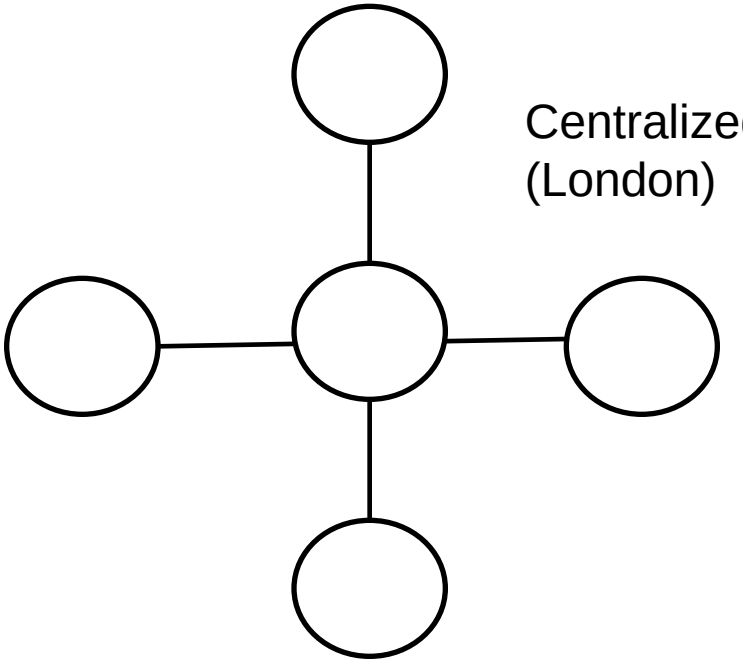
Mobile Payments
http://www.enst.fr/~urien/mp24.pdf
2024

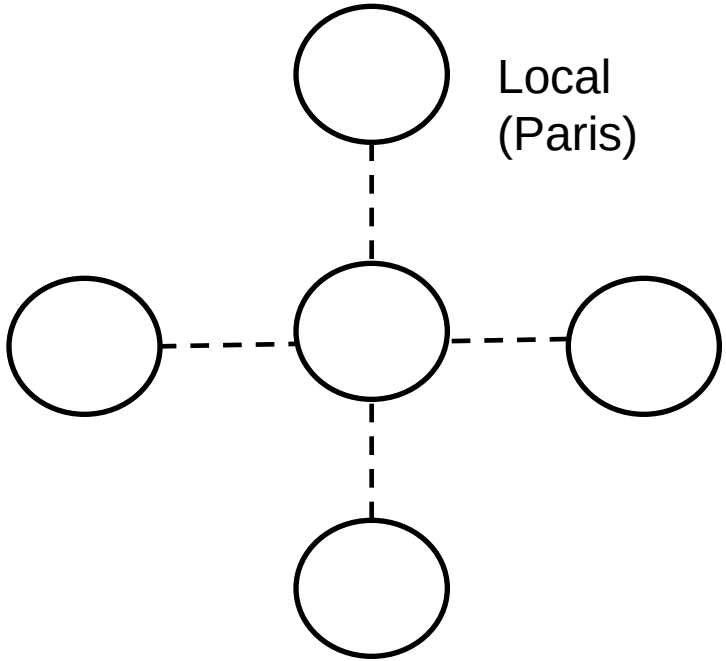
Pascal.Urien@telecom-paris.fr

TELECOM Paris

IP PARIS

Ticketing Systems





Content

- Electronic Bank Card story
- About Tamper Resistant Computing
- Smartcard Mask Age
  - B0’ bankcard
  - SIM Module
- Smartcard Application Age
  - Javacard Card
  - Global Platform
  - USIM
  - EMV overview
  - Global Platform Overview
- Smartcard Virtual Age
  - Near Field Communication
  - NFC Radio
  - NFC type A
  - NDEF, NFC Tags
  - NFC for mobile, SIM NFC, SIM Centric
  - TS 03.48, OPEN MOBILE API

- About Mobile Payments
- Online payments
- 3D secure
- EMV
- PayPass Mag Stripe (PMS)
- Google Wallet2
- VISA VCPS (Visa Contactless Payment Specification) MSD(Magnetic Stripe Data)
- Apple PAY
- EMV Contactless Transaction
- Host Card Emulation (HCE)
- EMV payment Tokenisation
- Android Pay
- Trusted Execution Environment (TEE)
- About Android Security
- Blockchain
  - Main Concepts
  - Bitcoin
  - Ethereum

Genesis

Bank Cards

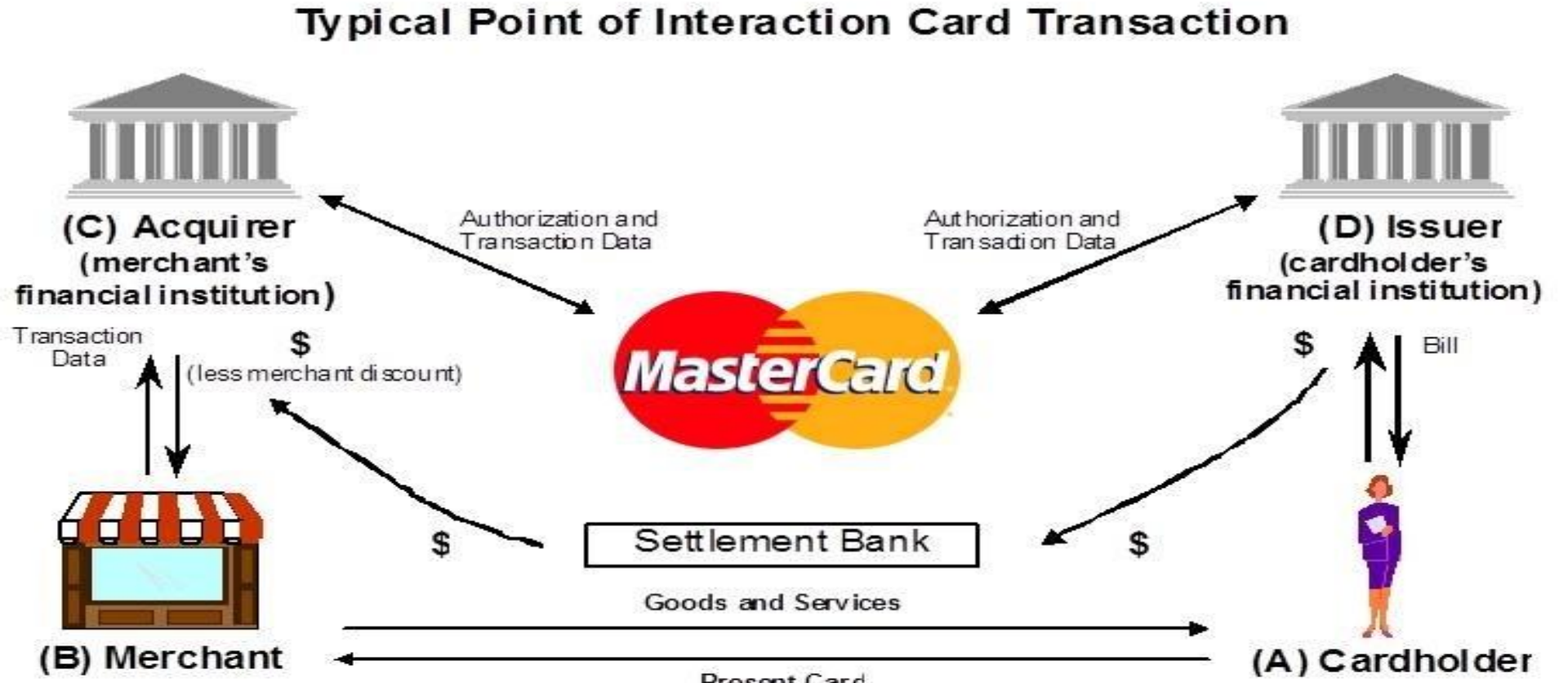
Diner's Club 1950

American express, plastic card, 1958

Carte Bleu 1967, two magnetic stripes
2x 200 bytes

1984, carte bancaire
First chip card

Foor Corner Transaction



British Museum (London)

HSBC
MONEY GALLERY

Case 17, Section 4
Actual size 85 x 54mm

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.

Carte

Lecteur

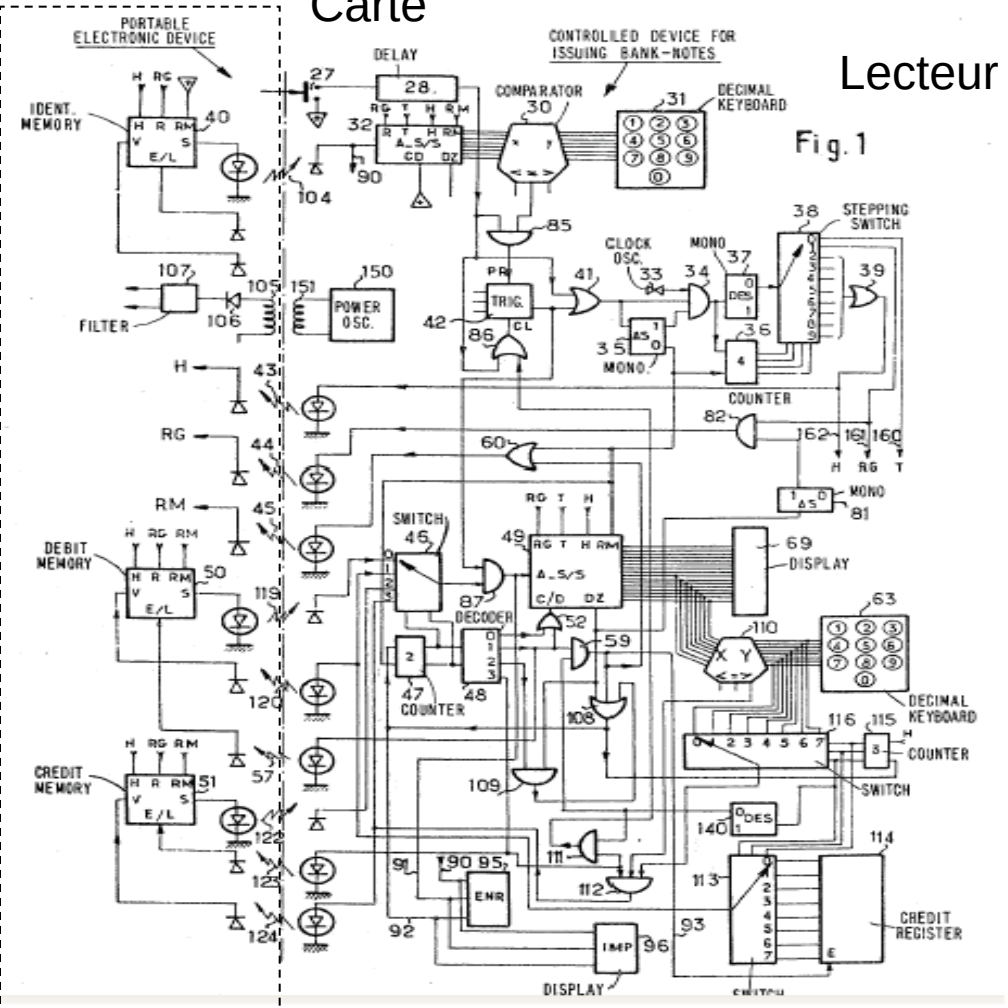
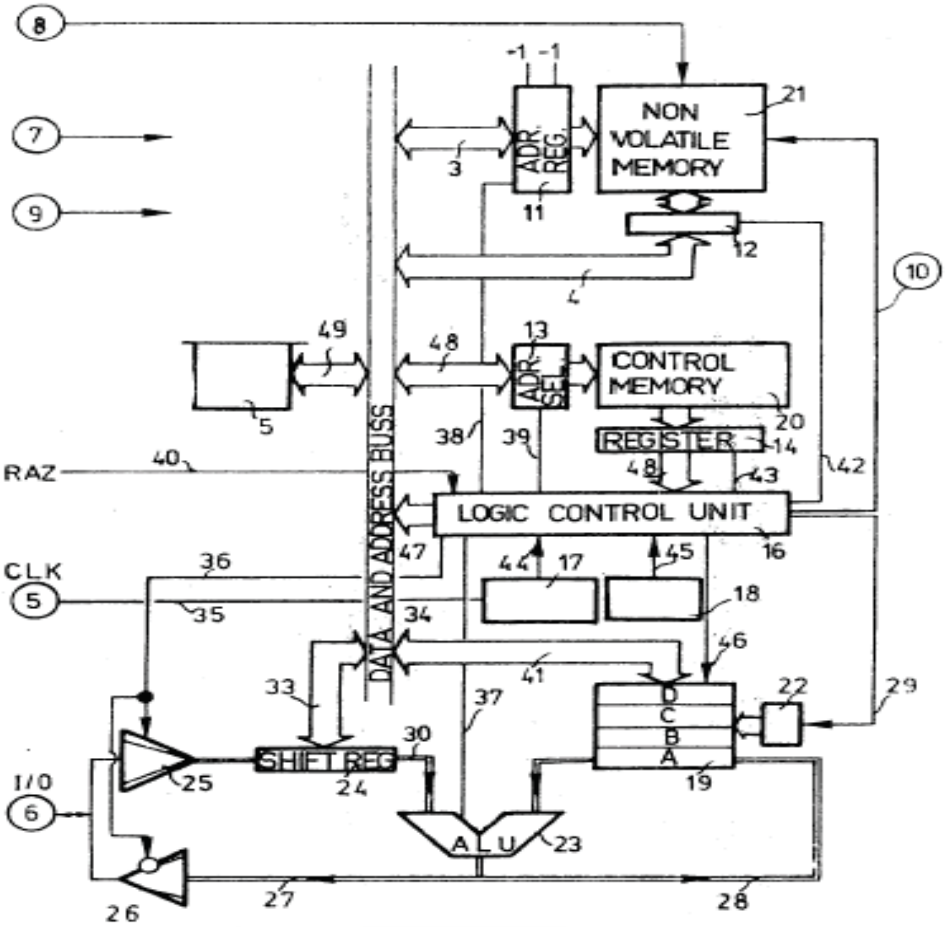


Fig. 1

Roland Moreno

- 25 march 1974, Patent 74.10191
- 21 march 1975, US 4,007,355

1978, a 1Kbit TELECARTE

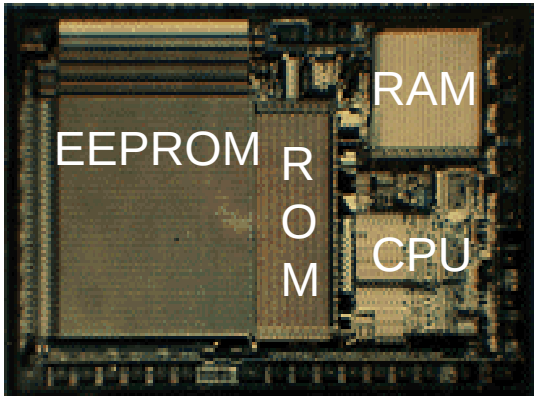


Michel Ugon

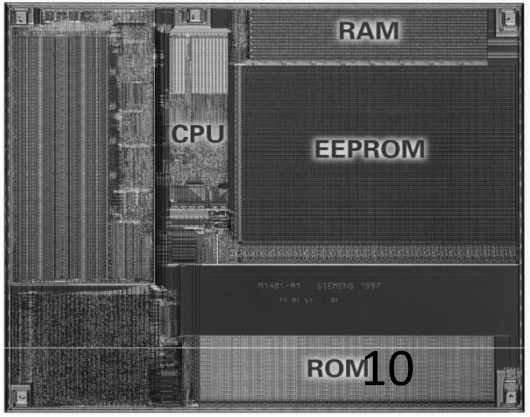
- 26 août 1977, brevet 77.26107
- 25 août 1978, US 4,211,919

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



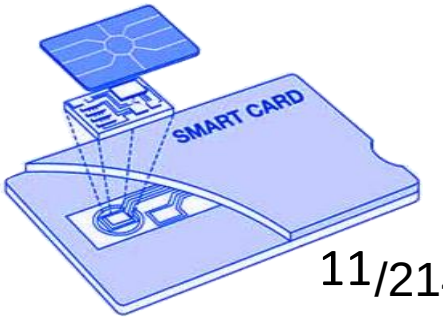
Siemens (SIM) chip, 1997

What is a Smartcard ?

- A smartcard is a SPOM
  - Self Programmable One Chip Microcomputer, born in 1980
  - 7 billions smartcards produced in 2012
  - 0,6 billion of contactless devices (*)
- Tamper resistant device
  - A Secure Microcontroller
  - Security is enforced by physical and logical countermeasures
- Typical chip area 5mm x 5mm
- Memories size
  - ROM 28 - 256 KB Area Factor 1
  - E^2PROM 64 - 128 KB Area Factor 4
  - RAM 4 - 8 KB Area Factor 16

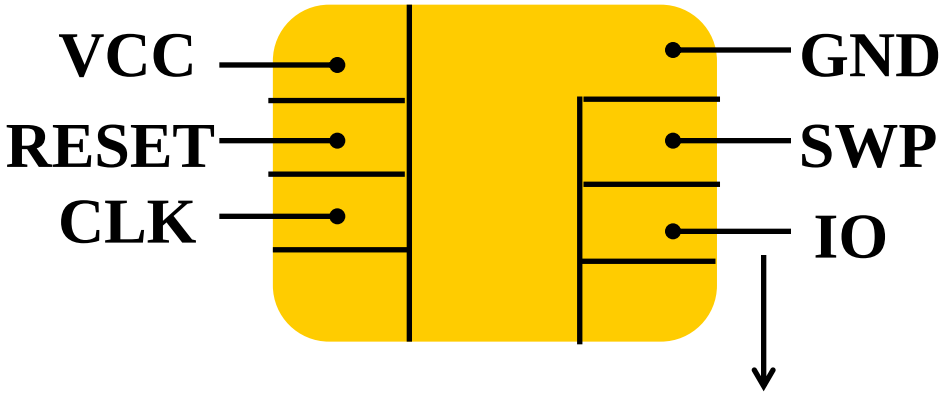
- CPU
  - Classical 8 bits processors, 1 - 3 MIPS (Clock 3.3 MHz)
  - 32 bits RISC processors
- Communication port
  - ISO7816 serial link 9600 to 230,400 bauds
  - USB (ISO7816-12), 10 Mbit/s
- Binary Encoding Rules
  - A five bytes Header
    - CLA INS P1 P2 P3
  - An optional payload of P3 (LC) bytes
  - An optional response of P3 (LE) bytes,
    - which ends with a two bytes status word SW

ISO 7816-4 commands are called APDU, their maximum size is about 256 bytes.



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."



ISO 7816-2 Serial Link

CLA INS P1 P2 Le

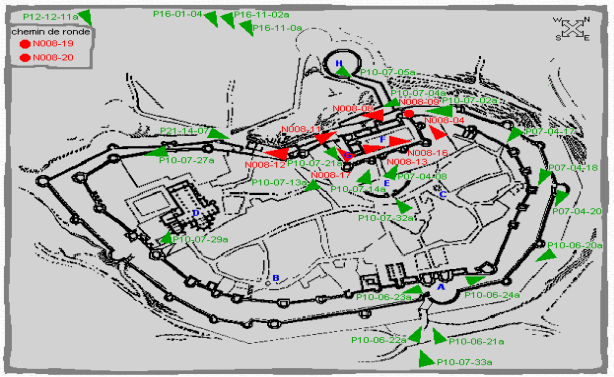
[Le Bytes] SW1 SW2

CLA INS P1 P2 Lc [Lc Bytes]

SW1 SW2

Tamper Resistant Computing

Security is a construction

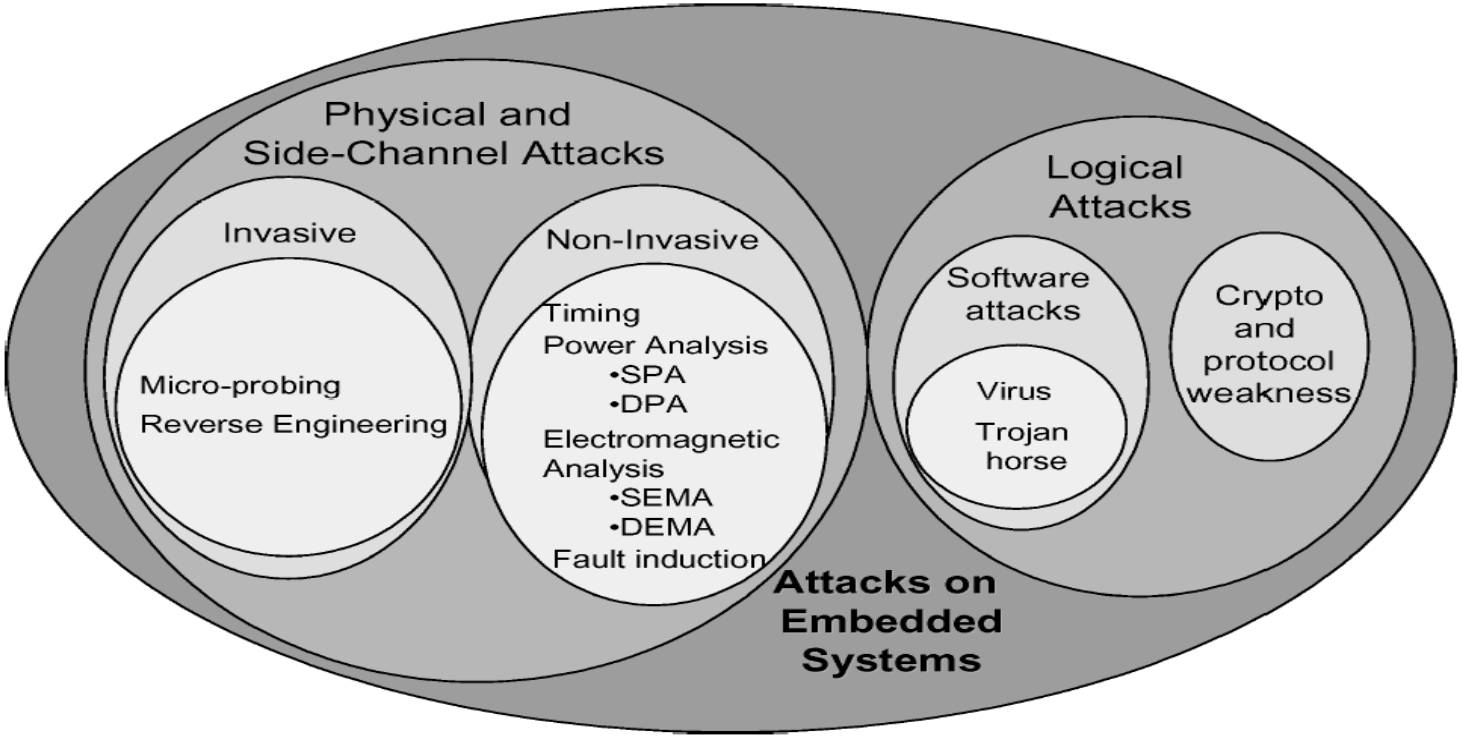


2 lines of rampart

1 bastion

*the city of Carcasonne

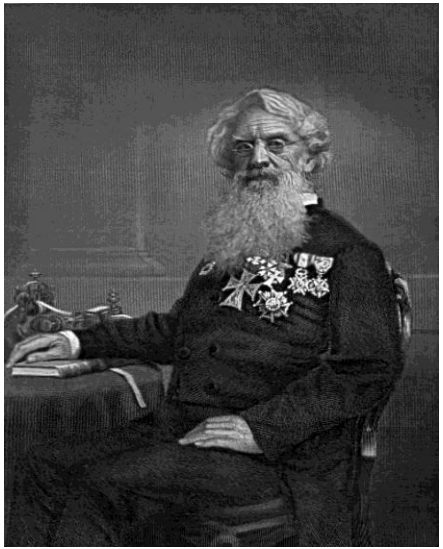
Secure Embedded Systems, S.Ravi, 2004



Examples of attack threats faced by embedded systems.

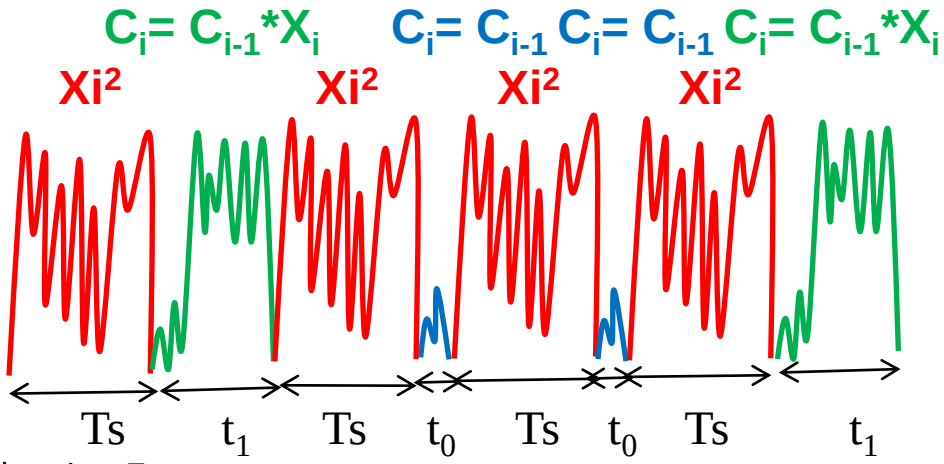
RSA & *Morse Samuel F*

A	● ▬	N	▬ ●	1	● ▬ ▬ ▬ ▬
B	▬ ● ● ●	O	▬ ▬ ▬	2	● ● ▬ ▬ ▬
C	▬ ● ▬ ●	P	● ▬ ▬ ●	3	● ● ● ▬ ▬
D	▬ ● ●	Q	▬ ▬ ● ▬	4	● ● ● ● ▬
E	● (1 unit)	R	● ▬ ●	5	● ● ● ● ●
F	● ● ▬ ●	S	● ● ●	6	▬ ● ● ● ●
G	▬ ▬ ●	T	▬ (3 units)	7	▬ ▬ ● ● ●
H	● ● ● ●	U	● ● ▬	8	▬ ▬ ▬ ● ●
I	● ●	V	● ● ● ▬	9	▬ ▬ ▬ ▬ ●
J	● ▬ ▬ ▬	W	● ▬ ▬	0	▬ ▬ ▬ ▬ ▬
K	▬ ● ▬	X	▬ ● ● ▬		
L	● ▬ ● ●	Y	▬ ● ▬ ▬		
M	▬ ▬	Z	▬ ▬ ● ●		



a^b modulus m

- $C = M^e = M * M * \ldots * M$ (e operations)
- $e = e_0 2^0 + e_1 2^1 + e_i 2^i + \ldots + e_{p-1} 2^{p-1}$, ei = 0 or 1, $di = e_i 2^i$
- $C = M^{d0} * M^{d1} * M^{di} * \ldots * M^{dp-1}$
  - $C_0 = M_0 = M^{d0} =$ 1 or M
  - $C_i = C_{i-1} * M^{di}$
  - $C = C_{p-1}$



- Algorithm
  - Begin i=0
    - $X_0 = M$
    - $C_0 = X_0^{d0} = 1$ or M, this calculation needs a time T_0
  - Loop i<p
    - $X_i = X^2_{i-1} = X_{i-1} * X_{i-1}$, this calculation needs a time Ts
    - If e_i=0 Then $C_i = C_{i-1}$, needs a "short" time t_0
    - If e_i=1 Then $C_i = C_{i-1} * X_i$, this this calculation needs a "long" time t_1

SPA
Single Power Attack

- $T = T_0 + Ts + t_{e1} + Ts + t_{ei} + \ldots + Ts + t_{ep-1}$

Differential Power Analysis

- Paul C. Kocher, Joshua Jaffe, Benjamin Jun: Differential Power Analysis. CRYPTO 1999: 388-397
  - Covariance, $cov(X,Y) = \sigma_{X,Y} = E(XY) - E(X)E(Y)$
  - Correlation coefficient, $\rho_{X,Y} = \sigma_{X,Y} / \sqrt{V(X)V(Y)}$, $\rho_{X,Y} \in [-1, 1]$
  - $E(XY) = E(X)E(Y) + \rho_{X,Y} \sqrt{V(X)V(Y)}$
  - $E(XY) = E(X)E(Y) + \rho_{X,Y} \sigma(X) \sigma(Y)$
- Let's assume :
  - A key domain of 2^p values, $i \in [0, 2^p - 1]$
  - A physical effect, such as power consumption, with an input value k, $X_i(k,t)$
  - A function Y correlated to the secret key i, and working for all input value k, $Y_i(k)$, and for each key i, $\langle Y_i(k) \rangle_k = 0$
  - For each wrong key
    - $\rho_{X,Y} = 0$, $\langle X_i(k,t).Y_i(k) \rangle_k = \langle X_i(k,t) \rangle_k \langle Y_i(k) \rangle_k = 0$
  - For the right key (j), $\rho_{X,Y} \neq 0$
    - $\langle X_j(k,t).Y_j(k) \rangle_k = \rho_{X,Y} \sigma(X) \sigma(Y)$

Bellcore Attack, D.Boneth 1997

$E1 = x^s \bmod p$, $E2 = x^s \bmod q$

$y = a.E1 + b.E2 \bmod pq$

$a = 1 \bmod p$, $a = 0 \bmod q$

$b = 1 \bmod q$, $b = 0 \bmod p$

$y = a.E1 + b.E2 \bmod pq$

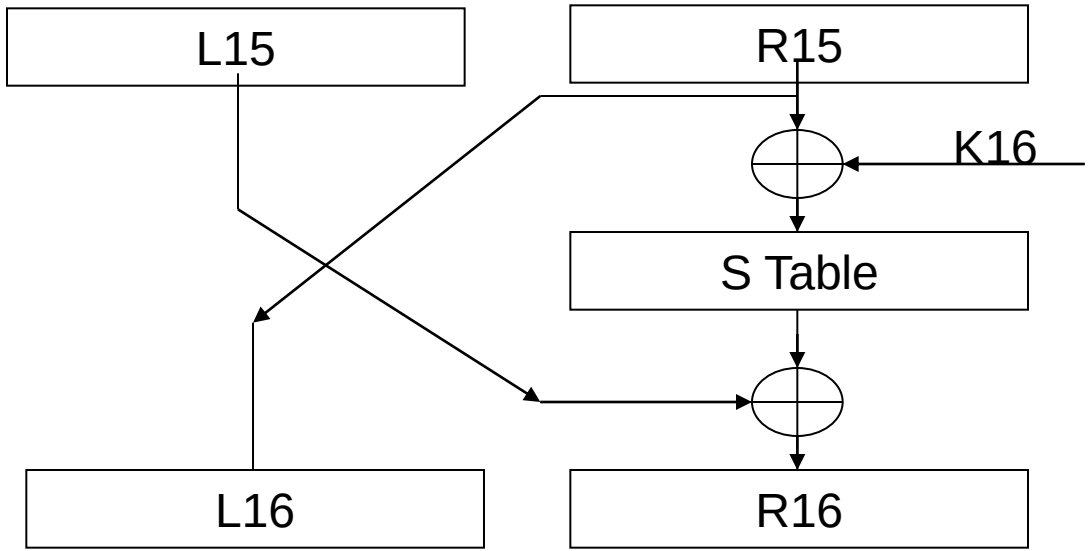
$y' = a.E1' + b.E2$, computation fault on $E1'$

$y - y' = a.(E1 - E1')$

if $E1 - E1'$ can not be divided by p

$GCD(y - y', n) = q$ $(n = p.q)$

DES fault injection - 1/2

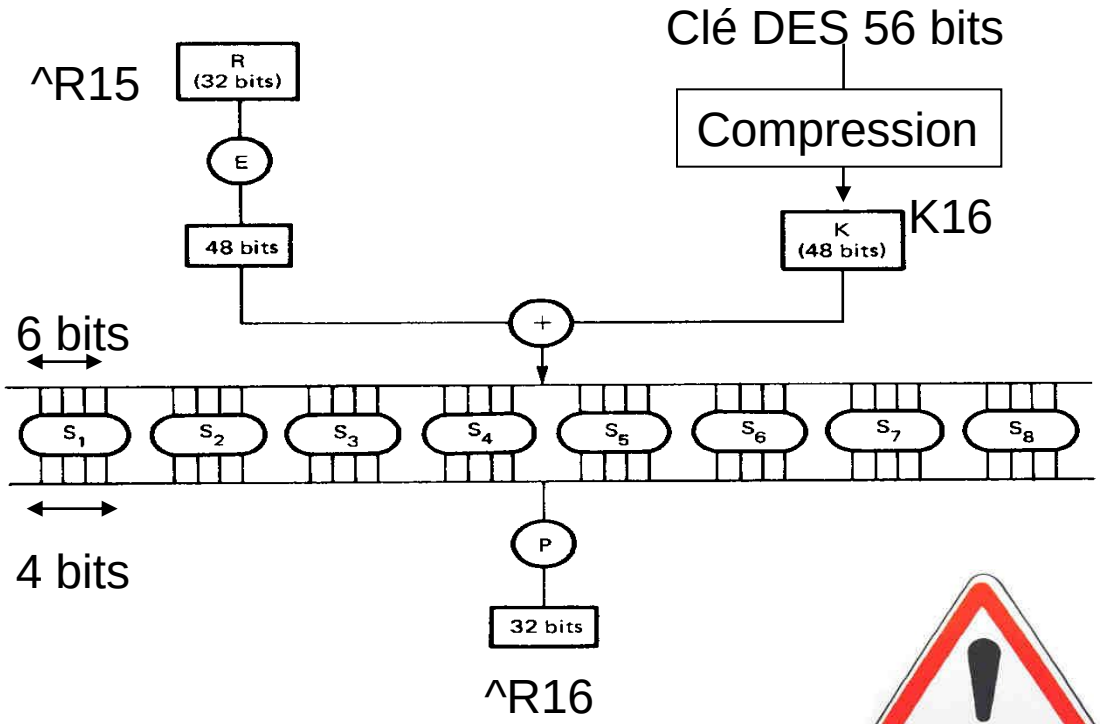


- Attacker knowns the right value R16
- He creates a fault ^R15, which implies a ^R16 value

Contrainte (C)

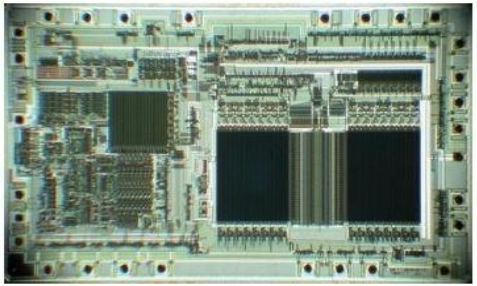
$$R_{16} \oplus \hat{R}_{16} = S(L_{16} \oplus K_{16}) \oplus S(\hat{L}_{16} \oplus K_{16})$$

DES fault injection DES - 2/2

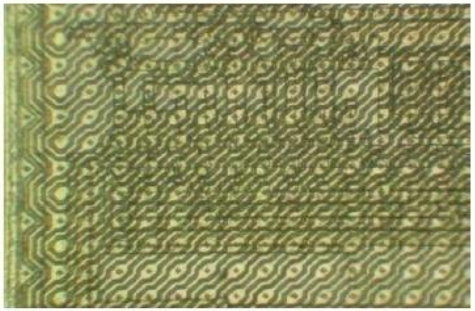


- About 2^{18} values of K16, verify the (C) relation
- About 2^{24} DES keys verify the (C) relation

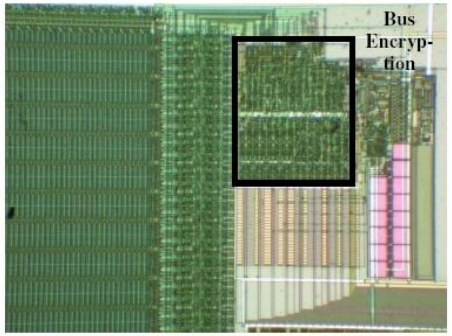
Secure Microcontroller



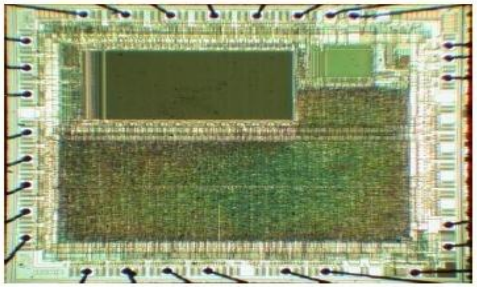
MC68HC705PA microcontroller with clearly distinguishable blocks



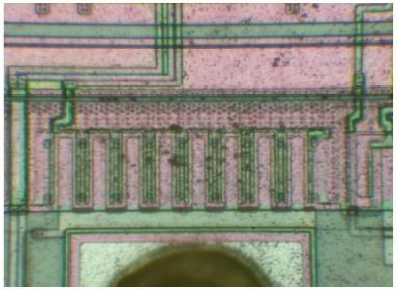
Top metal sensor mesh on ST16 smartcard



Hardware bus encryption module in Infineon SLE66 family smartcard chip [27] preserves EEPROM data from being microprobed. 100× magnification



SX28 microcontroller with 'glue logic' design



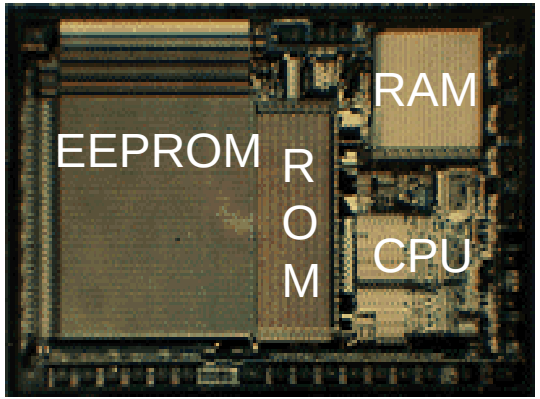
Second metal layer and polysilicon layer can be seen through top metal layer on Microchip PIC16F877 microcontroller [36]. 500× magnification

Nothing can be seen under the top metal layer on Microchip PIC16F877A microcontroller [37] as the layers inside the chip were planarised during fabrication. 500× magnification

Some figures for the ST22 microcontroller

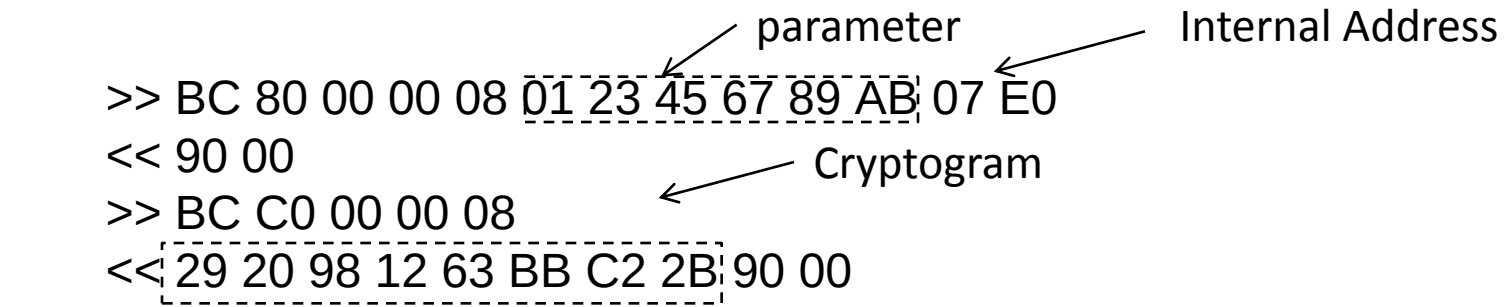
Algorithm	Function	Time
RSA 1024 bits	Signature with CRT	79.0 ms
	Signature without CRT	242.0 ms
	Verification (e=0x10001)	3.6 ms
RSA 2048 bits	Signature with CRT	485.0 ms
	Signature without CRT	1.7 s
	Verification (e=0x10001)	11.0 ms
DES	Triple	18 µs
	Single	8 µs
SHA-1	512-bit Block	194 µs
AES-128	Encryption including subkey computation	85 µs
Key generation	1024 bits key 2048 bits key	2.7 s 23.1 s

