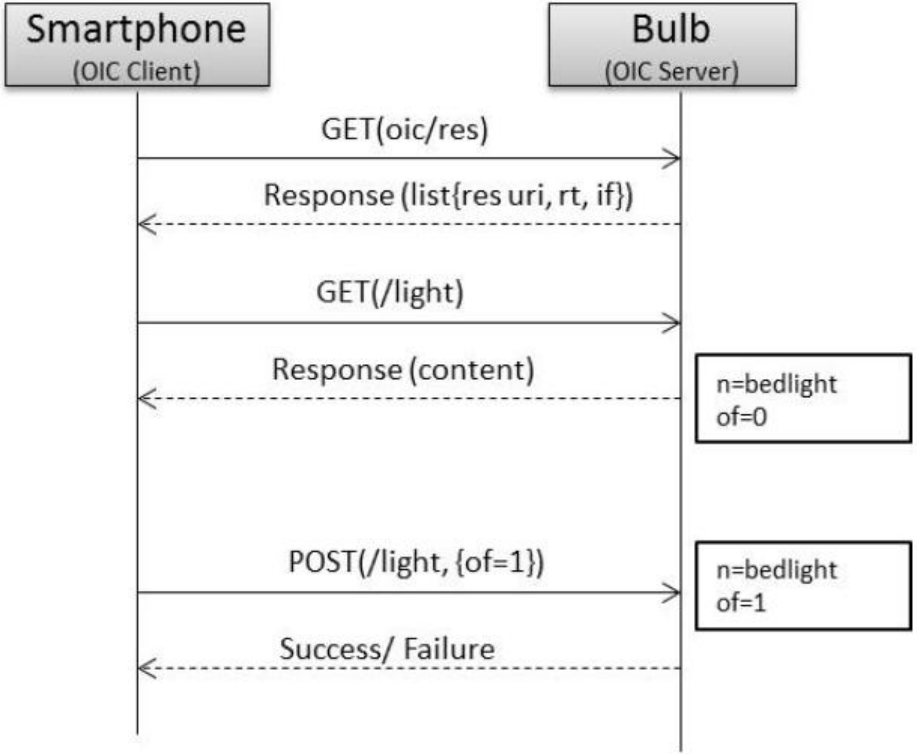
Thin Block (TB) stack



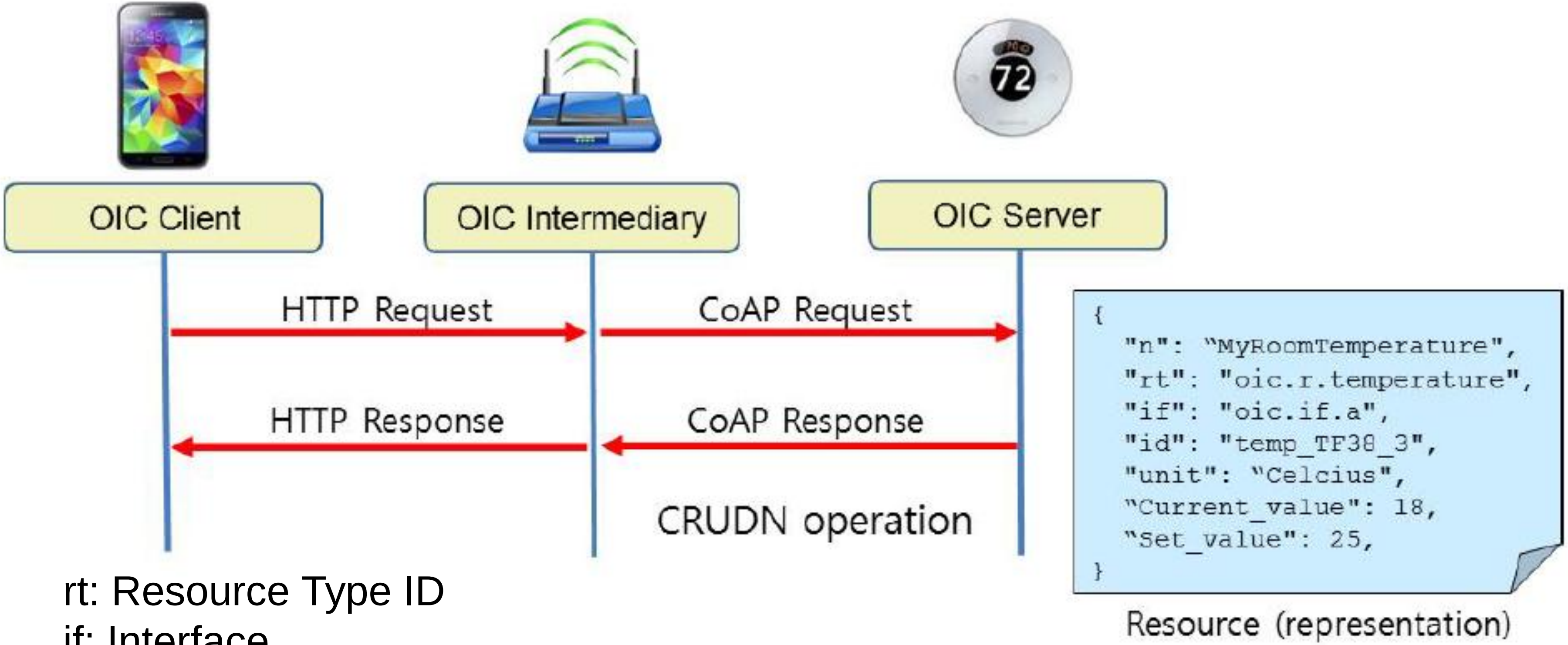


IoTivity is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. It supports multiple operating systems : Linux, Android, Tize, Arduino

Smartphone Bulb Interaction



CoAP /HTTP



```
{
  "n": “MyRoomTemperature",
  "rt": "oic.r.temperature",
  "if": "oic.if.a",
  "id": "temp_TF38_3",
  "unit": “Celcius",
  “Current_value": 18,
  “Set_value": 25,
}
```

Resource (representation)

rt: Resource Type ID
if: Interface

Access Control List (ACL)

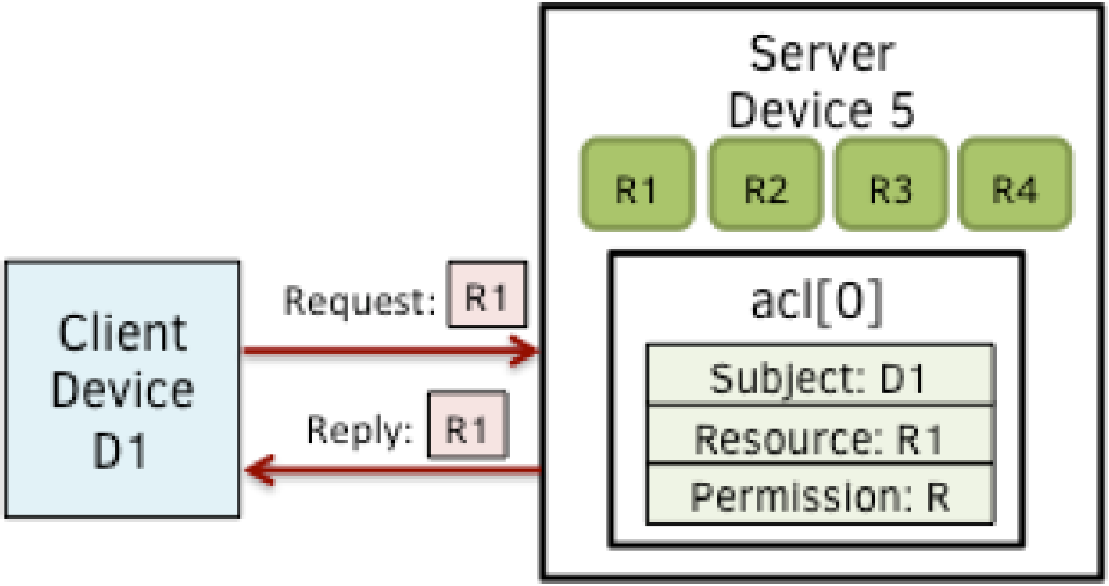


Figure 2 – Use case-1 showing simple ACL enforcement

Secure Storage

It is strongly recommended that IoT device makers provide reasonable protection for Sensitive Data so that it cannot be accessed by unauthorized devices, groups or individuals for either malicious or benign purposes.

In addition, since Sensitive Data is often used for authentication and encryption, it must maintain its integrity against intentional or accidental alteration

Device Authentication with DTLS

Device Authentication with Symmetric Key Credentials

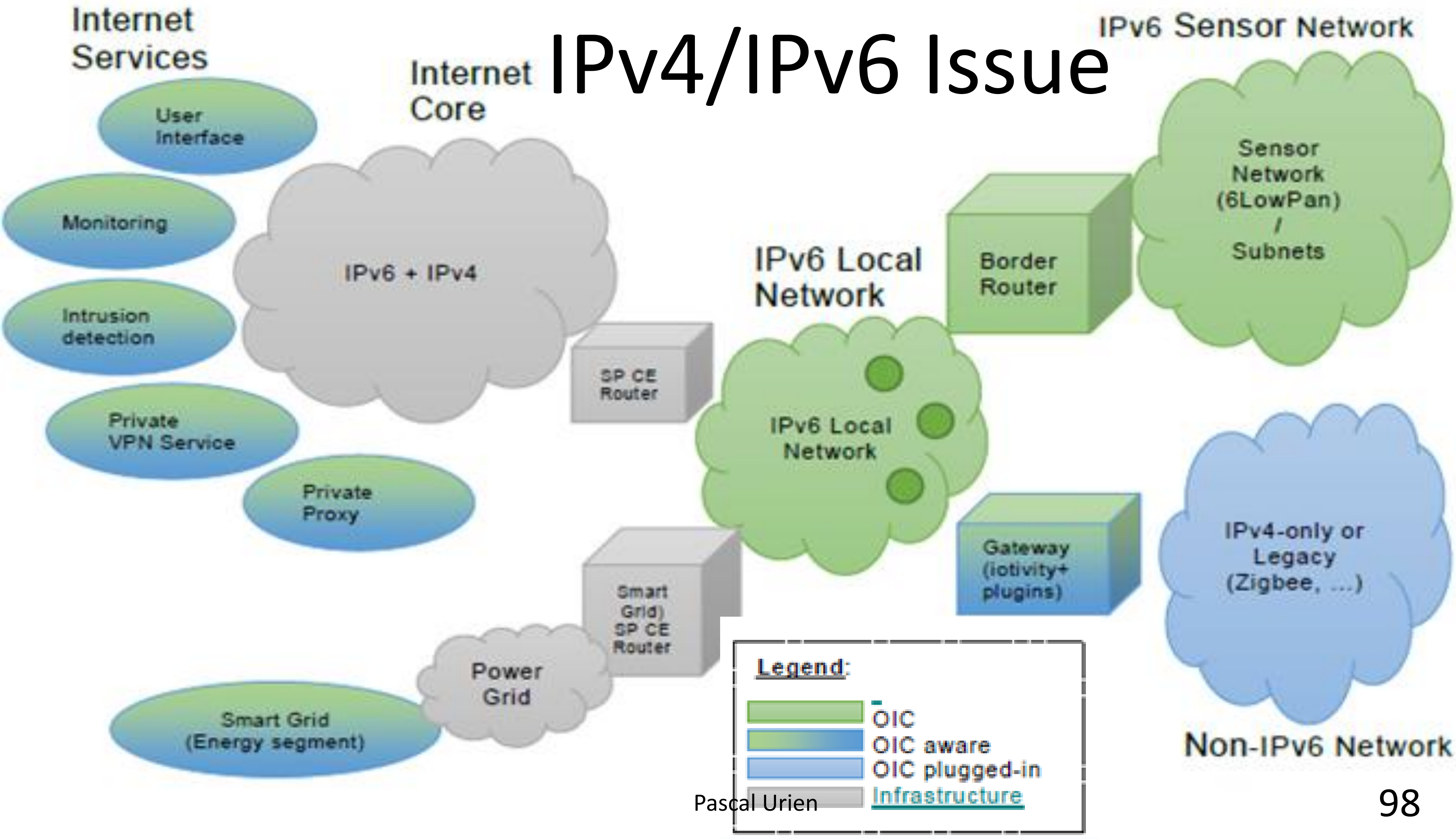
Device Authentication with Raw Asymmetric Key Credentials

Device Authentication with Certificates

Secure Boot

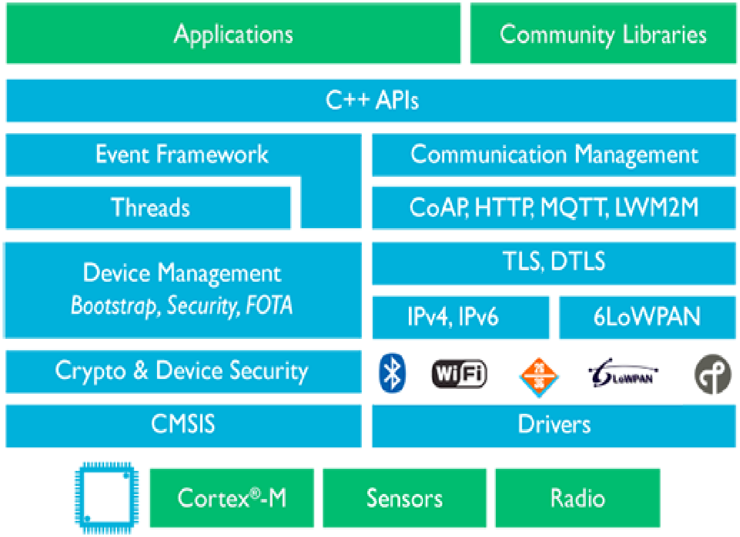
In order to ensure that all components of a device are operating properly and have not been tampered with, it is best to ensure that the device is booted properly. There may be multiple stages of boot. The end result is an application running on top an operating system that takes advantage of memory, CPU and peripherals through drivers.

IPv4/IPv6 Issue



Example 3. MBED

MBED stack from the ARM company



Applications

Community Libraries

C++ APIs

Event Framework

Threads

Communication Management

CoAP, HTTP, MQTT, LWM2M

Device Management
Bootstrap, Security, FOTA

TLS, DTLS

IPv4, IPv6

6LoWPAN

Crypto & Device Security

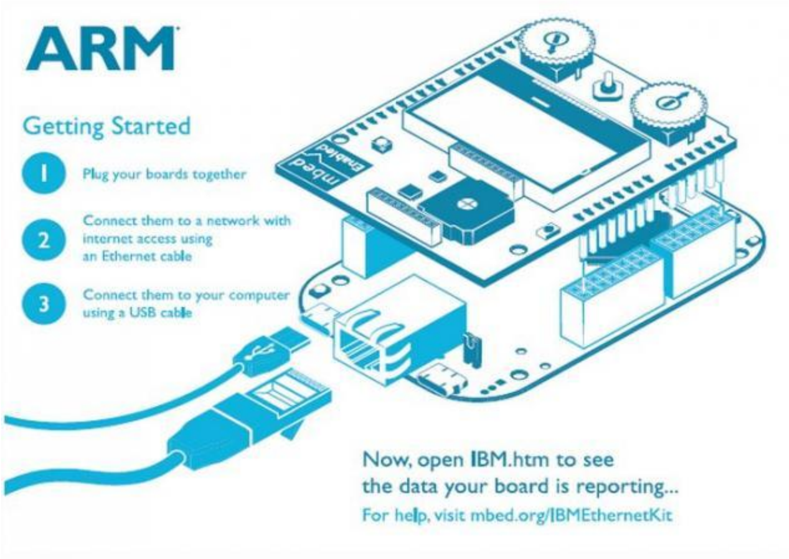
CMSIS

Drivers

Cortex®-M

Sensors

Radio



Getting Started

1 Plug your boards together

2 Connect them to a network with internet access using an Ethernet cable

3 Connect them to your computer using a USB cable

Now, open IBM.htm to see the data your board is reporting...

For help, visit mbed.org/IBMEthernetKit

IoT Protocols

- HTTP (most of today IP objects)
  - As an illustration some connected plugs work with the HNAP (*Home Network Administration Protocol*) protocol based on SOAP and used in CISCO routers. In 2014 HNAP was infected by" The Moon".
- MQTT protocol, is a Client Server publish/subscribe messaging transport protocol that is secured by TLS.

CoAP, RFC 7252

- CoAP (Constrained Application Protocol) , RFC 7252 is designed according to the Representational State Transfer (REST) architecture , which encompasses the following six features:
  - 1) Client-Server architecture;
  - 2) Stateless interaction;
  - 3) Cache operation on the client side;
  - 4) Uniform interface ;
  - 5) Layered system ;
  - 6) Code On Demand.
- CoAP is an efficient RESTfull protocol easy to proxy to/from HTTP, but which is not understood in an IoT context as a general replacement of HTTP.
  - It is natively secured by DTLS (the datagram adaptation of TLS), and works over a DTLS/UDP/IP stack. Nerveless the IETF is currently working on a CoAP version compatible with a TLS/TCP/IP stack.

CoAP Details

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td colspan="2">V</td><td colspan="2">T</td><td colspan="4">TKL</td><td colspan="8">Code</td><td colspan="16">Message ID</td></tr>
<tr><td colspan="32">Token (if any)</td></tr>
<tr><td colspan="32">Options (if any)</td></tr>
<tr><td colspan="8">1 1 1 1 1 1 1 1</td><td colspan="24">Payload (if any)</td></tr>
</table>

Version (V): protocol version (01).

Type (T) message type :

Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) or Reset.

Token Length (TKL)/ is the length of the Token field (0-8 bytes).

The Code field: identifies the method and is split in two parts a 3-bit class and a 5-bit detail

documented as "c.dd" where "c" is a digit from 0 to 7 and "dd" are two digits from 00 to 31.

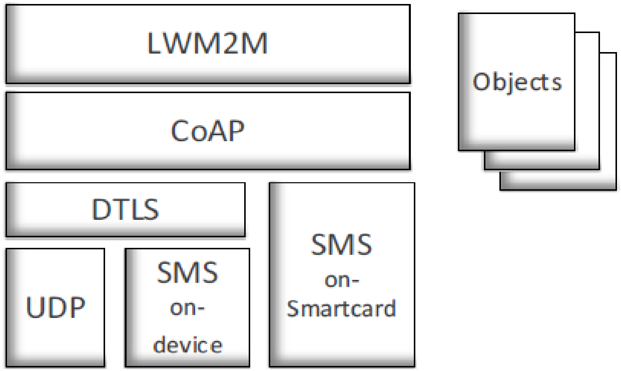
0.01 GET, 0.02 POST, 0.03 PUT and 0.04 DELETE.

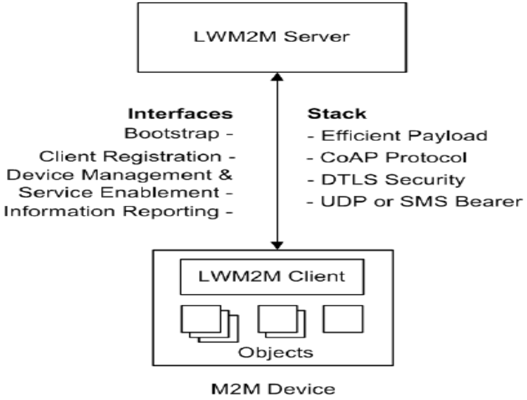
Message ID: matches messages ACK/Reset to messages CON/NON previously sent.

The Token (0 to 8 bytes): is used to match a response with a request.

Options: give additional information such as Content-Format dealing with proxy operations.

LWM2M

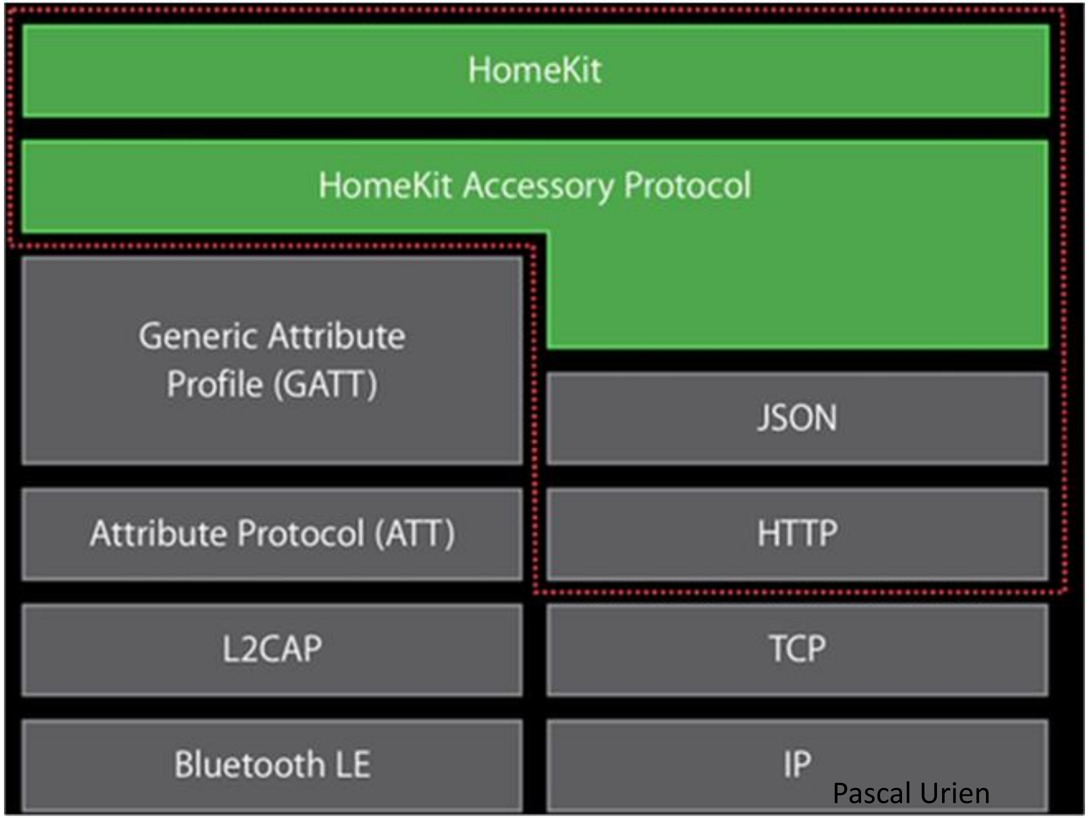




- LWM2M (*Lightweight Machine to Machine Technical Specification)* is a framework based on CoAP dealing with objects hosted by LWM2M clients and communicating with LWM2M servers
- LWM2M manages the following interfaces
  - Bootstrap
  - Client Registration (with servers)
  - Device management
  - Information Reporting
- Two transport mechainsm ("*transport channel bindings“)*
  - *UDP/IP*
  - *SMS*