The Smartcard MASK Age



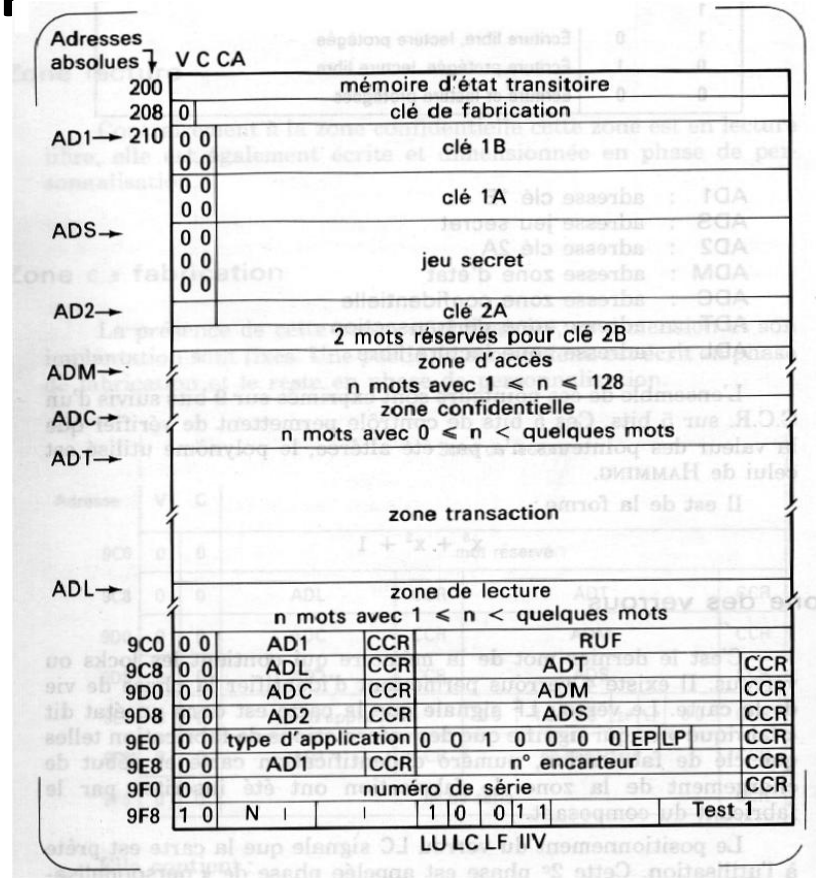
The B0’ banking card

- Was used in France during the period 1985-2000
- A memory area with
  - public,
  - (PIN) restricted,
  - and private zones.
- The Public zone content is signed with an RSA private key (static authentication)
- Transaction parameters (date, amount) are written in the card
- A function (The *TELEPASS* function), whose access is protected by a PIN code, generates cryptograms based on the 3xDES algorithm



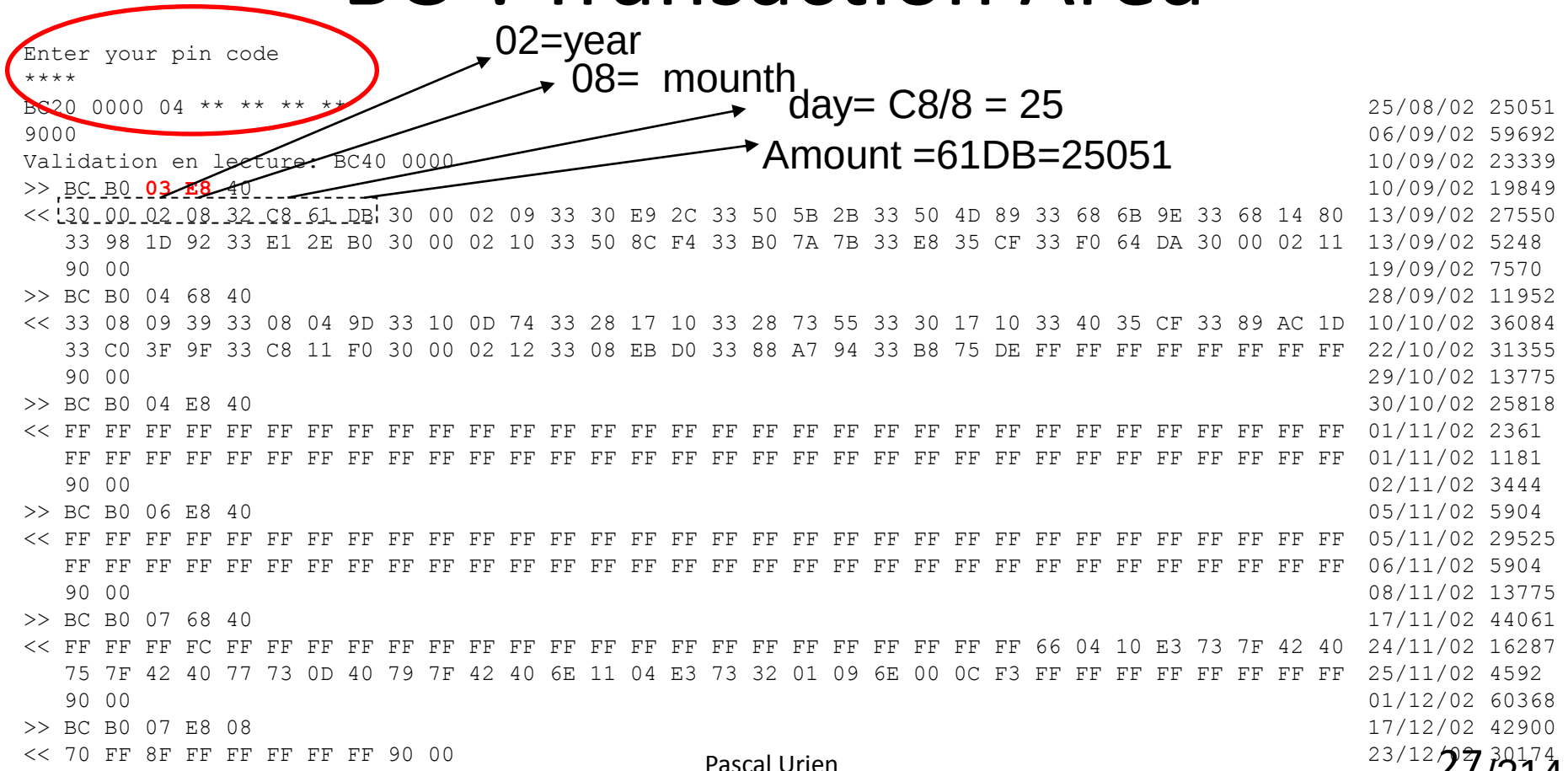
Reading BO’ pointers

```
>> BC B0 09 E8 04
<< 08 4D 00 25 90 00
>> BC B0 09 DC 04
<< 08 D9 3F E5 90 00
>> BC B0 09 D8 04
<< 09 F1 08 D9 90 00
>> BC B0 09 D4 04
<< 0A 57 09 F1 90 00
>> BC B0 09 D0 04
<< 0A C3 0A 57 90 00
>> BC B0 09 CC 04
<< 0F B0 0A C3 90 00
>> BC B0 09 C8 04
<< 1F F4 0F B0 90 00
AD1=210 ADS=230 AD2=278 ADM=290
ADC=2b0 ADT=3e8 ADL=7f8
Key 1B + Key 1A      AD1-Length= 16 bytes
Secret Keys          ADS-Length= 36 bytes
Key 2A + Key 2B      AD2-Length= 12 bytes
Access Area          ADM-Length= 16 bytes
Secret Area          ADT-Length= 520 bytes
Reading Area   (VS)  ADL-Length= 228 bytes
```



Pascal Urien

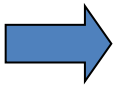
BO’: Transaction Area



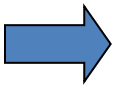

```
Enter your pin code
****
BC20 0000 04 ** ** ** **                                                        25/08/02 25051
9000                                                                            06/09/02 59692
Validation en lecture: BC40 0000                                                10/09/02 23339
>> BC B0 03 E8 40                                                               10/09/02 19849
<< 30 00 02 08 32 C8 61 DB 30 00 02 09 33 30 E9 2C 33 50 5B 2B 33 50 4D 89 33 68 6B 9E 33 68 14 80  13/09/02 27550
   33 98 1D 92 33 E1 2E B0 30 00 02 10 33 50 8C F4 33 B0 7A 7B 33 E8 35 CF 33 F0 64 DA 30 00 02 11  13/09/02 5248
   90 00                                                                        19/09/02 7570
>> BC B0 04 68 40                                                               28/09/02 11952
<< 33 08 09 39 33 08 04 9D 33 10 0D 74 33 28 17 10 33 28 73 55 33 30 17 10 33 40 35 CF 33 89 AC 1D  10/10/02 36084
   33 C0 3F 9F 33 C8 11 F0 30 00 02 12 33 08 EB D0 33 88 A7 94 33 B8 75 DE FF FF FF FF FF FF FF FF  22/10/02 31355
   90 00                                                                        29/10/02 13775
>> BC B0 04 E8 40                                                               30/10/02 25818
<< 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 FF FF  01/11/02 2361
   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 FF FF  01/11/02 1181
   90 00                                                                        02/11/02 3444
>> BC B0 06 E8 40                                                               05/11/02 5904
<< 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 FF FF  05/11/02 29525
   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 FF FF  06/11/02 5904
   90 00                                                                        08/11/02 13775
>> BC B0 07 68 40                                                               17/11/02 44061
<< FF FF FF FC FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 66 04 10 E3 73 7F 42 40  24/11/02 16287
   75 7F 42 40 77 73 0D 40 79 7F 42 40 6E 11 04 E3 73 32 01 09 6E 00 0C F3 FF FF FF FF FF FF FF FF  25/11/02 4592
   90 00                                                                        01/12/02 60368
>> BC B0 07 E8 08                                                               17/12/02 42900
<< 70 FF 8F FF FF FF FF FF 90 00                                                23/12/02 30174
```

BO' Signature Value Humpich Attack (2000)

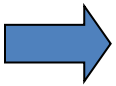
```
BigInteger  kpriv = new BigInteger
("142399135728060672159668180333308586470073022788225731360 9\
  900555054188454201824800920161049221243");
BigInteger  kpub =  new BigInteger("03") ;
BigInteger  kmod =  new BigInteger
("213598703592091008239502270499962879705109534182\
  641740644252416500858395774644508840500943086599 9");
String test= "5D 08 69 7D 8E 99 3B C9 F7 DF 46 9D 36 B3 2F 2A E0 10 18 01
              38 D1 52 34 03 00 F9 04 3C 1D EF 6A 72 69 39 5E B5 2C C7 38";
x = new BigInteger("1234") ;
x = x.modPow(kpriv,kmod)    ;
x = x.modPow(kpub,kmod)     ;

x = test.modPow(kpub,kmod) ;

BCB0 07F8 34
2E 03 30 33

30 00 05 D0
38 69 7D 8E
39 93 BC 9F
37 DF 46 9D
33 6B 32 F2
3A E0 10 18
30 13 8D 15
32 34 03 00
3F 90 43 C1
3D EF 6A 72
36 93 95 EB
35 2C C7 38
90 00
```



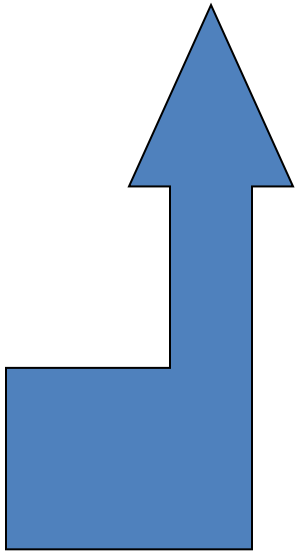
```
0 00 05 D0
8 69 7D 8E
9 93 BC 9F
7 DF 46 9D
3 6B 32 F2
A E0 10 18
0 13 8D 15
2 34 03 00
F 90 43 C1
D EF 6A 72
6 93 95 EB
5 2C C7 38
```



```
00 00 5D 08
69 7D 8E 99
3B C9 F7 DF
46 9D 36 B3
2F 2A E0 10
18 01 38 D1
52 34 03 00
F9 04 3C 1D
EF 6A 72 69
39 5E B5 2C
C7 38
```



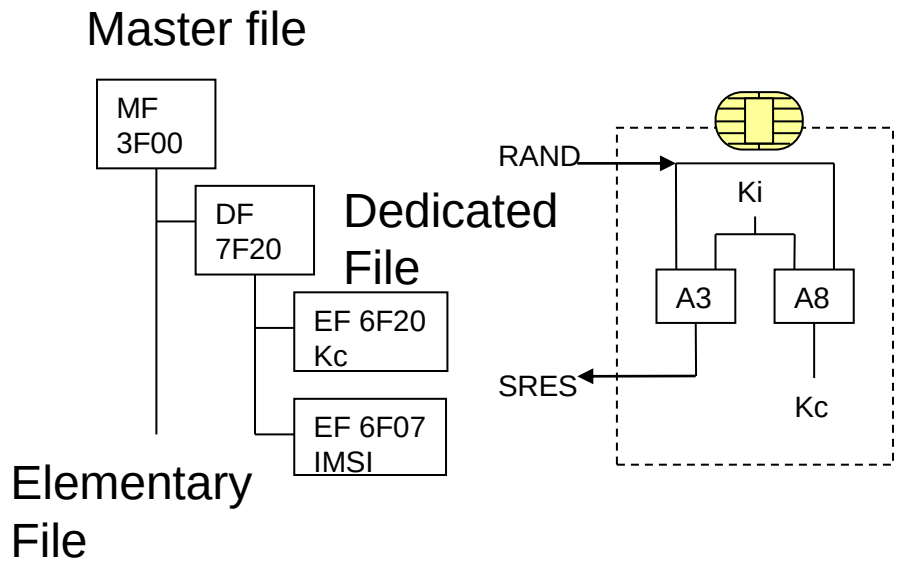
```
5D 08 69 7D
8E 99 3B C9
F7 DF 46 9D
36 B3 2F 2A
E0 10 18 01
38 D1 52 34
03 00 F9 04
3C 1D EF 6A
72 69 39 5E
B5 2C C7 38
```



```
00010a000004978461030099869fff1099120211\ (20 bytes)
00010a000004978461030099869fff1099120211\ (20 bytes)

10 a0 00 00
4978461030099869 // card number
fff10
9912   // issuance   date
0211   // expiration date
```

SIM for GSM radio networks

- Information is organized according to repertories and files
- Some Information
  - IMSI
  - Two phone books
  - One SMS file
- Some Procedures
  - RUN_GSM_ALGORITHM, computes the A3/A8 algorithm



// 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**

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

SRES

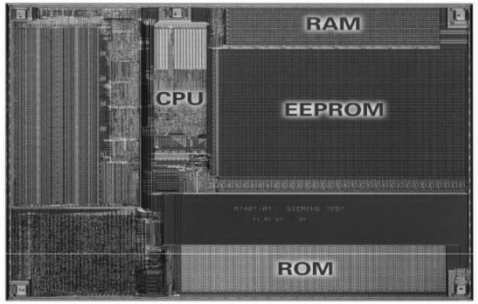
The Smartcard Application Age

The Applications Age

- Invention of Java Card, in 1996
- A java virtual machine (JVM) runs applications written in *javacard*, a subset of the java language.
- Embedded applications are identified by a number, the Application Identifier (AID), whose maximum size is 16 bytes.
- This milestone marks the birth of Secure Elements (SE)
  - **A SE is a secure microcontrollers equipped with general purpose operating systems, supporting a virtual machine and able to download, delete, and execute applications.**
- It also enables the splitting up between hardware and software.
- These components may also be included in electronic chips with classical pinout, performing additional tasks such as NFC controller

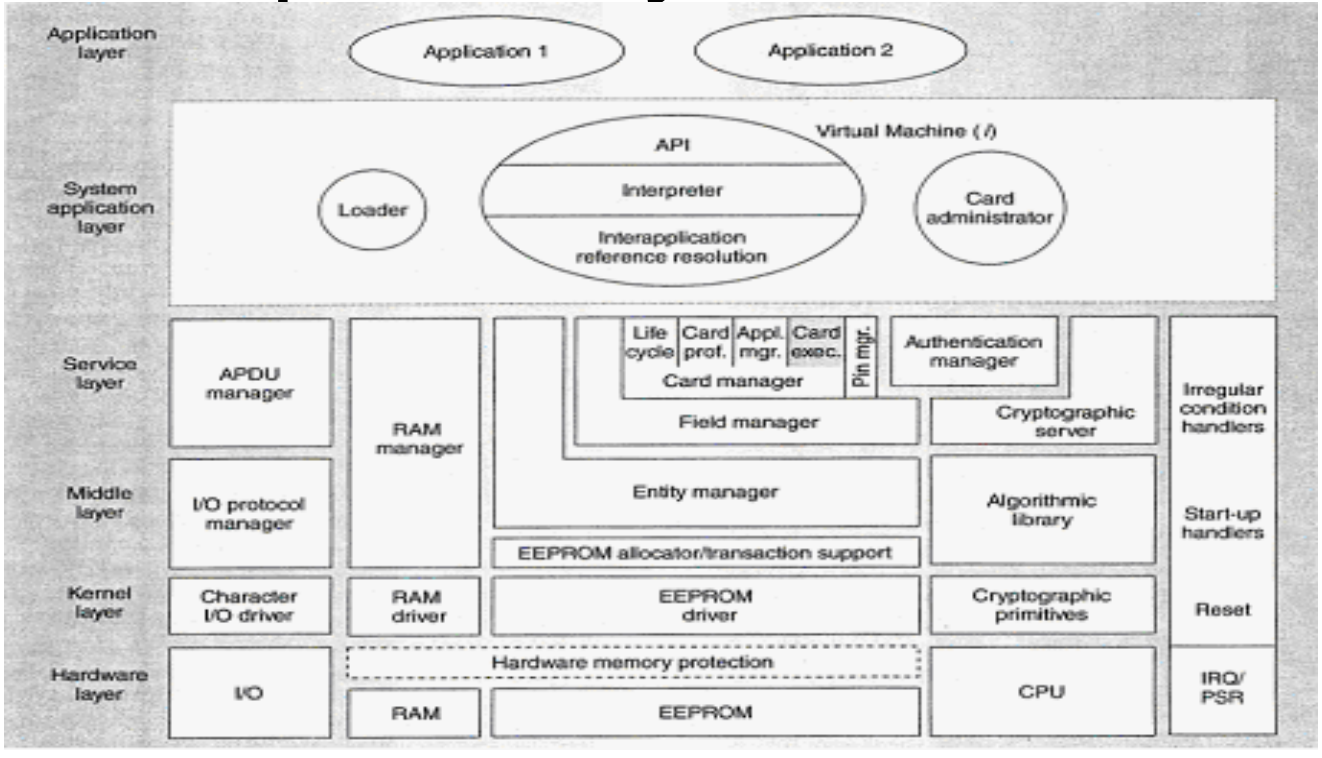
The Application Age Milestones

- 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



Siemens (SIM) chip, 1997

Example of a javacard OS



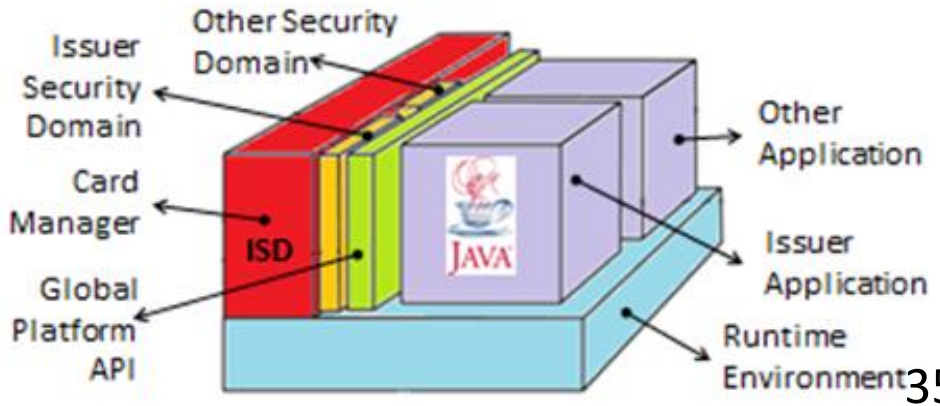
Tual, J.-P. "MASSC: a generic architecture for multiapplication smart cards", Micro, IEEE Volume: 19 , Issue: 5, 1999

Example of Javacard Code

```
package Demo;
import javacard.framework.*;
public class DemoApp extends Applet
{
final static byte  BINARY_WRITE   = (byte)  0xD0     ;
final static byte  BINARY_READ    = (byte)  0xB0     ;
final static byte  SELECT         = (byte)  0xA4     ;
final static short NVRSIZE        = (short) 4096     ;
final static short ERROR_WRITE    = (short) 0x6D02   ;
final static short ERROR_READ     = (short) 0x6D01   ;
byte[] NVR                        =  null            ;
public void process(APDU apdu)  throws ISOException
  { short adr,len;

  byte[] buffer = apdu.getBuffer() ;
  byte cla = buffer[ISO7816.OFFSET_CLA];
  byte ins = buffer[ISO7816.OFFSET_INS];
  byte P1  = buffer[ISO7816.OFFSET_P1] ;
  byte P2  = buffer[ISO7816.OFFSET_P2] ;
  byte P3  = buffer[ISO7816.OFFSET_LC] ;
   adr = Util.makeShort(P1,P2) ;
  len = Util.makeShort((byte)0,P3)  ;
   switch (ins){
  case SELECT: return;
  case BINARY_READ:
   if (adr <(short)0)   adr =(short) (NVR.length+adr);
   Util.arrayCopy(NVR,adr,buffer,(short)0,len);
   apdu.setOutgoingAndSend((short)0,len);
   break;
case BINARY_WRITE:
short readCount = apdu.setIncomingAndReceive();
if (readCount <= 0) ISOException.throwIt(ERROR_WRITE) ;
 if (adr <(short)0)  adr =(short) (NVR.length+adr);
Util.arrayCopy(buffer,(short)5,NVR,adr,len);
break;
 default:
ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
  }}
protected DemoApp(byte[] bArray,short bOffset,byte bLength)
 {  NVR = new byte[NVRSIZE] ;
    register(); }
public static void install( byte[] bArray, short bOffset, byte bLength )
{   new DemoApp(bArray,bOffset,bLength);  }
  public boolean select()   {   return true;  }
  public void deselect()  {  } }
```

The Applications Age

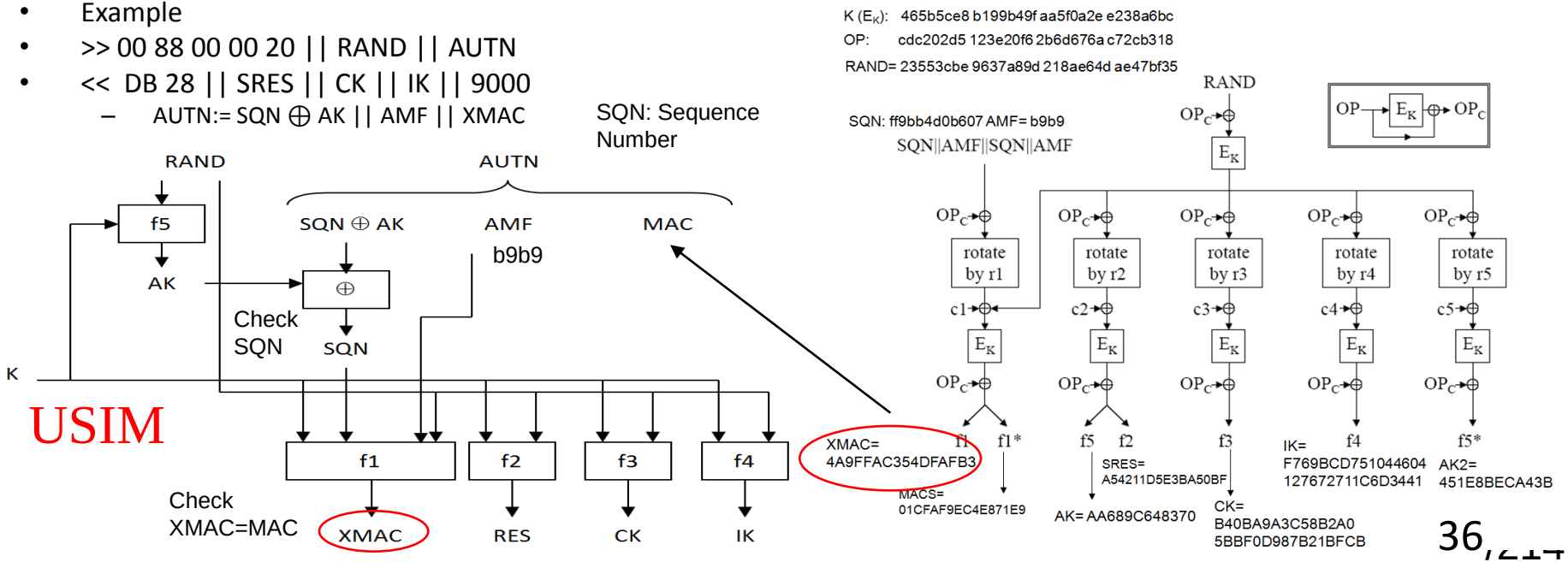
- The development of the javacard technology creates a need for trusted management of embedded software.
  - **The Global Platform specifications enforce the applications lifecycle i.e. downloading, activation and deletion in secure elements.**
  - All these operations are handled by a card manager entity, hosting an Issuer Security Domain (ISD) identified bv an AID
- Applications
  - EMV cards
  - ICAO passports
  - USIM cards
  - PKCS Cards



USIM for 3G/4G radio networks

- UICC modules host USIM applications
  - The list of USIM applications is stored in the EF-DIR file
- At least two concurrent applications may run at the same time
  - The application index is identified by the last two bits of the CLA byte
- 3G/4G authentication is performed thanks to the AUTHENTICATE (INS=88) command
- Example
- >> 00 88 00 00 20 || RAND || AUTN
- << DB 28 || SRES || CK || IK || 9000
  - AUTN:= SQN ⊕ AK || AMF || XMAC

HLR



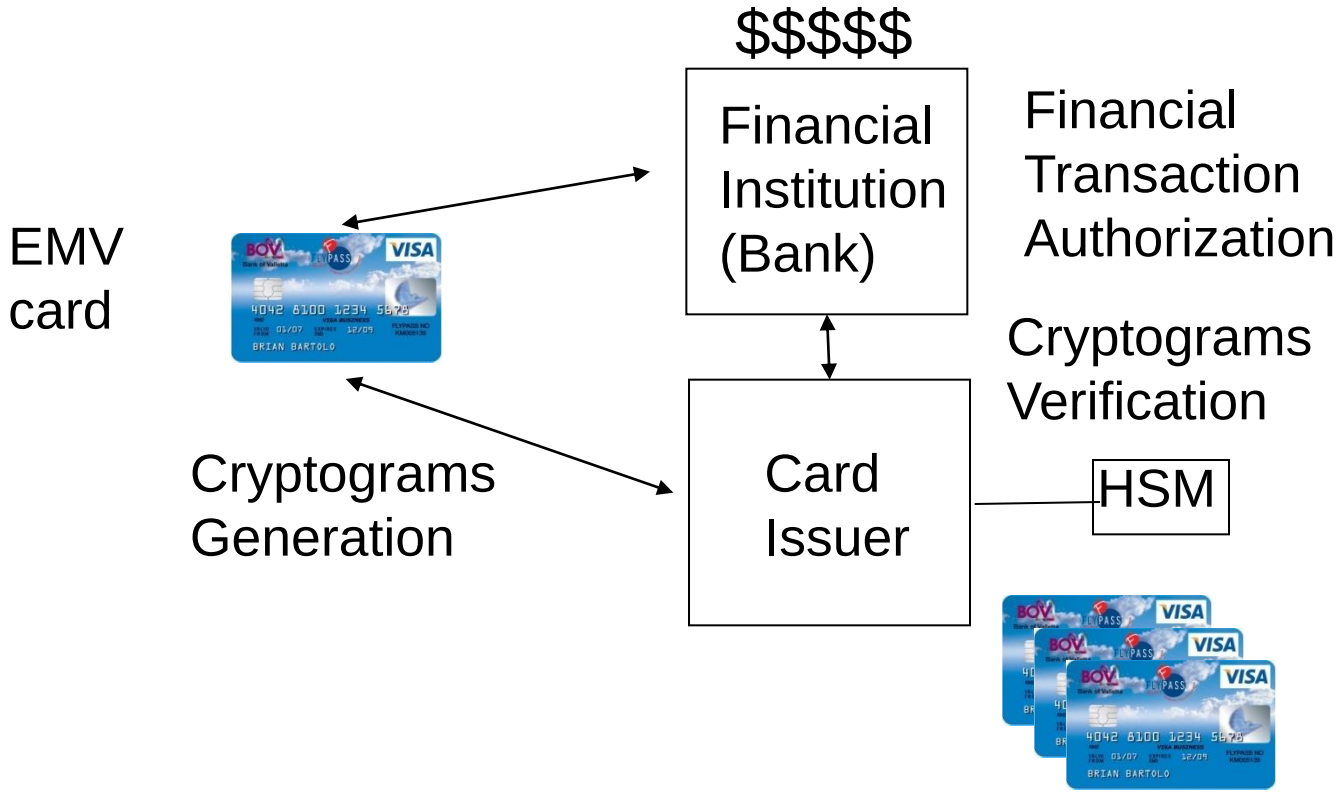
About EMV cards

- EMV means *Europay MasterCard and Visa*
  - These three companies initiated in 1994 the development of this specification.
  - More than 1 billion EMV cards deployed every year in the world.

About EMV standards

- The EMV specifications are published in four documents:
  - Book 1, *Application Independent ICC to Terminal Interface Requirement*. This document details transmission protocols, files commands and application selection.
  - Book 2, *Security and Key Managemen*t. This document defines all security procedures available in EMV chips.
  - Book 3, *Application Specification*. This specification presents EMV data elements and commands; it describes debit and credit applications.
  - Book 4, *Cardholder, Attendant and Acquirer Interface Requirements*. This document contains a set of recommendations for the support of EMV devices in payment systems.

Understanding the EMV model



EMV-Device structure

- An EMV smart card contains one or several EMV applications.
- An EMV application manages a set of information that can be freely read.
  - These data are encoding according to the ASN.1 syntax.
- An EMV application produces cryptograms that authenticate payment transactions.
- A Data Object List (DOL), is a list of tuples
  - TAG (one or two bytes)
  - and object length (one byte)

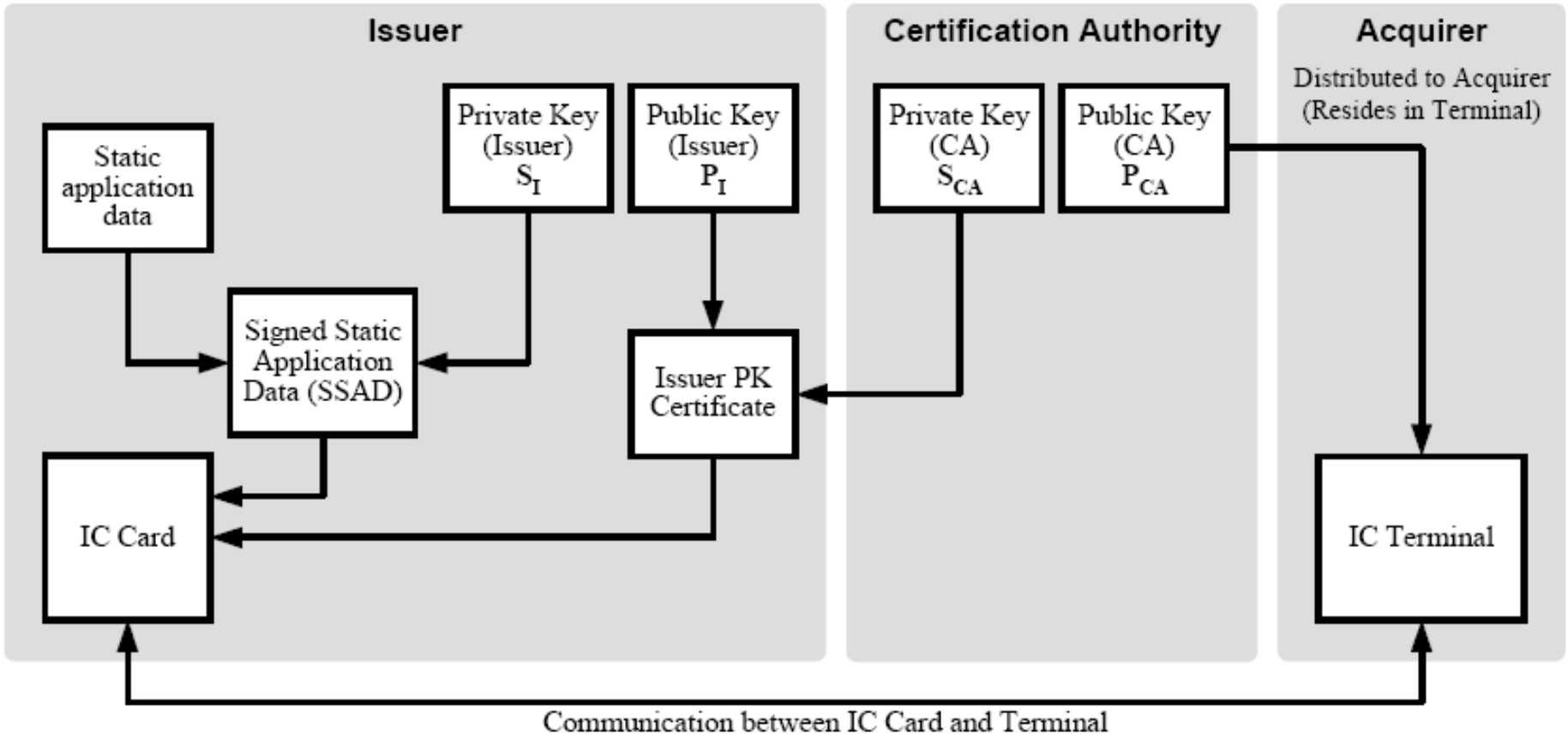
EMV smart card

- Stores multiple EMV applications
- Each application manages a set of data and procedures
- Information is organized according to files and records
  - Records store a set of ASN.1 encoded object (Data Object, DO)
- Some Data Objects
  - Application Primary Account Number (PAN)
    - The card number
  - The Signed Static Application Data (SSAD)
    - A signature (with the ISSUER private key) for a set of information stored in the card.
- Some procedures
  - DDA, Dynamic Data Authentication, encoding of a random 32 bits value with an optional smart card private key
  - Cryptogram generation, based on 3xDES algorithm, working with input parameters such as transaction amount and date
    - ARQC, The Authorization Request Cryptogram (ARQC) starts an EMV transaction.
    - AAC, The Application Authentication Cryptogram ends an EMV transaction.

EMV details

- Application Primary Account Number (PAN)
  - The card number
  - Tag 5A, Length 08, Value: 49 73 01 97 61 90 02 78
- Application PAN Sequence Number (PSN)
  - An additional identifier for the card
  - Tag 34, Length 01, Value: 00
- Signed Static Application Data (SSAD)
  - A signature for a set of information stored in the card.
  - Tag 93
  - The list of signed objects is found in the AFL (*Application File Locator*) or in the optionnal tag 9F4A

Static Authentication



Example of decrypted SSAD with the Issuer Public Key

```
6A03010000BBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBB03B996EDCC
8D893EDF039915C1B22344ACD2C588BC
```

=SHA1

Decrypted SSAD

Data to Hash

```
03010000BBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBB5713 45330182
69340345D11122018653485750016F5F
25030811015F24031112315A08453301
82693403455F3401009F0702FF008E14
000000000000000042014403010302031
1E031F039F0D05F848FCA0009F0E0500
100000009F0F05F868FCF8005F280202
509F4A01827C00
```

Dynamic Authentication

- DDA (*Dynamic Data Authentication)* is an encryption procedure dealing with the ICC private key.
- DDOL or *Dynamic Data Authentication Data* contains a list of data to be included in the DDA
- The DDOL is included in the tag 9F49
  - A typical value of 9F3704 means the use of a 32 bits (04 bytes) random value

DDA

- Command Internal Authenticate :
  - 00 88 00 00 04 5FAD5DF7

response

```
8081809ABE30A14921B400C8361E532A
0D81EFB5DBF206DE17725C11FCAB19BF
5EBB4FA7CFA90B68130E7B2E5FB6BF29
E9D57B42BB61480ECFF69CE6E2826C32
0C437A2B183A35E94B7599229AC758B2
D53B7CBEEE4D35C6794C6BE2DB4AE37E
32B8A2FA62BE80A349D4D366A91EA264
1258045E035CF5BCF52F4EA7B65C54B9
85DF5D
```

Decrypted Response

ICC Dynamic Number

```
6A05010908C8ED932E93721CB1BBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBB0653343900
96405CDE78DB8B120584EBCB2926FFBC
```

=SHA1

Data to hash

```
05010908C8ED932E93721CB1BBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBB5FAD5DF7
```

EMV Details

- Card Risk Management Data 1 (CDOL1)
  - CDOL1 is the list of objects required for the generation of a transaction cryptogram
    - Tag 8C, Length 1B, Value: 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 45 02 9F 4C 08
- Card Risk Management Data 2 (CDOL2)
  - CDOL2 is the list of objects required for the completion of a transaction.
  - It is the concatenation of the Authorization Response Code (TAG 8A, length 02) and the CDOL1 value.
    - Tag 8D, Length 1A, Value: 8A 02 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 4C 08

ARQC & AAC

- ARQC
  - The Authorization Request Cryptogram (ARQC) starts an EMV transaction.
  - A set of values, whose elements are listed by the CDOL1 object, and without TAG or length information, are pushed towards the card.
  - The content of CDOL1 is noted xCDOL1 and the response to ARQC is noted yCDOL1.
    - xCDOL1 comprises an Unpredictable Number (TAG 9F37, length 04), i.e. a random value of 32 bits.
    - The response yCDOL1, includes an 8 bytes cryptogram and additional information.
- AAC
  - The Application Authentication Cryptogram ends an EMV transaction.
  - A set of values, whose elements are listed by the CDOL2 object, and without TAG or length information, are pushed towards the card.
  - The content of CDOL2 is noted xCDOL2 and the response to AAC is noted xCDOL2.

ARQC & AAC example 1/2

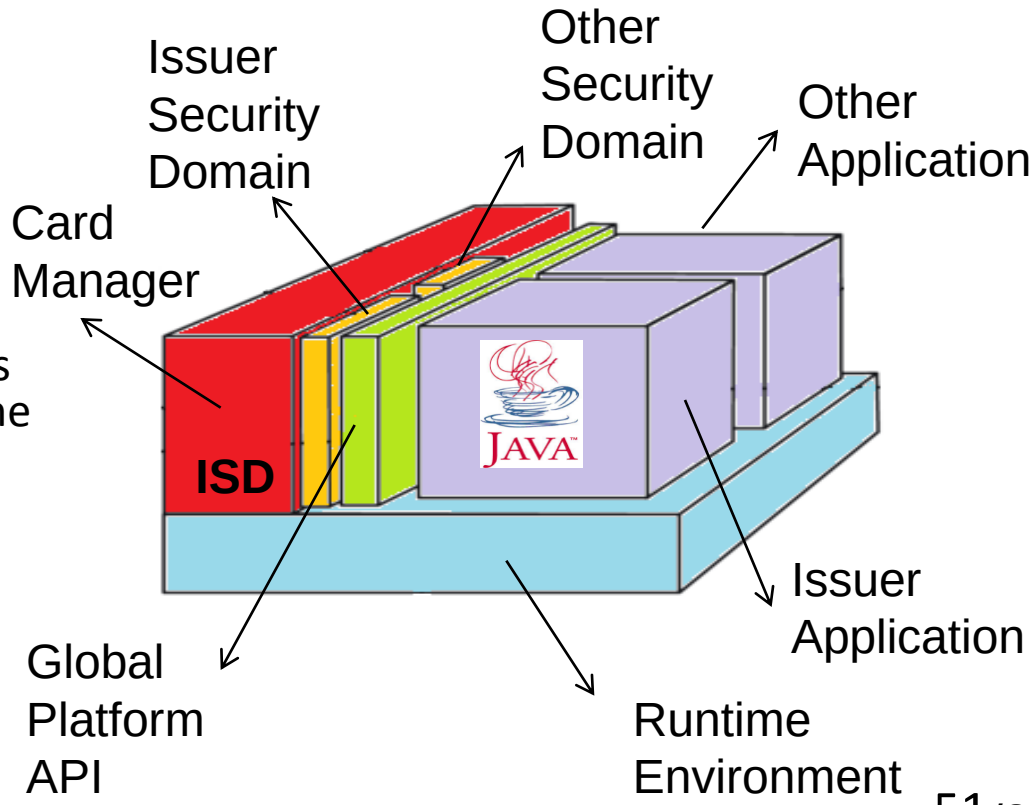
- ARQC
  - xCDOL1
    - 00 00 00 00 00 00, the transaction amount
    - 00 00 00 00 00 00, the cash back
    - 00 00, the national code of the payment terminal
    - 80 00 00 00 00, the terminal verification result
    - 00 00, the transaction currency code
    - 01 01 01, the transaction date
    - 00, the type of transaction
    - 12 34 56 78, a four bytes random value: r32
  - yCDOL1
    - 77 1E
      - 9F 27 01 80, Cryptogram Information Data
      - 9F 36 02 00 18, Apllication 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

ARQC & AAC example 2/2

- AAC
  - xCDOL2
    - 5A 33, Authorization Response Code
    - 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: r32
  - yCDOL2
    - 77 21
    - 9F 27 01 00, Cryptogram Information Data
    - 9F 36 02 01, Apllication Transaction Counter (ATC)
    - 2E 9F 26 08 82 8E 0A 3E 70 D4 4A D4, Application Cryptogram
    - 9F 10 0A 06 16 0A 03 25 80 00 02 00 00, Issuer Application data

Global Platform

- **The Global Platform standards (GP) specify the mechanisms for the management of the applications embedded in secure elements, i.e. software downloading, activation or deletion.**
- Mutual authentication procedures based on shared secrets enable the creation of secure channels used to store and install JAVACARD programs.
- Global Platform messages are shuttled by ISO7816 requests and responses.



Main Administration (GP) Commands

Commands	Topic
SELECT	Activate an embedded application. The Issuer Security Domain (ISD) is the application in charge of GP operations
INITIALIZE UPDATE	Initialize mutual authentication with ISD
EXTERNAL AUTHENTICATE	Ends mutual authentication with ISD
DELETE	Delete a package or an application
INSTALL	Allocate memory before package loading or instantiate an application
LOAD	Load a package

Javacard binaries are stored in CAP (Converted Applet) file, i.e. a set of .class files
Packages and applications are identified by AID

SCP01 Mutual Authentication

- Select ISD: A0 00 00 00 03 00 00 00 00
  - **>> 00 A4 04 00 08 A0 00 00 00 03 00 00 00 00**
  - **<< 6F 19**
  - **>> 84 08 A0 00 00 00 03 00 00 00 A5 0D 9F 6E 06 40 51 23 05 21 14 9F 65 01 FF 90 00**
- initialize-update
  - CLA=80 INS=50 P1=00 (key version), P2=00 P3=08, host challenge = 9D B1 90 58 6D 84 B6 96
    - **>> 80 50 00 00 08 9D B1 90 58 6D 84 B6 96**
    - **<< 00 00 23 25 00 47 30 90 18 09 FF 01 57 99 34 CB BC AE 75 9B 90 4C 79 38 1B 9A E2 79 90 00**
  - Key diversification data 10 bytes 00 00 23 25 00 47 30 90 18 09
  - Key information 2 bytes FF 01 FF=Key Version Number 01=Channel Protocol Identifier,
  - Card challenge 8 bytes 57 99 34 CB BC AE 75 9B
  - Card cryptogram 8 bytes 90 4C 79 38 1B 9A E2 79
- EXTERNAL AUTHENTICATE
  - P1 = Security level = 00 = No secure messaging expected, P2 = 0
  - Host cryptogram = 29 E5 5B 81 89 02 99 E0
  - Host MAC = E8 4A 14 89 66 54 7A 6C
    - **>> 84 82 00 00 10 29 E5 5B 81 89 02 99 E0 E8 4A 14 89 66 54 7A 6C**
    - **<< 90 00**
- Authentication Done

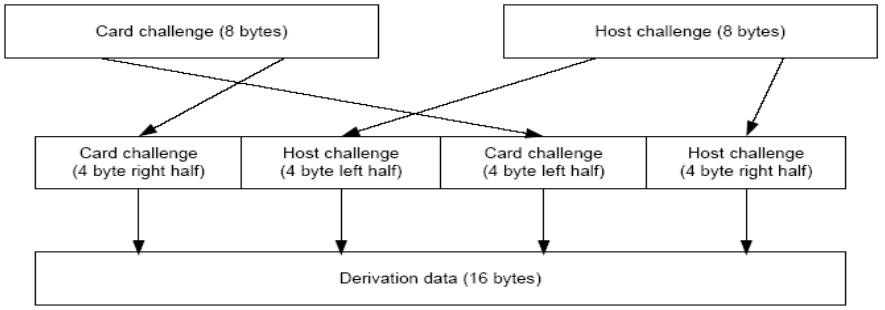


Figure D-3: Session Key - Step 1 - Generate Derivation data

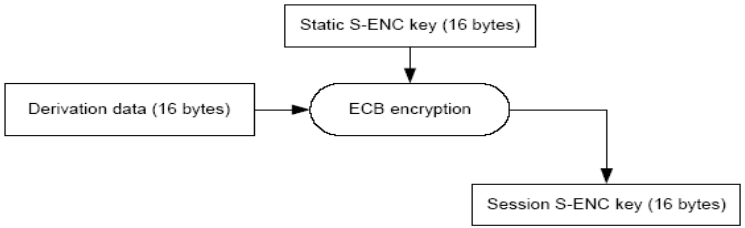


Figure D-4: Session Key - Step 2 - Create S-ENC Session Key

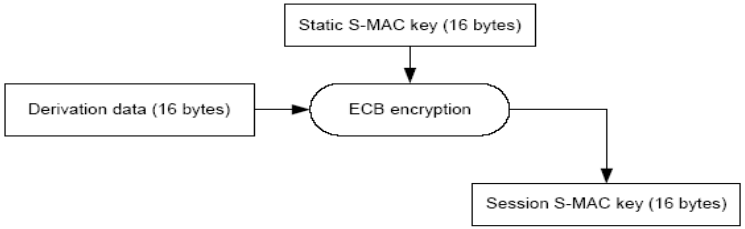


Figure D-5: Session Key – Step 3 - Create S-MAC Session Key

The VISA GP Model (SCP02)

KMC-ID (6B) CSN (Chip Serial Number, 4Bytes)

KMC (DES Master Key for Personalization Session Keys, 6 Bytes)

KEYDATA = KMC_{ID} || CSN = Key diversification data (10 Bytes)

KENC := $DES3_{KMC}$ [Six least significant bytes of the KEYDATA || F0 || 01] ||
$DES3_{KMC}$ [Six least significant bytes of the KEYDATA || 0F || 01].

KMAC := $DES3_{KMC}$ [Six least significant bytes of the KEYDATA || 'F0' || 02] ||
$DES3_{KMC}$ [Six least significant bytes of the KEYDATA || '0F' || 02].

KDEK := $DES3_{KMC}$ [Six least significant bytes of the KEYDATA || 'F0' || 03] ||
$DES3_{KMC}$ [Six least significant bytes of the KEYDATA || '0F' || 03]

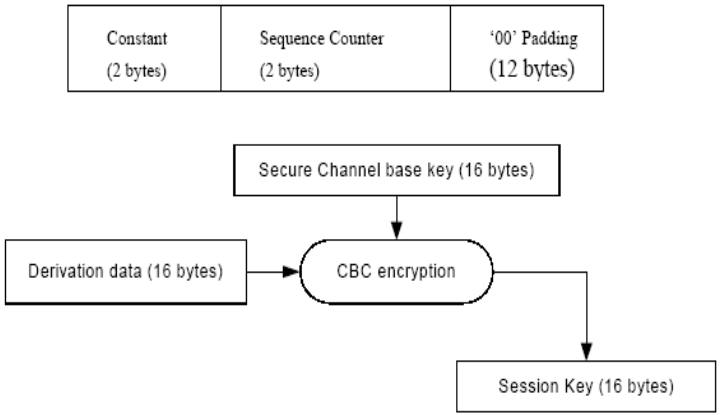


Figure E-2: Create Secure Channel Session Key from the Base Key

Session Key	IC Card Key	Derivation Data
SKU_{ENC}	K_{ENC}	'0182' || sequence counter || '00000000000000000000'
SKU_{MAC}	K_{MAC}	'0101' || sequence counter || '00000000000000000000'
SKU_{DEK}	K_{DEK}	'0181' || sequence counter || '00000000000000000000'

*EMV Card Personalization Specification Version 1.1 July 2007

The Smartcard Virtual Age

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

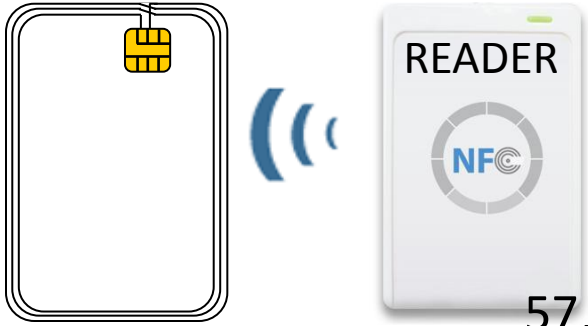
- The basic idea of Wi-Fi design was Wireless Ethernet.
- The basic idea of ISO 14443 design was Wireless (ISO 7816) Smartcard.
  - **Contrary to IEEE 802.11 there is no security features at the radio frame level.**

ISO 7816
Contact Mode

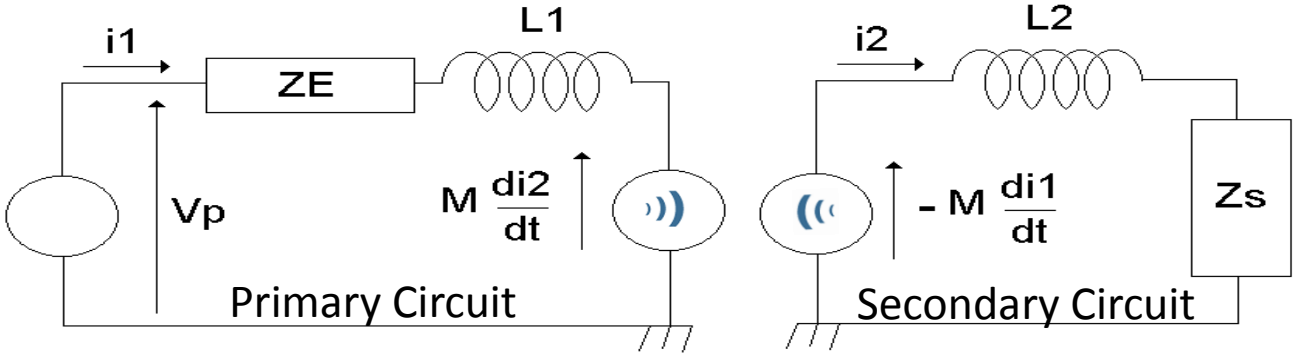
$$V = 2 \pi f_c S \mu_o H$$

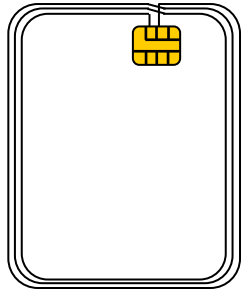
H = 5 A/m
fc= 13,56 10^6 Hz
S= 40.10^{-4}
V= 2,2V

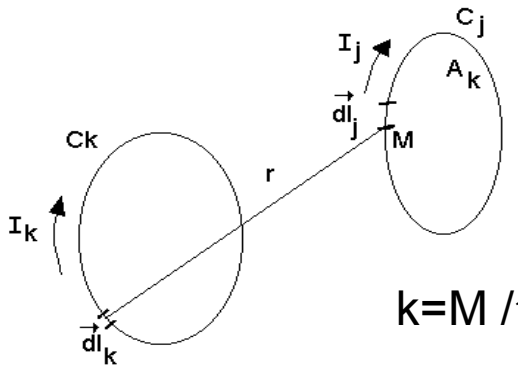
ISO 14443
Contactless Mode



About Inductive Coupling







Neumann formula

$$\Phi_{jk} = M_{jk} I_k = \frac{\mu_0 I_k}{4\pi} \oint_{C_j} \vec{dl}_j \oint_{Ck} 1/r \, \vec{dl}_k$$

$$M_{jk} = M_{kj}$$

$$k = M / \sqrt{L_1 L_2}$$

$$\begin{vmatrix} \Phi 1 \\ \Phi 2 \end{vmatrix} = \begin{vmatrix} L1 & M \\ L2 & M \end{vmatrix} \begin{vmatrix} i1 \\ i2 \end{vmatrix}$$

No energy propagation, implies vicinity proof.

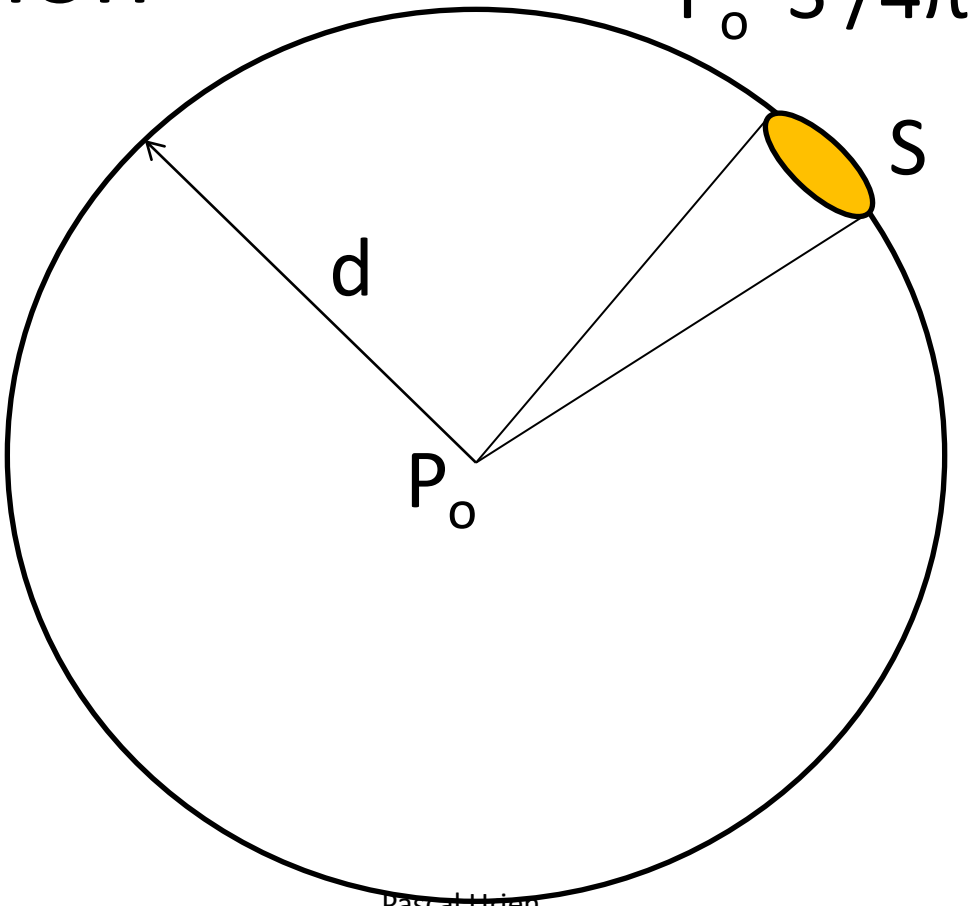
The Energy is conservative, i.e.

The Energy delivered by the primary circuit $P_{PRI} = i1 \cdot (M \omega i2)$,

is equal to the energy consumed by the secondary circuit $P_{SEC} = i2 \cdot (M \omega i1)$.

Propagation

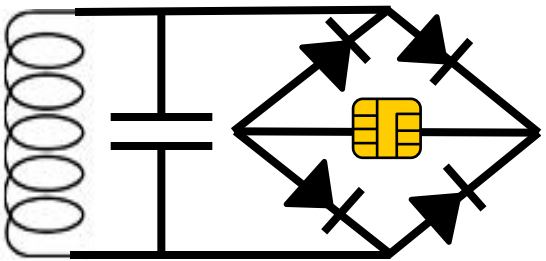
$P_o \; S / 4\pi d^2 \; e^{-d/\lambda}$



The Near Field Communication Age

- The early Near Field Communication technology was normalized by the ISO 14443 standards at the beginning of the 21st century
- It was targeting contactless applications, mainly for ticketing services.
  - It is possible to draw a parallel with the Wi-Fi technology that delivers seamless IP services; **however and contrary to Wi-Fi framework, ISO 14443 doesn't natively support security facilities (i.e. data privacy and integrity)** for radio packets,.
- ISO 14443 works at the 13,56 MHz frequency, the secure microcontroller is fed by an inductive process.
  - For a magnetic field of 5A/m and a loop area of 8x5 cm^2 the resulting voltage is about 2,2V (per loop).
  - Several data coding schemes are defined (referred as A, B, and F), which support a throughput ranging from 104 to 848 Kbits/s.

- NFC standards build an extended framework over ISO 14443. They defined three working modes:
  - **reader/writer,** it is a CAD device, compatible with ISO 14443 standards, feeding a contactless device, and supporting **typeA, typeB and typeF** radio coding schemes.
  - **card emulation**, it is a set of protocols working with reader/writer appliances, which are realized by secure elements or by pure software means.
  - **Peer to Peer (P2P**), two devices, the "Initiator" and the "Target" establish a NFC session using protocols (incompatible with ISO 14443) defined by the NFCIP-1 standard



NFC Standards Overview

French B'

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

NFC-SEC

DEP

Passive Mode
Active Mode
NFCIP-1

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

DEP: Data Exchange Protocol

NFC Radio

ISO 14443
106 kbps
212 kbps
424 kbps
848 kbps

Standard	Reader to Card	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

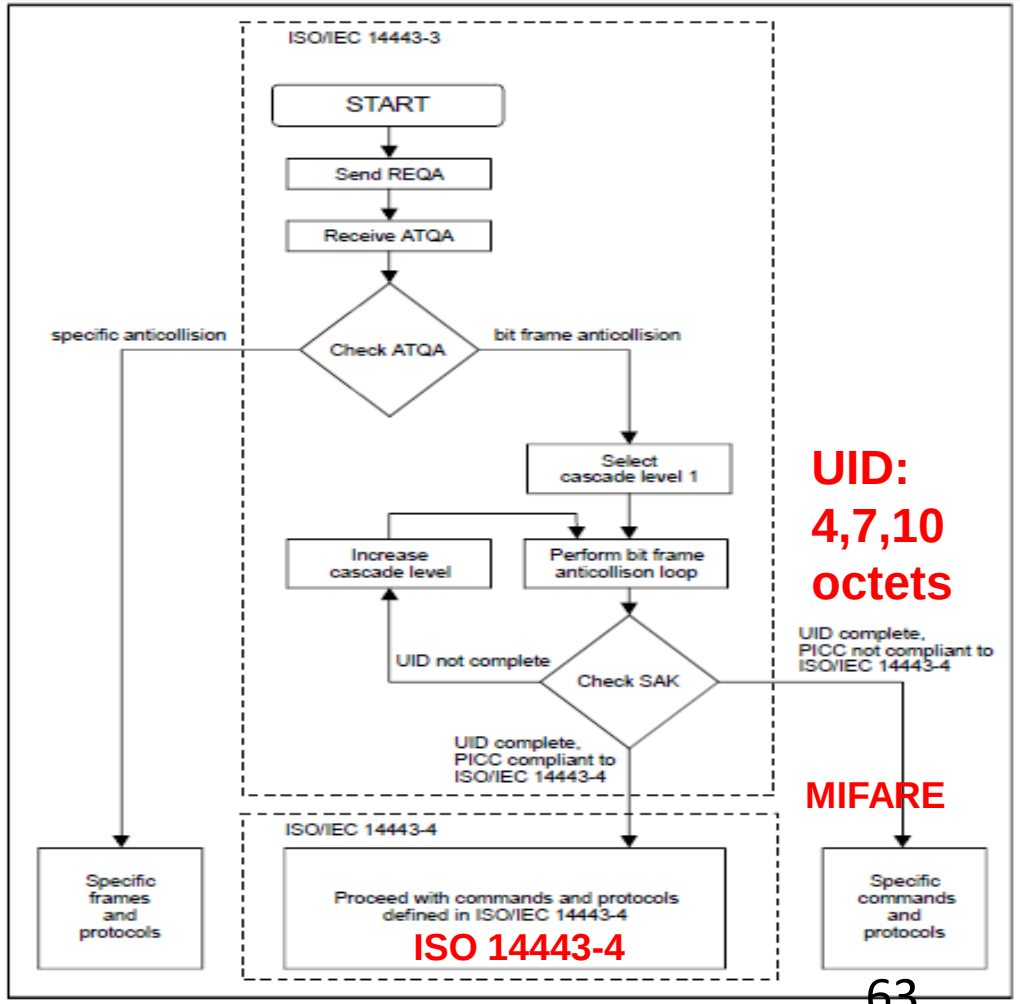
ISO 14443-3A State Machine

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

About Anti-collision

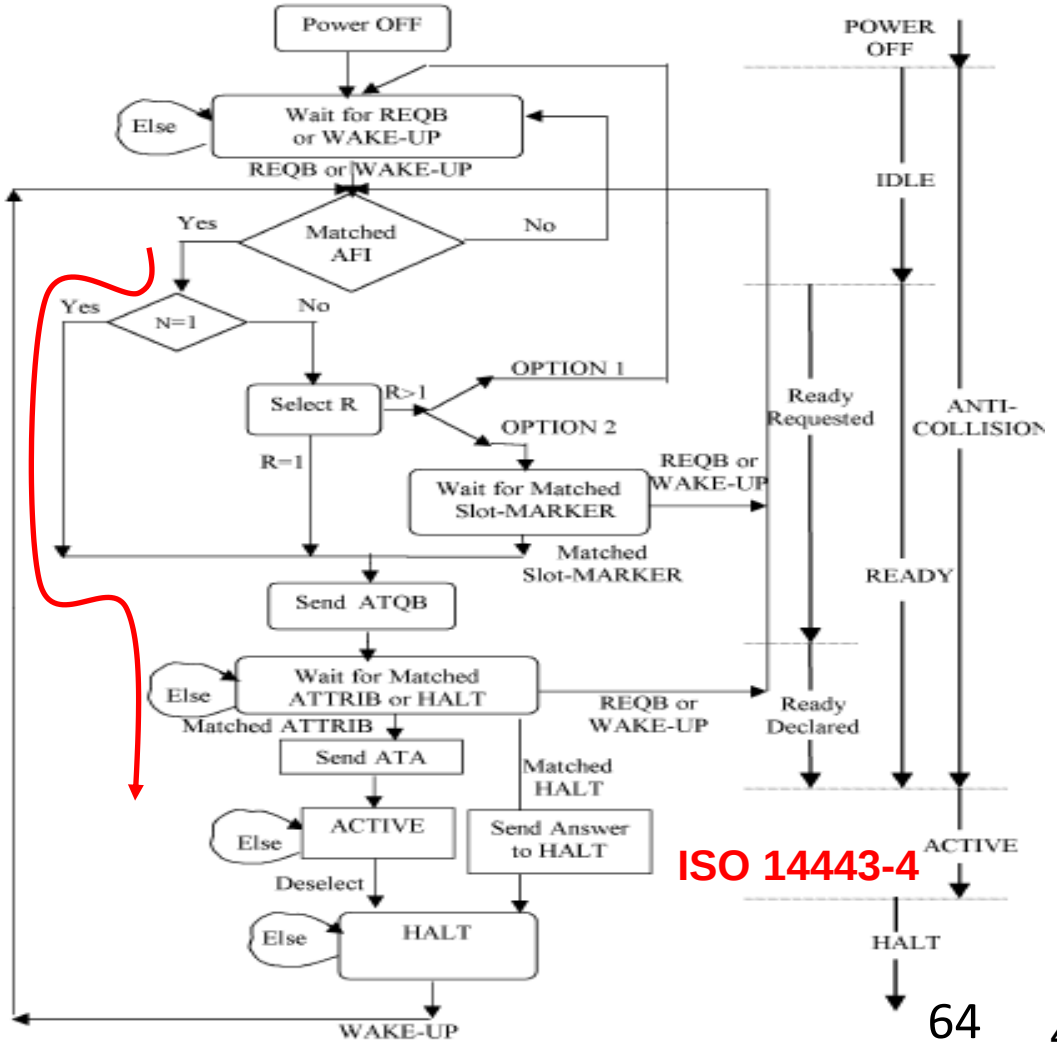
Several devices may be detected at the same time.

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

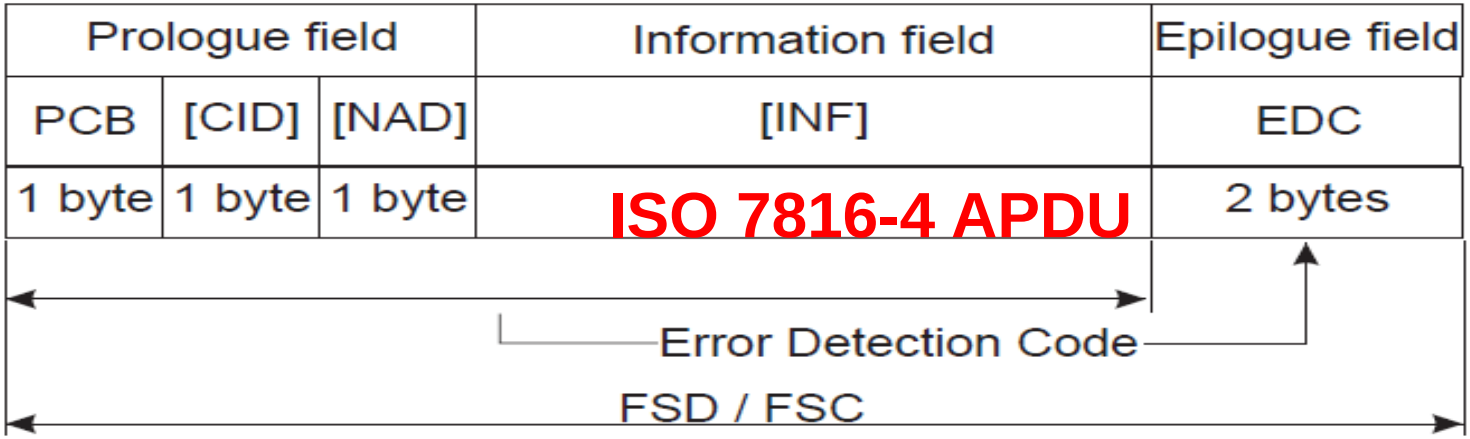


Figure 14 — Block format

ISO 14443-4

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

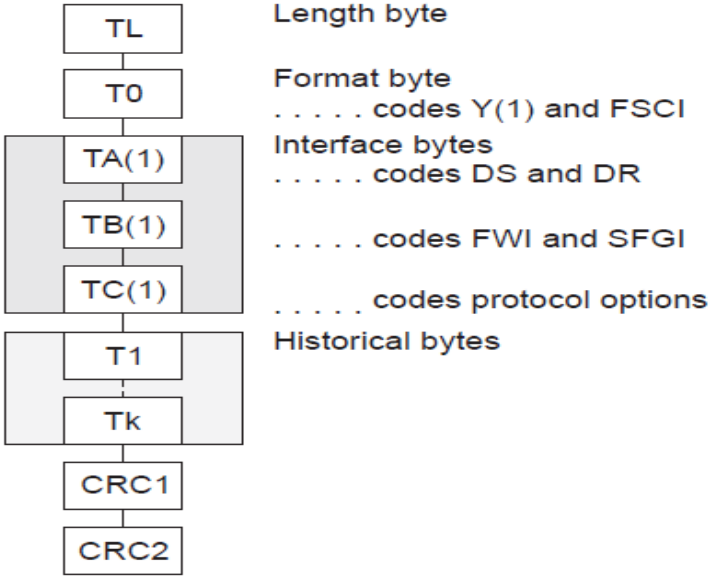


Figure 4 — Structure of the ATS

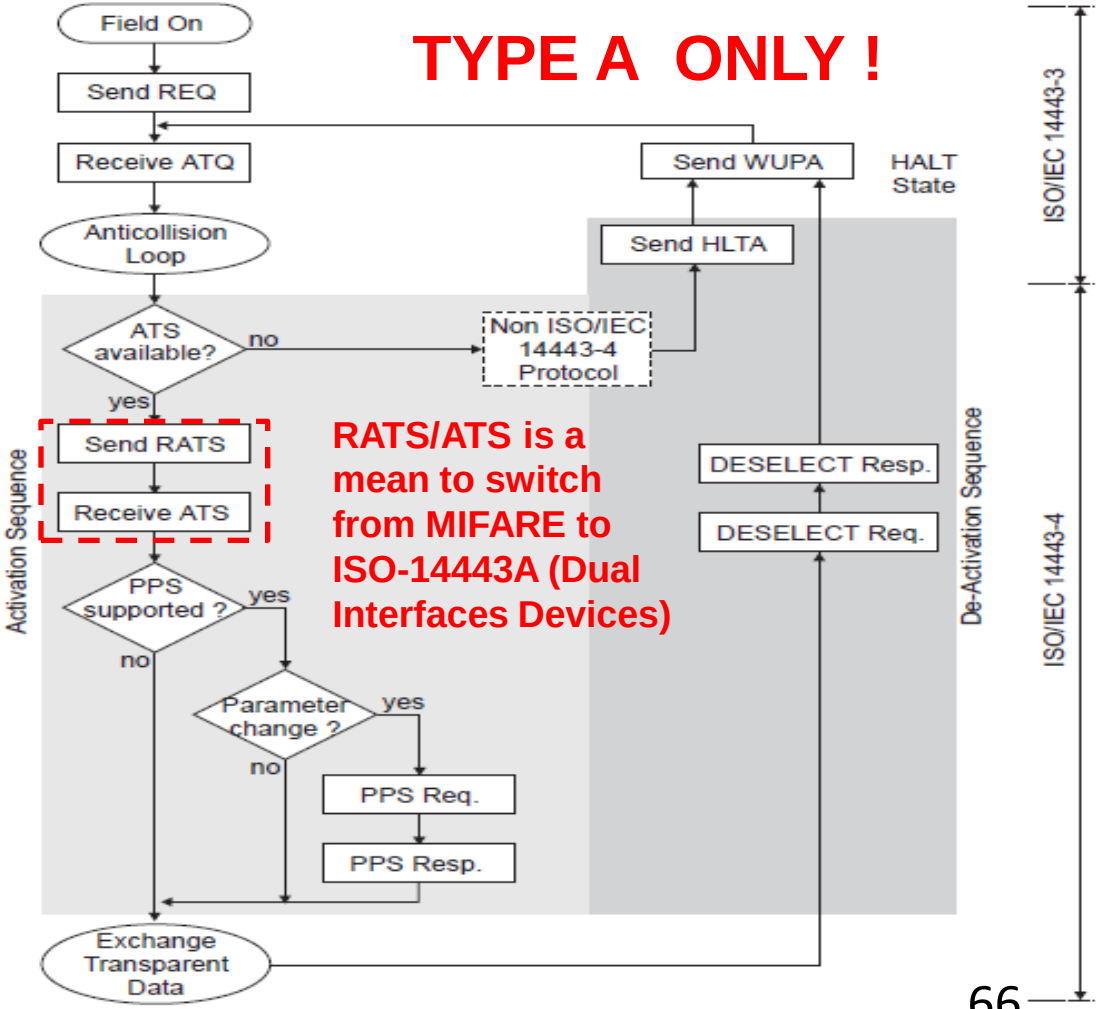


Figure 1 — Activation of a PICC Type A by a PCD

NDEF: NFC Data Exchange Format

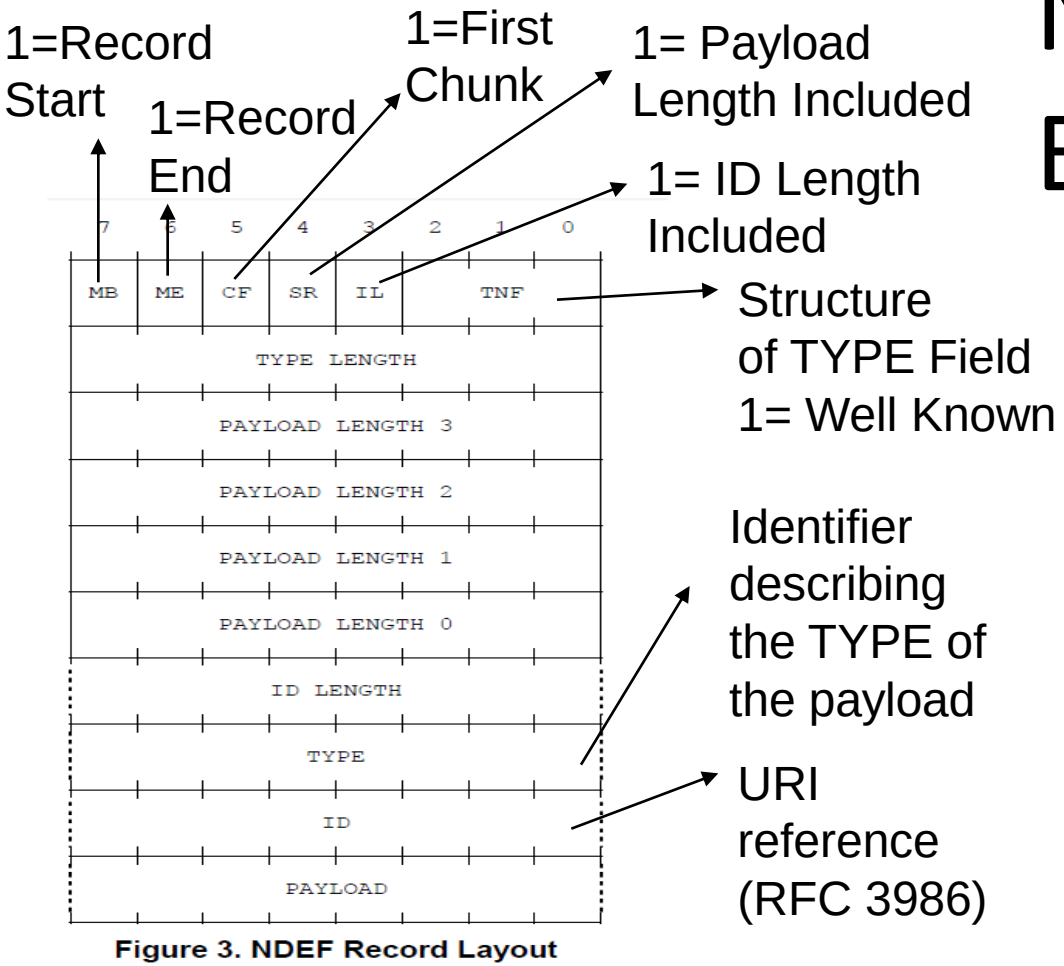


Figure 3. NDEF Record Layout

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 "

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”

Example of Type2 Tag with Mifare

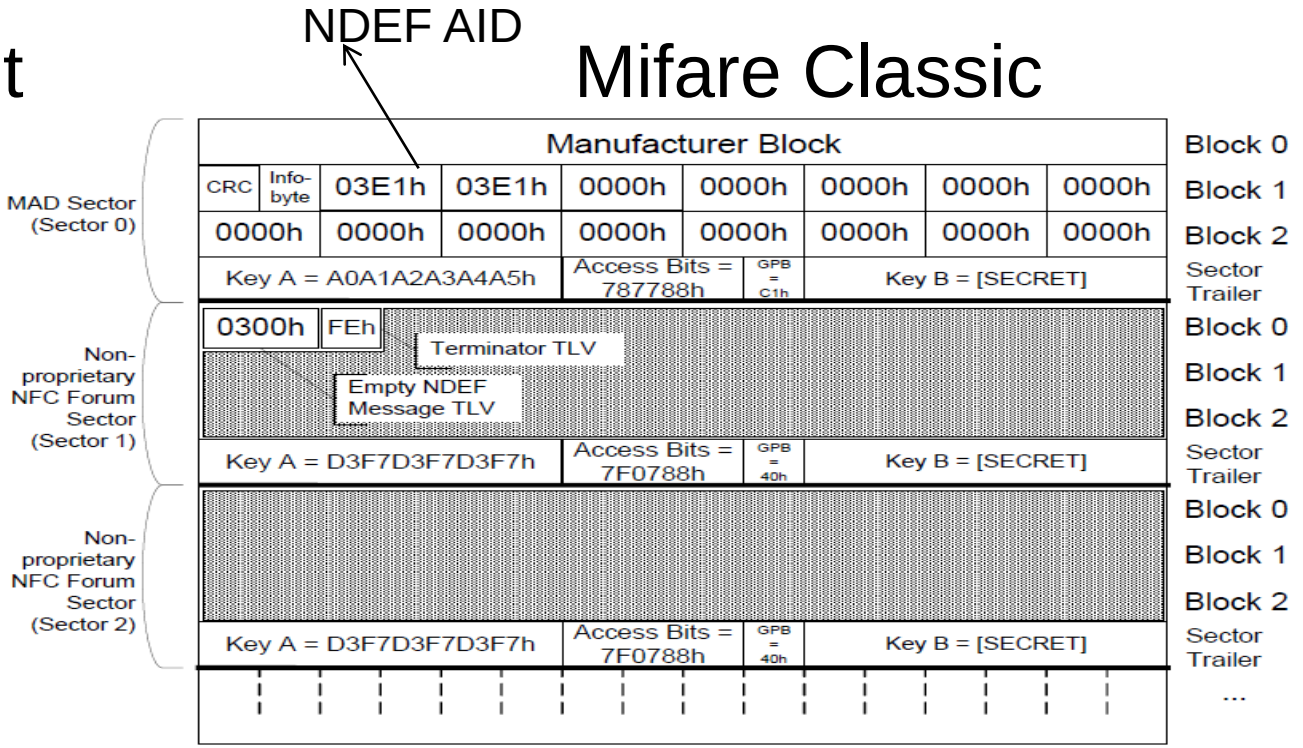
Mifare Ultralight

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



Mifare Application Directory, MAD

Mifare tags are read by Android Mobiles

Type 4 - Tag

```
// select NDEF File
>> 00 A4 00 0C 02 E104
<< 9000
// Read Binary (file length is encoded in the two first bytes of the file)
>> 00 B0 00 00 02
<< 0130 9000
>> 00B0 0000 FF
<< 0130
<< 940FE462636172642E6E65743A626361726433393933233331E313238361F
<< 1F50415343414C1F555249454E1F1F1FA94AA86F9817B8B4372042A7D9F37BB7
<< DC871E694AF83523401004FB19177A136A025AC082DF0DA833A9B9D9D4AB030F
<< E60BE6EB36B00DADD89604B4BE0B19F9B2498E849846D2678981F06E3FAA6F16
<< 0F414F2B1AE3C8EB35280F280D00E4241EA132A6D18BCC886898536E2B8E8D24
<< AA838E58400D7C897C8216D0CD85F94ABFF7C4E3E8AA0036870FA88885FB9089
<< D58DB6A373BFB87B2A525A38DC13677BC5393F5956C461DCEE3409ED8C43C37F
<< C9B9A2BA472CE98EA70D326EA7AE3B9DDEF3D499A6EA4EE751013655006874
<< 9000

>> 00B0 00FF 31
<< 74703A2F2F6D6F62696C652E62636172642E6E65742F617070732F3F453D3132
<< 383626553D383033343438433237313742
<< 9000

// Write Binary (NULL NDEF)
// 00 D6 00 00 05 00 03 D0 00 00
```

http://www.ndefparser.net/

Record .1:
HEADER: 0x94 (MB:1, ME:0, CF:0, SR:1, IL:0, TNF:4)
TYPE LENGTH: 0xf
PAYLOAD LENGTH: 0xe4
ID LENGTH: No ID LENGTH field
TYPE: 0x62636172642e6e65743a6263617264 ("bcard.net:bcard")
ID: No ID field
Payload as Ascii: "399233 1286 PASCAL URIEN J o 7 B { iJ 5#@ z j Z 3 6 I F g n? o AO+ 5(($ 2 h Sn+ $ X@ | | J 6 s {*RZ8 g{ 9?YV a 4 C G, 2n ; N "

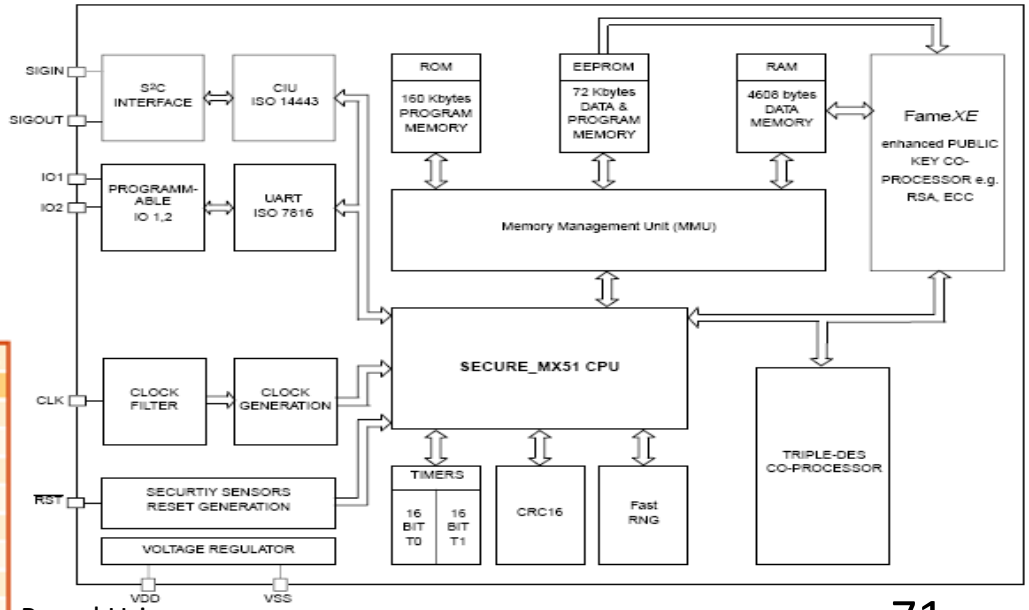
What is a Secure Element ?