Example 4. Home Kit

HOME Kit (Apple)



Protocol Security
- End-to-end encryption
- Initial setup secured directly between iOS and accessory
- Perfect forward secrecy
- Standard cryptography

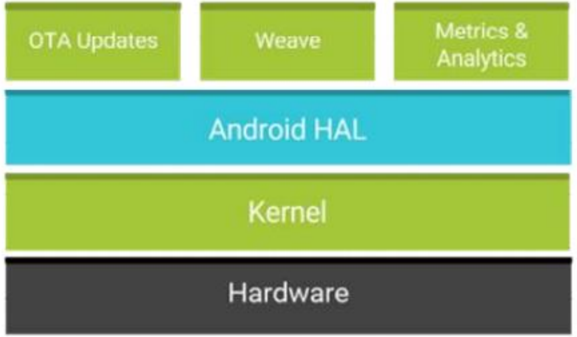
The HAP (*HomeKit Accessory Protocol*) initial pairing exchange is based on the Secure Remote Password procedure (SRP, RFC 5054) which deals with a 8 digits PIN code available for every accessory.

Example 5. Brillo & Weave

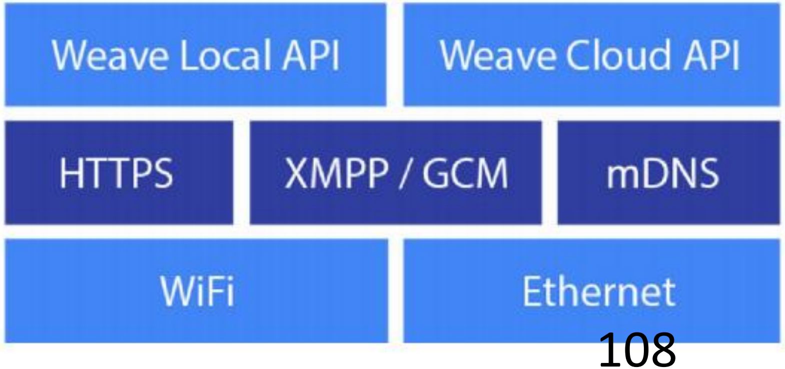
Brillo & Weave

The Intel® Edison Board Made for Brillo.

Brillo is an OS from Google for building connected devices. 35MB Memory Footprint (minimum)



Weave is a communications protocol that supports discovery, provisioning, and authentication so that devices can connect and interact with one another, the Internet, and your mobile platforms.

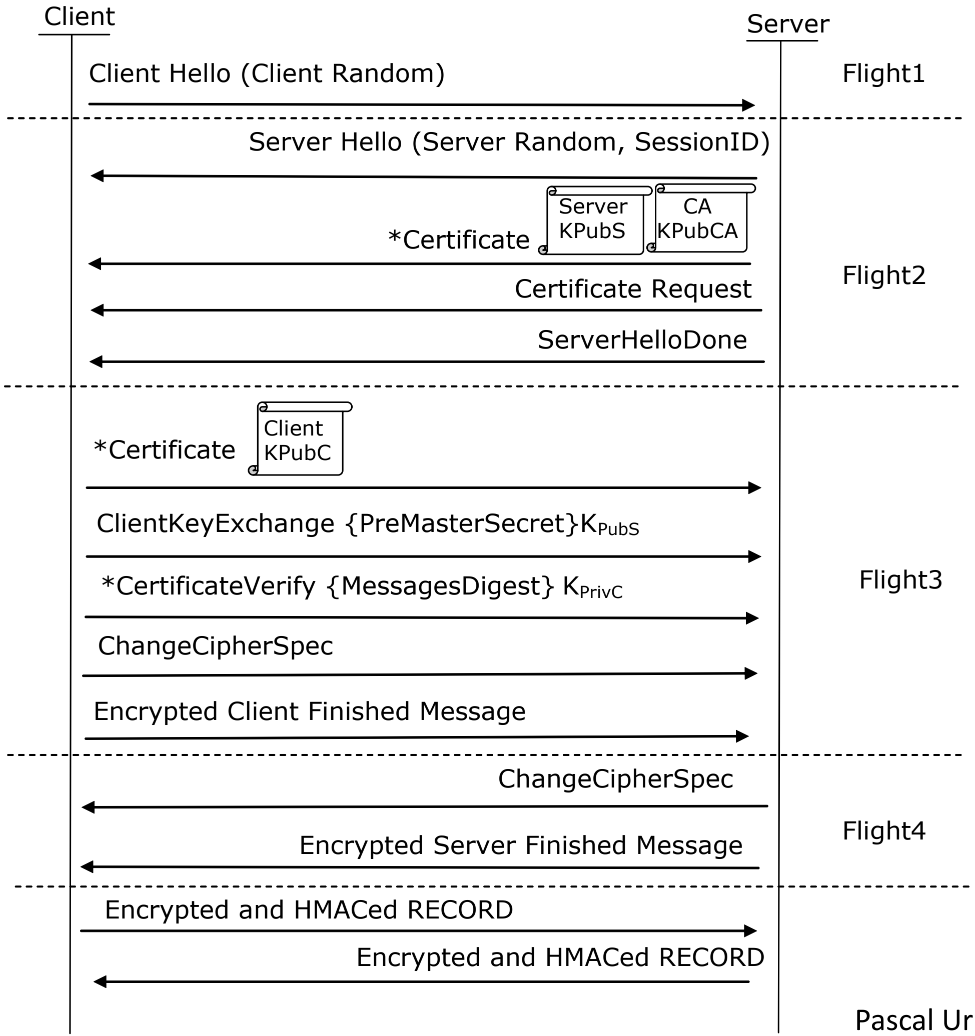


Brillo and Weave

Weave is a communications platform for IoT devices
- Device setup, phone-to-device-to-cloud communication
- User interaction from mobile devices and the web
- Transports: 802.15.4 (zigbee, threads), BLE, WiFi, Ethernet, Others possible
- Schema Driven (json) Associates Weave XMPP requests with application function invocations
- Web apps may be written with Google* API support
- OAuth 2.0 Authentication, Google as AS

Brillo is Simpler… Smaller…IoT Focused
- C/C++ environment
- Binder IPC No Java Applications, framework, runtime
-No Graphics
- 35MB Memory Footprint (minimum)

About TLS



Client

Server

Client Hello (Client Random)

Flight1

Server Hello (Server Random, SessionID)

*Certificate

Server KPubS

CA KPubCA

Certificate Request

Flight2

ServerHelloDone

*Certificate

Client KPubC

ClientKeyExchange {PreMasterSecret}K_{PubS}

*CertificateVerify {MessagesDigest} K_{PrivC}

Flight3

ChangeCipherSpec

Encrypted Client Finished Message

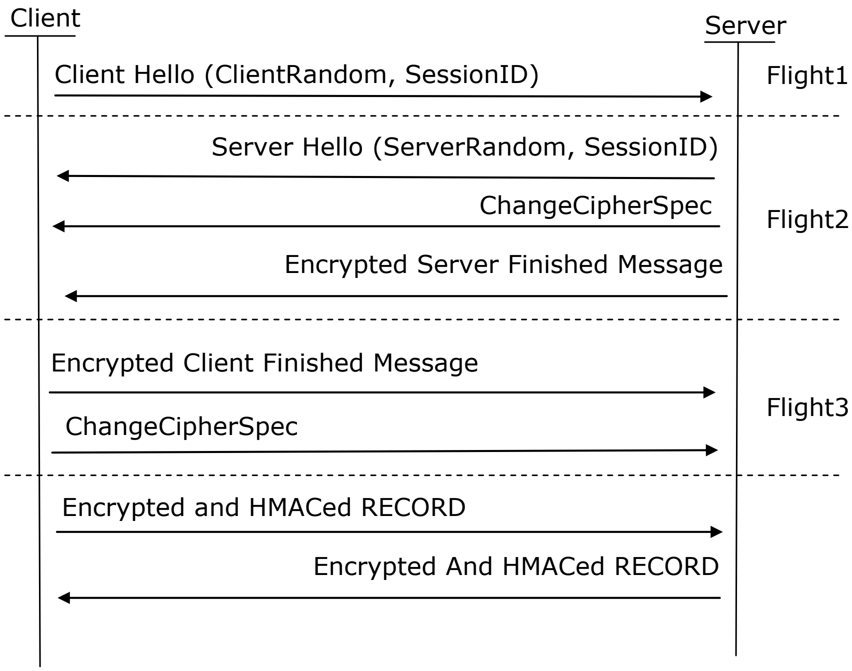
ChangeCipherSpec

Flight4

Encrypted Server Finished Message

Encrypted and HMACed RECORD

Encrypted and HMACed RECORD



Client

Server

Client Hello (ClientRandom, SessionID)

Flight1

Server Hello (ServerRandom, SessionID)

ChangeCipherSpec

Flight2

Encrypted Server Finished Message

Encrypted Client Finished Message

Flight3

ChangeCipherSpec

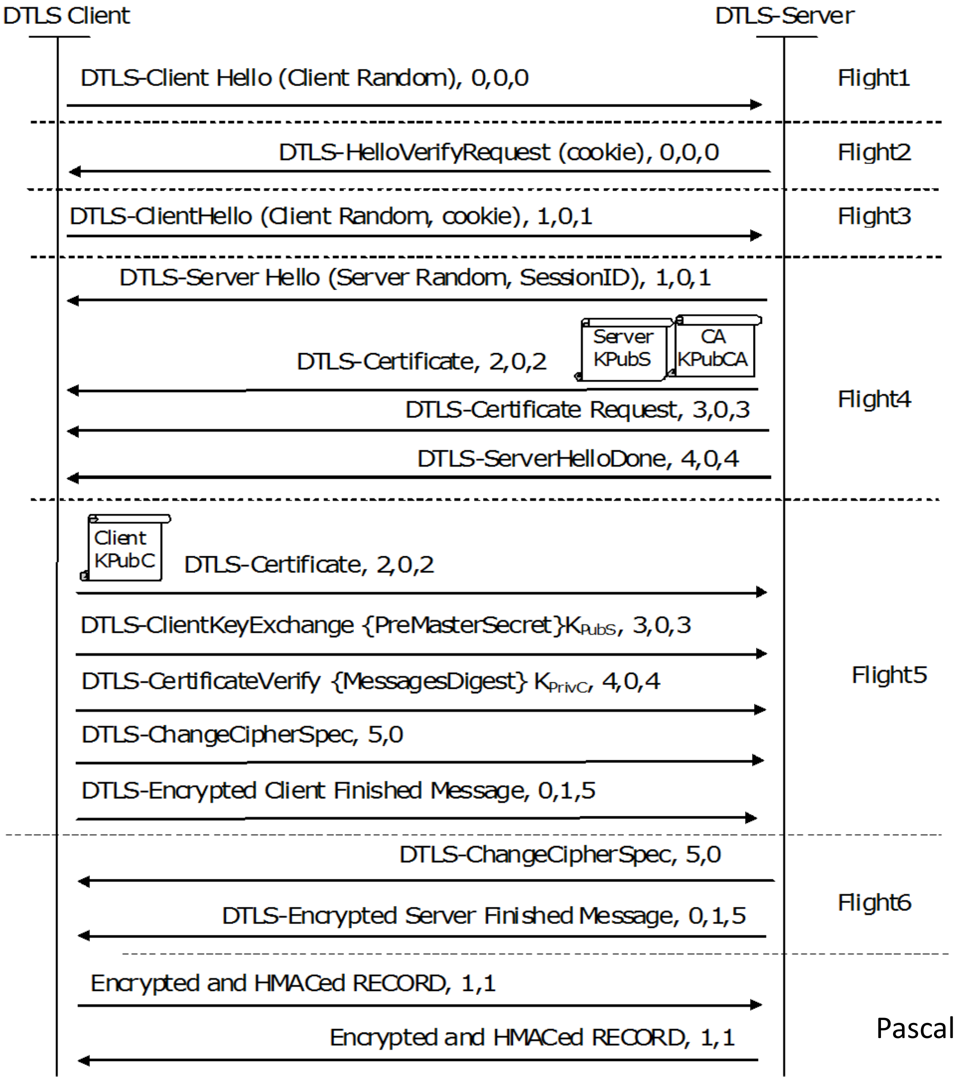
Encrypted and HMACed RECORD

Encrypted And HMACed RECORD

Pascal Urien

TLS/DTLS Security Modules

About DTLS



DTLS Client

DTLS-Server

DTLS-Client Hello (Client Random), 0,0,0

Flight1

DTLS-HelloVerifyRequest (cookie), 0,0,0

Flight2

DTLS-ClientHello (Client Random, cookie), 1,0,1

Flight3

DTLS-Server Hello (Server Random, SessionID), 1,0,1

DTLS-Certificate, 2,0,2

Server KPubS

CA KPubCA

DTLS-Certificate Request, 3,0,3

DTLS-ServerHelloDone, 4,0,4

Flight4

Client KPubC

DTLS-Certificate, 2,0,2

DTLS-ClientKeyExchange {PreMasterSecret}K_{PubS}, 3,0,3

DTLS-CertificateVerify {MessagesDigest} K_{PrivC}, 4,0,4

DTLS-ChangeCipherSpec, 5,0

DTLS-Encrypted Client Finished Message, 0,1,5

Flight5

DTLS-ChangeCipherSpec, 5,0

DTLS-Encrypted Server Finished Message, 0,1,5

Flight6

Encrypted and HMACed RECORD, 1,1

Encrypted and HMACed RECORD, 1,1

The two first number are respectively the record sequence number and the *epoch* field.

The optional third number is the message sequence used by a handshake message.

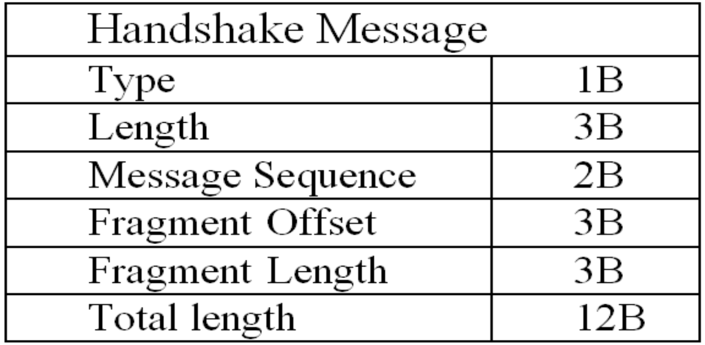
DTLS cryptographic details

Handshake cryptographic calculations are insensitive to fragmentation operations.

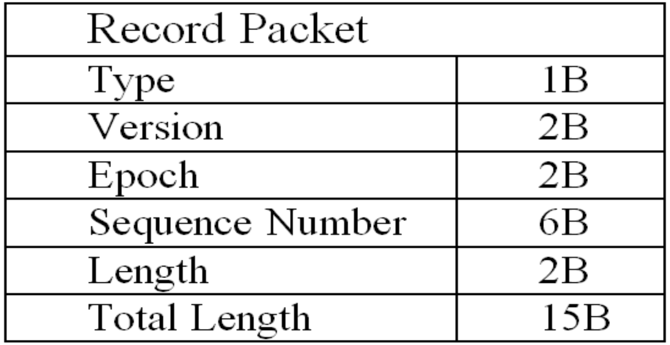
According to finished messages (either client or server) have no sensitivity to fragmentation. There are computed as if each handshake message had been sent as a single fragment, i.e. with *Fragment-Length* set to Length, and *Fragment-Offset* set to zero ; the *Message-Sequence* field is not used in these procedures.

It also should be noticed that the DTLS-HelloVerifyRequest message and the previous associated DTLS-ClientHello are not taken into account by the Handshake cryptographic calculation.

DTLS Handshake and Record Layer

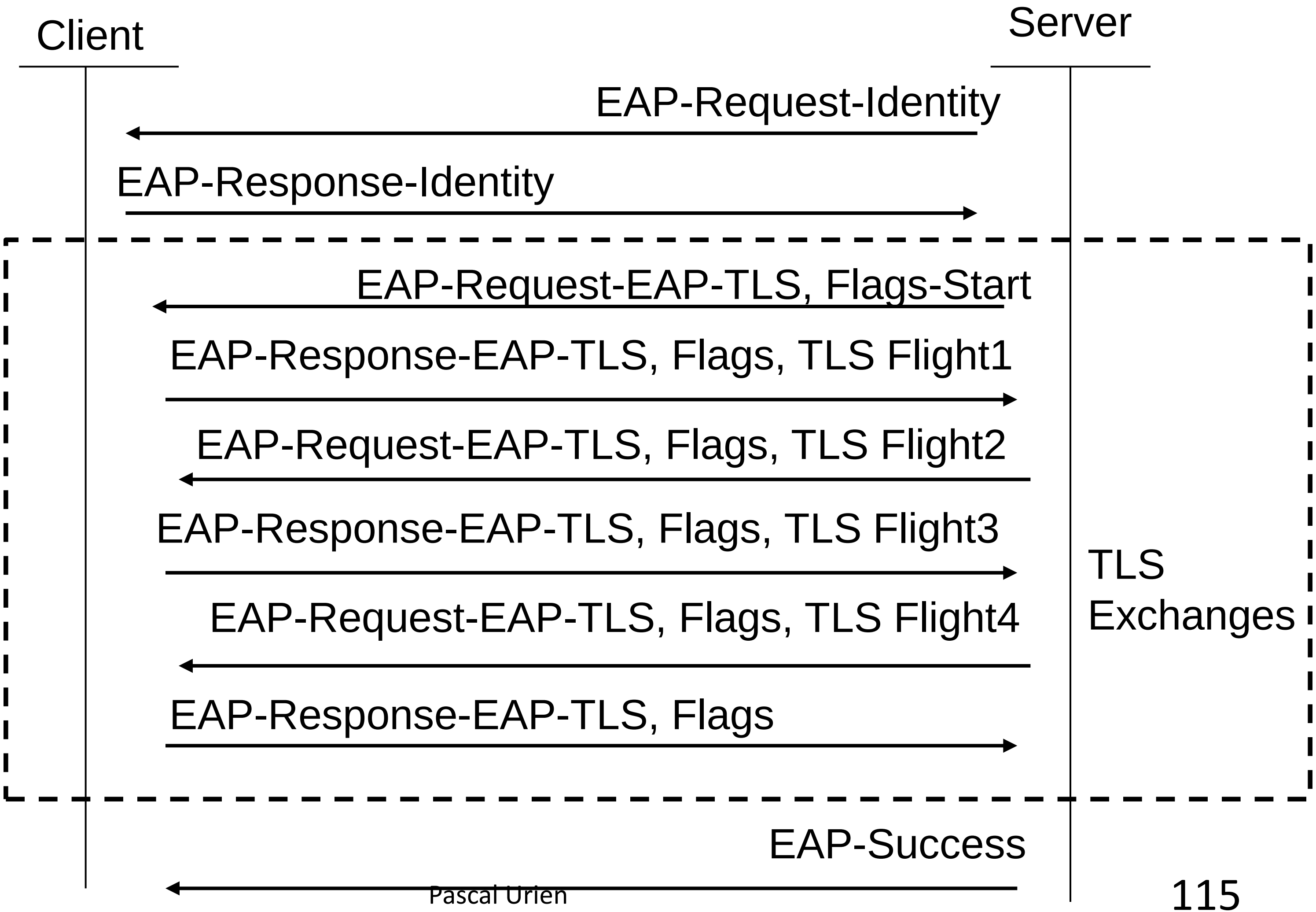


Handshake Message	
Type	1B
Length	3B
Message Sequence	2B
Fragment Offset	3B
Fragment Length	3B
Total length	12B

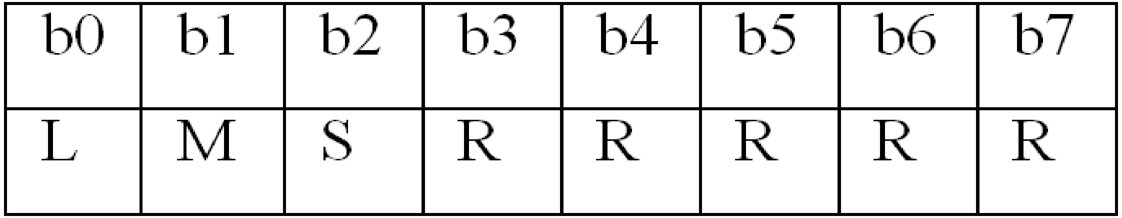


Record Packet	
Type	1B
Version	2B
Epoch	2B
Sequence Number	6B
Length	2B
Total Length	15B

About EAP-TLS


Client
Server
EAP-Request-Identity
EAP-Response-Identity
EAP-Request-EAP-TLS, Flags-Start
EAP-Response-EAP-TLS, Flags, TLS Flight1
EAP-Request-EAP-TLS, Flags, TLS Flight2
EAP-Response-EAP-TLS, Flags, TLS Flight3
EAP-Request-EAP-TLS, Flags, TLS Flight4
EAP-Response-EAP-TLS, Flags
TLS Exchanges
EAP-Success