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

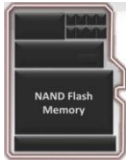
OS JAVACARD JCOP
GP (Global Platform)
ROM 160 KB
EEPROM 72 KB
RAM 4KB
Crypto-processor
3xDES, AES, RSA, ECC
Certification CC (Common Criteria) EAL5+
Security Certificates EMVCo

	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



Secure Elements Market



Eurosmart Estimated WW MP TAM – (Mu)

	2021	2022 forecasts
Telecom*	4700	4600
Financial services	3250	3200 – 3300
Government – Healthcare	510	550
Device manufacturers **	490	520
Transport	220	220-245
Others***	155	150
Total	**9325**	**9240 – 9360**

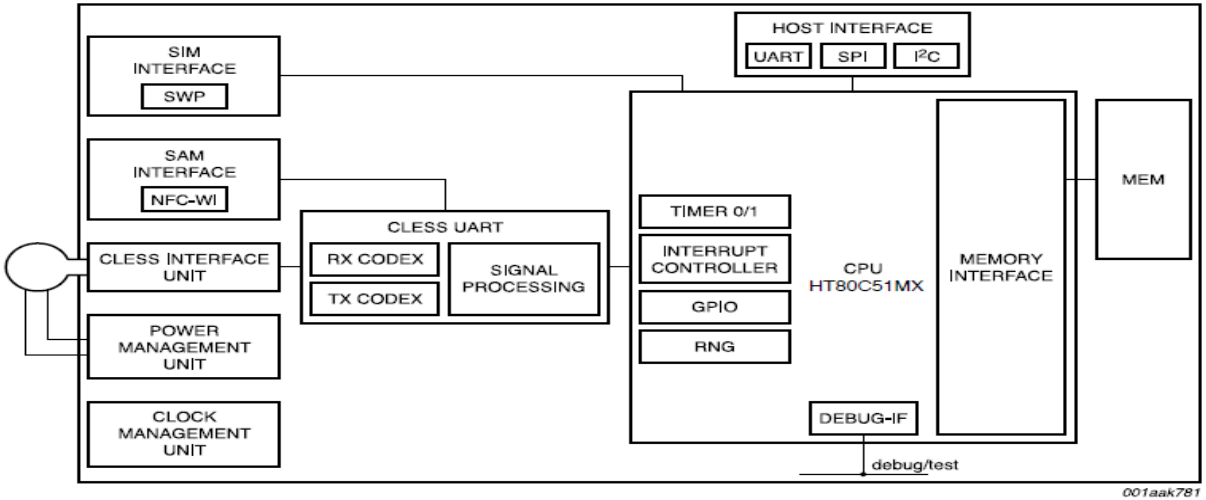
Eurosmart Estimated WW MP TAM – (Mu) Contactles

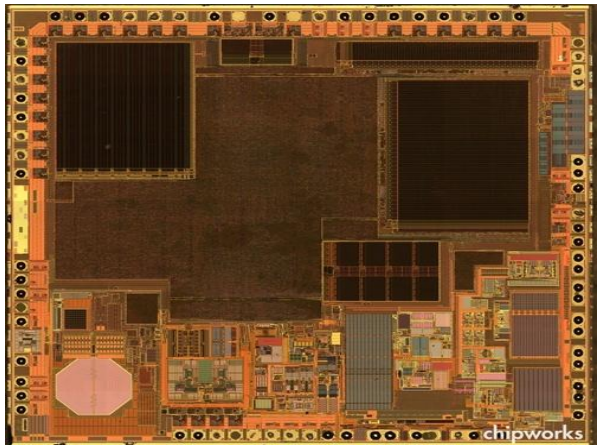
	2021	2022 fo
Financial	2450	2530 -
share of total	*75,4%*	*79,0%*
Government – Healthcare	350	385
share of total	*68,6%*	*70,0%*
Transport	220	220-245
Total	**3020**	**3135 – 3235**

NFC and Secure Elements

- Some NFC Controllers embed a Secure Element
- In that case the card emulation mode may be managed by the embedded secure element
- This is the Google Secure Element Android Model

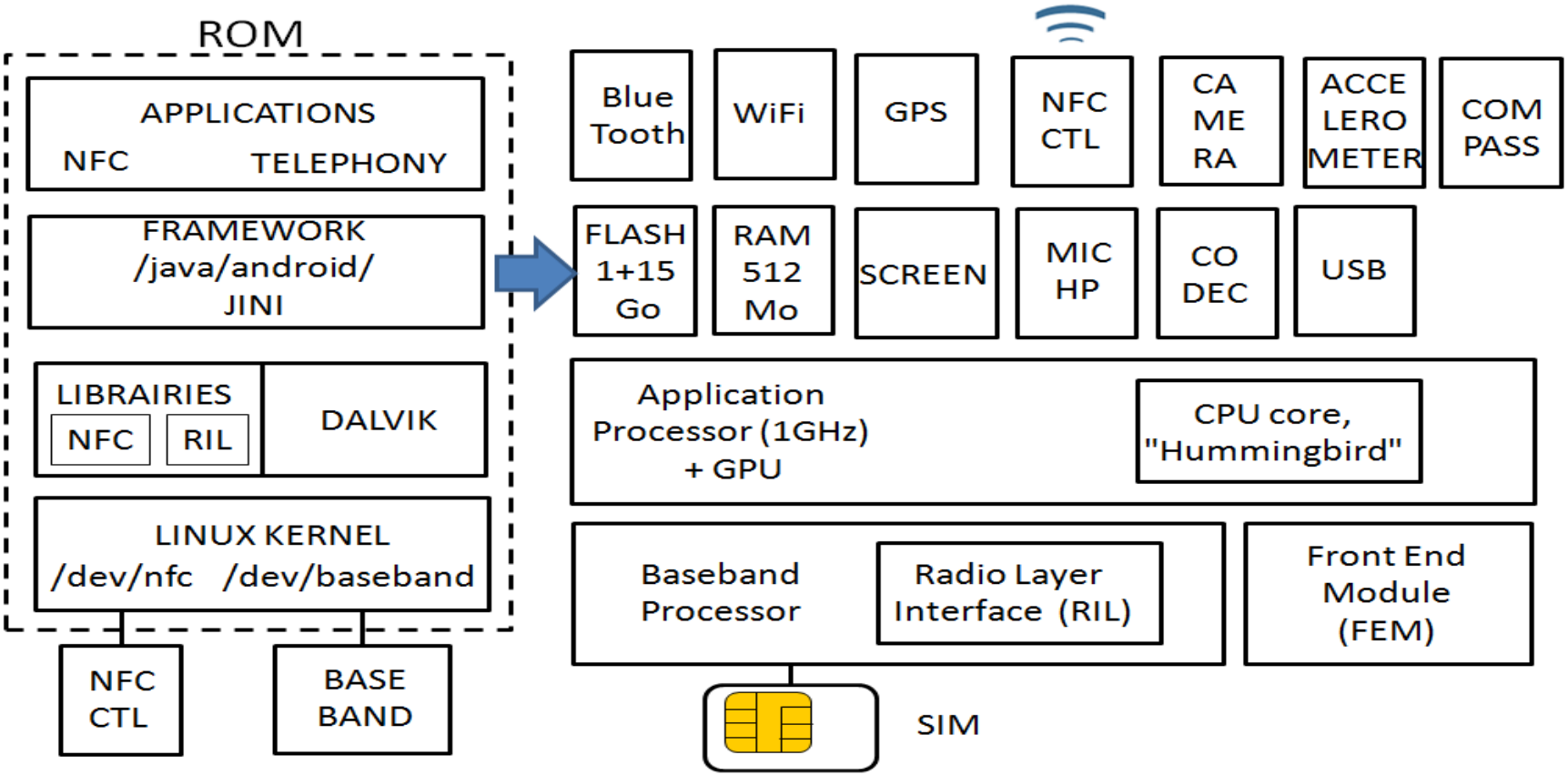
www.chipworks.com





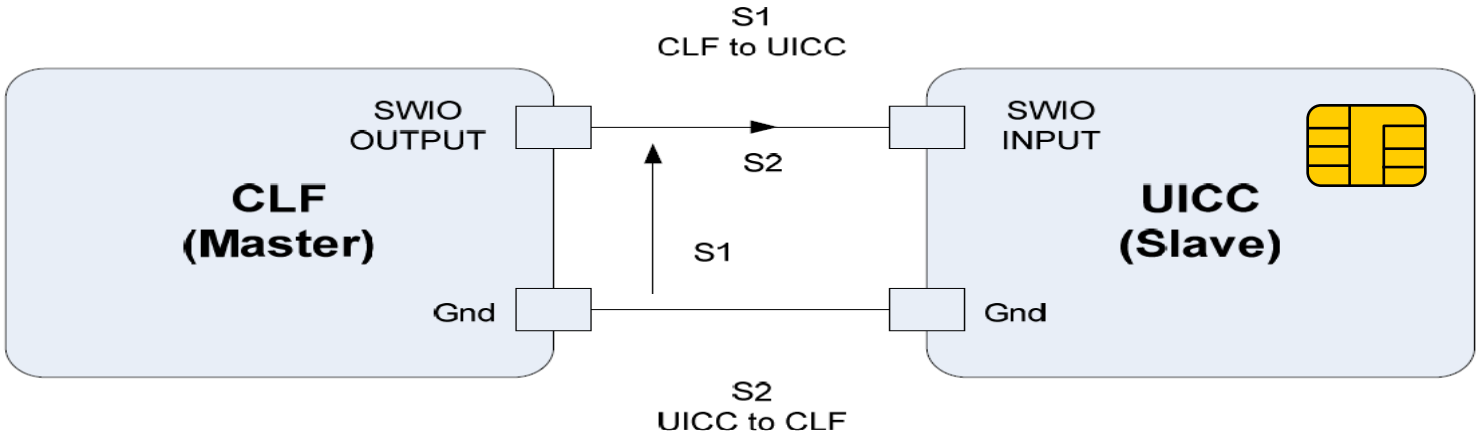
Reader/writer ISO 14443 –A-B, MIFARE, FeliCa®, NFC Forum tags, ISO 15693
Card Emulation ISO 14443 –A-B-B', MIFARE, FeliCa RF , SWP
RAM 5Ko, ROM 128 Ko, EEPROM 52 Ko

Android, Nexus S, 2011



The SIM card becomes an NFC device: the Contactless Front-end (CLF)

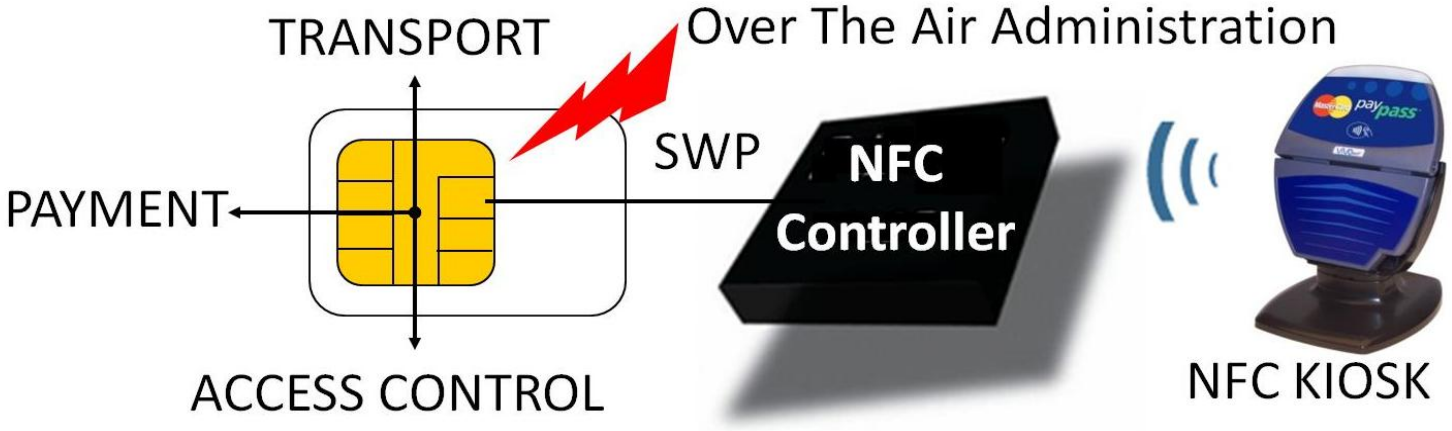
The ETSI TS 102 613 Standard



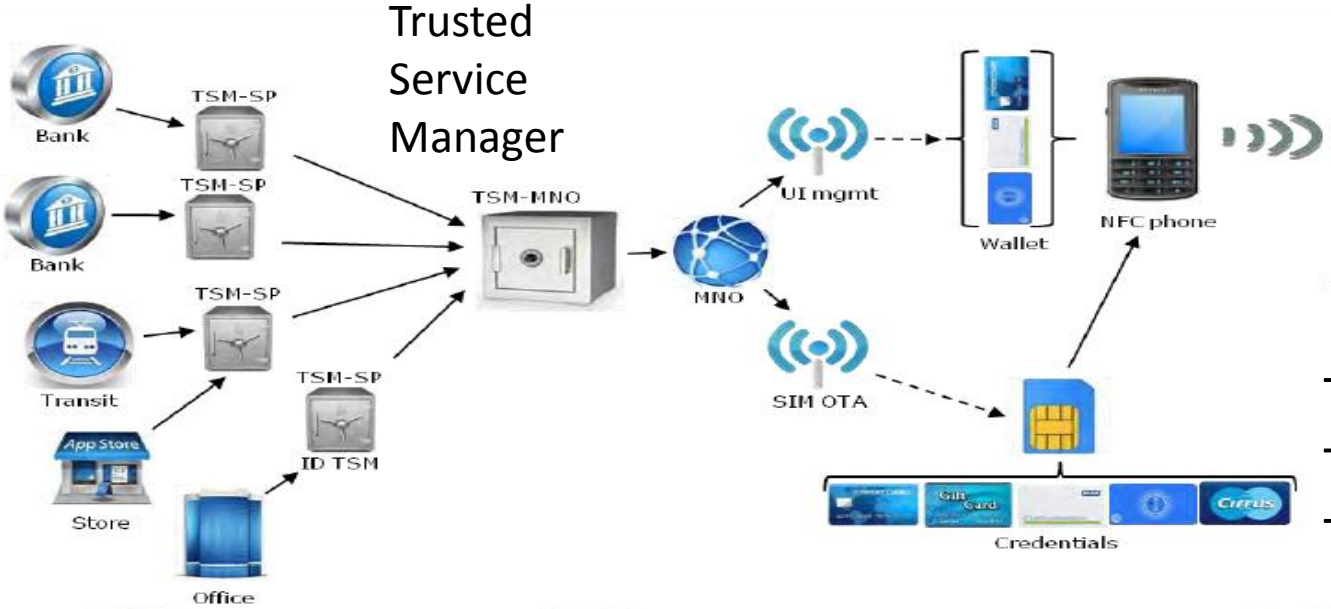
A simplified HDLC protocol: SHDLC

A physical Link: Single Wire Protocol (SWP)

SIM-Centric Legacy Paradigm



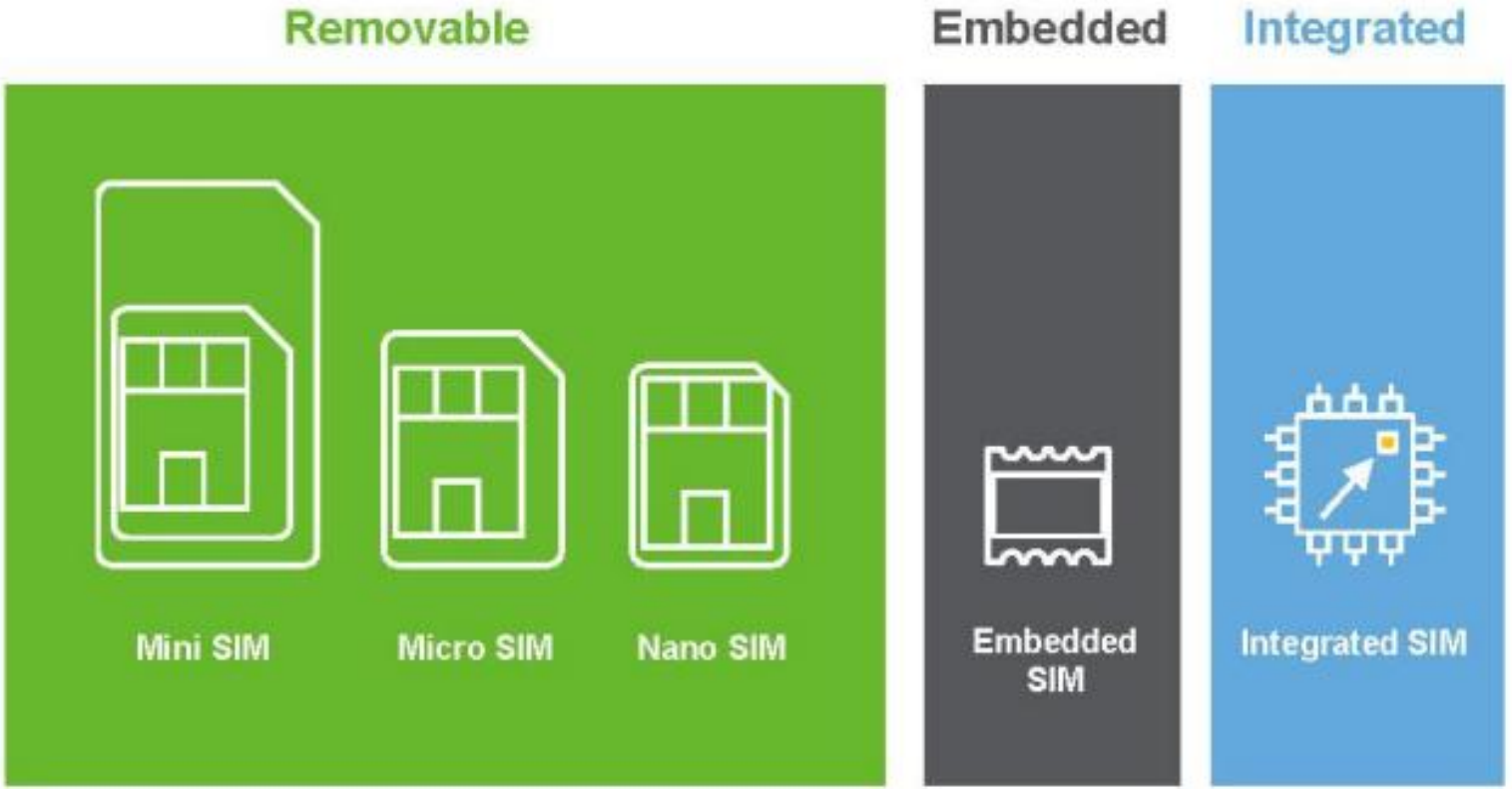
HID NFC White Paper: SIM Centric Services



- Payment
- Access Control
- Transport

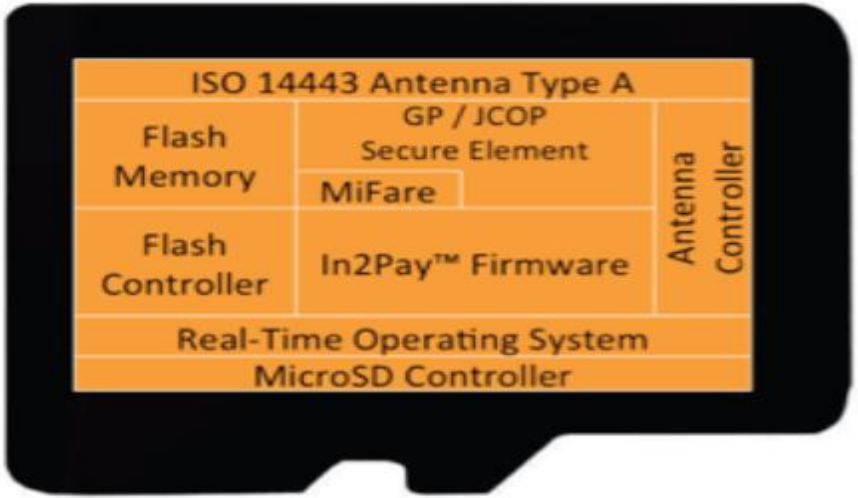
NFC ecosystem with the Secure Element in the SIM and one MNO

SIM Form Factor



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

Google Vault : a SD Card

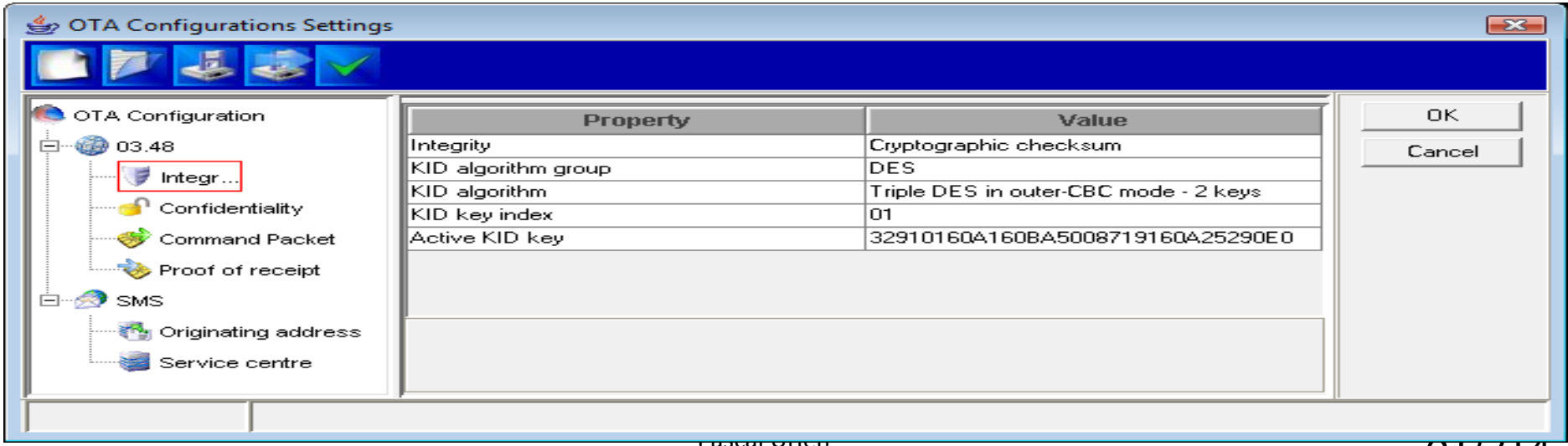
Research Hardware / Development Kit

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

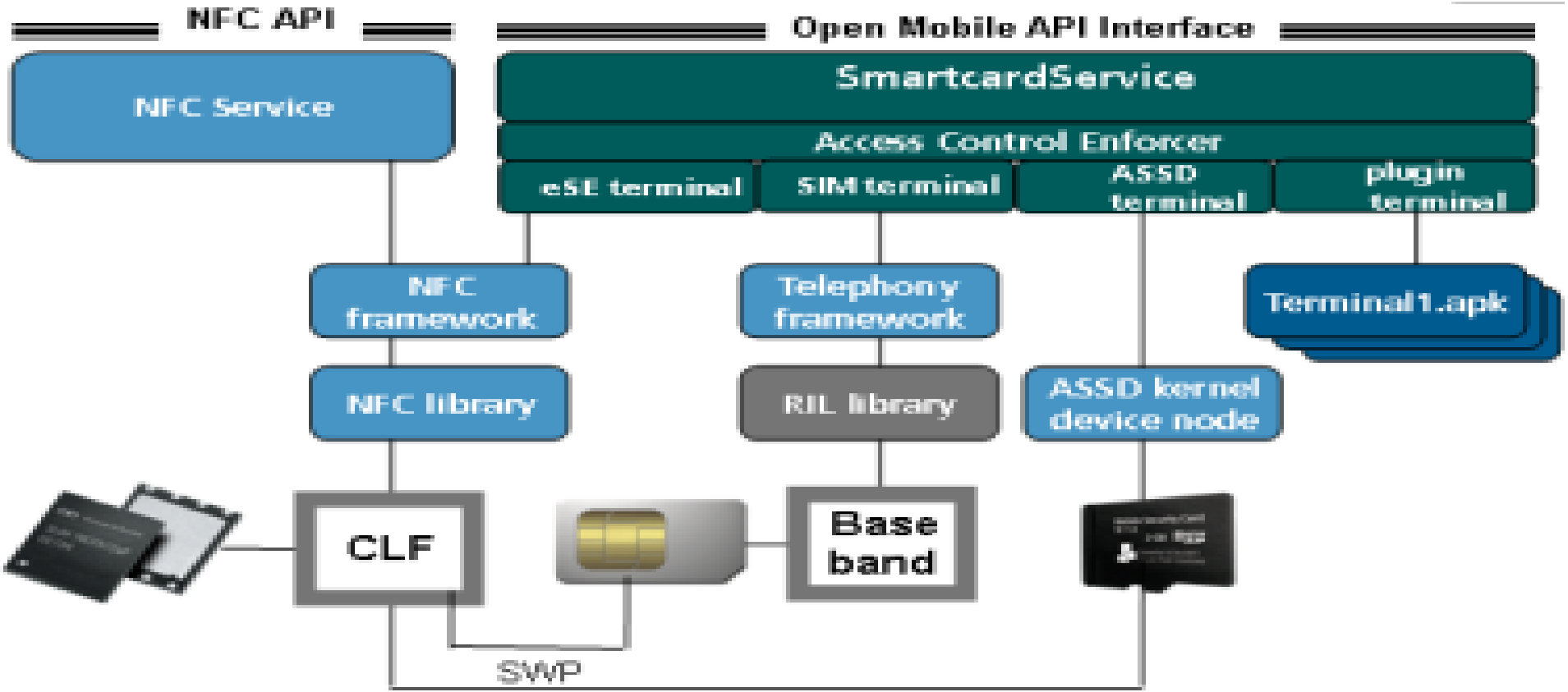
- SD card
- Only two files: WFILE and RDFILE
- Cryptographic procedures
- GB of storage
- MB of throughput
- NFC controller

3GPP TS 03.48

- This standard defines a secure transport of APDU over SMS. It enables the remote management (via the Global Platform framework) of SIM card



2011, Open Mobile API



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()

Open Mobile API & Security Policy

- In order to protect the USIM from a non-authorized Android application, an access control (AC) mechanism based on the PKCS#15 standard is used.
  - The PKCS#15 repertory (hosted by the SIM) contains three files defined for the access rules
  - The Access Control Main File (EF-ACMF) gives a reference to the Access Control Rules File (EF-ACRF)
  - The Access Control Rules File (EF-ACRF) stores a list of Access Control Conditions File (EF-ACCF), each of them being associated to a particular AID.
  - **Each Access Control Conditions File (EF-ACCF), contains a SHA1 digest of the mobile application whose access to embedded software running in the Secure Element (and identified by its AID) is authorized.**

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 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

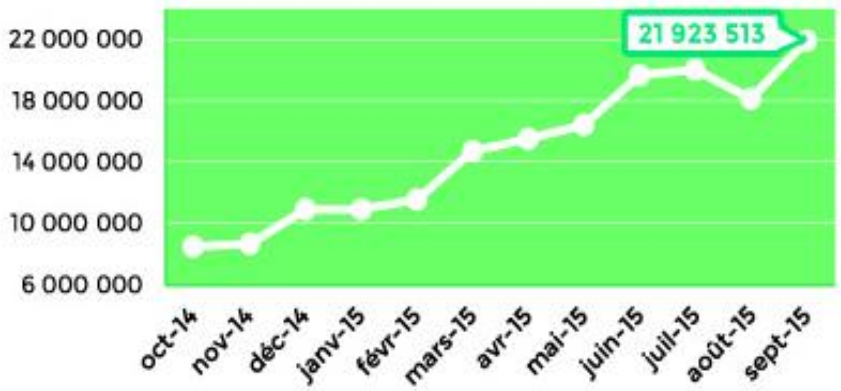
Tx: 00 A4 00 04 02 **43 12** // Select AC-Conditions3
Rx: 61 20
Tx: 00 C0 00 00 20
Rx: 62 1E 82 02 41 21 83 02 43 12 A5 06 C0 01 00 DE 01 00 8A 01 05 8B 03 6F 06 02 80 02 00 1E 88 00 90 00
Tx: 00 B0 00 00 00 // Read AC-Conditions3, access by any application
Rx: 6C 1E
Tx: 00 B0 00 00 1E
Rx: 30 00 // empty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

About Mobile Payments

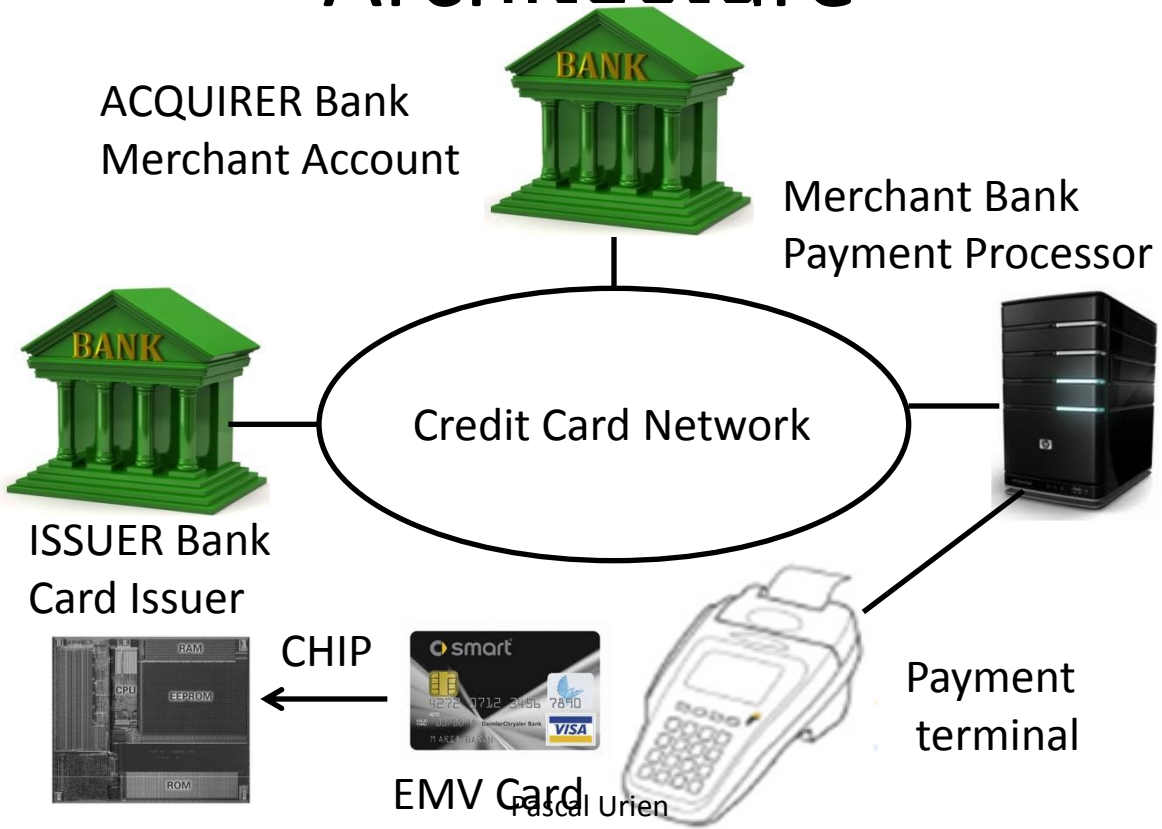
US payment cards market
2011: 21 trillion $

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



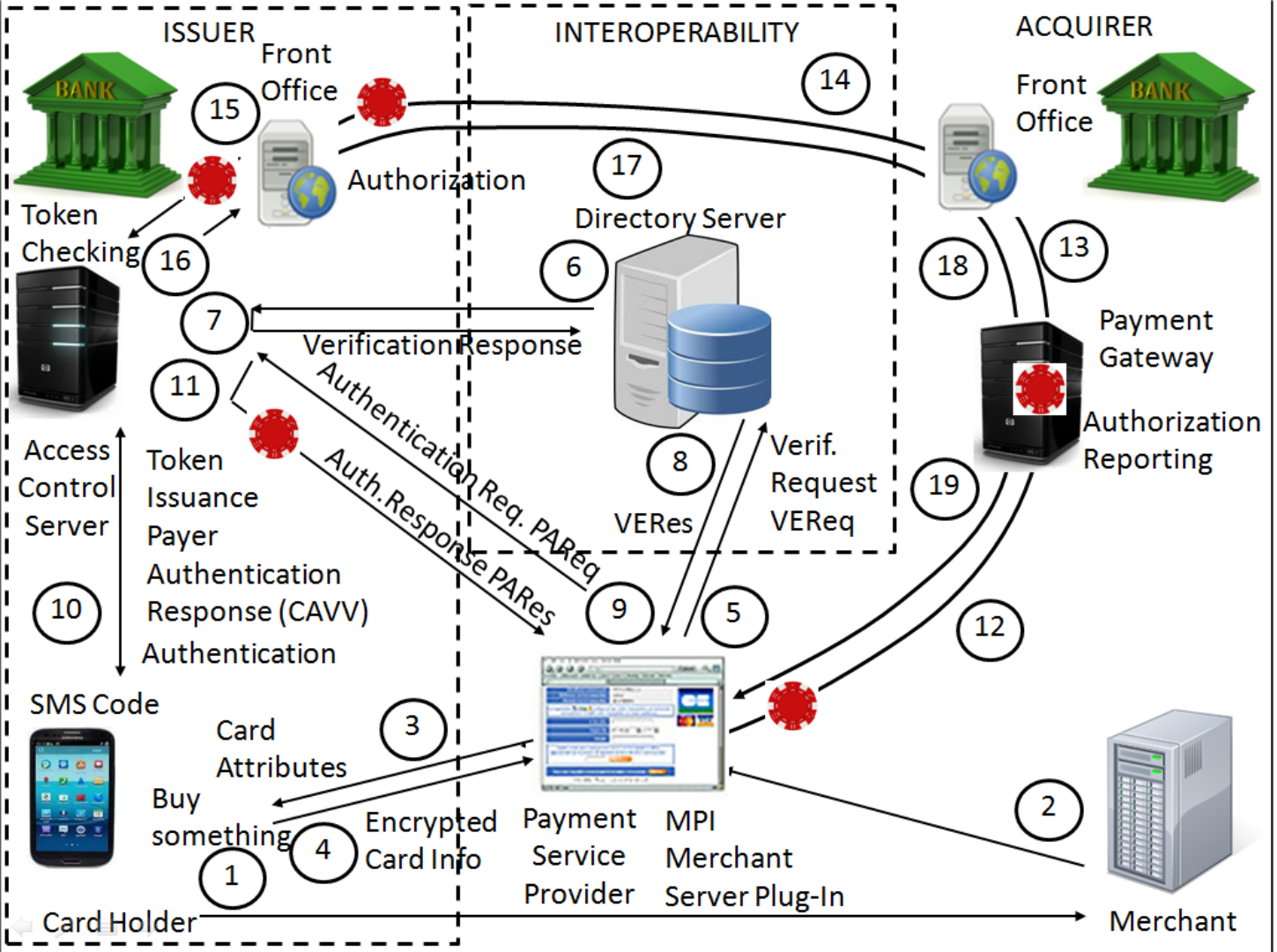
About the EMV Payment Four Corner Architecture



A payment processor is a company (often a third party) appointed by a merchant to handle transactions from various channels such as credit cards and debit cards for merchant acquiring banks. They are usually broken down into two types: front-end and back-end.