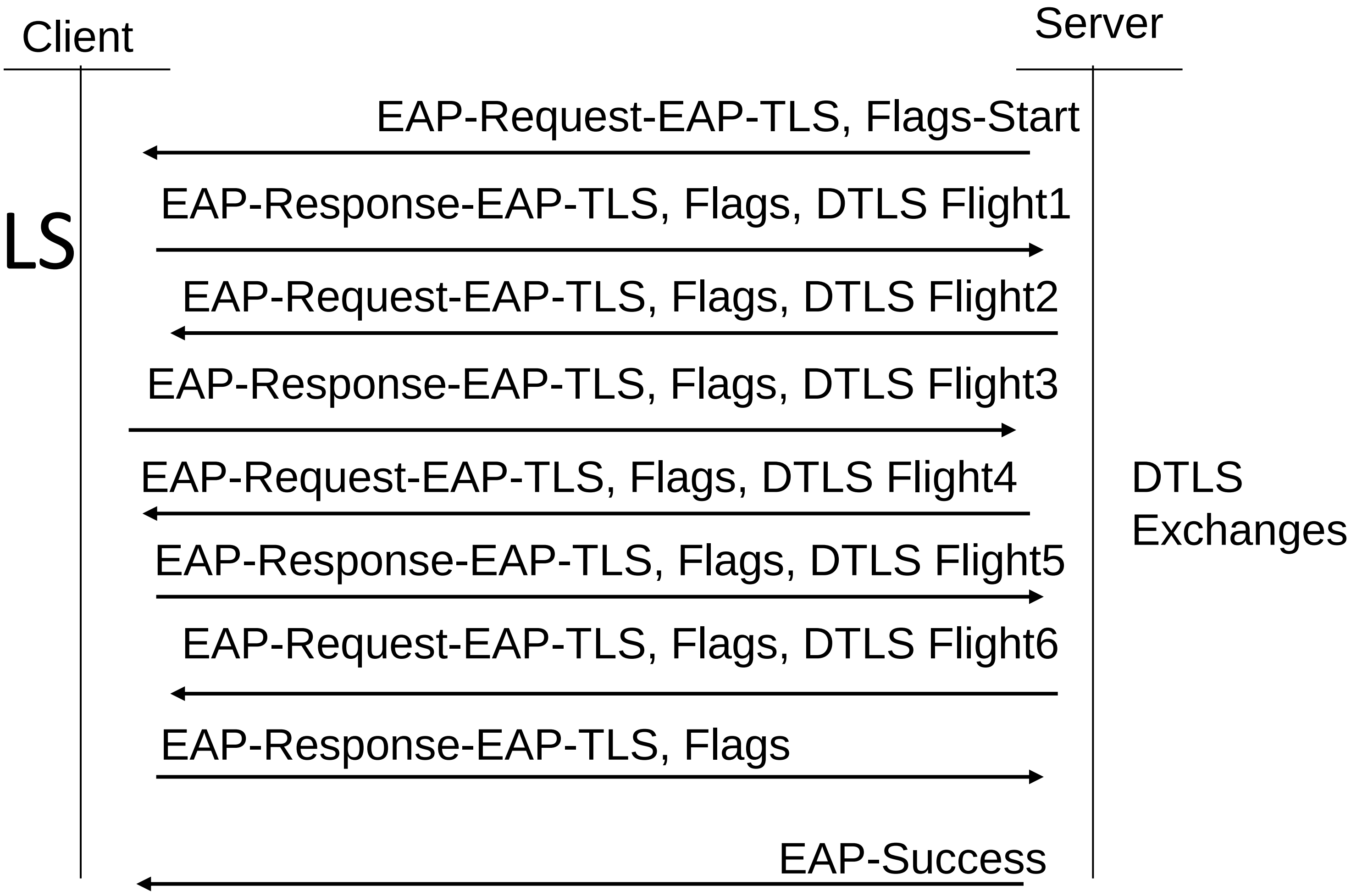
EAP-TLS Flags Field Segmentation Reassembly Procedures



b0	b1	b2	b3	b4	b5	b6	b7
L	M	S	R	R	R	R	R

- The L bit (length included) is set to indicate the presence of the four-octet TLS flight length field, and is set for the first fragment of a fragmented TLS message or set of messages.
- The M bit (more fragments) is set on all but the last fragment.
- The S bit (EAP-TLS start) is set in an EAP-TLS Start message.

EAP-DTLS



Client

Server

EAP-Request-EAP-TLS, Flags-Start

EAP-Response-EAP-TLS, Flags, DTLS Flight1

EAP-Request-EAP-TLS, Flags, DTLS Flight2

EAP-Response-EAP-TLS, Flags, DTLS Flight3

EAP-Request-EAP-TLS, Flags, DTLS Flight4

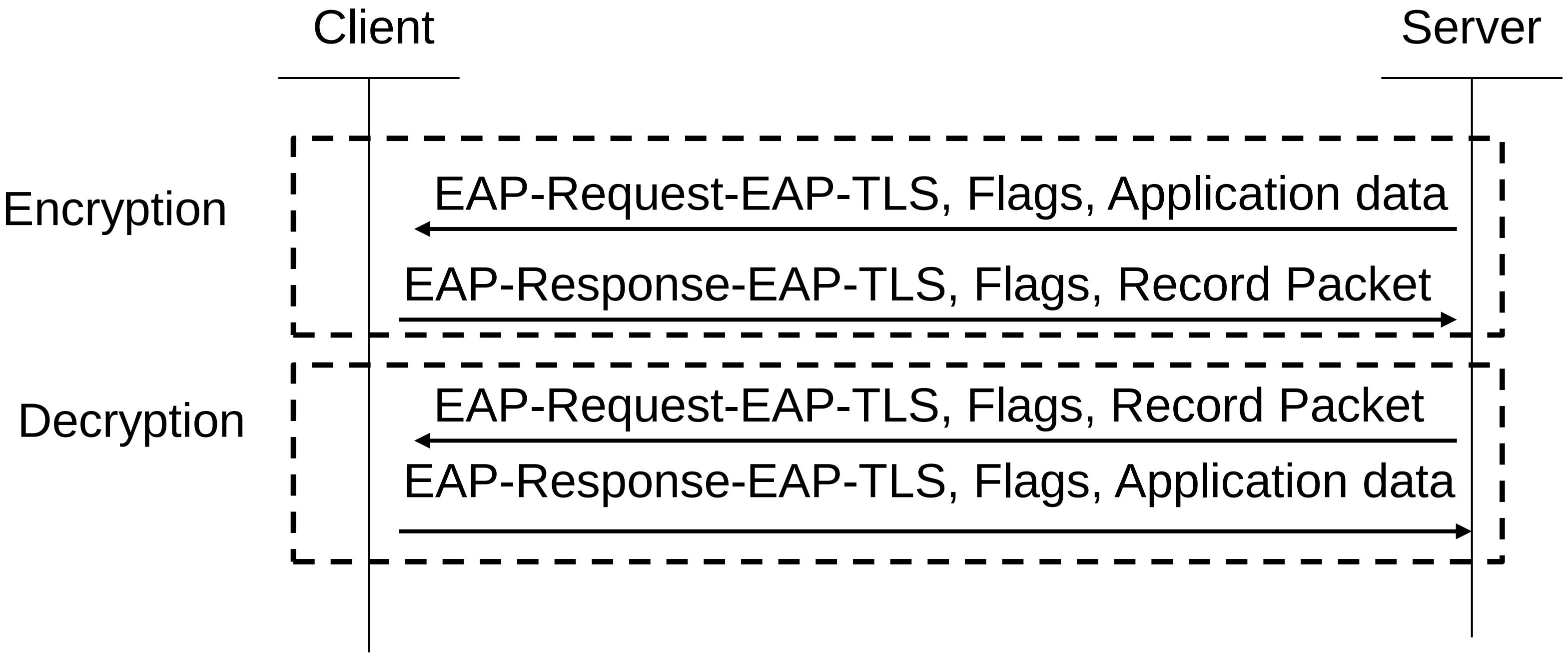
EAP-Response-EAP-TLS, Flags, DTLS Flight5

EAP-Request-EAP-TLS, Flags, DTLS Flight6

EAP-Response-EAP-TLS, Flags

EAP-Success

DTLS
Exchanges


Client
Server
Encryption
EAP-Request-EAP-TLS, Flags, Application data
EAP-Response-EAP-TLS, Flags, Record Packet
Decryption
EAP-Request-EAP-TLS, Flags, Record Packet
EAP-Response-EAP-TLS, Flags, Application data

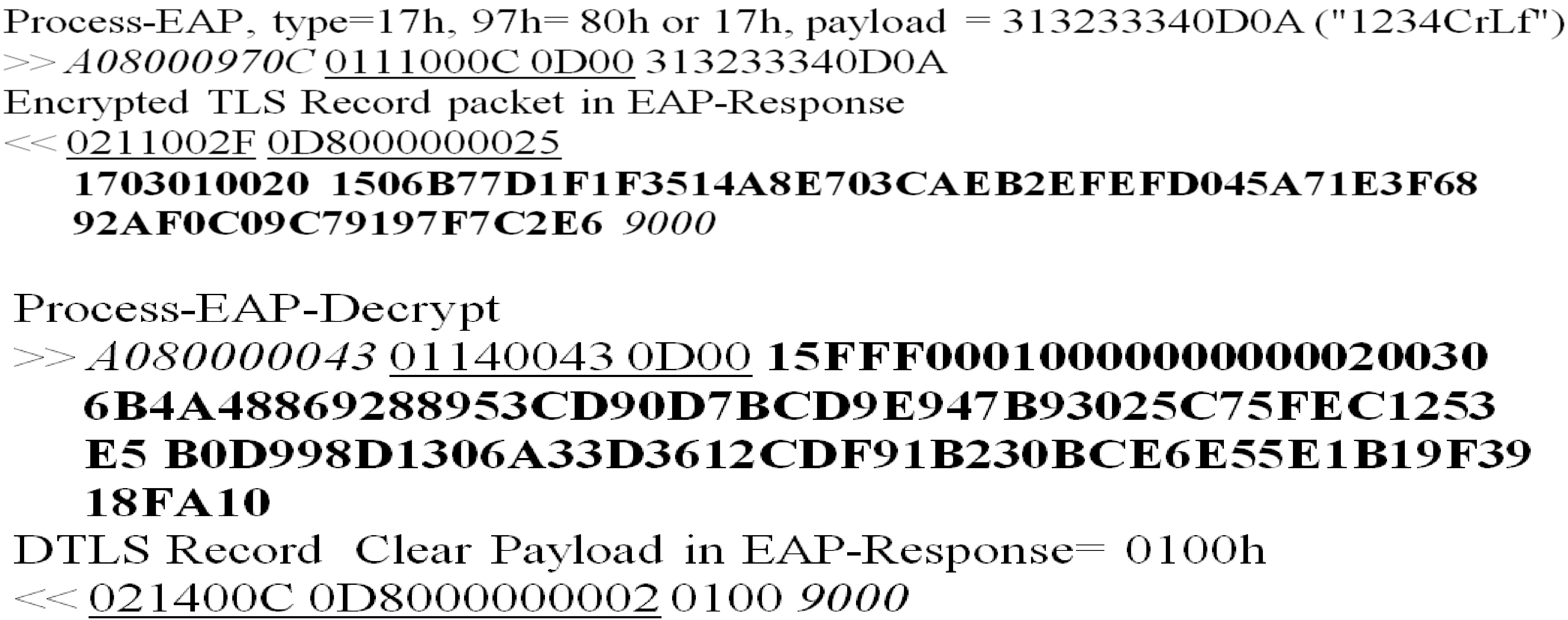
About Secure Elements

- Secure Elements are tamper resistant microcontrollers, whose security is enforced by multiple hardware and software countermeasures.
- Their security level is ranked by evaluations performed according to the Common Criteria standards, whose level range from one to seven.
- The chip area is typically 25mm^2 (5mm x 5mm). The power consumption is low , as an illustration for SIM module 1.8V-0,2 mA (3.6mw) in idle state and no more than 1.8V-60mA (108 mW) in pike activity.