Classical Online Payments

- Card Not Present (CNP) transaction
  - Work with a Payment Service Provider (PSP)
  - Card number, validity date, bearer name
  - But it the predominant fraud
    - Implies insurance costs
- In order to limit the fraud, the Visa and MasterCard companies developed the 3-D Secure protocol
  - It uses an Access Control Server (ACS) that performs additional authentication.
  - A transaction code is typically sent via SMS, to a mobile phone number bound to a credit card



3D Secure

A payment gateway is a merchant service provided by an e-commerce application service provider that authorizes credit card or direct payments processing for e-businesses, online retailers, bricks and clicks, or traditional brick and mortar.
The payment gateway may be provided by a bank to its customers, but can be provided by a specialized financial service provider as a separate service, such as a **payment service provider**.

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

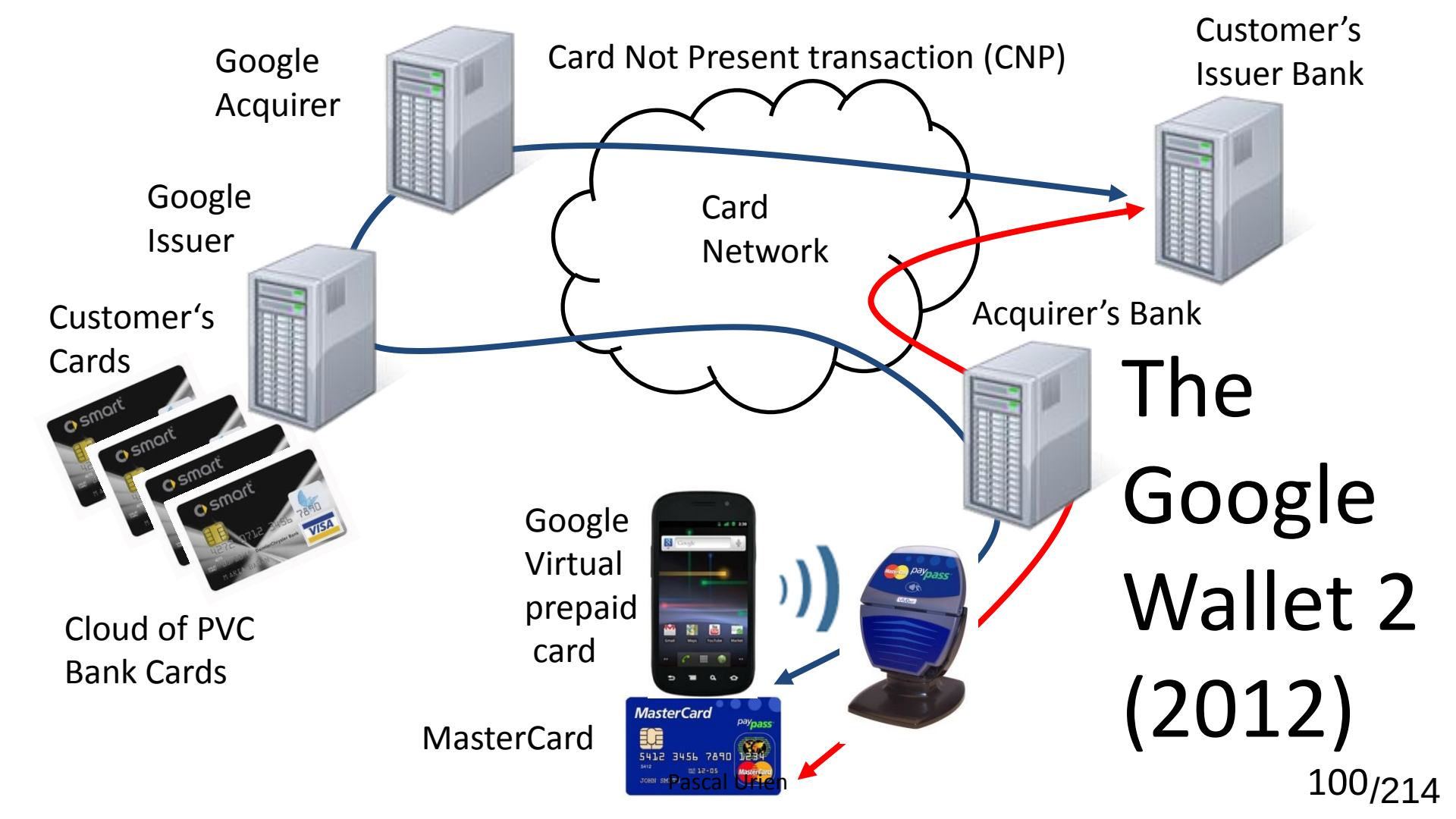
- Financial transactions are associated with cryptogam generation based on symmetric 3xDES cryptographic algorithm.
- One or two dedicated commands (*GenerateAC*) are required by a payment operation, whose input parameters include, among others, the amount and the date.
- DDA and *GenerateAC* may be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



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

M a s t e r C a r d

Google PrePaid Card Transaction

// Get Processing Options
>> 80A80000028300
<< 770A 8202 **0000** 9404 **08010100** 9 000
AIP=0000 AFI= 08010100

// Reader Record one File one
>> 00B2010C00
<< 706A9F6C0200019F6206000000000389F6306000000003C64 **5629** 235343330
3939393930393937393939395E202F5E31373131313031303031303030303030
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

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.

SELECT 2PAY.SYS.DDF01

SELECT Visa App

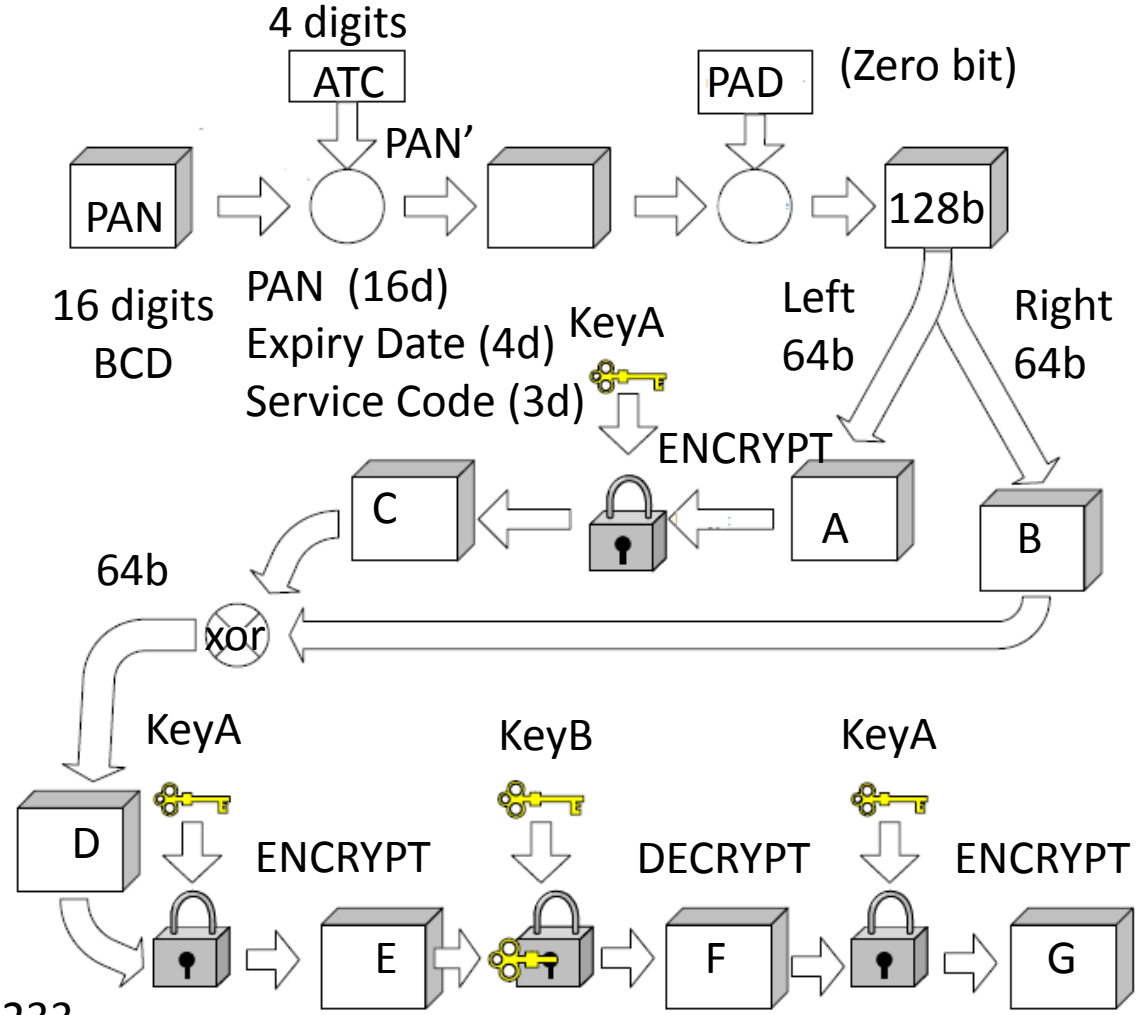
GET PROCESSING OPTIONS

READ first record

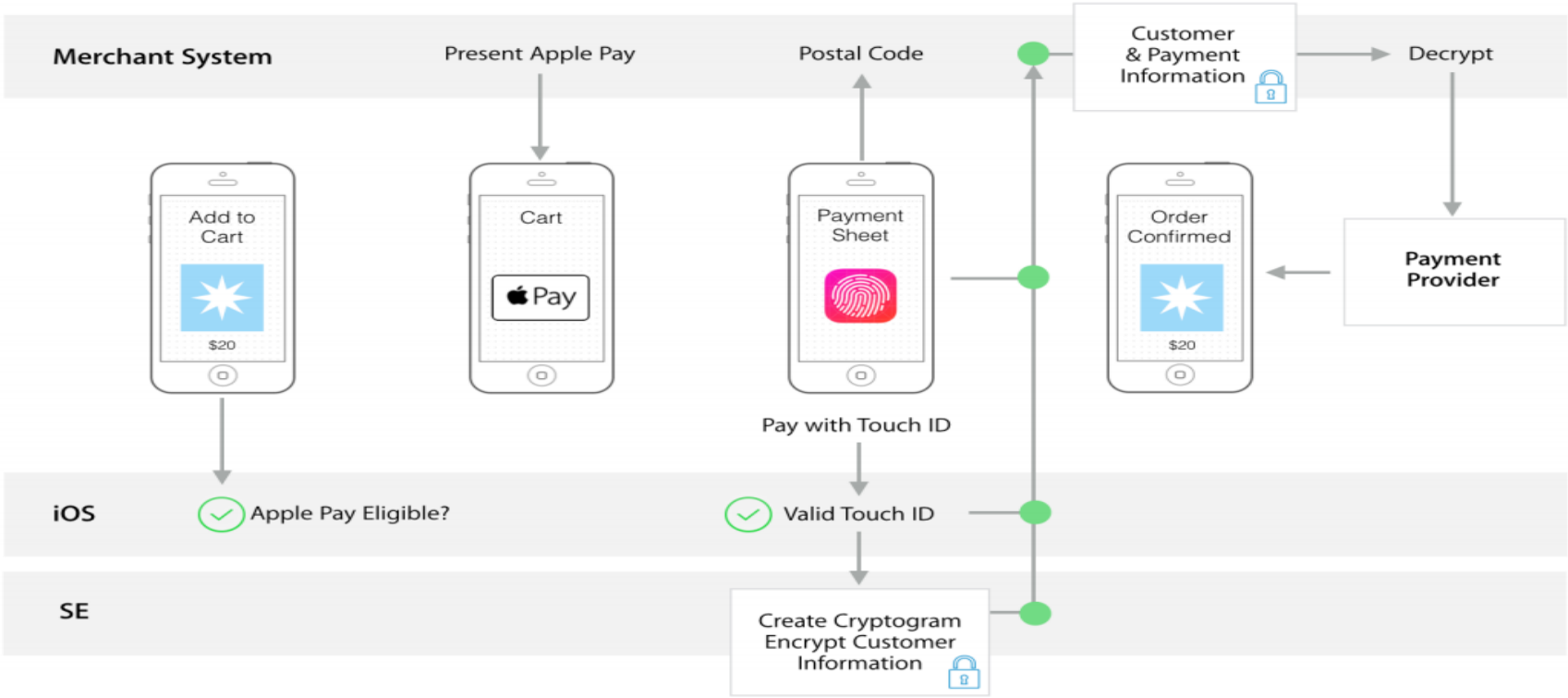
COMPUTE Cryptographic Checksum

G=A23FB35FC89AE3A9
23358939AFBFCAEA
2335893905152040

CVC3=233



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

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

57 Track 2 Equivalent Data

407123131122334 4D 2003 201 0 0000 **509** 00025 F

5F20 Cardholder Name

/

DAN

dCVV

EMV

//Select PPSE
00A404000E325041592E5359532E444446303100

6F23840E325041592E5359532E4444463031A511BF0C0E610C4F07A00000000
41010870101900 0

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 NFC

// Select Master Card
>> 00A4040007A000000004101000
<< 6F388407A0000000041010A52D500A4D617374657243617264870101 5F2D0266
729F1101019F120A4D617374657263617264BF0C059F4D020B0A9000

EMV NFC

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

GPO
80A8000002830000
7716 8202 1980 9410 080101001001010118010200200101000 9000

77 Response Message Template Format 2
82 Application Interchange Profile
94 Application File Locator (AFL)

EMV NFC

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

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 NFC Records and Files Reading

EMV NFC 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
      - Issuer PK Certificate 90 — 176 — 4A9B535D89E798C859F615FC449414008760C4CBD916586ADB36A5E7F353E1F4 1F2B9CAF5C80936BC2C2F74E15C9CC8BD3932B486A9A065AD6449051CD64877A 5544F4B174A9B1904F7EAC75066944EE370C0C79D3C1CD536067606851FD90E1 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
      - ICC Public Key Certificate 9F46 — 176 — AECB52A5B77A12CDCFF814DEA18072OODDE6CFE4C2A5D51CF365741FO66138C6 BO6O2DAA31377A9ABDBO9AB954F28EB78E6B3128D9B46FCCB1261292D4D52EOO 963685B2A3FB7B308DO4FB165E972CEO676A3AO4954E83717621D53DF7C2C208 30B12A14DB624OD5D52D501C1DOO9835BO13244F383C1F80159944E37A4661OF [48 bytes follow...]h
      - ICC Public Key Exponent 9F47 — 1 — 03h

// P1= Generate TC (01xx) + CDA signature Request (xxx1)= 50
80AE**50**002B 000000006290 000000000000 250 0000000000 0978 150610 00 90B4 E0D2 25 0000 0000000000000000 1F0302 00

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

EMV NFC

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

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
A040032230000000000000000000FF

9000

EMV NFC 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**

EMV NFC CDA

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

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbb Padding

154fb6d9d774f5e6f7eef512b557eaf754c9c8f3bc Signature

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

EMV NFC CDA

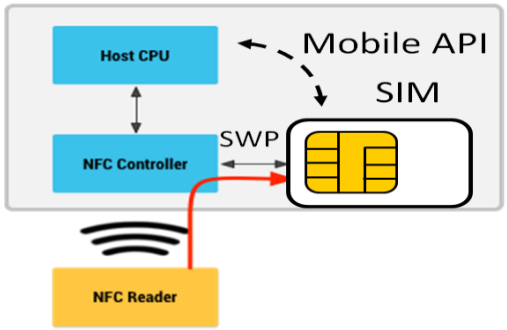
Transaction Data Hash code = hash of
-The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command
- The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal 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 NFC CDA

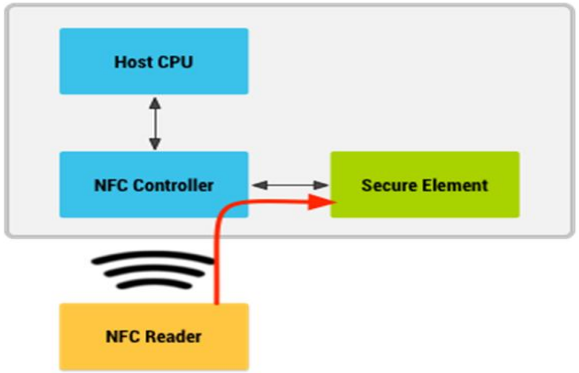
Transaction Data Hash code= sha1 (
00000000062900000000000025000000000009781506100090B4E0D2220000000000000000001F0302
|| 9F27 01 80
|| 9F36 02 001F
|| 9F10 12 0F10A040032230000000000000000000FF)
= 09 a1 86 bd eb 56 60 ba 15 b2 b2 8d 9f 1c b2 e4 74 a6 8d 8c