About Secure Elements

- Secure microcontrollers comprise a few hundred KB of ROM, about one hundred KB of non volatile memory (E^2PROM, Flash) and a few KB of RAM.
- Most of them include a Java Virtual Machine and therefore run applications written in the Javacard language, a subset of the java language.
- A TLS/DTLS stack is an application, typically a javacard application, stored and executed in a secure element. Its logical interface is a set of APDUs exchanged over the IO link.
- We previously designed EAP-TLS smartcards, which compute TLS flights encapsulated in EAP-TLS messages, until the generation of server and client finished messages.

Illustration of (TLS) Encryption and (DTLS) Decryption Operations



Process-EAP, type=17h, 97h= 80h or 17h, payload = 313233340D0A ("1234CrLf")

>> *A08000970C* 0111000C 0D00 313233340D0A

Encrypted TLS Record packet in EAP-Response

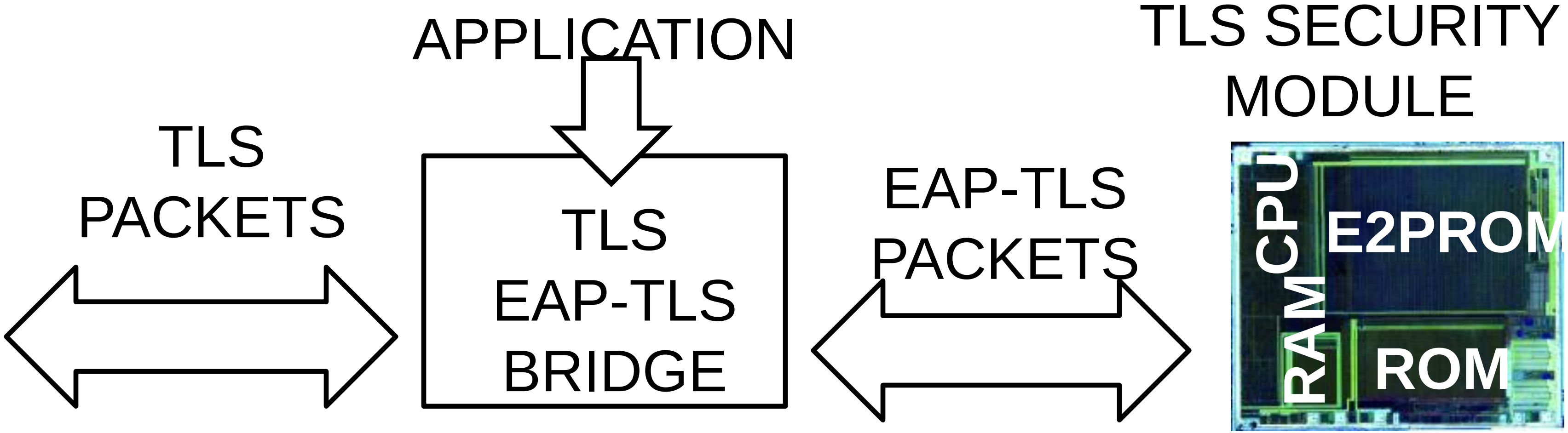
<< 0211002F 0D8000000025
1703010020 1506B77D1F1F3514A8E703CAEB2EFEFD045A71E3F68
92AF0C09C79197F7C2E6 *9000*

Process-EAP-Decrypt

>> *A080000043* 01140043 0D00 **15FFF00010000000000020030**
6B4A488692889 53CD90D7BCD9E947B93025C75FEC1253
E5 B0D998D1306A33D3612CDF91B230BCE6E55E1B19F39
18FA10

DTLS Record Clear Payload in EAP-Response= 0100h

<< 021400C 0D8000000002 0100 *9000*



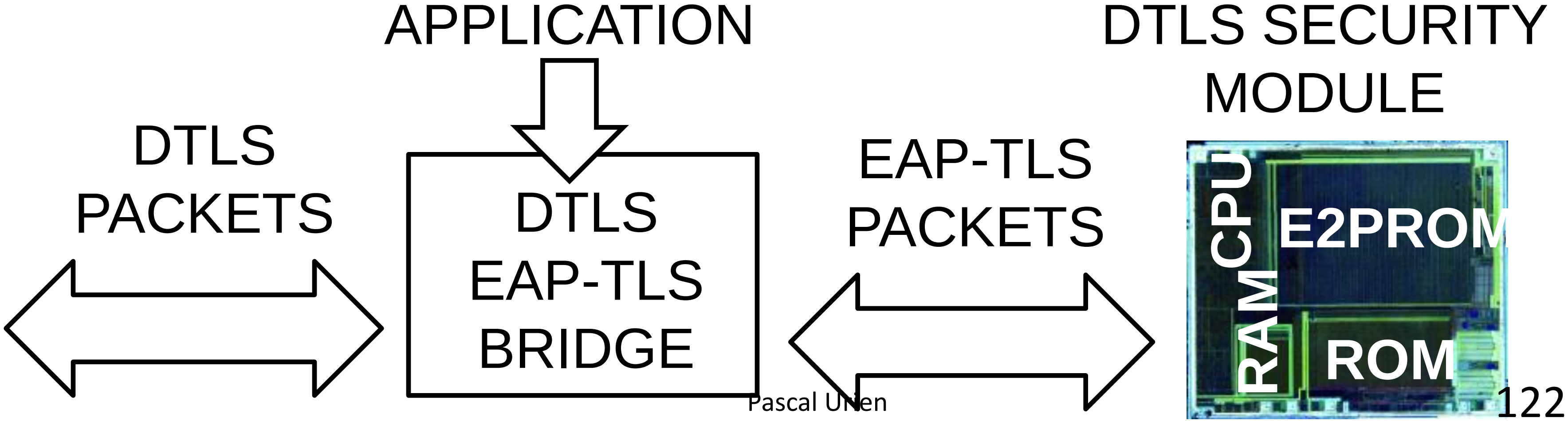
APPLICATION

TLS SECURITY MODULE

TLS PACKETS

TLS EAP-TLS BRIDGE

EAP-TLS PACKETS



APPLICATION

DTLS SECURITY MODULE

DTLS PACKETS

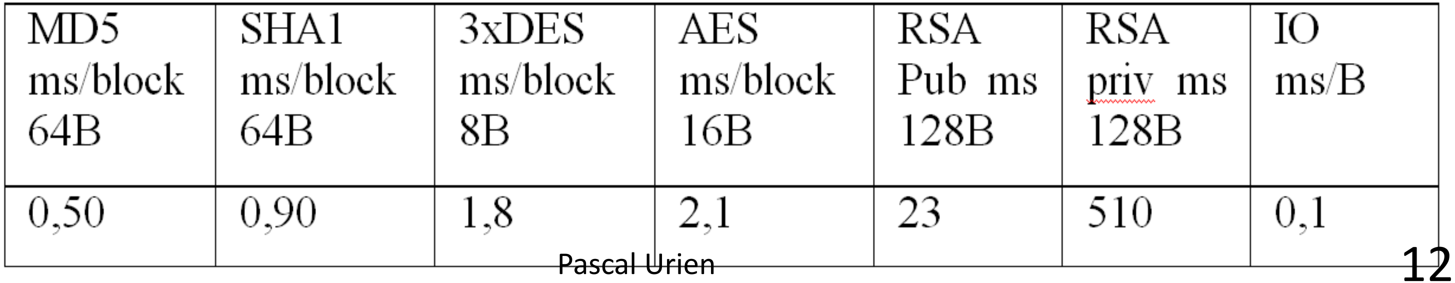
DTLS EAP-TLS BRIDGE

EAP-TLS PACKETS

Experimental Platform

The cryptographic module (Gemalto TOP-IM_GX4)is based on the Samsung S3CC9TC chip. It includes:

- a 16 bits CPU
- 72 KB of EEPROM
- 384 KB of ROM
- 8 KB of RAM for the CPU
- 2 KB of RAM for the crypto processor



MD5 ms/block 64B	SHA1 ms/block 64B	3xDES ms/block 8B	AES ms/block 16B	RSA Pub ms 128B	RSA priv ms 128B	IO ms/B
0,50	0,90	1,8	2,1	23	510	0,1

Performances

The booting of a TLS/DTLS session (until the delivering of finished messages) should cost about 878 ms (1300 ms measured) consumed by the following operations:

- 556 ms for RSA procedures, one RSA private key encryption and two public key decryptions (510+ 23 + 24)
- -322 ms for hash procedures, requiring the computing of 230 MD5 et 230 SHA1 block

The measured time for a resume session (75 SHA blocks+ 75 MD5 blocks = 105 ms) setting is 360 ms

Performances

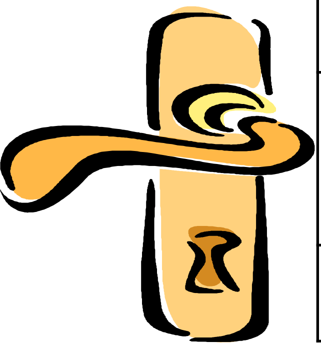
The processing of encrypted record packets, with a 1024 bytes size, should require about 143 ms (415 ms measured), according to the following relations :

- 135 ms (64 x 2,1) for the encryption/decryption of 64 blocks of data.