Host Card Emulation

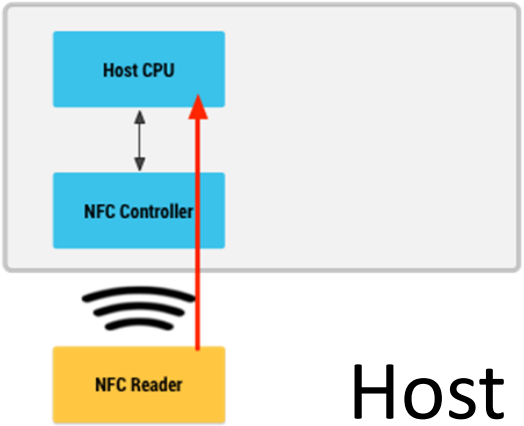
Host Card Emulation



LEGACY



Google Nexus S



Host Card Emulation

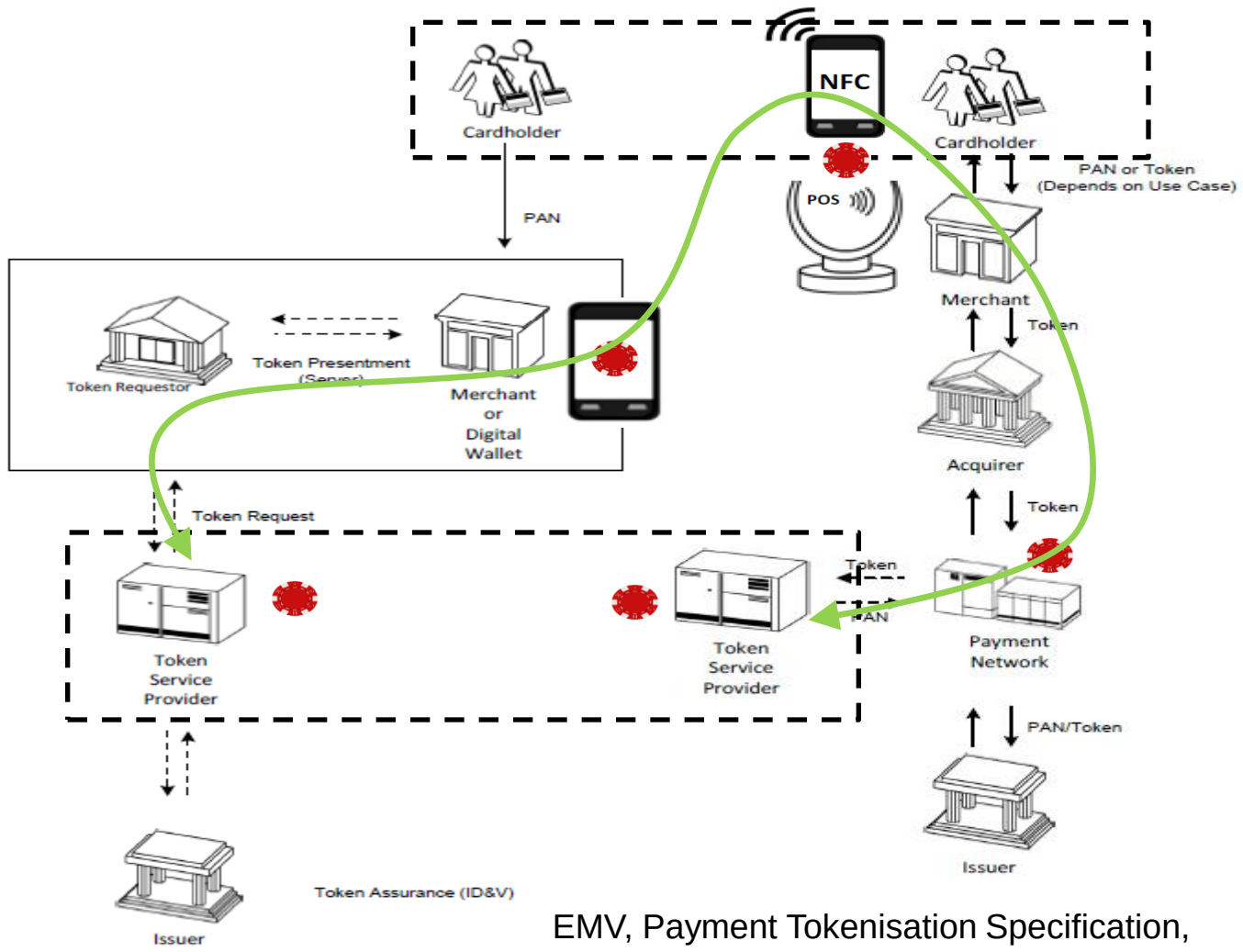
HCE Service

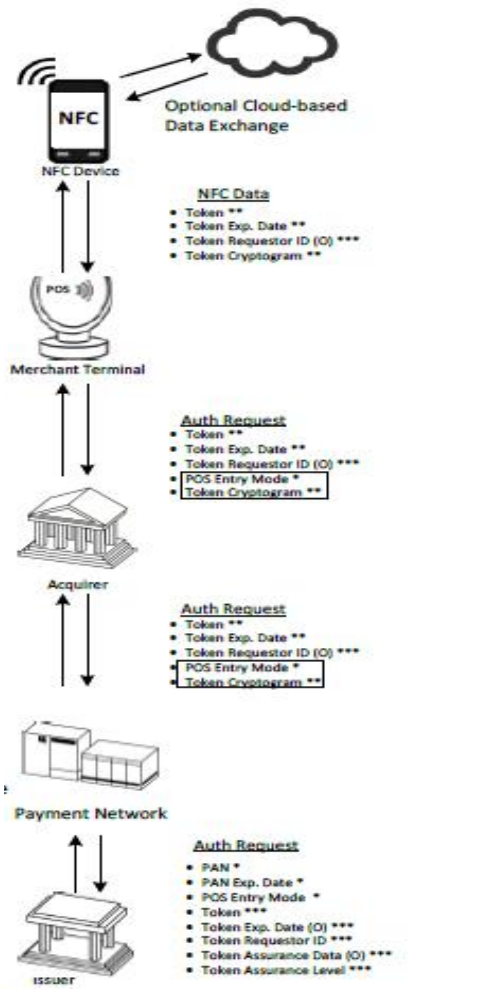
```
<service
  android:name=".MyHostApduService"
   android:exported="true"
   android:permission="android.permission.BIND_NFC_SERVICE" >
  <intent-filter>
 <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
 </intent-filter>
 <meta-data
   android:name="android.nfc.cardemulation.host_apdu_service"
   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= "a0000000041010aa54303200ff01ffff" />
</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).

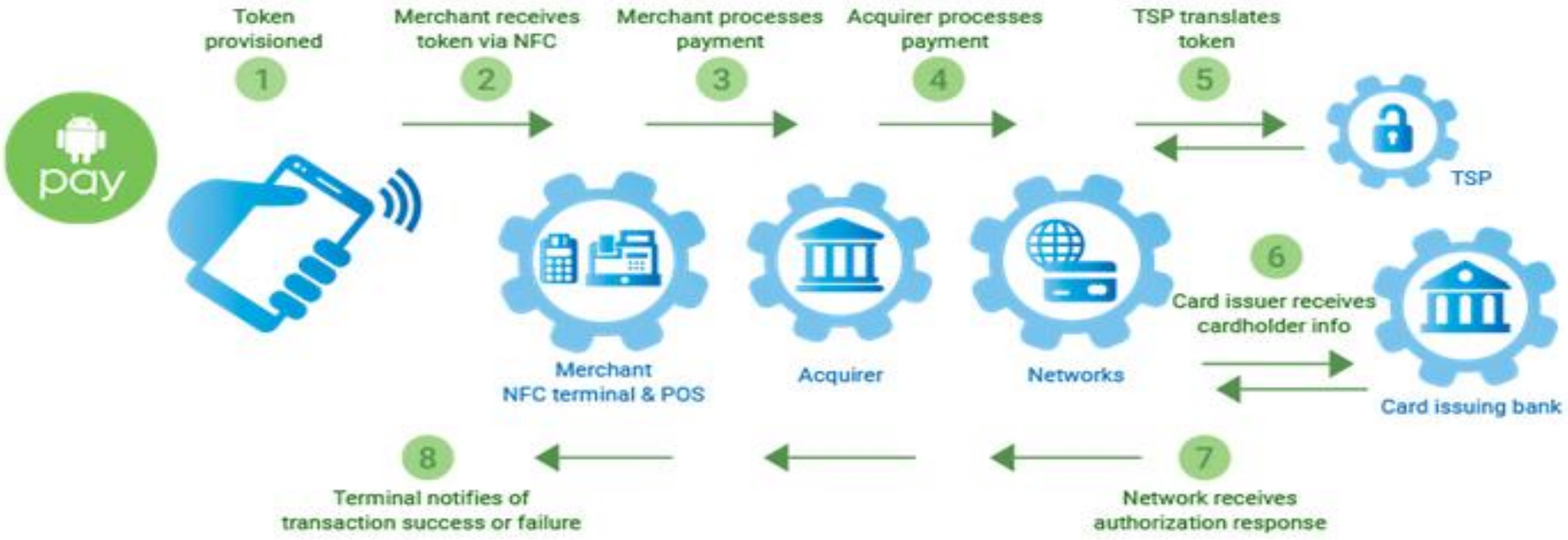




EMV, Payment Tokenisation Specification, Technical Framework, Version 1.0, March 2014

Note: The Token Service Provider may be located at the Issuer, Payment Network, or a third party

Android Pay


Token provisioned
1
Merchant receives token via NFC
2
Merchant processes payment
3
Acquirer processes payment
4
TSP translates token
5
TSP
6
Card issuer receives cardholder info
Card issuing bank
Merchant NFC terminal & POS
Acquirer
Networks
7
Network receives authorization response
8
Terminal notifies of transaction success or failure

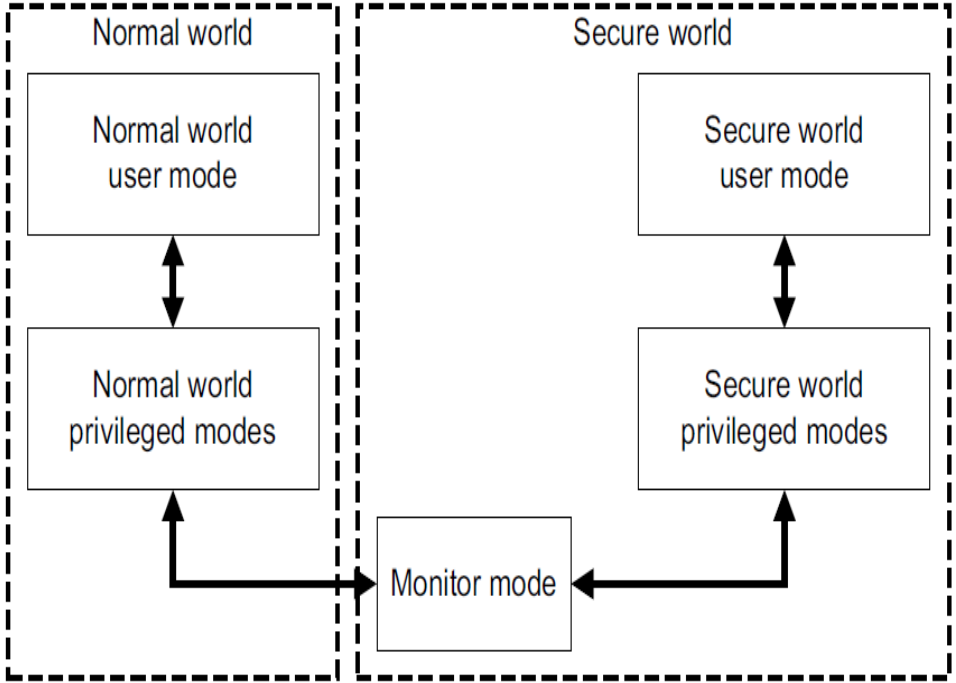
Android Pay Token

Name	Type	Description
`dpan`	`string` (digits only)	The device-specific personal account number (i.e., token)
`expirationMonth`	`number`	The expiration month of the `dpan` (1 = January, 2 = February, etc.)
`expirationYear`	`number`	The four-digit expiration year of the `dpan (e.g., 2015)`
`authMethod`	`string`	The constant "`3DS`". This may change in the future.
`3dsCryptogram`	`string`	3DSecure cryptogram
`3dsEciIndicator`	`string` (optional)	ECI indicator per 3DSecure specification

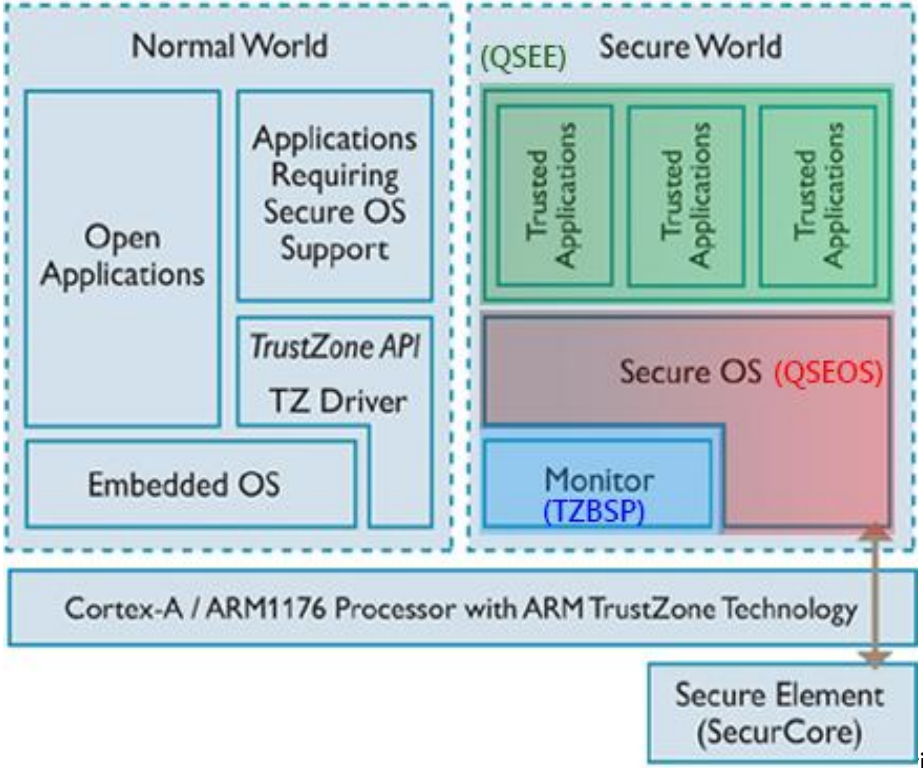
Trusted Execution Environment

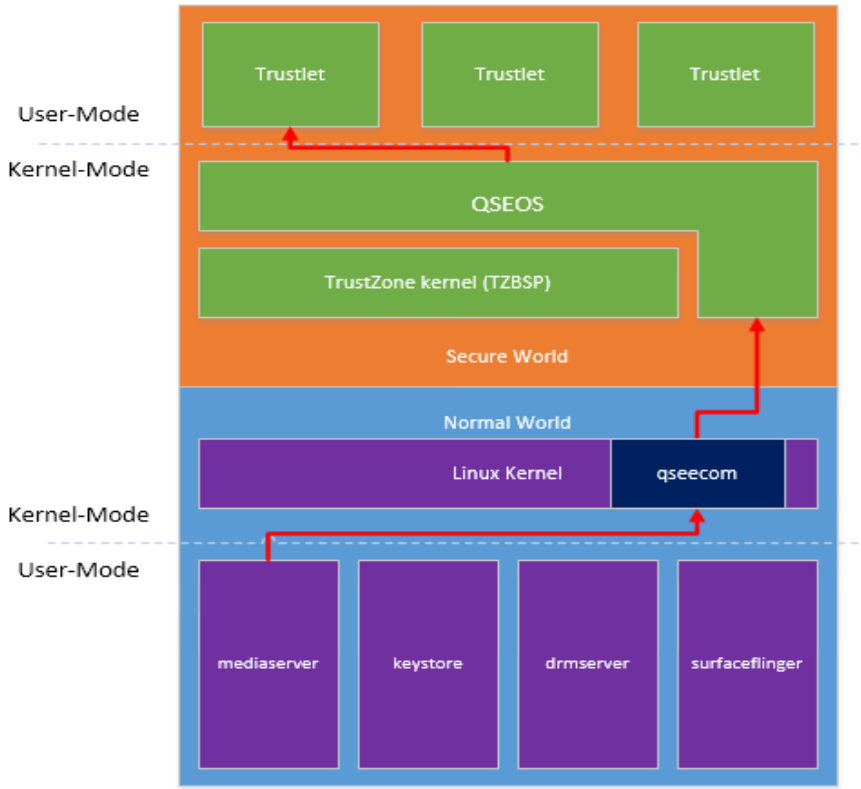
Trusted Execution Environment

- Trusted Zone is an ARM patent
- A processor with two modes:
  - Normal Mode
  - Secure Word
- A monitor entity manages the environment switch between the
- Normal and the Secure word
- Each mode has its own interrupt procedures and a set of registers.
- Never less CPU and memories (SRAM, ROM) are common
- The SRAM size is about 10KB and the ROM size is about 128 KB
- An fuse OTP memory of about 256 bits, stores the hash of a public key
- Trusted Application (Trustlet) are signed

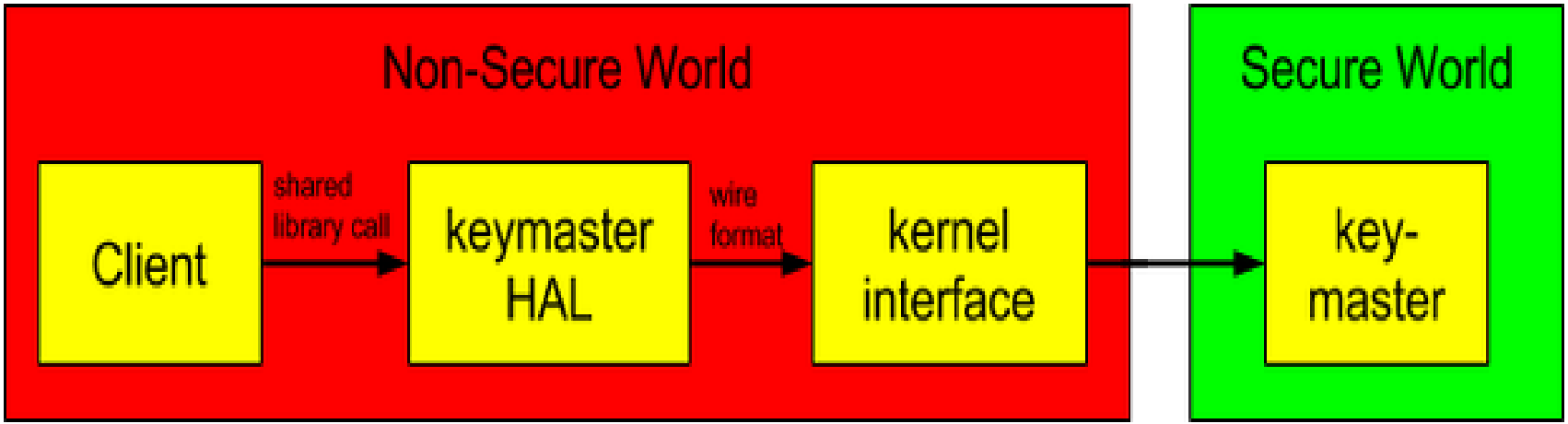


Android QSEE (*Qualcomm's Secure Execution Environment)*

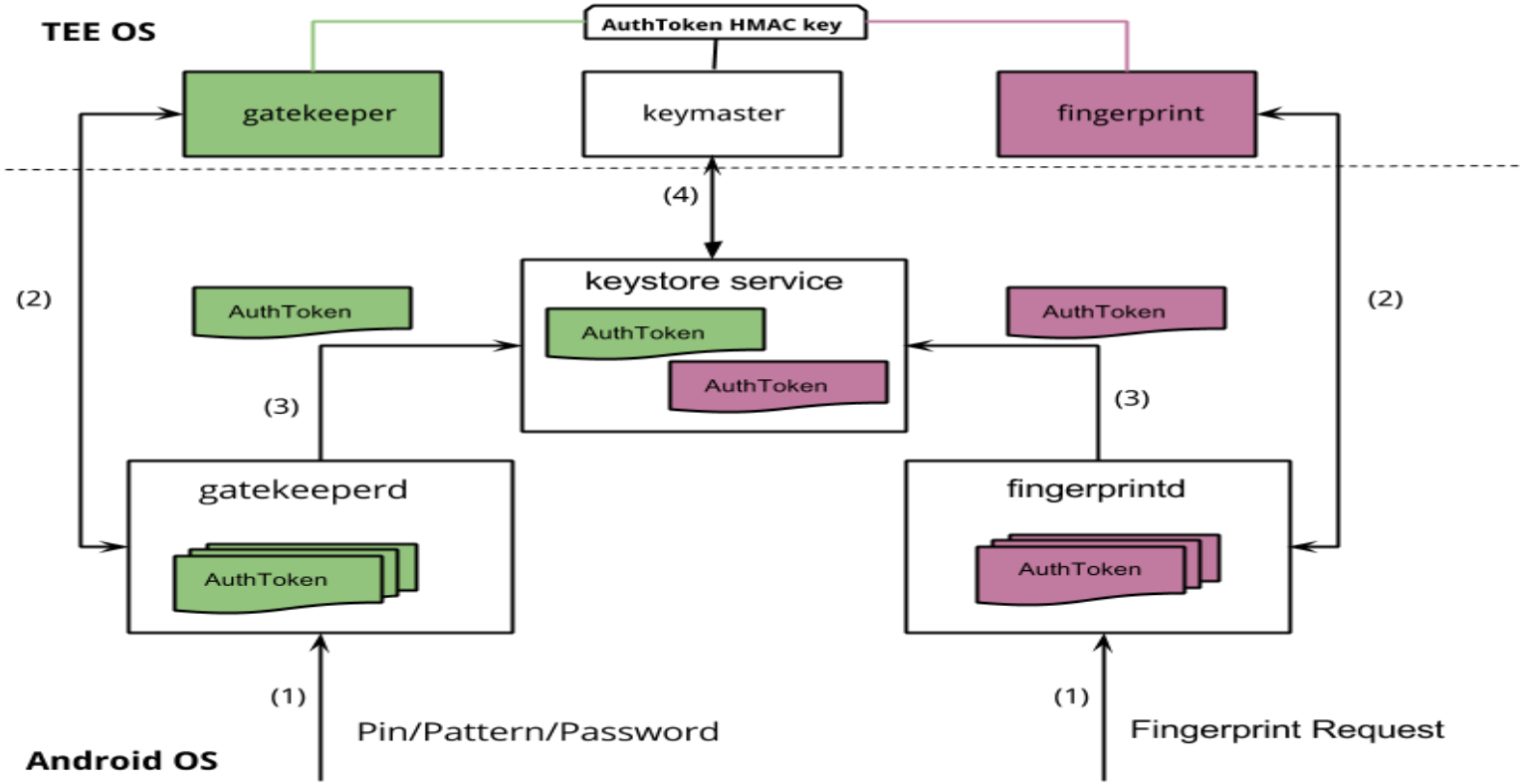




Android Hardware-Backed Keystore



TEE support in Android

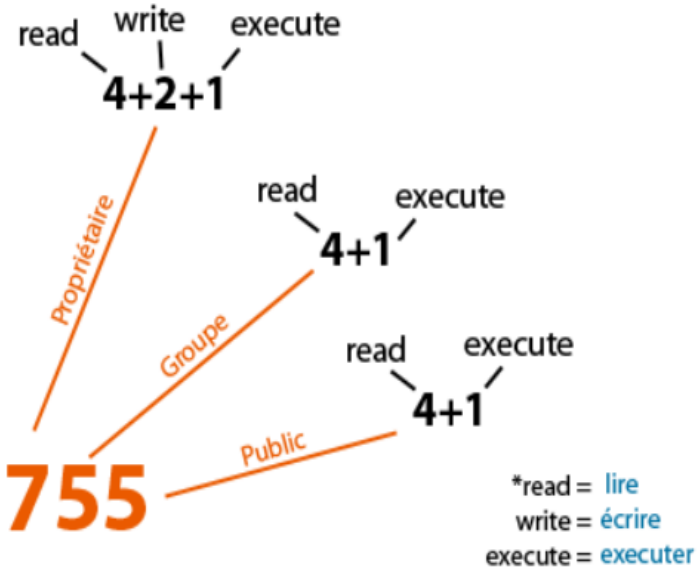


About Android Security

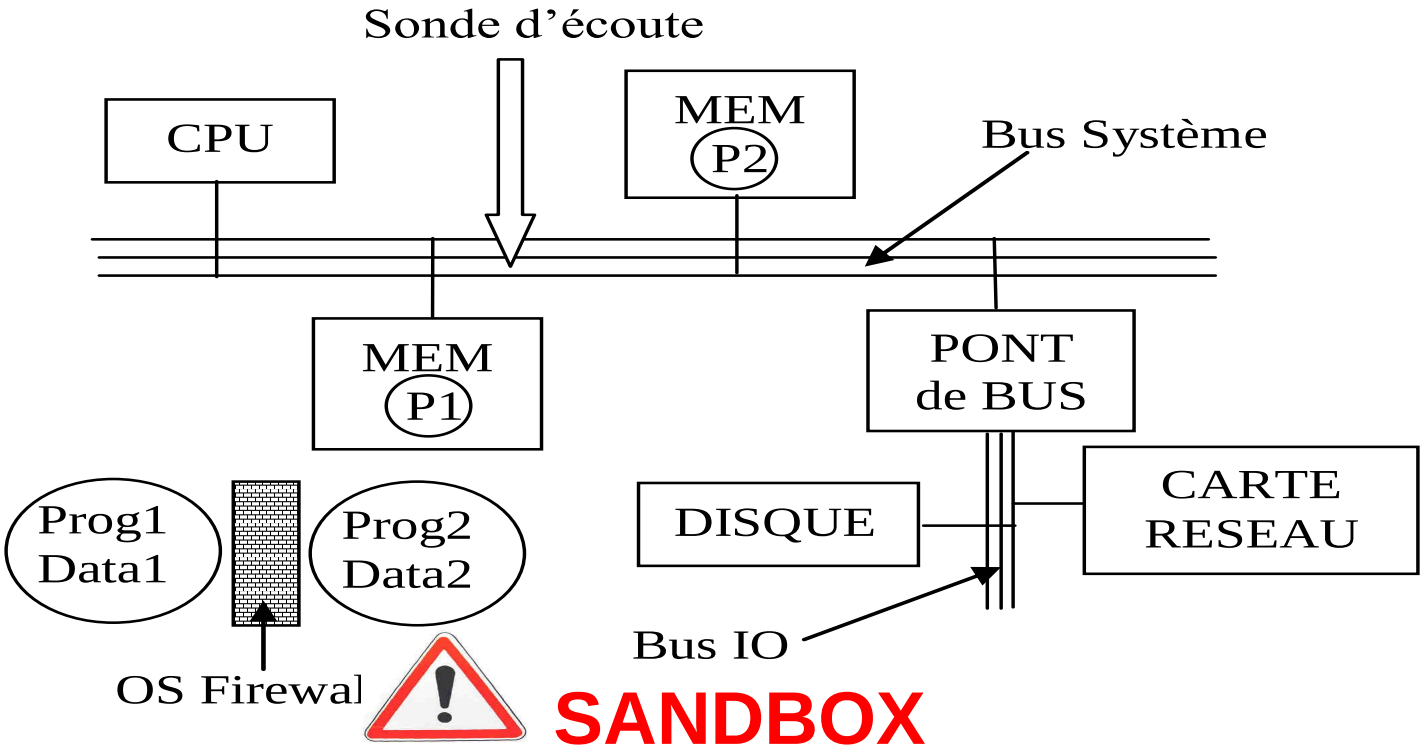
- Unix Sandboxes
  - One Dalvik Virtual machine per application
  - One UserID (UID) per application
  - Unix Discretionary Access Control (**DAC**)
- Application Signature
  - Applications are signed
- Permissions
- File Encryption

Android Application Storage

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



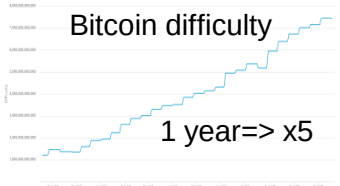
GlueWare


Sonde d'écoute
CPU
MEM
P2
Bus Système
MEM
P1
PONT
de BUS
Prog1
Data1
Prog2
Data2
DISQUE
CARTE
RESEAU
Bus IO
OS Firewal
SANDBOX

About Blockchain

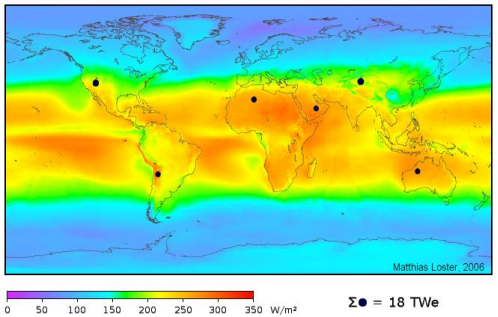
What does a blockchain ?

- Exchange of **signed transactions** in P2P networks
  - Blockchain addresses are computed from public keys
  - Signature over elliptic curve (secp256k1…)
- "Consensus" procedure for the building of transaction blocs
  - Checking of transaction coherence and **authenticity**
- Competition for remunerated certification of blocks
  - Proof of Work, *Proof of Stake
- **No third party**
- Publication/duplication of databases



A 36 MegaWatts crypto farm under construction in Maroco .

The long term objective is to reach a **900 MW** production capacity .



World Electricity:	$2{,}5\ 10^4$	TWh
World Sun Energy:	$1{,}6\ 10^9$	TWh
Bitcoin 2018	73.1	TWh

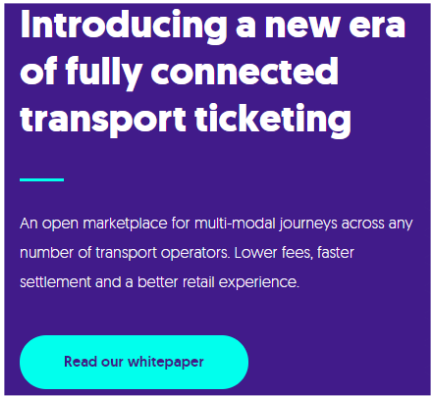
***Proof of Stake** (PoS) concept states that a person can mine or validate block transactions according to how many coins he or she holds.

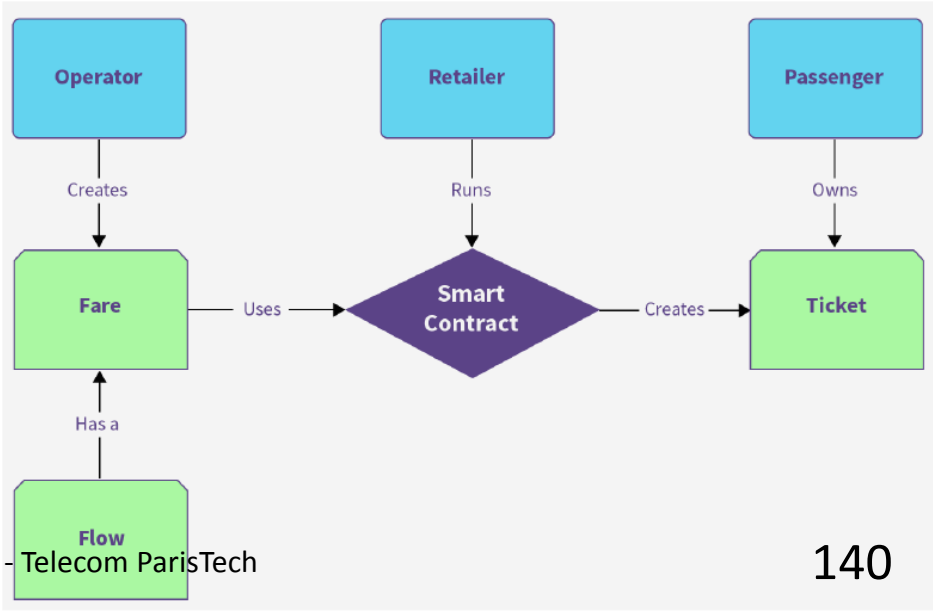
Blockchain: a young technology

- 1974: First TCP/IP paper
- 1986: First IETF meeting
- 1991: First WEB paper
- 1995: TCP/IP over Windows 95
- 1999: First Google data center
- 2008: iPhone
- 2008 : Bitcoin paper
- 2009: Bitcoin.exe
- 2012: Coinbase
- 2013: Ethereum white paper
- 2015: Ethereum blockchain

The Planar Network Blockchain Transport

https://planar.network/





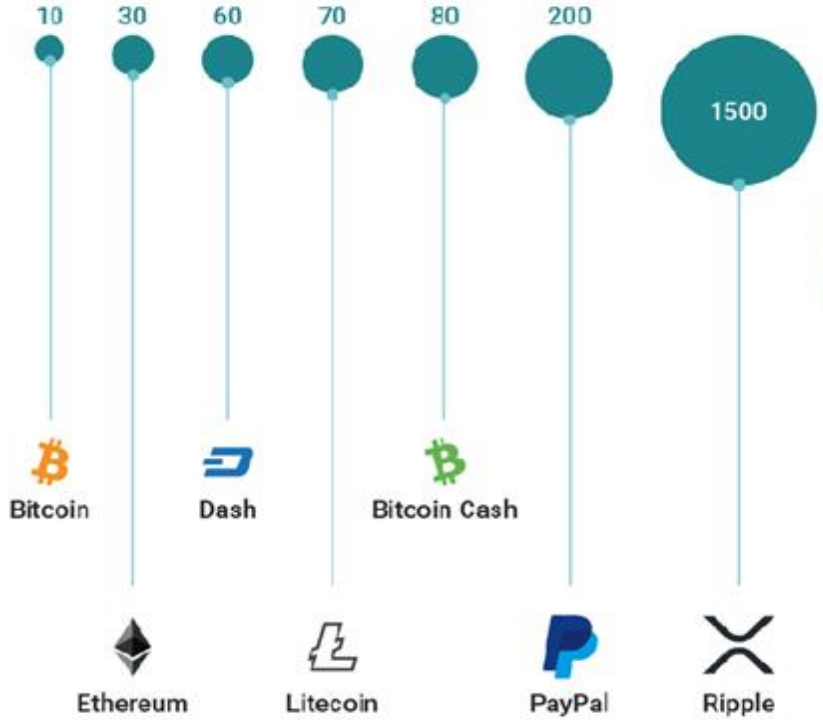

JURIS
Vision
Mission
Team
Tools
Blog
Learn More
Imagine a better
legal system.
At Juris we are working on the next evolution of legal and dispute resolution services by combining people and computers. Looking for help? Check out the tools we're building below. Want to help? Sign up, verify your credentials, and get started working toward our goal:
Make The World More Fair.
Learn More
Legal Professional? Sign Up for the Beta!
IN ASSOCIATION WITH
VE RI TAS
HARVARD
Advanced Leadership Initiative
COLUMBIA
ENTREPRENEURSHIP
INNOVATION AND DESIGN


LOUIS VUITTON
Louis Vuitton Owner LVMH Is Launching a Blockchain to Track Luxury Goods

Blockchain Services

- The transfer of crypto money from a blockchain address to another;
- The execution of programs;
- The storage and timestamp of data, charged with crypto money;
- The use of multi-signatures, i.e. in blockchain such as Bitcoin a transaction may be signed by several entities.

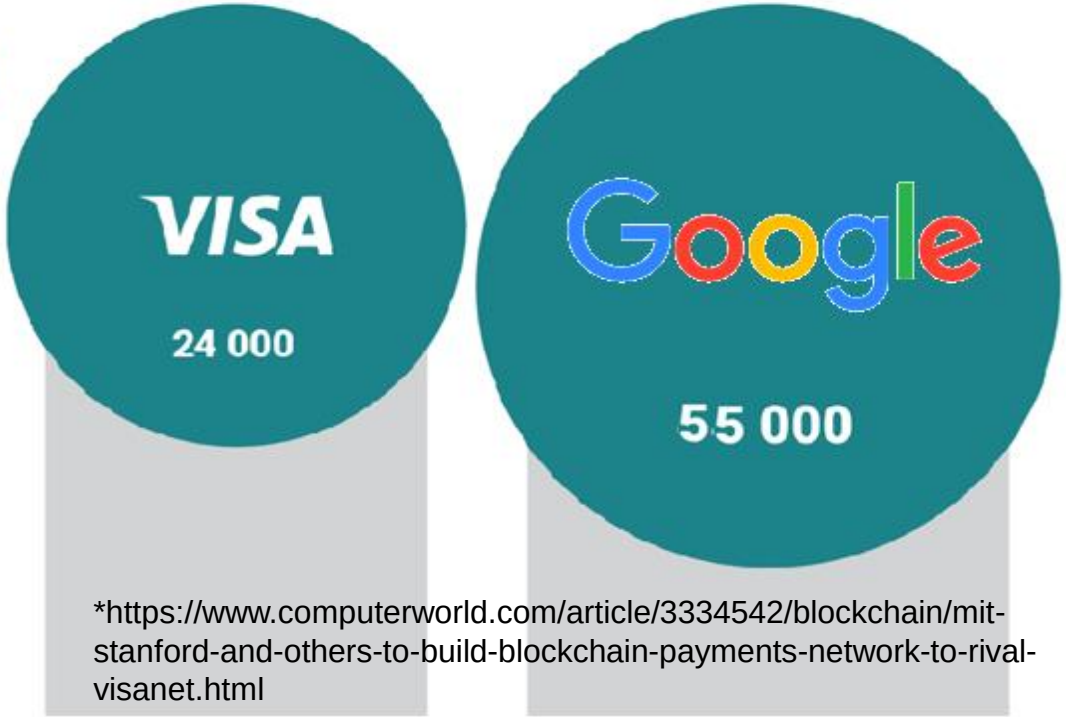
Scalability Issue: Transaction/s



NEWS ANALYSIS

MIT, Stanford and others to build blockchain payments network to rival VisaNet

Seven universities are collaborating to create a blockchain-based online payment system that will solve issues of scalability, privacy, security and performance, enabling up to 10,000 transactions per second.



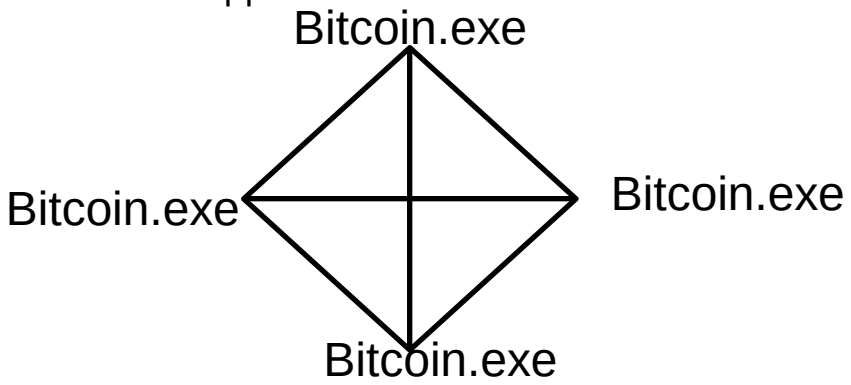
*https://www.computerworld.com/article/3334542/blockchain/mit-stanford-and-others-to-build-blockchain-payments-network-to-rival-visanet.html

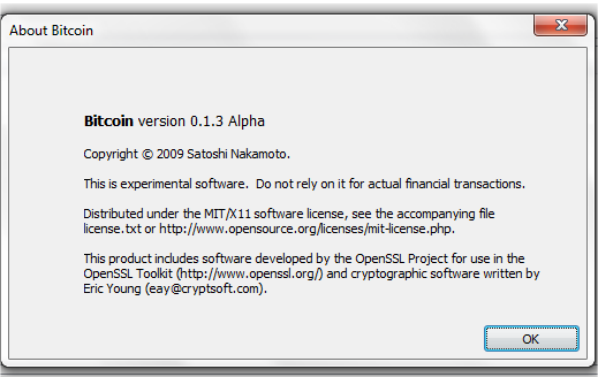
"Bitcoin: A Peer-to-Peer Electronic Cash System." Satoshi Nakamoto

- In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions
- The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation.
- In our case, it is CPU time and electricity that is expended

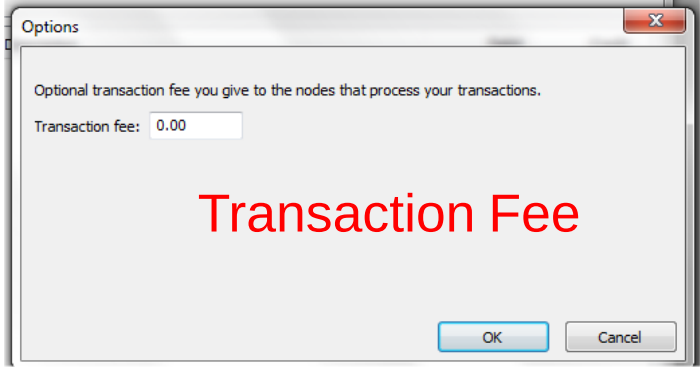
Bitcoin.exe

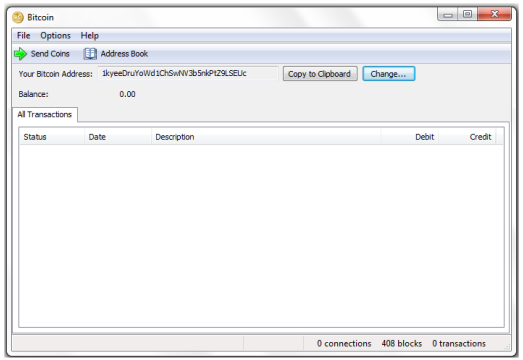
- A win32 software, written in 2009 by Satoshi Nakamoto
- About 16,000 lines of C++ code, using OPENSSL, and 6 MB binary size.
- This software realizes all the functions needed by the bitcoin blockchain. It manages four major tasks:
  - **Transaction Generation**.
    - Transactions are signed according to the ECDSA algorithm, dealing with elliptic curve private keys.
  - **Communication** with other bitcoin nodes running the bitcoin.exe application.
    - Bitcoin Protocol
    - Incoming transactions checking
    - Management of transaction pools
  - **Block Mining.**
    - Include the CoinBase in the block to be mined
    - A full node always mines a block
    - Always work on the greatest blockchain
  - **Blockchain Management**.
    - Database management
    - A set of data files managed by a non SQL database, the Berkeley DB**.** ***In particular the private keys are stored in the file named wallet.dat.***

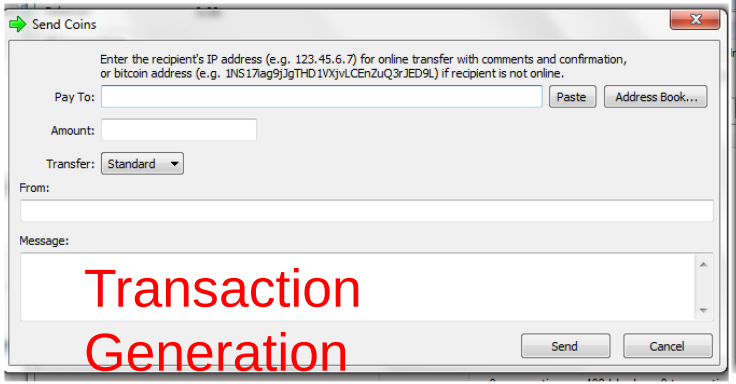



About Bitcoin
Bitcoin version 0.1.3 Alpha
Copyright © 2009 Satoshi Nakamoto.
This is experimental software. Do not rely on it for actual financial transactions.
Distributed under the MIT/X11 software license, see the accompanying file license.txt or http://www.opensource.org/licenses/mit-license.php.
This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) and cryptographic software written by Eric Young (eay@cryptsoft.com).
OK

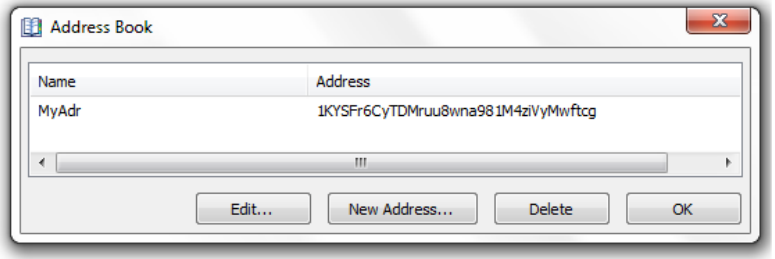

Bitcoin
File Options Help
Generate Coins
Options...
Copy to Clipboard
Change...
Balance: 0.00
All Transactions
Status Date Description Debit Credit
0 connections 408 blocks 0 transactions
Mining
Coin Generation


Options
Optional transaction fee you give to the nodes that process your transactions.
Transaction fee: 0.00
Transaction Fee
OK
Cancel


Bitcoin
File Options Help
Send Coins
Address Book
Your Bitcoin Address:
Copy to Clipboard
Change...
Balance: 0.00
All Transactions
Status Date Description Debit Credit
0 connections 408 blocks 0 transactions


Send Coins
Enter the recipient's IP address (e.g. 123.45.6.7) for online transfer with comments and confirmation, or bitcoin address (e.g. 1NS17ag9jJgTHD1VXjvLCEnZuQ3rJED9L) if recipient is not online.
Pay To:
Paste
Address Book...
Amount:
Transfer: Standard
From:
Message:
Send
Cancel
Transaction
Generation

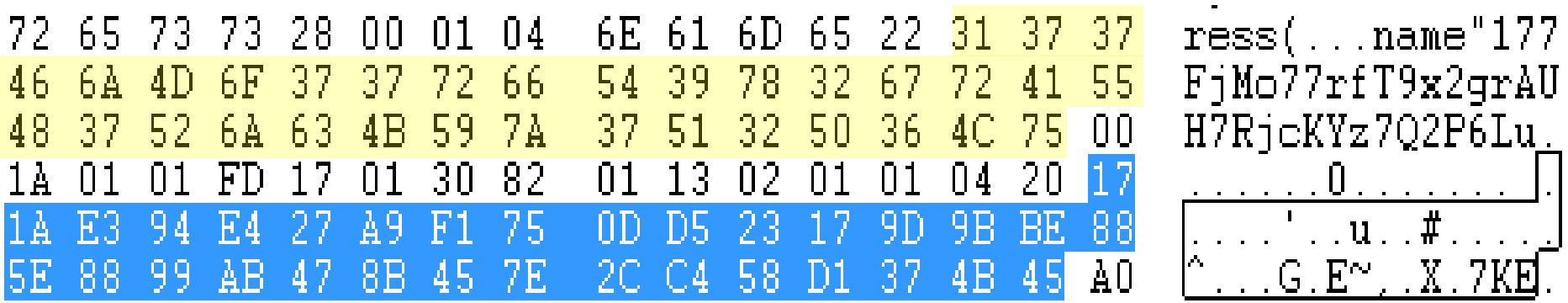

Your Bitcoin Address
These are your Bitcoin addresses for receiving payments.
You may want to give a different one to each sender so you can keep track of who is paying you.
Name
Bitcoin Address
test1 13adv1s7cpgph4NMY7ZnGpuo2juQsUpxrz
test2 1kyeeDruYoWd1ChSwNV3b5nkPt29LSEUc
test3
Your Address 177FjMo77rfT9x2grAUH7RjcKYz7Q2P6Lu
Bitcoin Address
Generation
Rename...
New Address...
Copy to Clipboard
OK
Cancel


Address Book
Name
Address
MyAdr 1KYSFr6CyTDMruu8wna981M4ziVyMwftcg
Edit...
New Address...
Delete
OK

Bitcoin.exe

- At boot
  - try to connect to Internet Relay Chat (IRC) server (example orwell.freenode.net)
  - JOIN #bitcoin group orwell.freenode.net
  - Get a list of Bitcoin.exe nodes

Secure key storage and Secure key use are critical issues



The wallet.dat, a database file from bitcoin.exe.
The PrivateKey is
171AE394E427A9F1750DD523179D9BBE885E8899AB478B457E2CC458D1374B4
and the address is
177FjMo77rfT9x2grAUH7RjcKYz7Q2P6Lu

Three Security Pilars

- ***Transactions***.
  - Transactions are signed according to an asymmetric mechanism (ECDSA) relying on elliptic curves. Blockchain entities are identified by an address, computed from a public key zG, G being a generator over the secp256k1 curve and z the 32 bytes private key
- ***Block mining***.
  - Transactions are gathered in blocks according to *hash trees* such as *Merkle tree*, *Merkle Patricia tree*, and blocks are linked according to a *blockchain* structure, whose integrity relies on a *Proof of Work* (PoW) computation.
  - Given the computation difficulty D, and the hashrate (h(t), in computations per second), the probability Δp of solving the PoW in Δt second is therefore : $\Delta p = \Delta t\ h(t)/D$,
  - The mining duration follows an exponential distribution, whose probability density function $\rho(t)$ is : $\rho(t) = \lambda e^{-\lambda t}$ with $\lambda = h(t)/D$ in s^{-1}
- ***Distributed Ledger***.
  - Blocks and transactions are stored in full node databases. A full node checks transaction validity according to its database content, before including it in the next block to be mined. According to **the *Byzantine Generals Problem*** a fully connected (P2P) network works if 2/3 of participants are honest.

SECP 256k1

$y^2 = x^3 + 7,\ x,y \in Z/pZ \quad p = 2^{256} + 2^{32} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 1$

G: GENERATOR
04 79BE667E F9DCBBAC 55A06295 CE870B07
029BFCDB 2DCE28D9 59F2815B 16F81798
483ADA77 26A3C465 5DA4FBFC 0E1108A8
FD17B448 A6855419 9C47D08F FB10D4B8

n: Curve Order
FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFE
BAAEDCE6 AF48A03B
BFD25E8C D0364141

ECDSA(r,s):

Two integer values

x private key ϵ [1,n-1]

P= xG = public key,

k ϵ [1,n-1] , kG= (x_R, y_R)

r = x_R mod n, e = H(M) 32 bytes LSB

s = k^{-1} (e + x r) mod n

VERIFY (r,s):

$u_1 = e\ s^{-1} \bmod n$

$u_2 = r\ s^{-1} \bmod n$

$(x_R, y_R) = u_1G + u_2P$

$v = x_R \bmod n$

Check v = r

RECOVER (r,s):

For R(x=r, y=y+) to R(x=r, y=y-)

$Q = r^{-1}(sR - eG)$

VERIFY(r,s) with Q as public Key

Bitcoin Address

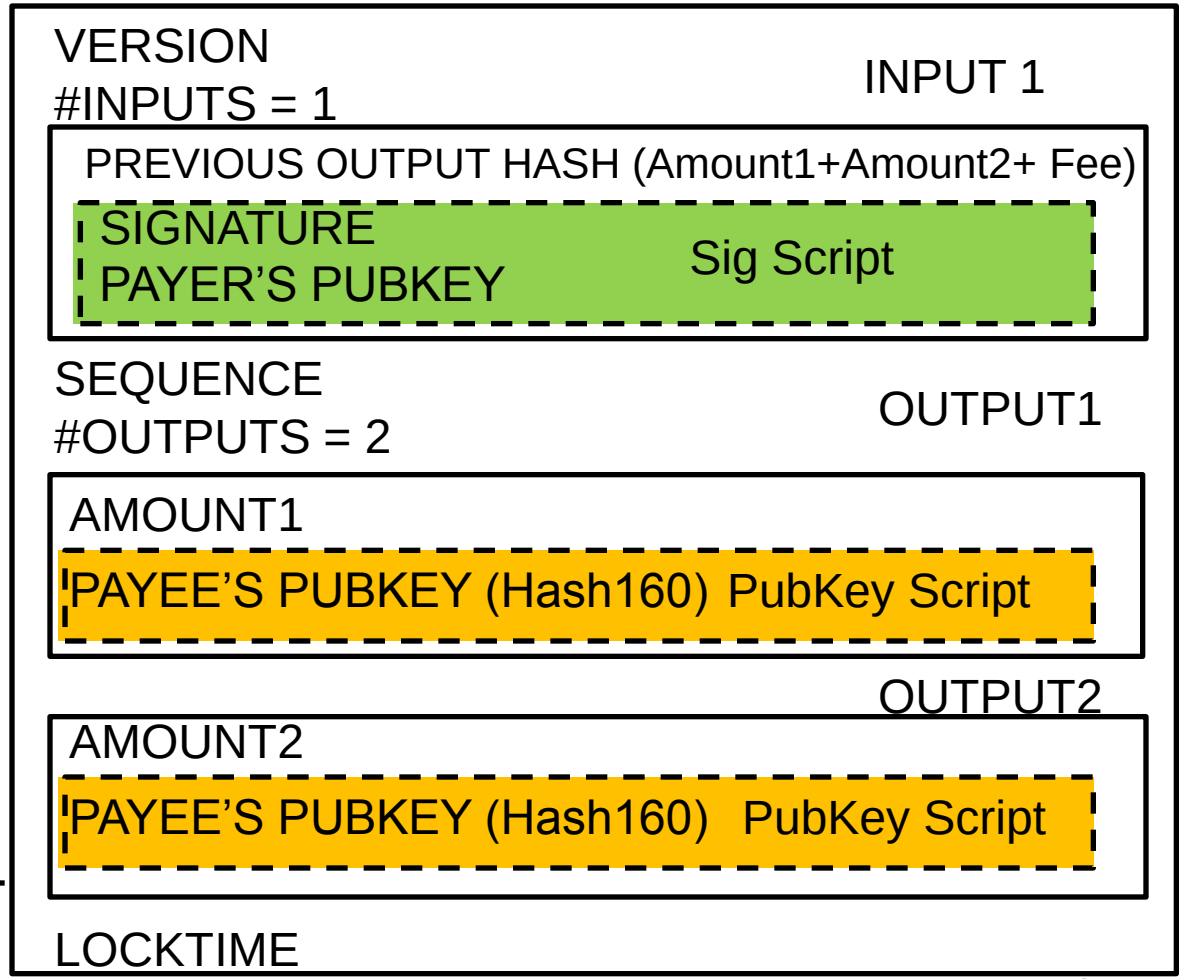
- Bitcoin addresses (BA) are computed from ECDSA public key ($Y^2 = X^3 + 7$)
- A private key, i.e. a 32 byte number x, is generated, according to a true random number generator (RNG).
- Thereafter a public key is computed according to the relation P=xG.
- **The uncompressed form uF(P) is a set of 65 bytes {4,xP,yP},** a prefix (one byte 0x04) and a point (xP,yP) of the curve (2x32 bytes, in Z/pZ).
- The bitcoin address is computed according to the following procedure:
  - 1) a1 = SHA256(uF(P)), 32 bytes
  - 3) **hash160= a2 = RIPEMD160(a1), 20 bytes**
  - 3) a3 = Network-ID || a2, 25 bytes
  - 4) a4 = SHA256(SHA256(a3)), 32 bytes
  - 5) a5= checksum = 4 rightmost bytes of a a4
  - 6) a6 = a4 || a5, 25 bytes
  - 7) bitcoin address = a7 = encoding of a6 in base 58
- The base 58 encoding uses the following digits
  - {1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H,J,K,L,M,N,P,Q,R,S,T,U,V,W,X,Y,Z,a,b,c,d,e,f,g,h,i,j,k,m,n,o,p,q,r,s,t,u,v,w,x,y,z}.

Transaction

Σ input =
Σ output + Fee

Fee= Σ input
- Σ output

txHash_____
(double SHA256)



Bitcoin Transaction

```
01000000 // Version
01 // number of inputs
 DE2D211EF429909B0AB8D2E7D25826A0 //TransactionID
 EDD6281EC6DEDF2B822CE5014A349E72
 01000000 // index
 8A // length of the signature Script
  47 // ECDSA Signature length
   30 44 // Sequence of (r,s) integer values
    02 20 // integer r value
     0772ABD5D37D0CAAB881DBC8912628F9
     3461839CC8D4BC007A355831A6061ED7
    02 20 // integer s value
     4CCCC34B34A9075FC09C9777EAB7A6F5
     612DA2130C1FF1C0E376AD9B2209D51D
  01 41 // Public key length
   04 // uncompressed format
   CFD7A542B8C823992AF51DA828E1B693
   CC5AB64F0CACF0F80C31A1ECA471786E
   285BDD3F1FE0A006BD70567885EF57EB
   149C8880CB9D5AF304182AC942E176CC
FFFFFFFF // sequence
```

```
01 // number of outputs
 D4180400000000000 // amount in BTC
 19 // Public Key Script
  76 // OP_DUP
  A9 // OP_HASH160
  14 // hash160 length
   CB643DD608FB5C323A4A6342C1A6AC8048B409EB
  88 // OP_EQUALVERIFY
  AC // OP_CHECKSIG
00000000 // Locktime
```

The *pay-to-pubkey-hash* script is defined as:
OP_DUP [76] OP_HASH160 [A9]
<length=14> <hash160>
OP_EQUALVERIFY[88] OP_CHECKSIG[AC]

Raw Transaction

```
01000000 // Version
01 // number of inputs
 DE2D211EF429909B0AB8D2E7D25826A0
 EDD6281EC6DEDF2B822CE5014A349E72
 01000000 // index
 00 // vi= length of the Signature Script
FFFFFFFF // sequence
01 // number of outputs
 D4180400000000000 // amount in BTC
 19 // Public Key Script (Pk script)
  76 // OP_DUP
  A9 // OP_HASH160
  14 // hash160 length
   CB643DD608FB5C323A4A6342C1A6AC8048B409EB
  88 // OP_EQUALVERIFY
  AC // OP_CHECKSIG
00000000 // Locktime
01000000 // hash Type
```

1) Build a raw transaction, in which, for every input, the SigScript is removed, i.e. the length value (vi) is set to zero.

Signing a raw transcation

2) For every input:
2.1) Copy the *pay-to-pubkey-hash Script* in the *Signature Script* location, and modify the length (initially set to 0) accordingly (length =25 in decimal)
2.2) The hash160 inserted in *pay-to-pubkey-hash* is computed from the payer's public key.
2.3) Compute the double SHA256 of the modified transaction
2.4) Generate the ECDSA signature with the payer's private key
2.5) Insert the final *Signature Script* in the input, and modify the length accordingly

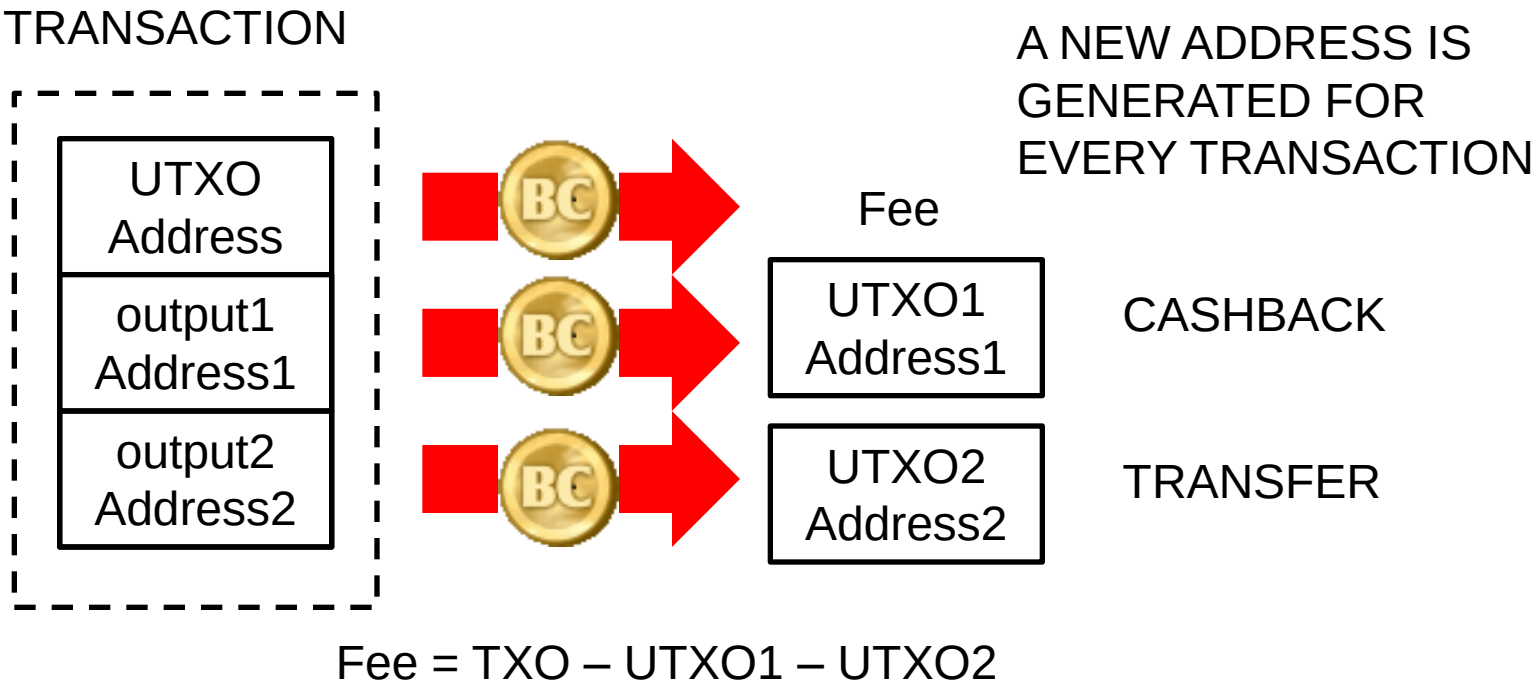
The ECDSA signature is encoded using the following ASN.1 structure

```
Ecdsa-Sig-Value  ::=  SEQUENCE  {  r     INTEGER,
                                   s     INTEGER
                                }
```

Canonical Signature

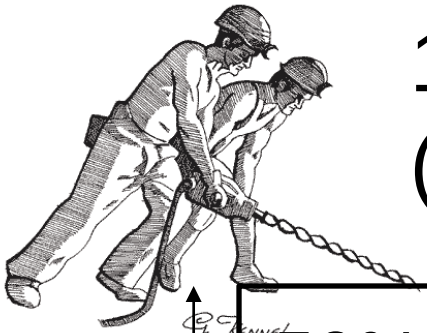
- blockchain.io error
  - " Non-canonical signature: High S Value"
- The signature is composed of two values, the r value and the s value. In your third input, the s value is greater than N/2, which is not allowed. Just add in some code that if s is greater than N/2, then s = N - s.
- bitcoinj 0.12 API
  - public static boolean isEncodingCanonical(byte[] signature)

BITCOIN CASHBACK MODEL



Fee = TXO – UTXO1 – UTXO2

Mining Bitcoin



144 blocks/day
(Fix Mining Rate)

$10{,}5 \times 10^6$ BTC

4 years	50%	50 BTC/block (144x50)
4 years	25%	25 BTC/block (144x25)
4 years	12,5%	12,5 BTC/block (144x12,5)

Difficulty (i.e. costs) increases

The number of Bitcoin is finite

210,000 blocks

Initial Block Reward (IBR)

$$N_S = N_B \times \sum_{i=0}^{i=32} 50 \times 10^8 / 2^i$$

(in satoshi)

1 BTC = 10^8 satoshi

The block reward started at 50 BTC in 2009
It halves every 210,000 blocks (about 4 years, = 144x 1461)
It will stop with the block number 6,930,000 (=33x 210,000, $33 = 1 + \log_2(5.10^9)$)
This mechanism limits the total number of Bitcoins in circulation
to 21 millions (210,000 x 50 x 2)

About the 25 $ pizza for 10,000 BTC

- May 22nd 2010
- Hashrate = 0,1 GH/s
- Daily Cost = 82,9 x HashRate = 8,29 $
- 50 x 144 = 7200 BTC/day
- Cost for 10,000 BTC = 11,52 $

- August 17th 2010, 1 BTC = 0,077 $
- February 11th 2011, 1BTC= 1,0$

The Difficulty of the PoW

- A nonce value that make a double SHA-256 hash of the block's header that is less
  - (65535 << 208) / difficulty
- So the entropy of this calculation is closed to 32+log2(difficulty), about 75 bits in March 2019
- The difficulty is scaled every 2016 blocks in order to maintain a block production every 10 minutes, i.e. about 144 (6x24) per day.

The PoW Magic

Hash rate= Computation number/s

$$\lambda = h(t)/D$$

Difficulty= computation number

In average a block is mined every 1/λ second (600s)

Density of probability

$$\rho(t) = \lambda e^{-\lambda t}$$

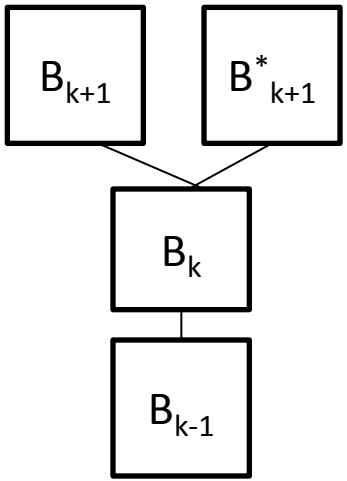
Probability of mining a block in t second

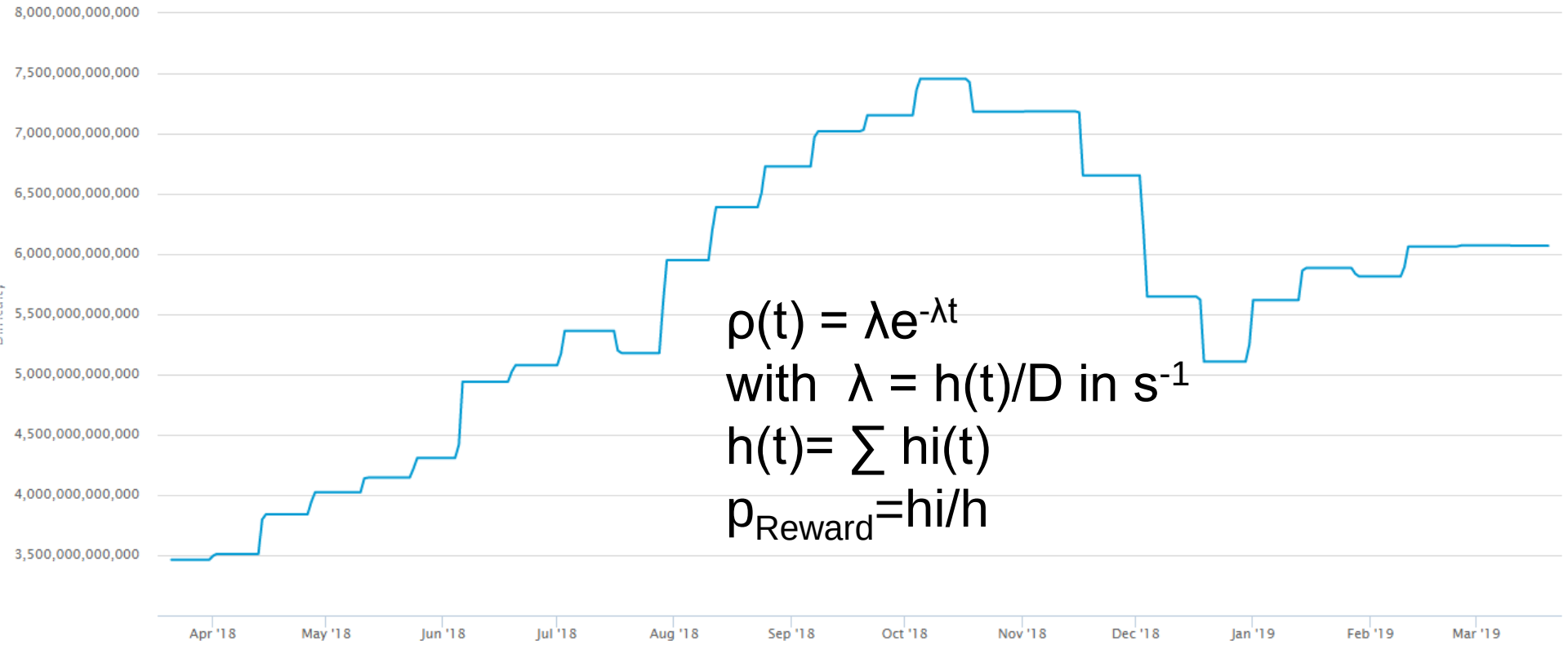
$$\mathrm{p(t)} = 1 - e^{-\lambda t}$$

$$\sum_{i=1}^{n} \lambda_i = \lambda = \sum_{i=1}^{n} h_i(t)/D = h(t)/D$$

hi ~ number of rigs
cost ~ number of rigs
Success probability = hi/h
Income ~ hi/h

Blockchain Fork




8,000,000,000,000
7,500,000,000,000
7,000,000,000,000
6,500,000,000,000
6,000,000,000,000
5,500,000,000,000
5,000,000,000,000
4,500,000,000,000
4,000,000,000,000
3,500,000,000,000
Apr '18
May '18
Jun '18
Jul '18
Aug '18
Sep '18
Oct '18
Nov '18
Dec '18
Jan '19
Feb '19
Mar '19
$\rho(t) = \lambda e^{-\lambda t}$
with $\lambda = h(t)/D$ in s^{-1}
$h(t) = \sum hi(t)$
$p_{Reward} = hi/h$

The HashRate Cost (estimation)

Energy
(per day)

Rig Cost/day

Year	C_1=W / GH/s	C_2=$/ GH/s /day
2009-2010	4000	68,5
2011	500	2,73
2012	100	2,05
2013	10	0,055
2014	1	$2,3\ 10^{-3}$
2015-2017	0,1	$1,6\ 10^{-4}$

The Costs (Estimation)

Year	2009 2010	2011	2012	2013	2014	2015 2017
C_1 x E $/Gh/s	14,4	1,8	$3,6\ 10^{-1}$	$3,6\ 10^{-2}$	$3,6\ 10^{-3}$	$3,6\ 10^{-4}$
C_2 $/Gh/s	68,5	2,73	2,05	$5,5\ 10^{-2}$	$2,3\ 10^{-3}$	$1,6\ 10^{-4}$

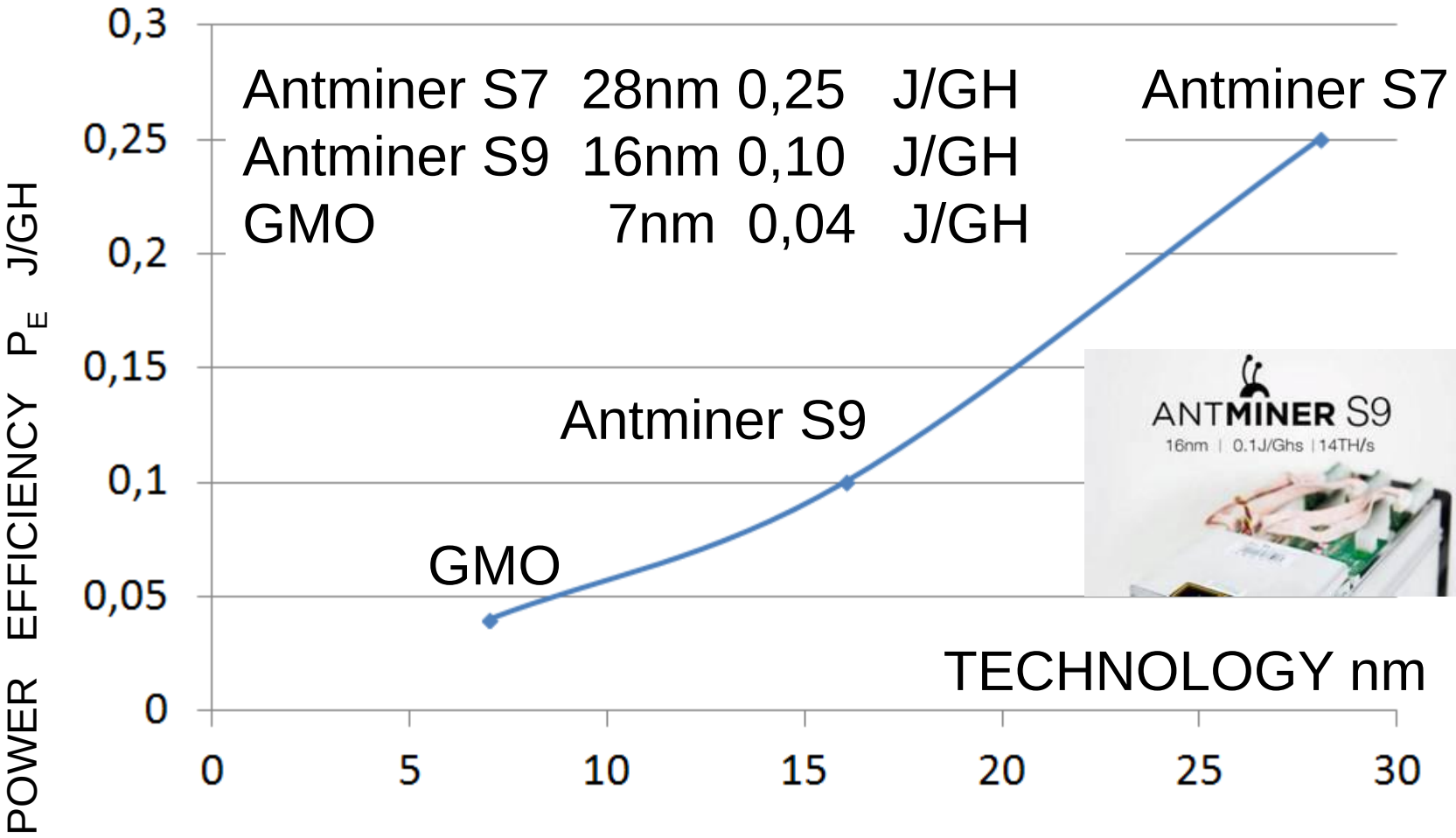
$E = 0,0036 = 0,15\ 10^{-3} \times 24$ $\quad$ $0,15\ \$ / KWh = 0,15\ 10^{-3}\ \$ / Wh$

Total Costs/day = $C_1 \times E \times HashRate + C_2 \times HashRate$

About the 25 $ pizza for 10,000 BTC

- May 22nd 2010, Hashrate = 0,1 GH/z
- Daily Cost = 82,9*x HashRate = 8,29 $
- 50 x 144 = 7200 BTC/day
- Cost for 10,000 BTC = 11,52 $
- August 17th 2010, 1 BTC = 0,077 $
- February 11th 2011, 1BTC= 1,0$

*82,9= 14,4+ 68,5


Antminer S7 28nm 0,25 J/GH
Antminer S9 16nm 0,10 J/GH
GMO 7nm 0,04 J/GH
Antminer S7
Antminer S9
GMO
ANTMINER S9
16nm | 0.1J/Ghs | 14TH/s
TECHNOLOGY nm
POWER EFFICIENCY P_E J/GH
0,3
0,25
0,2
0,15
0,1
0,05
0
0
5
10
15
20
25
30

J/GH/s CJ	HashRate TH/s HR	TWh /year PW	Cents /KWh EP	Electricity Cost billion $ / year EC	BTC /year BP	Fee BTC/year 10% FP	$Market Price $/BTC MP	Miners' Revenus billion $ MR
0,1	50,000,000	**43,8**	15	**6,57**	657000	65700	4000	**2,89**

France electricity production 563 TWh / year
1 nuclear reactor production # 6 TWh / year

PW = CJ x HR x 24 x 365 x 0,001 = 8,76 10^{-6} CJ * HR (TWh)
EC = PW x EP / 100 billion$/year
BP = 12,5 x 144 x 365 = 657000 BTC/year
FP = BP x 0,1 = 65700 BTC/year
MR = (BP+FP)* MP / 10^9 billion$/year

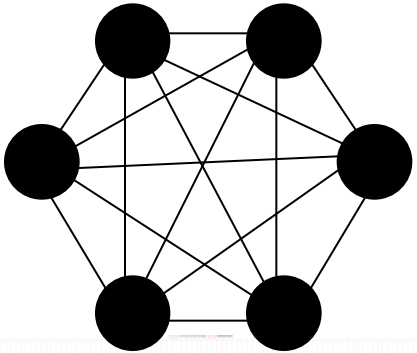
ANTMINER S9 HashRate = 14,000 GH/s
HR=50 10^6 TH/s 3,571,429 xS9

Miners
- Buy rigs
- Pay for Energy
- Produce Hashes
- Get BTC

RA, rig amortization $/year β, rig hashrate TH/s

$$\mu = (R_A/\beta + 8.760\, P_E\, E_C)/31.536.000 \; in \; \$/TH$$

PE, rig power efficiency J/GH EC, Energy Cost $/KWh



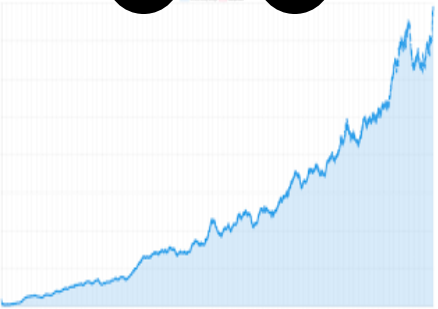
Network
- Fixes the Hash price in BTC
- Fixes the BTC production rate
- Provides the blockchain
- Stores blocks and transactions

HR hashrate TH/s

BR, Block Reward in currency

$$c(t) \approx \mu\, H_{R(t)}/B_R \;\; in \; \$/BTC$$

c(t), production cost $/currency



Market
- Fixes the BTC change in $

$$p(t) = c(t)(1+m) \;\; , in \; \$/BTC$$

m(t), market valorization coefficient

Currency	market price $/currency	hashrate TH/s	TR, block mining time, in s	BR, Block Reward in currency
Bitcoin	4000	5,00E+07	600	12,5
Ether	140	144	15	2
Litecoin	60	250	150	12,5

Currency	Bitcoin	Ether	Litecoin
market price $/currency	4 000	140	60
hashrate TH/s	50 000 000	144	250
TR, block mining time, in s	600	15	150
BR, Block Reward in currency	12,50	2,00	12,50
EC, Energy Cost $/KWh	0,15	0,15	0,15
Best Rig	Antminer S9	Radeon RX 480	Antminer L3+
rig price $	1 800	540	4 500
β, rig hashrate TH/s	1,4E+01	3,0E-05	5,0E-04
Rig Power Consumption, Watt	1 400	165	800
PE, rig power efficiency, J/GH	0,10	5 500,00	1 600,00
RA, rig amortization $/year	600	180	1 500
μ, income per TH, $/TH	5,5E-06	4,2E-01	1,6E-01
EnergyCost/RA, ratio EnergyCost/RigAmortization	3,1	1,2	0,7
number of rigs	3 571 429	4 800 000	500 000
c(t), production cost, $/currency	13 262	453	485
m(t), market valorization coefficient	-0,7	-0,7	-0,9
Energy TWh/year	43,8	6,9	3,5

Bitcoin.exe

- First block (Genesis Block) is store in the software
- It comprise only a *CoinBase* transaction
- A CoinBase transaction is the first transaction of a block that tranfers the block rewards to a given address.


THE TIMES

Saturday January 3 2009 timesonline.co.uk No 69525 £1.50

Eat Out from £5

More than 900 great restaurants, including four Gordon Ramsay favourites from £15

Start collecting tokens today Pullout inside

Israel prepares to send tanks and troops into Gaza

Israel allowed foreigners to flee the Gaza Strip as it prepared for a ground offensive. At least 430 Palestinians were killed in a week of airstrikes News, page 7

Michael Sheen Frost, Nixon and me

Working mums So that's how she does it

Detox in style The best spas on the planet

Salman Rushdie I won't marry again

Giant killing? Guide to the FA Cup third round

Chancellor on brink of second bailout for banks

Billions may be needed as lending squeeze tightens

99p

Pub chain cuts the price of a pint from £1.69 to 1989 levels

Business, page 67

Continued on page 6, col 1

Leading article, page 2

CoinBase

04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73

(décodé) ┘��↔┌EThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks

Block #0

Scripts de sortie

PUSHDATA(65)[04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f]
CHECKSIG

Blockchain: a public ledger

Bitcoin 12LZjvQBy31ABRpqvMZQbu7S9K5SxaifjW in Block 96188

Full Bitcoin Block 96188

Share:

block, address, transaction Search

PREVIOUS BLOCK

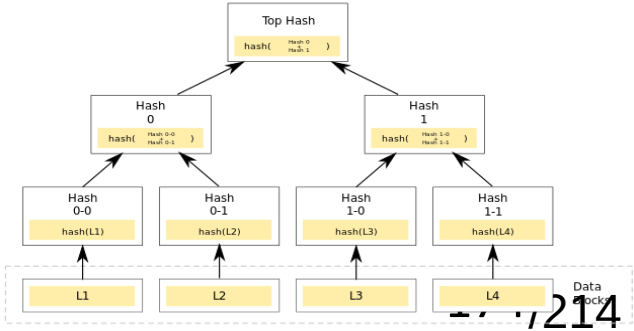
NEXT BLOCK

Number Of Transactions	4
Output Total	91.24 BTC
Height	96188
Time	2010-12-07 13:57:40
Difficulty	8,078.19525793
Bits	453516498
Version	1
Nonce	944968343
Block Reward	50 BTC
Days Destroyed	2

PoW

Hash	000000000004d6e22b42bf66fd9cad1977bdee13abab1e51a2d03ca22e6f71af
Previous Block	0000000000027c094bf08f7c27a6debcb5f36419bf53390e3e1c40a653e2195c
Next Block(s)	0000000000042c9d08f3f06d502a247f090625fb6c7623cf956a38e987c59e0f
Merkle Root	26ab6b697b8e06416b2fe5047f558c997af0a9c36e6a6eae79ff59745b5065a1

Transaction identifiers are stored in a Merkle Tree



tx:d4a73f51ab7ee7acb4cf0505d1fab34661666c461488e58ec30281e2becd93e2 33.59 BTC Fee: 0 BTC

prev tx	1689LPUuixaxSchENLMNaNbS3hYVgdpaSS -33.59 BTC	13RoCeq4K8ddPW6ugcheFoXK4GC2BLVuET 0.05 BTC	next tx
		12LZjvQBy31ABRpqvMZQbu7S9K5SxaifjW 33.54 BTC	next tx

https://bitinfocharts.com/bitcoin/search.html

Pascal Urien

Double Spent Attack

Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. **An attacker can only try to change one of his own transactions to take back money he recently spent**.

p = probability an honest node finds the next block , p= 1-q
q = probability the attacker finds the next block
Pz = probability the attacker will ever catch up from z blocks behind

$$Pz = 1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Solving for Pz
less than 0.1%...
q=0.10 z=5
q=0.15 z=8
q=0.20 z=11
q=0.25 z=15
q=0.30 z=24
q=0.35 z=41
q=0.40 z=89
q=0.45 z=340

About the Byzantine Generals Problems

The Byzantine Generals Problem
LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE, 1982

Byzantine Generals Problem. A commanding general must send an order to his n - 1 lieutenant generals such that
IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

A SOLUTION WITH ORAL MESSAGES

THEOREM 1. *For any m, Algorithm OM (m) satisfies conditions IC1 and IC2 if there are more than 3m generals and at most m traitors.*

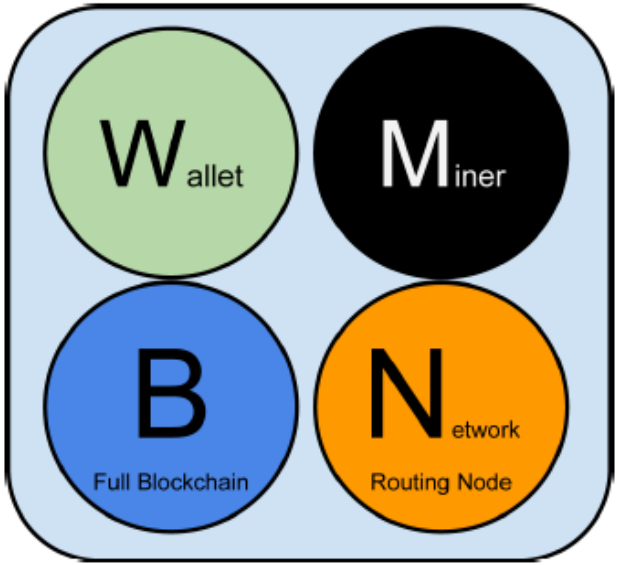
A SOLUTION WITH SIGNED MESSAGES

THEOREM 2. *For any m, Algorithm SM(m) solves the Byzantine Generals Problem if there are at most m traitors.*

https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

- Proof of Stake (PoS) is a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network
- CAP theorem
  - *in the cases that a network partition takes place, you have to choose (in case of failure) either consistency (error or data) availability(data)*
- FLP impossibility
  - *in an asynchronous setting (i.e. there are no guaranteed bounds on network latency even between correctly functioning nodes), it is not possible to create an algorithm which is guaranteed to reach consensus in any specific finite amount of time if even a single faulty/dishonest node is present*
- DLS paper
  - (i) protocols running in a partially synchronous network model (i.e. there is a bound on network latency but we do not know ahead of time what it is) can tolerate up to 1/3 arbitrary (i.e. "Byzantine") faults,
  - (ii) deterministic protocols in an asynchronous model (i.e. no bounds on network latency) cannot tolerate faults (although their paper fails to mention that randomized algorithms can with up to 1/3 fault tolerance),
  - (iii) protocols in a synchronous model (i.e. network latency is guaranteed to be less than a known d) can, surprisingly, tolerate up to 100% fault tolerance, although there are restrictions on what can happen when more than or equal to 1/2 of nodes are faulty.
  - Note that the "authenticated Byzantine" model is the one worth considering, not the "Byzantine" one; the "authenticated" part essentially means that we can use public key cryptography in our algorithms, which is in modern times very well-researched and very cheap.

Network



A bitcoin node is a collection of functions:
- Routing ,
- Blockchain database,
- Mining,
- Wallet Services.

https://en.bitcoin.it/wiki/Protocol_documentation

Common structures

Almost all integers are encoded in little endian. Only IP or port number are encoded big endian.

Message structure

Field Size	Description	Data type	Comments
4	magic	uint32_t	Magic value indicating message origin network, and used to seek to next message when stream state is unknown
12	command	char[12]	ASCII string identifying the packet content, NULL padded (non-NULL padding results in packet rejected)
4	length	uint32_t	Length of payload in number of bytes
4	checksum	uint32_t	First 4 bytes of sha256(sha256(payload))
?	payload	uchar[]	The actual data

- Message types
  - version, vercak, addr, inv, getdata, notfound, getblock, getheaders, block, headers, getaddr, mempool, checkorder, submitorder, reply, ping, pong, reject, filterload, filteradd, filterclear, merkleblock, alert, sendheaders, sendcmpct, cmpctblock, getblocktxn

Connecting to the bitcoin blockchain

Source	Destination	Protocol	Length	Info
137.194.23.117	5.178.68.215	Bitcoin	163	version
5.178.68.215	137.194.23.117	Bitcoin	181	version
5.178.68.215	137.194.23.117	Bitcoin	78	verack
5.178.68.215	137.194.23.117	Bitcoin	86	ping
5.178.68.215	137.194.23.117	Bitcoin	109	addr
137.194.23.117	5.178.68.215	Bitcoin	78	verack
137.194.23.117	5.178.68.215	Bitcoin	325	tx
137.194.23.117	5.178.68.215	Bitcoin	86	[unknown command]
5.178.68.215	137.194.23.117	Bitcoin	259	inv
5.178.68.215	137.194.23.117	Bitcoin	1159	inv
5.178.68.215	137.194.23.117	Bitcoin	259	inv
5.178.68.215	137.194.23.117	Bitcoin	439	inv
5.178.68.215	137.194.23.117	Bitcoin	223	inv
5.178.68.215	137.194.23.117	Bitcoin	1123	inv
5.178.68.215	137.194.23.117	Bitcoin	403	inv
5.178.68.215	137.194.23.117	Bitcoin	223	inv
5.178.68.215	137.194.23.117	Bitcoin	907	inv
5.178.68.215	137.194.23.117	Bitcoin	871	inv
5.178.68.215	137.194.23.117	Bitcoin	583	inv
5.178.68.215	137.194.23.117	Bitcoin	115	inv
5.178.68.215	137.194.23.117	Bitcoin	115	inv
5.178.68.215	137.194.23.117	Bitcoin	1339	inv
5.178.68.215	137.194.23.117	Bitcoin	1015	inv
5.178.68.215	137.194.23.117	Bitcoin	763	inv

version

```
Bitcoin protocol
  Packet magic: 0xf9beb4d9
  Command name: version
  Payload Length: 85
  Payload checksum: 0x7eb7e05f
  Version message
    Protocol version: 60002
    Node services: 0x0000000000000000
      .... .... .... .... .... .... .... ...0 = Network node: Not set
    Node timestamp: Jul 29, 2017 16:14:21.000000000 Paris, Madrid (heure d'été)
    Address as receiving node
      Node services: 0x0000000000000000
        .... .... .... .... .... .... .... ...0 = Network node: Not set
      Node address: ::ffff:127.0.0.1 (::ffff:127.0.0.1)
      Node port: 8333
    Address of emmitting node
      Node services: 0x0000000000000000
        .... .... .... .... .... .... .... ...0 = Network node: Not set
      Node address: ::ffff:127.0.0.1 (::ffff:127.0.0.1)
      Node port: 8333
    Random nonce: 0x0000026d000054d8
    User agent
      Count: 0
      String value:
    Block start height: 0
```

```
0020  44 d7 cf 50 20 8d 71 45  46 35 61 ff 75 18 50 18   D..P .qE F5a.u.P.
0030  44 10 1b 41 00 00 f9 be  b4 d9 76 65 72 73 69 6f   D..A.... ..versio
0040  6e 00 00 00 00 00 55 00  00 00 7e b7 e0 5f 62 ea   n.....U. ..~.._b.
0050  00 00 00 00 00 00 00 00  00 00 3d 98 7c 59 00 00   ........ ..=.|Y..
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 ff ff 7f 00  00 01 20 8d 00 00 00 00   ........ .. ....
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ........ ........
0090  7f 00 00 01 20 8d d8 54  00 00 6d 02 00 00 00 00   .... ..T ..m.....
00a0  00 00 00                                          ...
```

version

```
Packet magic: 0xf9beb4d9
Command name: version
Payload Length: 103
Payload checksum: 0x4c3b02b3
Version message
  Protocol version: 70014
  Node services: 0x0000000000000005
    .... .... .... .... .... .... .... ...1 = Network node: Set
  Node timestamp: Jul 29, 2017 16:14:29.000000000 Paris, Madrid (heure d'été)
  Address as receiving node
    Node services: 0x0000000000000000
      .... .... .... .... .... .... .... ...0 = Network node: Not set
    Node address: ::ffff:84.101.210.191 (::ffff:84.101.210.191)
    Node port: 53072
  Address of emmitting node
    Node services: 0x0000000000000005
      .... .... .... .... .... .... .... ...1 = Network node: Set
    Node address: ::ffff:5.178.68.215 (::ffff:5.178.68.215)
    Node port: 8333
  Random nonce: 0xc229404ba283a0c9
  User agent
    Count: 17
    String value: /Satoshi:0.13.99/
  Block start height: 478129
```

```
0030  39 08 1e b6 00 00 f9 be  b4 d9 76 65 72 73 69 6f   9.....·· ··versio
0040  6e 00 00 00 00 00 67 00  00 00 4c 3b 02 b3 7e 11   n.....g. ..L;..~.
0050  01 00 05 00 00 00 00 00  00 00 45 98 7c 59 00 00   ........ ..E.|Y..
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 ff ff 54 65  d2 bf cf 50 05 00 00 00   ......Te ...P....
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ........ ........
0090  05 b2 44 d7 20 8d c9 a0  83 a2 4b 40 29 c2 11 2f   ..D. ... ..K@)../
00a0  53 61 74 6f 73 68 69 3a  30 2e 31 33 2e 39 39 2f   Satoshi: 0.13.99/
00b0  b1 4b 07 00 01                                     .K...
```

verack

```
Bitcoin protocol
    Packet magic: 0xf9beb4d9
    Command name: verack
    Payload Length: 0
    Payload checksum: 0x5df6e0e2
```

```
0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 40 a5 18 40 00 36 06  92 4b 05 b2 44 d7 c0 a8   .@..@.6. .K..D...
0020  02 23 20 8d cf 50 61 ff  75 97 71 45 46 a2 50 18   .# ..Pa. u.qEF.P.
0030  39 08 b0 57 00 00 f9 be  b4 d9 76 65 72 61 63 6b   9..W.... ..verack
0040  00 00 00 00 00 00 00 00  00 00 5d f6 e0 e2         ........ ..]...
```

verack

```
    Packet magic: 0xf9beb4d9
    Command name: verack
    Payload Length: 0
    Payload checksum: 0x5df6e0e2
```

```
0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 40 39 36 40 00 80 06  b4 2d c0 a8 02 23 05 b2   .@96@... .-...#..
0020  44 d7 cf 50 20 8d 71 45  46 a2 61 ff 76 06 50 18   D..P .qE F.a.v.P.
0030  43 22 a5 ce 00 00 f9 be  b4 d9 76 65 72 61 63 6b   C"...... ..verack
0040  00 00 00 00 00 00 00 00  00 00 5d f6 e0 e2         ........ ..]...
```

```
    Packet magic: 0xf9beb4d9
    Command name: ping
    Payload Length: 8
    Payload checksum: 0x9a691ff6
```

```
0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 48 a5 19 40 00 36 06  92 42 05 b2 44 d7 c0 a8   .H..@.6. .B..D...
0020  02 23 20 8d cf 50 61 ff  75 af 71 45 46 a2 50 18   .# ..Pa. u.qEF.P.
0030  39 08 53 09 00 00 f9 be  b4 d9 70 69 6e 67 00 00   9.S..... ..ping..
0040  00 00 00 00 00 00 08 00  00 00 9a 69 1f f6 b8 e4   ........ ...i....
0050  71 31 46 27 d6 cb                                 q1F'..
```

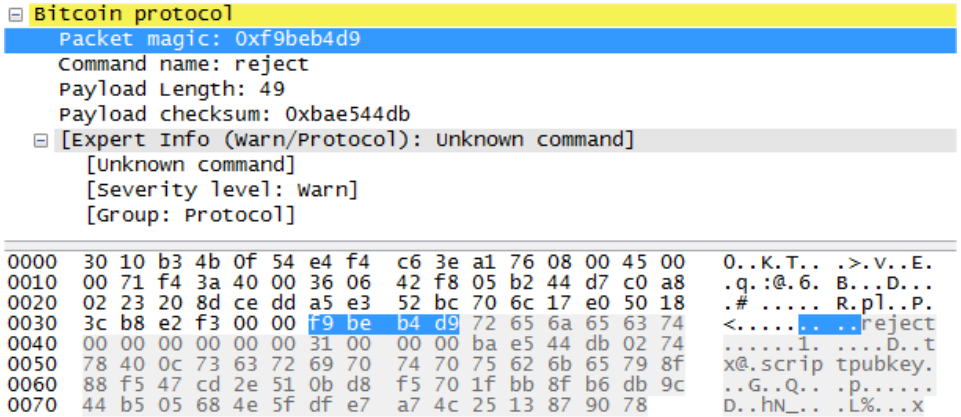
Bitcoin protocol

```
    Packet magic: 0xf9beb4d9
    Command name: pong
    Payload Length: 8
    Payload checksum: 0x9a691ff6
```

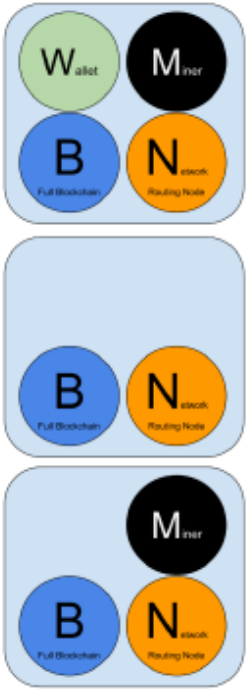
```
0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 48 39 38 40 00 80 06  b4 23 c0 a8 02 23 05 b2   .H98@... .#...#..
0020  44 d7 cf 50 20 8d 71 45  47 ce 61 ff 76 06 50 18   D..P .qE G.a.v.P.
0030  43 22 47 66 00 00 f9 be  b4 d9 70 6f 6e 67 00 00   C"Gf.... ..pong..
0040  00 00 00 00 00 00 08 00  00 00 9a 69 1f f6 b8 e4   ........ ...i....
0050  71 31 46 27 d6 cb                                 q1F'..
```

```
Packet magic: 0xf9beb4d9
Command name: tx
Payload Length: 252
Payload checksum: 0xe3a37c54
Tx message
  Transaction version: 1
  Input Count: 1
  Transaction input
    Previous output
      Script Length: 139
      Signature script: 483045022100f10ebbff6677f778d421d1baa656079b97e5..
      Sequence: 4294967295
  Output Count: 2
  Transaction output
      Value: 0
      Script Length: 19
      Script: 6a1154656c65636f6d20506172697354656368
  Transaction output
      Value: 126375
      Script Length: 25
      Script: 76a9143a40abac5b0c9ac4cb6471dce09f94eb11c0e4db88...
    Block lock time or block ID: 0
```