- 18 ms (20 x 0,9) for the HMAC (SHA1) processing of 20 (16+4) blocks of data

Example of Application

eLock



COAP

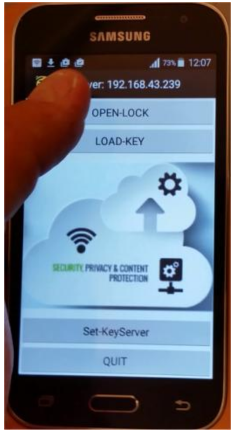
DTLS Server

NFC

COAP

SIM DTLS Client

NFC



HTTP

SIM TLS

TCP

IP

KeyServer

“Innovative DTLS/TLS Security Modules Embedded in SIM Cards for IoT Trusted and Secure Services”, to appear, IEEE CCNC 2016

USE Case 1. CoAP Key

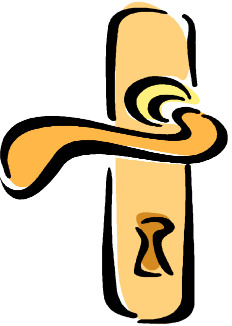
Secure Element as a CoAP Client

Secure Element as a TLS client

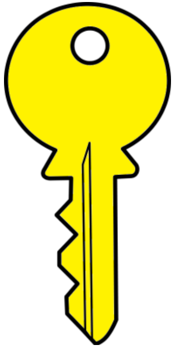
Urien, P.; "Innovative DTLS/TLS Security Modules Embedded in SIM Cards for IoT Trusted and Secure Services", IEEE CCNC 2016, Las Vegas, NV, USA

Urien, P.; "Towards Secure Elements For The Internet of Things: The eLock Use Case", IEEE MobiSecServ 2016, Gainesville, FL, USA

Issues to Solve



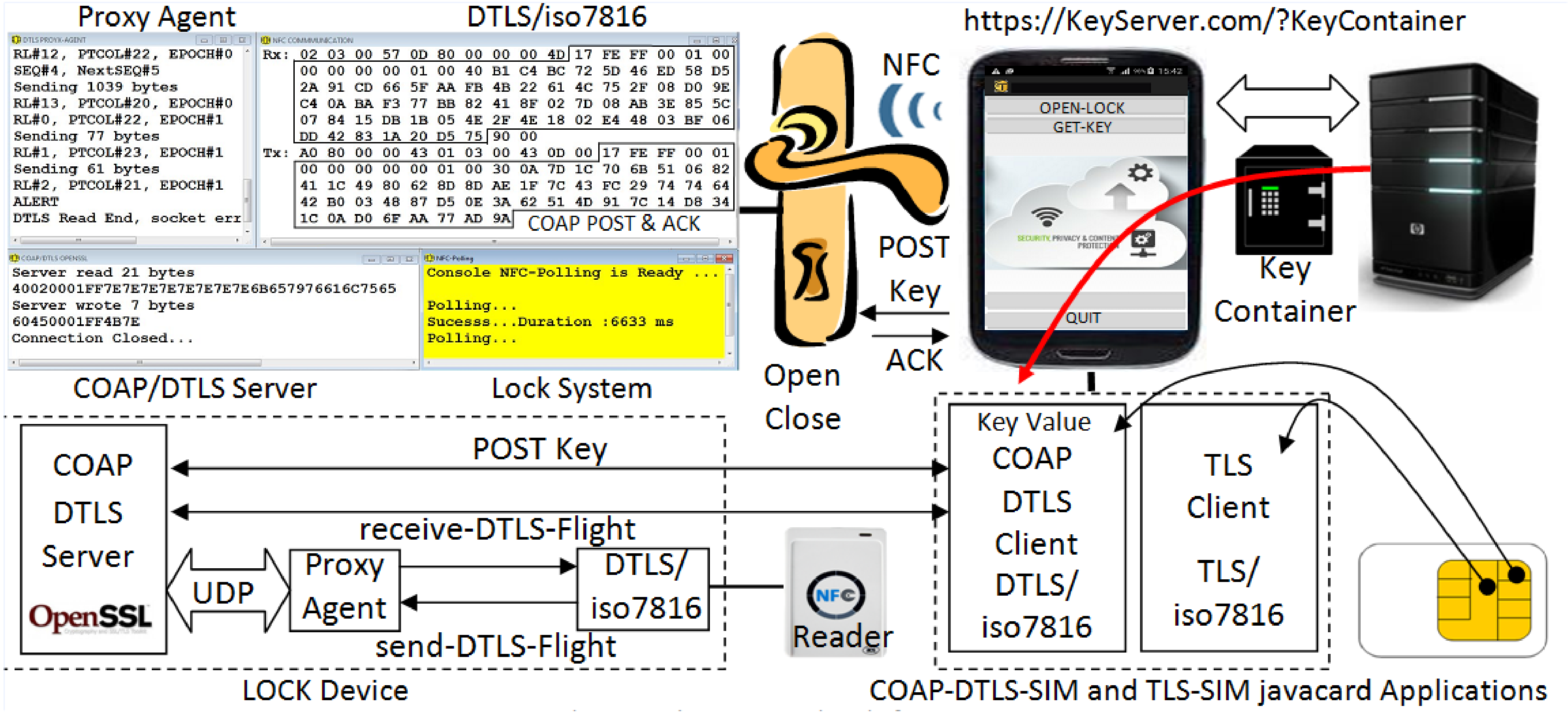
In the Internet of Thing (IoT) a lock is a COAP Server



So the Key is a COAP Client

- What is a Key in the Internet of Things ?
  - A Mobile Application ?
- Where is stored the Key ?
  - In a Secure Element (SIM)
- Who is generating the Key ?
  - A Key server generates KeyContainers
- What about security and trust
  - – COAP client and dual TLS/DTLS stack are running in a Secure Element

IEEE CCNC 2016 Demonstration



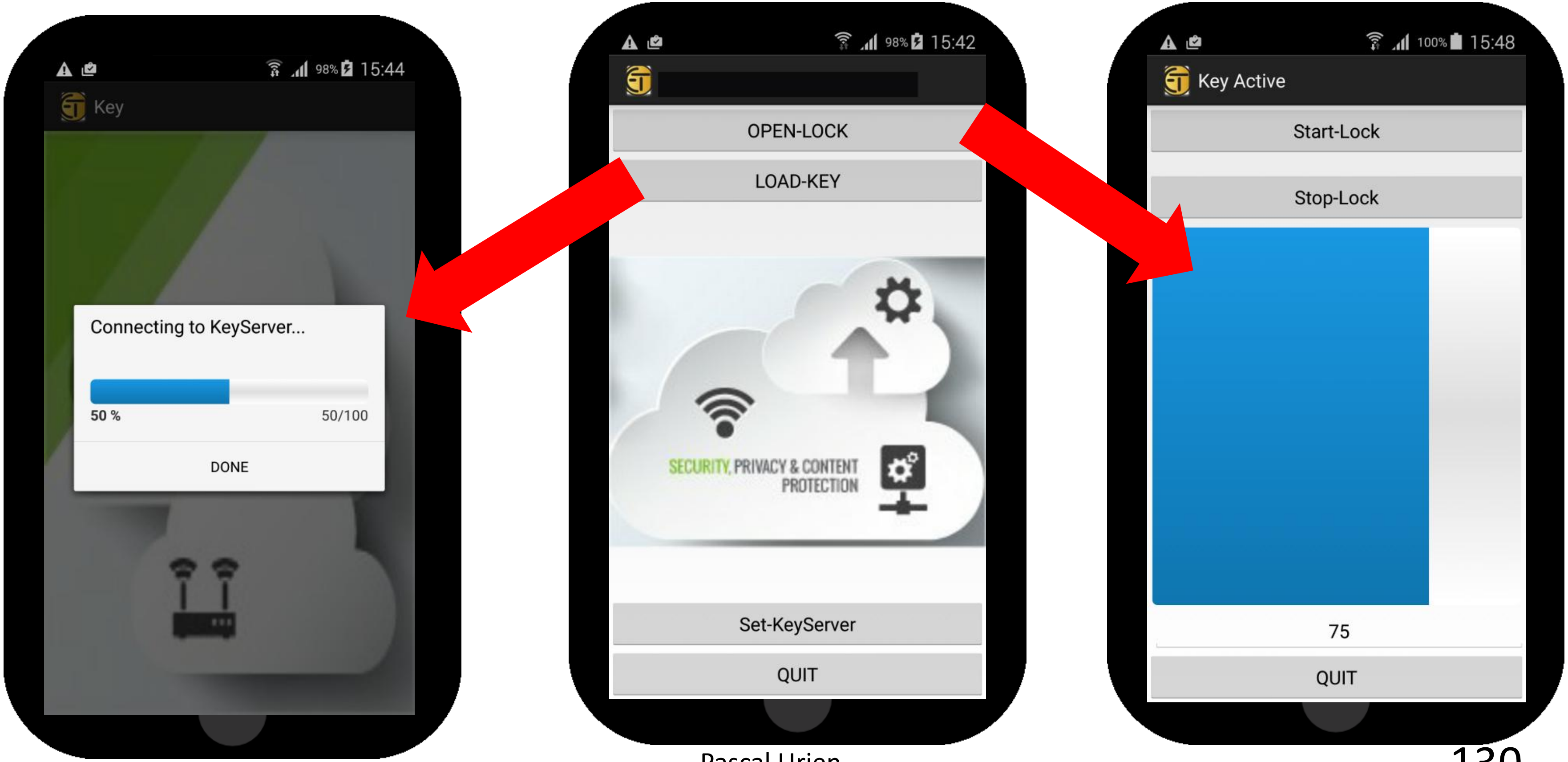
```
RL#12, PTCOL#22, EPOCH#0
SEQ#4, NextSEQ#5
Sending 1039 bytes
RL#13, PTCOL#20, EPOCH#0
RL#0, PTCOL#22, EPOCH#1
Sending 77 bytes
RL#1, PTCOL#23, EPOCH#1
Sending 61 bytes
RL#2, PTCOL#21, EPOCH#1
ALERT
DTLS Read End, socket err
```

```
Rx: 02 03 00 57 0D 80 00 00 00 4D 17 FE FF 00 01 00
    00 00 00 00 01 00 40 B1 C4 BC 72 5D 46 ED 58 D5
    2A 91 CD 66 5F AA FB 4B 22 61 4C 75 2F 08 D0 9E
    C4 0A BA F3 77 BB 82 41 8F 02 7D 08 AB 3E 85 5C
    07 84 15 DB 1B 05 4E 2F 4E 18 02 E4 48 03 BF 06
    DD 42 83 1A 20 D5 75 90 00
Tx: A0 80 00 00 43 01 03 00 43 0D 00 17 FE FF 00 01
    00 00 00 00 00 01 00 30 0A 7D 1C 70 6B 51 06 82
    41 1C 49 80 62 8D 8D AE 1F 7C 43 FC 29 74 74 64
    42 B0 03 48 87 D5 0E 3A 62 51 4D 91 7C 14 D8 34
    1C 0A D0 6F AA 77 AD 9A
```

```
Server read 21 bytes
40020001FF7E7E7E7E7E7E7E7E6B657976616C7565
Server wrote 7 bytes
60450001FF4B7E
Connection Closed...
```

```
Console NFC-Polling is Ready ...

Polling...
Sucesss...Duration :6633 ms
Polling...
```