```
0030  43 22 06 b2 00 00 f9 be  b4 d9 74 78 00 00 00 00   C"....?? ??tx....
0040  00 00 00 00 00 00 fc 00  00 00 e3 a3 7c 54 01 00   ........ ....|T..
0050  00 00 01 e5 c9 9c 31 d7  cf a5 15 95 d6 74 63 f9   ......1. .....tc.
0060  66 9f b9 fc a0 3d 24 0c  63 a2 24 16 0a 54 5c 24   f....=$. c.$..T\$
0070  ac 24 6c 01 00 00 00 8b  48 30 45 02 21 00 f1 0e   .$l..... H0E.!...
0080  bb ff 66 77 f7 78 d4 21  d1 ba a6 56 07 9b 97 e5   ..fw.x.! ...V....
0090  53 5f 3d bb 45 41 b1 f1  11 76 8f bd 6a b1 02 20   S_=.EA.. .v..j..
00a0  75 36 26 9f 2a f0 4f d5  b9 9c db 10 62 12 b0 70   u6&.*.O. ....b..p
00b0  2b e5 7f e7 1a e3 5a 7c  21 9f 96 7a 22 7b 98 81   +.....Z| !..z"{..
00c0  01 41 04 98 ff 3b ad 68  9f a2 84 39 f3 3c 89 b8   .A...;.h ...9.<..
00d0  b3 28 5c e6 6f d8 ba e7  e1 7d 4e 83 c5 2b 72 d9   .(\.o... .}N..+r.
00e0  10 c1 1a 0f 0b a5 b3 c1  51 b2 38 3b 74 dd 1e 80   ........ Q.8;t...
00f0  13 d1 d3 00 00 7e ea cd  73 1f b7 65 f1 b2 20 de   .....~.. s..e.. .
0100  c9 69 70 ff ff ff ff 02  00 00 00 00 00 00 00 00   .ip..... ........
0110  13 6a 11 54 65 6c 65 63  6f 6d 20 50 61 72 69 73   .j.Telec om Paris
0120  54 65 63 68 a7 ed 01 00  00 00 00 00 19 76 a9 14   Tech.... .....v..
0130  3a 40 ab ac 5b 0c 9a c4  cb 64 71 dc e0 9f 94 eb   :@..[... .dq.....
0140  11 c0 e4 db 88 ac 00 00  00 00                     ........ ..
```


Bitcoin protocol
Packet magic: 0xf9beb4d9
Command name: reject
Payload Length: 49
Payload checksum: 0xbae544db
[Expert Info (Warn/Protocol): Unknown command]
[Unknown command]
[Severity level: Warn]
[Group: Protocol]
0000 30 10 b3 4b 0f 54 e4 f4 c6 3e a1 76 08 00 45 00 0..K.T.. .>.v..E.
0010 00 71 f4 3a 40 00 36 06 42 f8 05 b2 44 d7 c0 a8 .q.:@.6. B...D...
0020 02 23 20 8d ce dd a5 e3 52 bc 70 6c 17 e0 50 18 .# ..... R.pl..P.
0030 3c b8 e2 f3 00 00 f9 be b4 d9 72 65 6a 65 63 74 <..... ..reject
0040 00 00 00 00 00 00 31 00 00 00 ba e5 44 db 02 74 ......1. ....D..t
0050 78 40 0c 73 63 72 69 70 74 70 75 62 6b 65 79 8f x@.scrip tpubkey.
0060 88 f5 47 cd 2e 51 0b d8 f5 70 1f bb 8f b6 db 9c ..G..Q.. .p......
0070 44 b5 05 68 4e 5f df e7 a7 4c 25 13 87 90 78 D..hN_.. .L%...x

Network



Reference Client (Bitcoin Core)

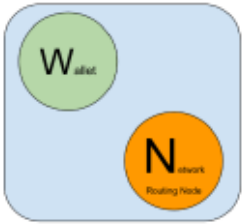
Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.

Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.

Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



Lightweight (SPV) wallet

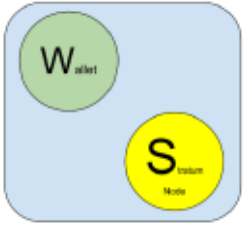
Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.

Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.

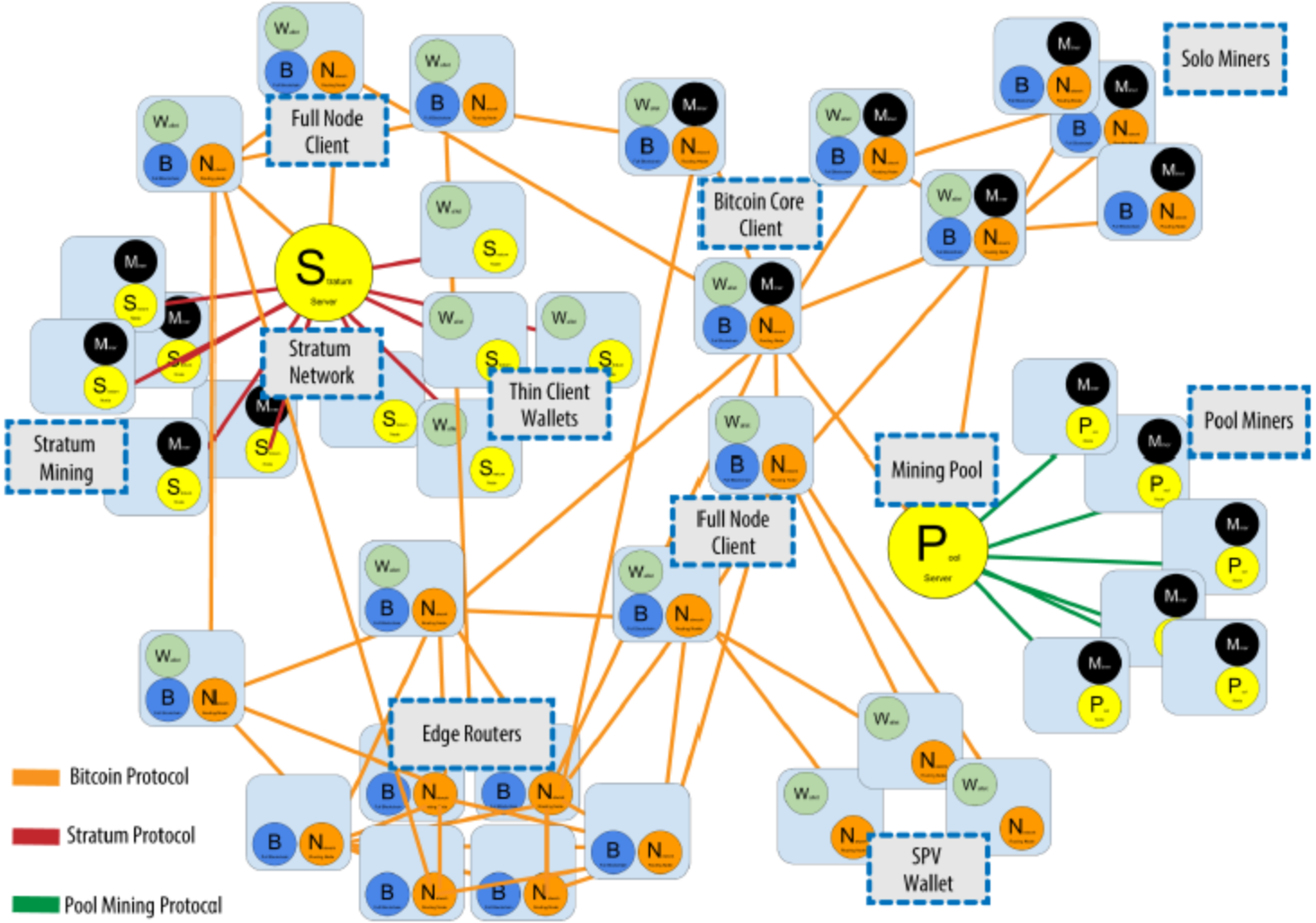
Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.

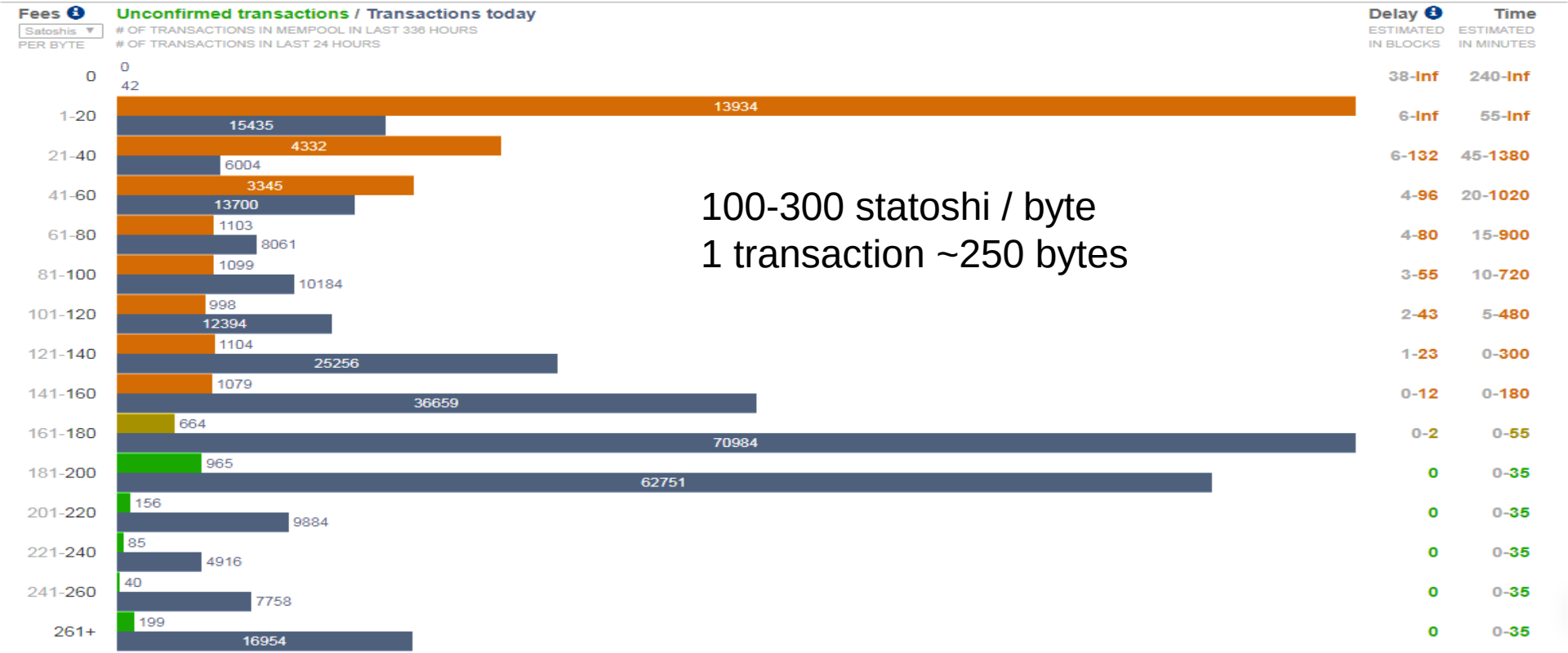


Lightweight (SPV) Stratum wallet

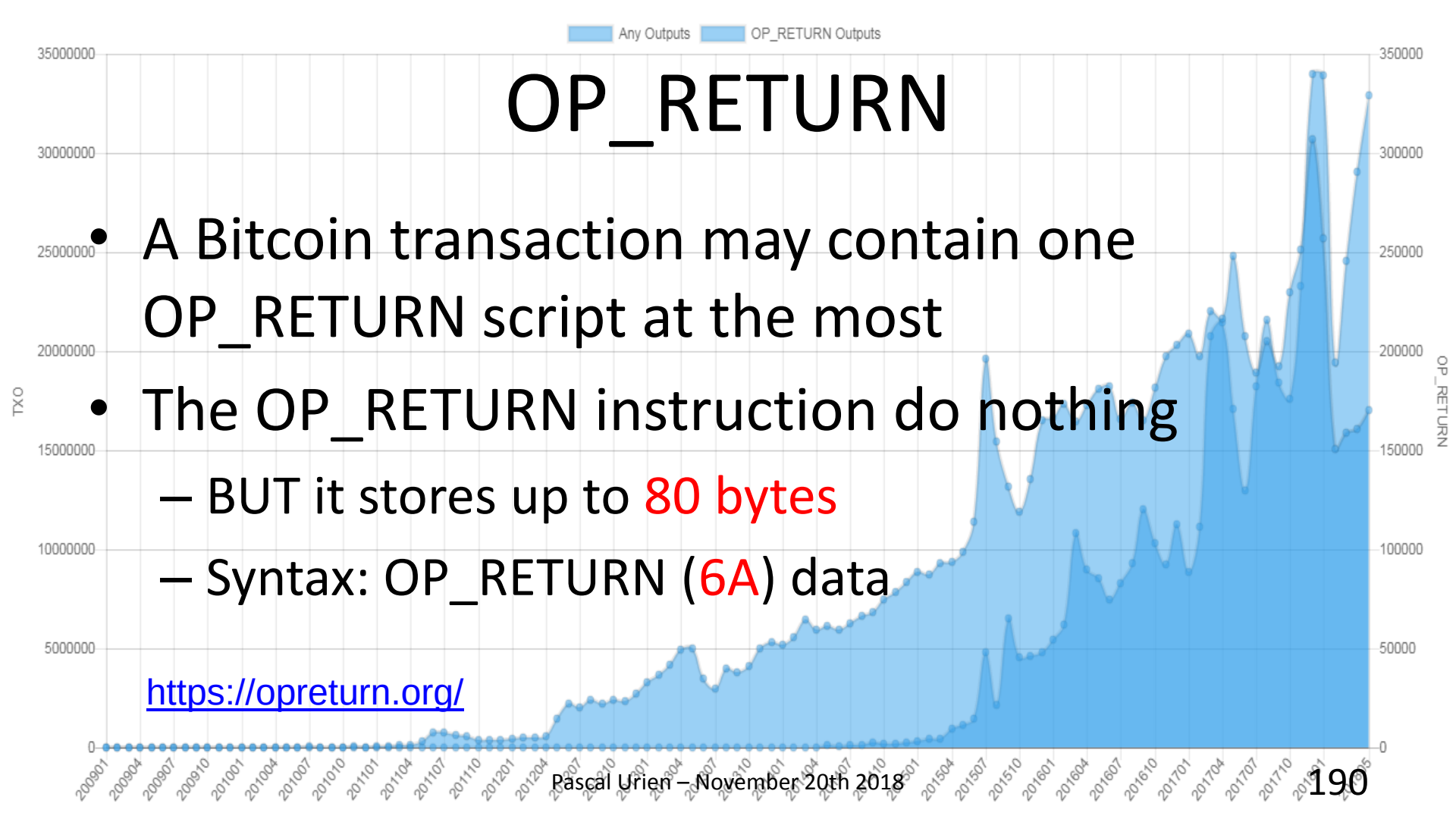
Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.


Solo Miners
Full Node Client
Bitcoin Core Client
Stratum Network
Thin Client Wallets
Stratum Mining
Pool Miners
Mining Pool
Full Node Client
Edge Routers
SPV Wallet
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocal

https://bitcoinfees.21.co/



100-300 statoshi / byte
1 transaction ~250 bytes

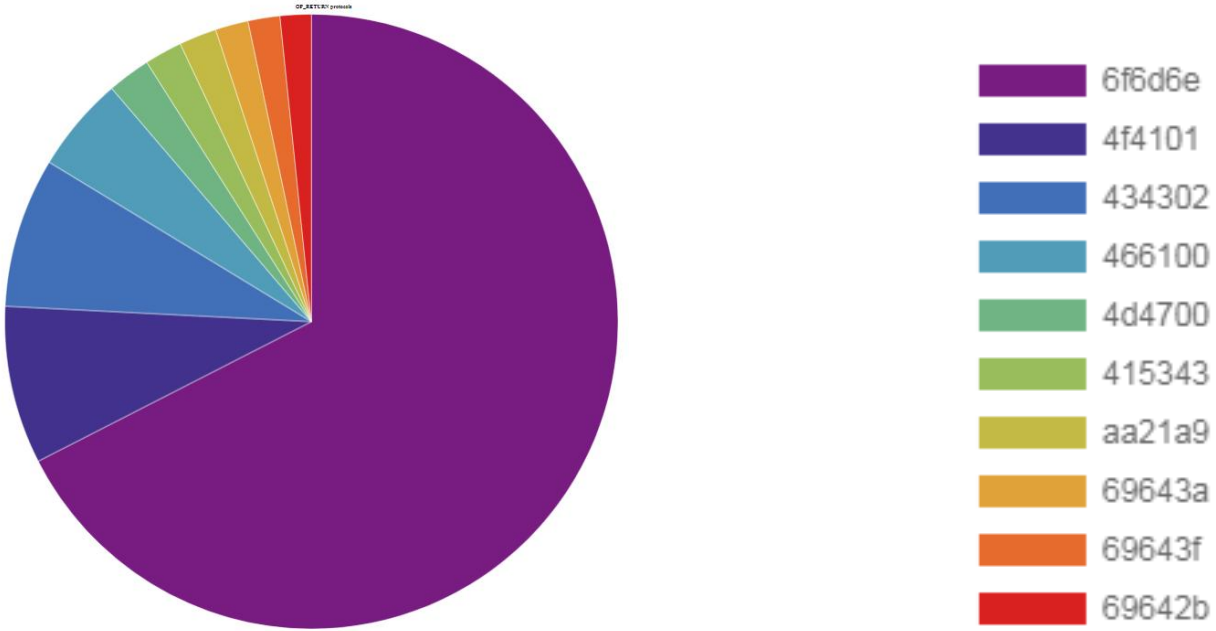


OP_RETURN

- A Bitcoin transaction may contain one OP_RETURN script at the most
- The OP_RETURN instruction do nothing
  - BUT it stores up to 80 bytes
  - Syntax: OP_RETURN (6A) data

https://opreturn.org/

Example of OP_RETURN protocols



OP_RETURN example 1/2

Transaction Afficher les informations d'une transaction bitcoin

01ee70227b26d14c3f33e1283cd66a6b7d6e8d8cb97b9cc77dd913d90f779cf1

16K1fX41VMvTW1DWyPw2rmXvvjrHMgL9q4 ($ 5.21 - Sortie) 16K1fX41VMvTW1DWyPw2rmXvvjrHMgL9q4 ($ 30.64 - Sortie)	→	Impossible de décoder l'adresse de sortie - (Non dépensé) 16K1fX41VMvTW1DWyPw2rmXvvjrHMgL9q4 - (Dépensé)
		$ 0.00 $ 29.03

$ 29.03

Récapitulatif	
Taille	488 (octets)
Poids	1952
Date de réception	2017-08-21 19:20:29
Inclue dans les blocs	481560 (2017-08-21 23:00:49 + 220 minutes)
Confirmations	45329 Confirmations
Visualiser	Voir le graphique

Entrées et sorties	
Total des entrées	$ 35.85
Total des sorties	$ 29.03
Taxes	$ 6.82
Frais par octet	204.918 sat/B
Frais par unité de poids	51.23 sat/WU
Estimation des BTC échangées	$ 0.00
Scripts	Cacher les scripts et coinbase

https://blockchain.info/fr/tx/01ee70227b26d14c3f33e1283cd66a6b7d6e8d8cb97b9cc77dd913d90f779cf1

OP_RETURN example 2/2

{ "transactionId": "01ee70227b26d14c3f33e1283cd66a6b7d6e8d8cb97b9cc77dd913d90f779cf1",
"index": 0,
"value": 0,
"scriptPubKey":
"6a4a687474703a2f2f7777772e656e73742e66722f7e757269656e2f38796e6855474b777234726e6f33517a425438533769
355539614448504a38515932757245676e5974336d652e6a7067", "redeemScript": null },

Scripts de sortie

RETURN PUSHDATA(74)
[687474703a2f2f7777772e656e73742e66722f7e757269656e2f38796e6855474b777234726e6f33517a425438533769355539614448504a38515932757245676e5974336d652e6a7067
(décodé) http://www.enst.fr/~urien/8ynhUGKwr4rno3QzBT8S7i5U9aDHPJ8QY2urEgnYt3me.jpg

DUP HASH160 PUSHDATA(20)[3a40abac5b0c9ac4cb6471dce09f94eb11c0e4db] EQUALVERIFY CHECKSIG

http://www.enst.fr/~urien/8ynhUGKwr4rno3QzBT8S7i5U9aDHPJ8QY2urEgnYt3me.jpg

http://www.enst.fr/~urien/Base58(SHA256(file)).jpg

About Ethereum

- Ethereum was introduced in a white paper by Vitalik Buterin in 2013
- The Ethereum software project was initially developed in early 2014 by the Swiss company, *Ethereum Switzerland GmbH,* and a a Swiss non-profit foundation, the Ethereum Foundation (*Stiftung Ethereum*).
- Ethereum's live blockchain was launched on 30 July 2015
- Ethereum is a blockchain platform supporting a digital currency *the Ether* and distributed applications called *Smart Contrats* written in *Serpent* or other languages.
  - The Ethereum Virtual Machine (EVM) supports a Turing complete language
- 1 ETHER = 10^{18} Wei.

Ethereum is a BlockChain

- A new block is mined every 14,0 s
- The Block reward is 5/3/2 Ethers
- Transactions are stored in the blockchain
- Every account is defined by a pair of keys (ECC sepc256k1), a private key and public key.
  - Accounts are indexed by their *address* which is derived from the public key by taking the last 20 bytes.

Account

- An Ethereum account contains four fields:
  - The **nonce**, a counter used to make sure each transaction can only be processed once
    - A scalar value equal to the number of transactions sent by the sender
  - The account's current **Ether balance**
  - The account's **contract code**, if present
  - The account's **storage** (empty by default)

Ethereum Address

- Ethereum addresses (EA) are computed from ECDSA public keys
- A private key, i.e. a 32 byte number x, is generated, according to a true random number generator (RNG).
- Thereafter a public key is computed according to the relation P=xG. The uncompressed form u'F(P) is a set of 64 bytes $\{x_P,y_P\}$, i.e. the point (x_P,y_P) of the curve (2x32 bytes, in Z/pZ).
- The Ethereum address is computed according to the following procedure:
  - Compute a1= Keccak(u'F(P)), a 32 byte value. SHA3 is this a subset the Keccak algorithm.
  - Extract a2, the 20 rightmost bytes of a1; a2 is the Ethereum address

Ethereum Transaction

- A transaction encodes the transfer of ethers or data between two entities, identified by their address. It includes the following fields:
  - The **recipient**'s address of the message
  - **nonce**, a scalar value equal to the number (>=0) of transactions generated by the sender.
  - **value**, a scalar value equal to the number of Wei (1 Wei=10^{-18} Ether) to be transferred to the message recipient, or in the case of contract creation, as an endowment for the newly created account.
  - A **gasLimit** value, representing the maximum number of computational steps that the transaction execution is allowed to take.
  - A **gasPrice** value, representing the fee the sender pays per computational step. A scalar value equal to the number of Wei to be paid per unit of gas.
  - An optional **data** field. A contract creation transaction contains an unlimited size byte array specifying the EVM (*Ethereum Virtual Machine*) code for the account initialization procedure. A message call transaction contains an unlimited size byte array specifying the input data of the message.
  - The ECDSA **signature**, used to identify the sender.

RLP Encoding

- All transaction attributes are encoding according to the RLP (*Recursive Length Prefix*) syntax, which supports *string* and *list* items.
- *String encoding*
  - A *string* is a byte array, it is encoded according to the following rules :
  - for one byte ϵ [0x00 0x7F] : a byte value
  - if the string length ϵ [0,55] : 0x80 + Length ϵ [0x80, 0xb7] || ByteArray[Length]
  - 0x80: = NULL String
  - if the string Length >55 : 0xb7 + Length-of-Lengh ϵ [0xb8, 0xbf] || Length-value || ByteArray[Length]
- *List encoding*
  - A list is a set of items, either *list* or *string*.
  - if the list Length <=55 : 0xc0 + Length ϵ [0xc0, 0xf7] || ListItems.
  - if the list Length > 55 : 0xf7 + Length-of-Length ϵ [0xf8, 0xff] || Length-value || ListItems.

RAW Transaction

```
E8 80 // list length = 40 bytes
80 // nonce = null (zero value)
85 04E3B29200 // gazPrice= 21 000 000 000 Wei)
82 9C40 // gazLimit= 40 000 Wei
94 777A07BAB1C119D74545B82A8BE72BEAFF4D447B //Recipient
87 2386F26FC10000 // value= 10 000 000 000 000 000 Wei
80 // data = null
```

Hash and Sign

Ethereum Transaction

```
F8 6B  // list length= 107 bytes
80 // nonce = null (zero value)
85 04E3B29200 // gazPrice= 21 000 000 000 Wei)
82 9C40 // gazLimit= 40 000 Wei
94 777A07BAB1C119D74545B82A8BE72BEAFF4D447B //Recipient
87 2386F26FC10000 // value= 10 000 000 000 000 000 Wei
80 // data = null
1C // signature recovery parameter = 28 (27+i)
A0 F1DD7D3B245D75368B467B06CAD61002 // r value
   67031935B7474ACB5C74FE7D8C904097 // 32 bytes
A0 772D65407480D7C45C7E22F84211CB1A // s value
   DF9B3F36046A2F93149135CADBB9385D // 32 bytes
```

Public key is recovered from the signature two solutions + (27) and (-) (28)

Overview | Comments

Transaction Information

Tools & Utilities

https://etherscan.io/tx/0xd6904d832462ae17718c69e9caa0c3f3bed458382ac1f4e43b1aadd8e94744ad

TxHash:	0xd6904d832462ae17718c69e9caa0c3f3bed458382ac1f4e43b1aadd8e94744ad
TxReceipt Status:	Success
Block Height:	4942834 (561272 block confirmations)
TimeStamp:	94 days 18 hrs ago (Jan-20-2018 09:52:42 PM +UTC)
From:	0x6bac1b75185d9051af740ab909f81c71bbb221a6
To:	0x6bac1b75185d9051af740ab909f81c71bbb221a6
Value:	0 Ether ($0.00)
Gas Limit:	80000
Gas Used By Txn:	22020
Gas Price:	0.00000003 Ether (30 Gwei)
Actual Tx Cost/Fee:	0.0006606 Ether ($0.41)
Nonce:	12
Input Data:	0xTemperature=25C

Switch Back

GasPrice= 15 bytes x 68 + 21000 = 22020

x

=

The yellow paper outlines the fees for various operations.

$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.

http://ethgasstation.info/index.php

Gas-Time-Price Estimator: For transactions sent at block: 4362424

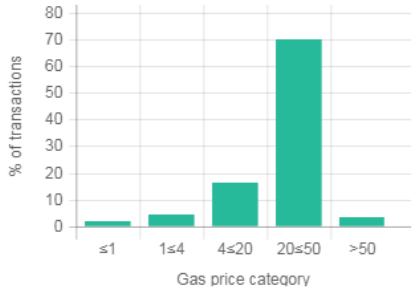
Adjust confirmation time

Avg Time (min)	4.13	Gas Used*	21000
95% Time (min)	10.33	Avg Time (blocks)	8.44
Gas Price (Gwei)*	20	95% Time (blocks)	21.1
Tx Fee (Fiat)	$0.142	Tx Fee (ETH)	0.00042

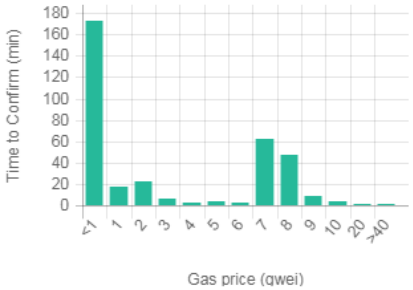
Real Time Gas Use: % Block Limit (last 10)

Last Block: 4362424

Transaction Count by Gas Price



Confirmation Time by Gas Price



Recommended Gas Prices

(based on current network conditions)

Speed	Gas Price (gwei)	Predicted Wait (minutes)
SafeLow (<20m)	9	8.6
Standard (<5m)	20	4.1
Fast (<2m)	27	1

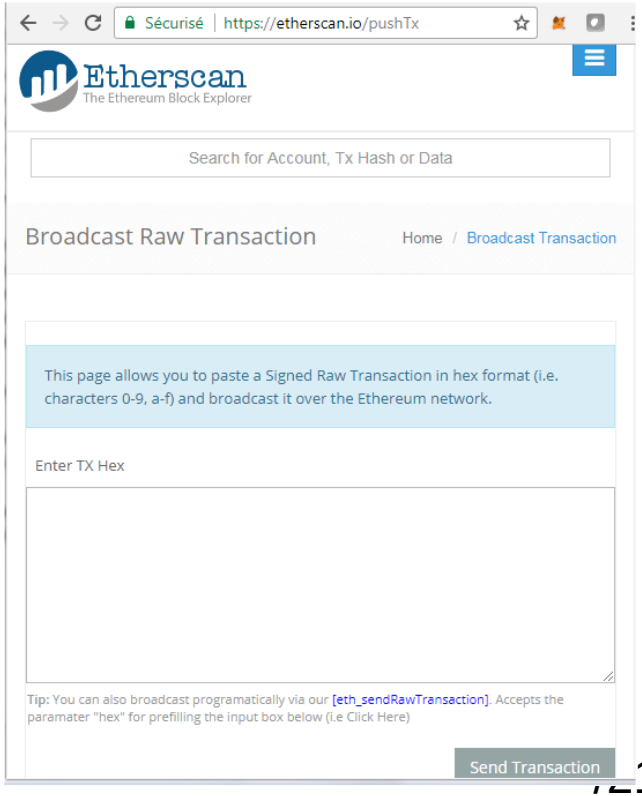
Ethereum RLPx:
Cryptographic Network & Transport Protocol

- https://github.com/ethereum/devp2p/blob/master/rlpx.md
- LPx is a cryptographic peer-to-peer network and protocol suite which provides a general-purpose transport and interface for applications to communicate via a p2p network. RLPx is designed to meet the requirements of decentralized applications and is used by Ethereum.
- The current version of RLPx provides a network layer for Ethereum.
- Roadmap Completed:
  - UDP Node Discovery for single protocol
  - ECDSA Signed UDP
  - Encrypted Handshake/Authentication
  - Peer Persistence
  - Encrypted/Authenticated TCP
  - TCP Framing
- Security
  - authenticated connectivity (ECDH+ECDHE, AES128)
  - authenticated discovery protocol (ECDSA)
  - encrypted transport (AES256)

ÐΞVp2p Wire Protocol

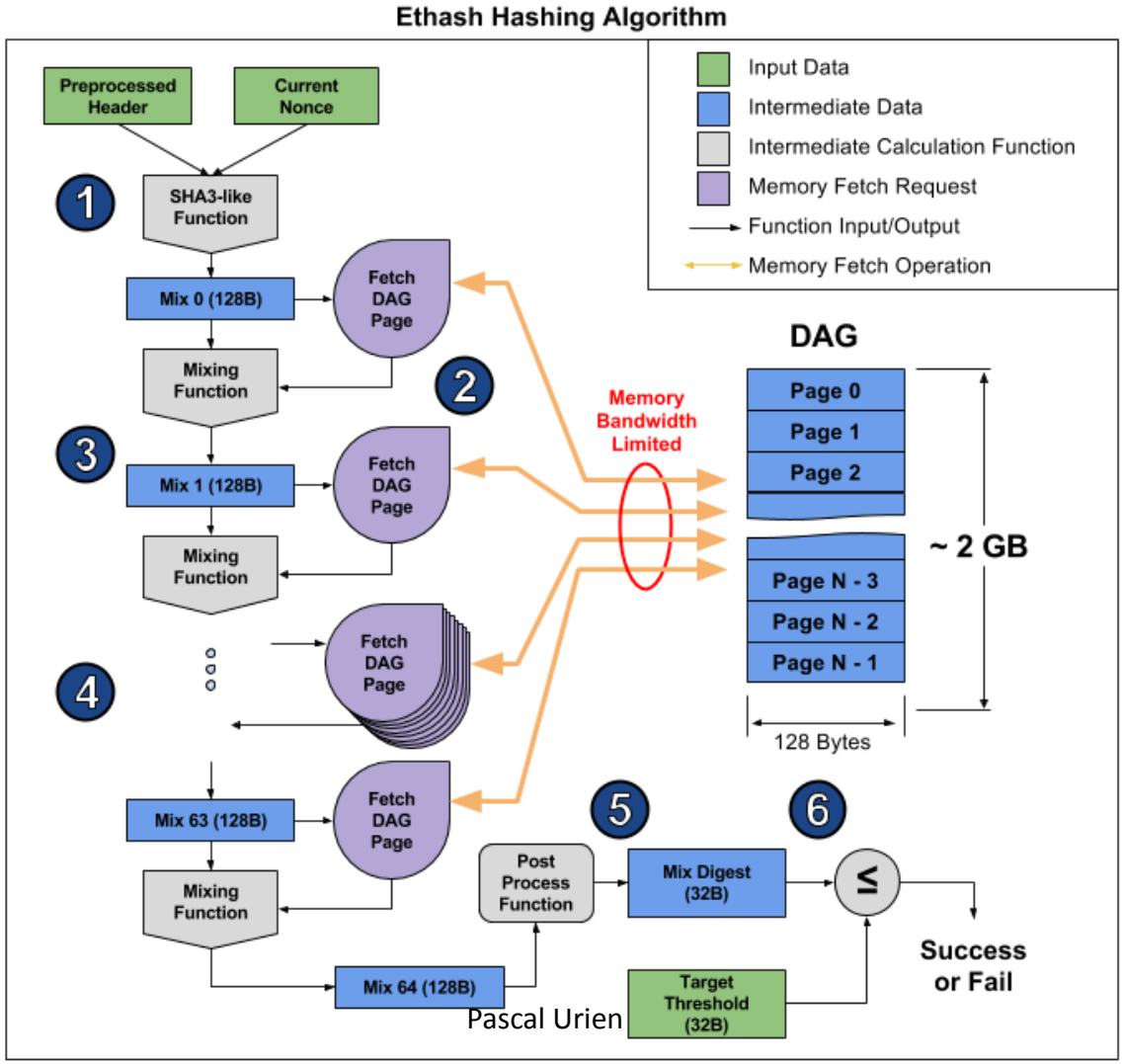
https://etherscan.io/pushTx

- https://github.com/ethereum/wiki/wiki/ÐΞVp2p-Wire-Protocol
- ÐΞVp2p nodes communicate by sending messages using RLPx, an encrypted and authenticated transport protocol. Peers are free to advertise and accept connections on any TCP ports they wish, however, a default port on which the connection may be listened and made will be 30303.



Mining

- Ethash is the PoW algorithm for Ethereum 1.0.
  - It is based on the sha3_512 hash function
  - "One plague of the Bitcoin world is ASICs. These are specialized pieces of compute hardware that exist only to do a single task. In Bitcoin's case the task is the SHA256 hash function. While ASICs exist for a proof-of-work function, both goals are placed in jeopardy. Because of this, a proof-of-work function that is ASIC-resistant (i.e. difficult or economically inefficient to implement in specialized compute hardware) has been identified as the proverbial silver bullet"
- Proof of stake is a consensus algorithm for public blockchains which is intended to serve as an alternative to proof of work.


Ethash Hashing Algorithm
Input Data
Intermediate Data
Intermediate Calculation Function
Memory Fetch Request
Function Input/Output
Memory Fetch Operation
Preprocessed Header
Current Nonce
1
SHA3-like Function
Mix 0 (128B)
Fetch DAG Page
Mixing Function
2
3
Mix 1 (128B)
Fetch DAG Page
Mixing Function
4
Fetch DAG Page
Mix 63 (128B)
Fetch DAG Page
Mixing Function
Mix 64 (128B)
Memory Bandwidth Limited
DAG
Page 0
Page 1
Page 2
Page N - 3
Page N - 2
Page N - 1
~ 2 GB
128 Bytes
5
Post Process Function
6
Mix Digest (32B)
≤
Target Threshold (32B)
Success or Fail

Contract Transaction

Transaction Information

TxHash:	0x79d910cc067334514e1f37ed2a3f2e1ad4ac13e46e97c208baa3f897bb74b336
Block Height:	2961230 (148972 block confirmations)
TimeStamp :	24 days 15 hrs ago (Jan-09-2017 12:42:49 AM +UTC)
From:	0xcafb10ee663f465f9d10588ac44ed20ed608c11e (Bitfinex_1)
To:	Contract 0xab7c74abc0c4d48d1bdad5dcb26153fc8780f83e
Value:	399,900 Ether ($4,254,936.00)
Gas:	122423
Gas Price:	0.00000002 Ether
Gas Used By Transaction:	22423
Actual Tx Cost/Fee:	0.00044846 Ether ($0.0048)
Cumulative Gas Used:	98037
Nonce:	437
Input Data:	0x

Pascal Urien

Creating a Contract

RLP(Address, nonce):
D6 (list)
94
3f406a15095669e63df80d21d54d12bdfa2141
87 (address)
80 (null byte)

Keccak(RLP)=
BF474FF3D5F9AC3393D3F6B9D4E29AD9AC3C8BA701C0FFAC566
117A2BBFDB177

Contract Code

Computing the contract Address

- The address for an Ethereum contract is **deterministically** computed from the address of its creator (sender) and how many transactions the creator has sent (nonce). The sender and nonce are RLP encoded and then **hashed with Keccak-256**.
- From pyethereum:
  - def mk_contract_address(sender, nonce): return sha3(rlp.encode([normalize_address(sender), nonce]))

Contrat Source

```
pragma solidity ^0.4.2;
address public owner;
string public log;
function storer()
{
    owner = msg.sender ;
}
modifier onlyOwner
{
        if (msg.sender != owner)
            throw;
        _;
}
function store(string _log) onlyOwner()
{
    log = _log;
}
function kill() onlyOwner()
{
  selfdestruct(owner); }
}
```

Example of online compiler
https://ethereum.github.io/browser-solidity/

Typing in solidity

Solidity includes 7 basic types, listed below:
hash: 256-bit, 32-byte data chunk, indexable into bytes and operable with bitwise operations.
uint: 256-bit unsigned integer, operable with bitwise and unsigned arithmetic operations.
int: 256-bit signed integer, operable with bitwise and signed arithmetic operations.
string32: zero-terminated ASCII string of maximum length 32-bytes (256-bit).
address: account identifier, similar to a 160-bit hash type.
bool: two-state value.

https://etherscan.io/opcode-tool

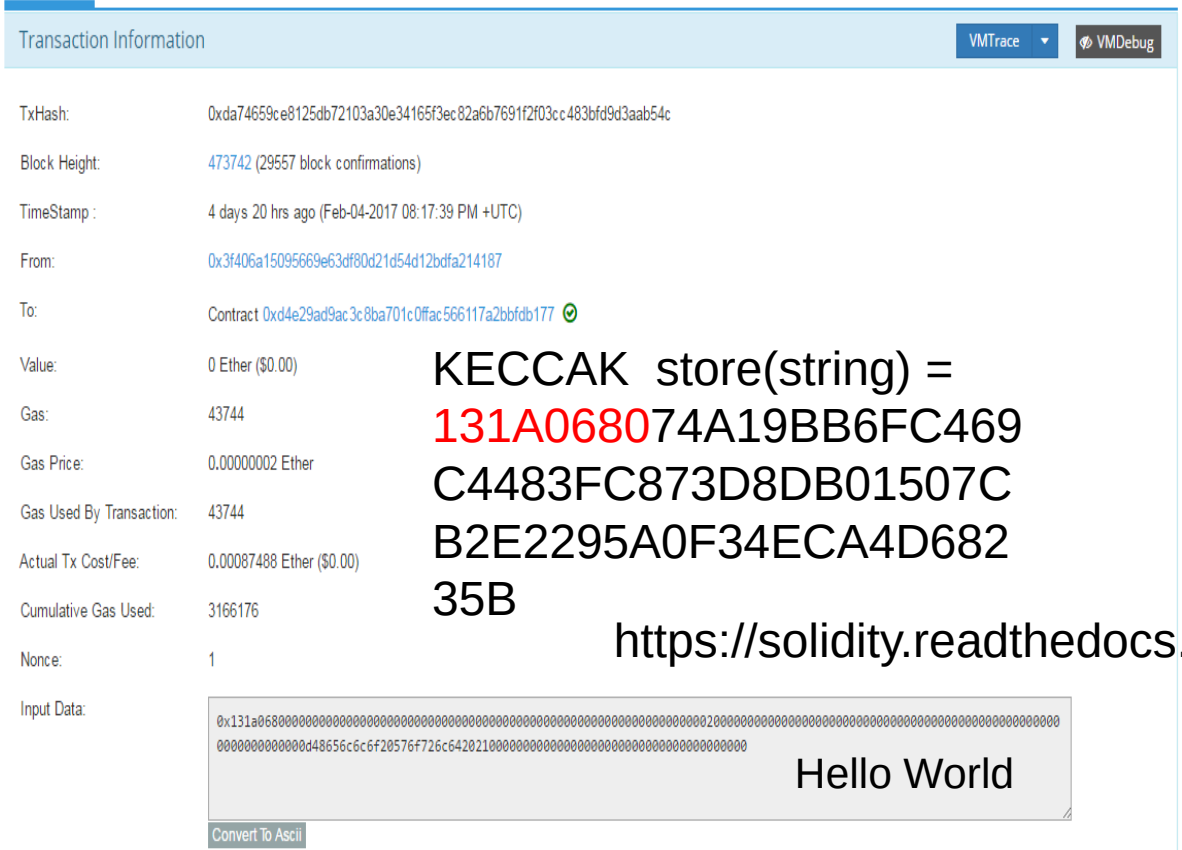
Decode

Decoded Bytecode :

[1] PUSH1 0x60
[3] PUSH1 0x40
[4] MSTORE
[5] CALLVALUE
[6] ISZERO
[9] PUSH2 0x000c

A contract's storage is at its most basic a key-value store with 2^256 possible keys and 2^256 values. This makes for sufficient possible storage to create database structures of any type imaginable.

Contract Transaction



KECCAK store(string) = 131A068074A19BB6FC469C4483FC873D8DB01507CB2E2295A0F34ECA4D68235B

https://github.com/ethereum/go-ethereum/wiki/Contracts-and-Transactions

https://solidity.readthedocs.io/en/develop/abi-spec.html

```
131a0680 the Method ID. This is derived as the first 4 bytes of the Keccak hash of function ASCII name
0000000000000000000000000000000000000000000000000000000000000020    // Offset of the parameter = 32
000000000000000000000000000000000000000000000000000000000000000d    // String lentgh= 13
48656c6c6f20576f726c6420210000000000000000000000000000000000000000    // string
```

Contract Transaction