User's Experience

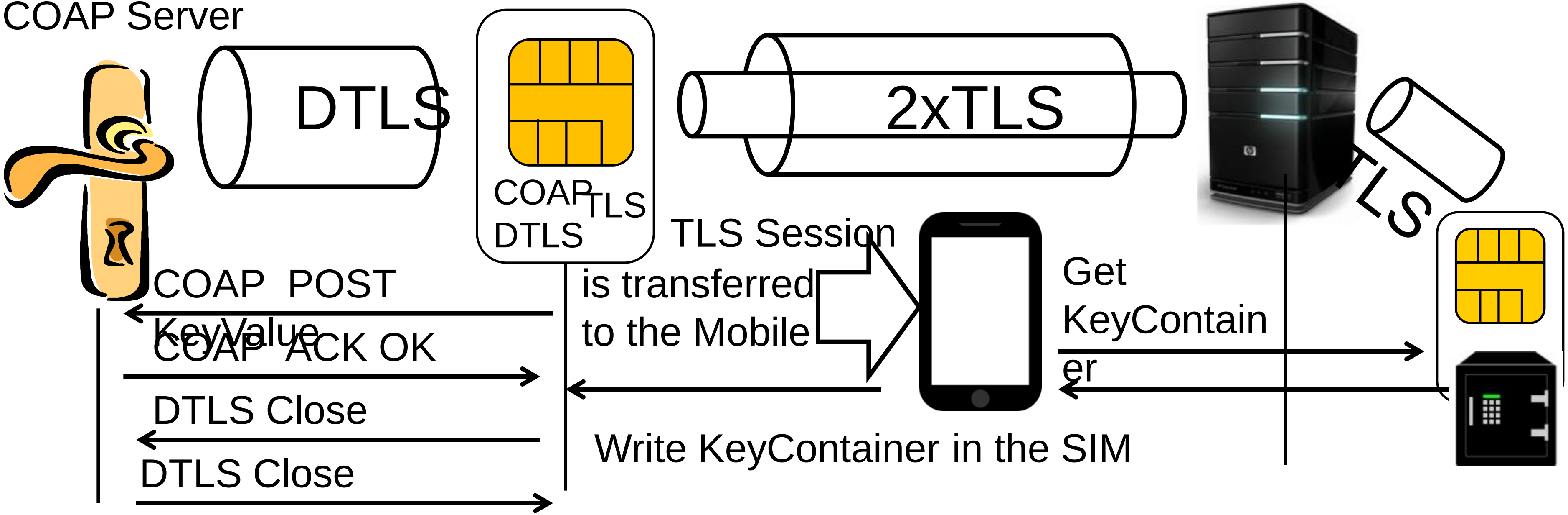


Connecting to KeyServer...

50 % 50/100

Pascal Urien

Double TLS with RACS Server for Key Provisionning



SIM COAP
TLS/DTLS Security Module

NFC Lock
COAP Server

RACS Server

DTLS

2xTLS

TLS

COAP
DTLS

TLS

TLS Session
is transferred
to the Mobile

COAP POST
KeyValue

COAP ACK OK

DTLS Close

DTLS Close

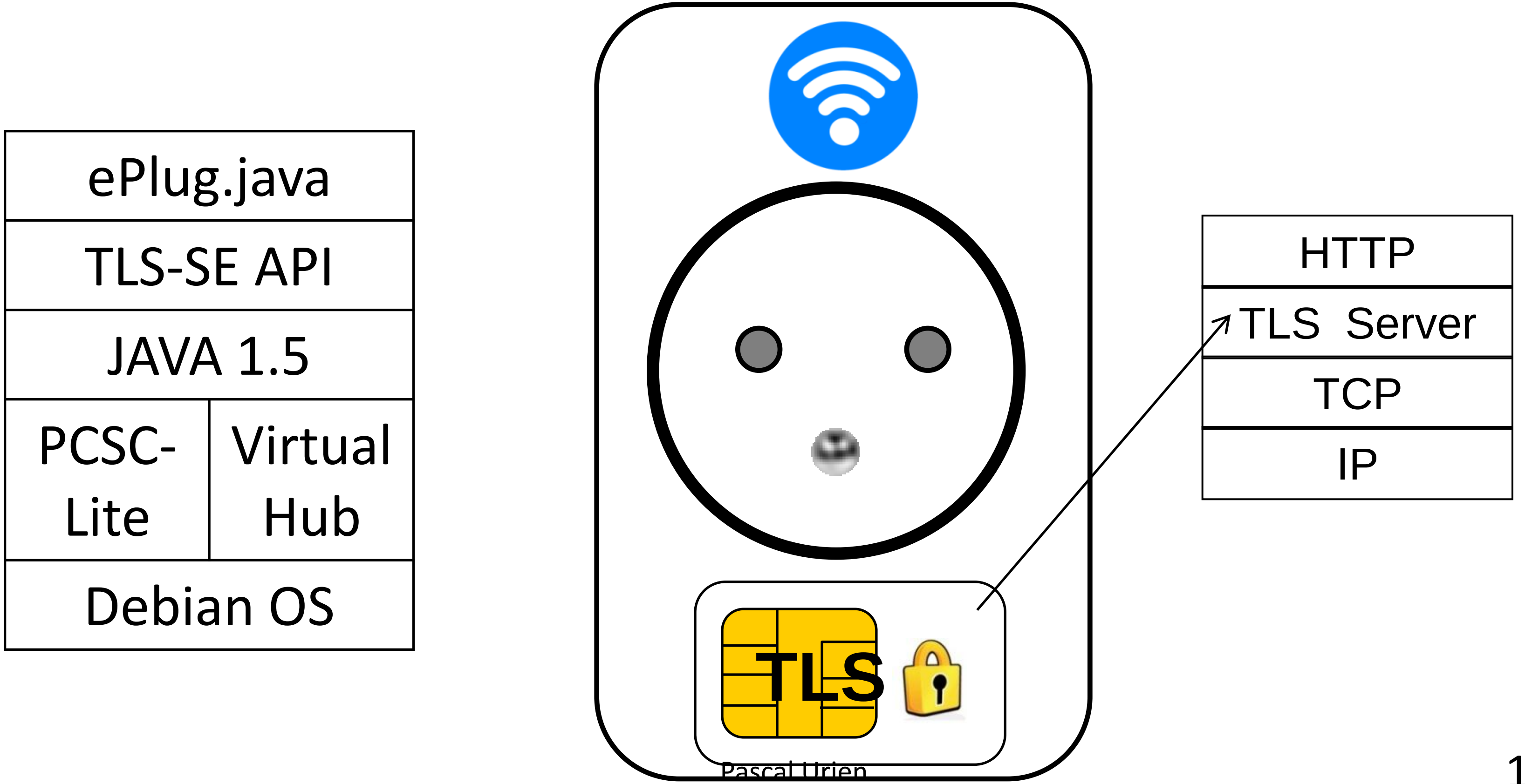
Get
KeyContain
er

Write KeyContainer in the SIM

Use Case 2. TLS Server for Operated Connected Plug

Secure Element as TLS Server Stack

A Connected Plug



ePlug.java

TLS-SE API

JAVA 1.5

PCSC-Lite | Virtual Hub

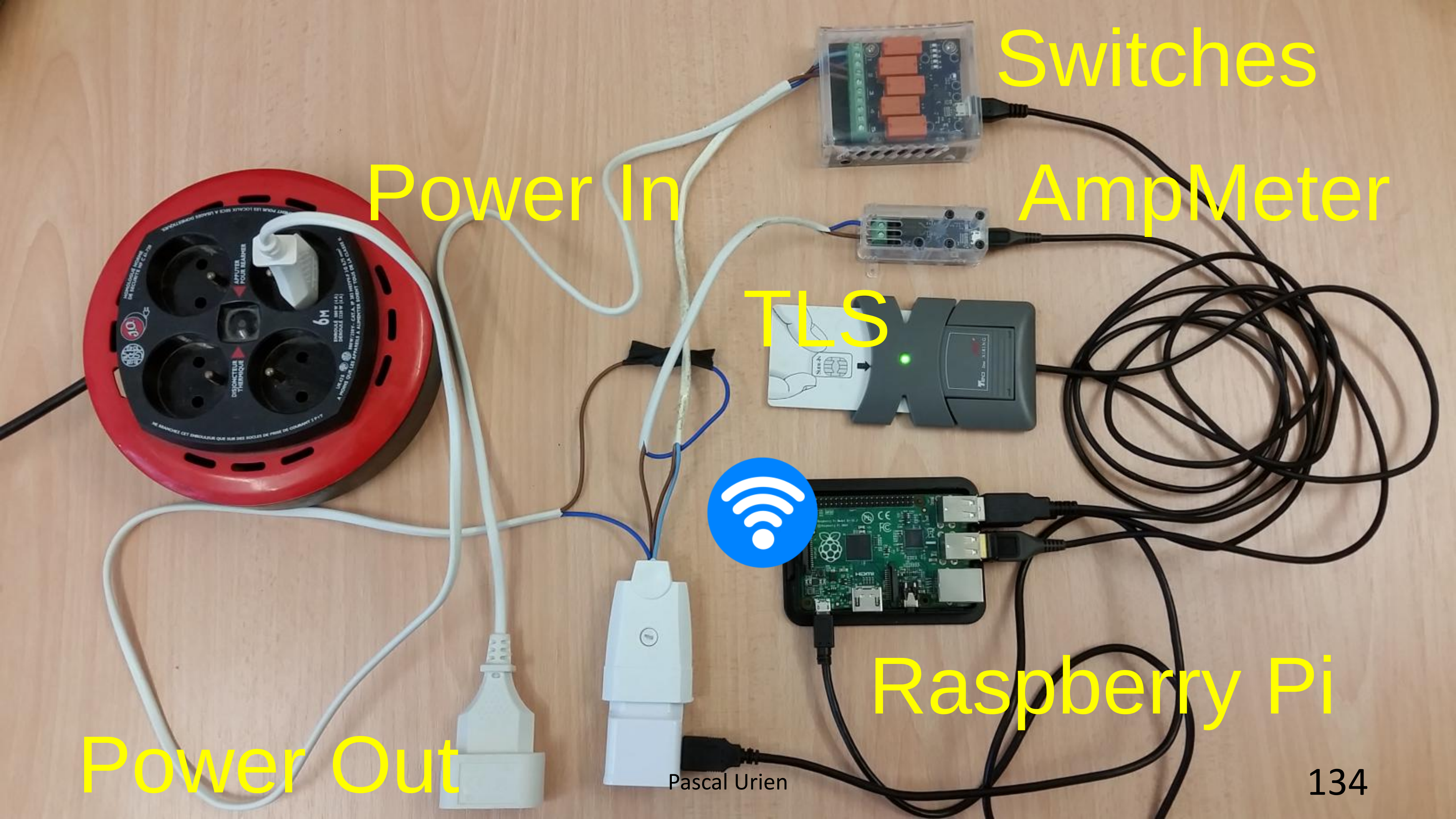
Debian OS

HTTP

TLS Server

TCP

IP


Switches
Power In
AmpMeter
TLS
Raspberry Pi
Power Out

Questions ?

Pascal Urien

October 9th 2016