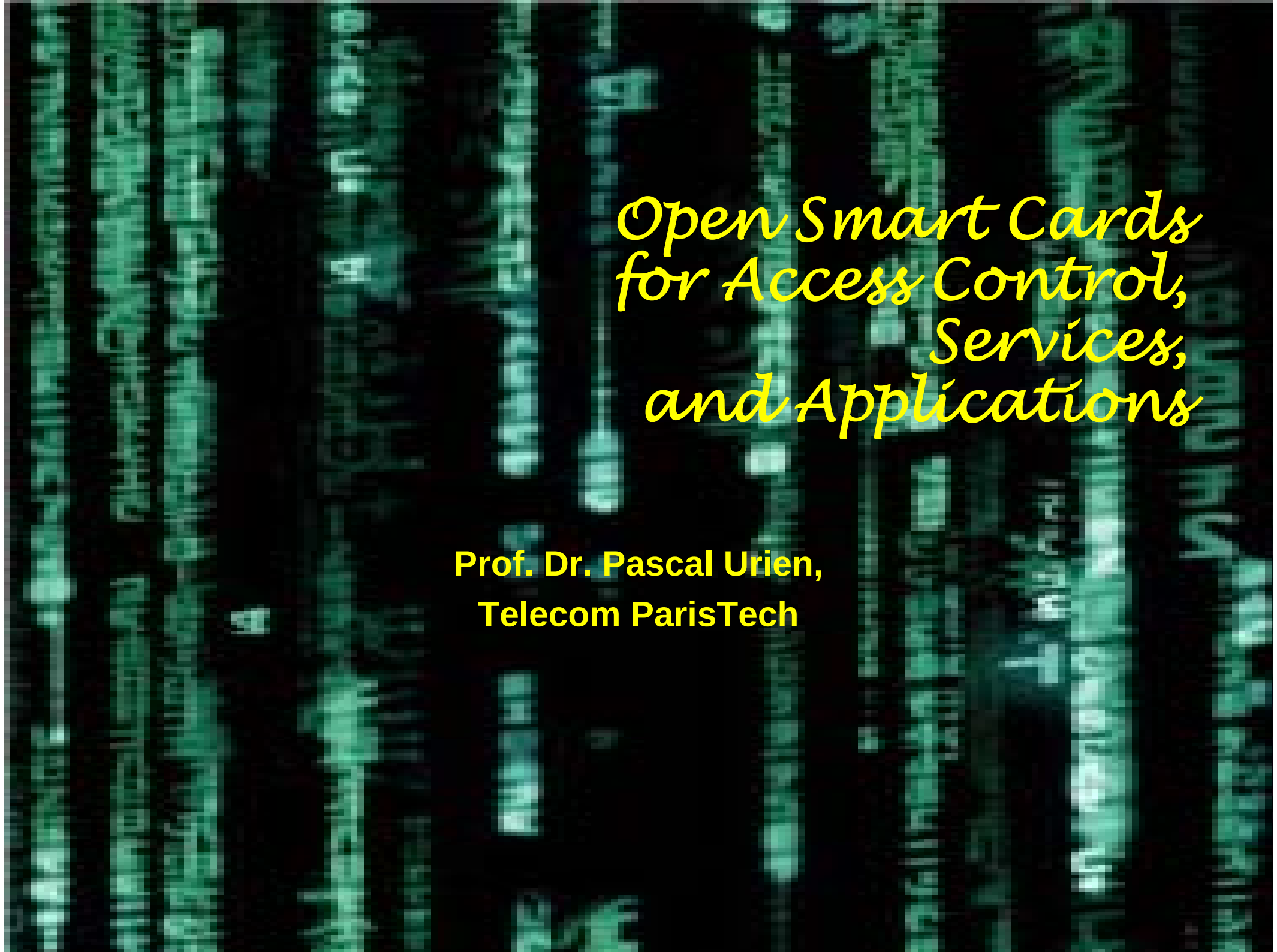

Open Smart Cards
for Access Control,
Services,
and Applications
Prof. Dr. Pascal Urien,
Telecom ParisTech

Agenda

- Introduction
- About smart cards
- EAP smart cards
- A practical use case for EAP smart cards
- Smart cards enabled RADIUS server
- Identity Protection
- Keys-Tree computing
- Smart cards for WEB applications
  - SSL and OpenID use cases
- Conclusions

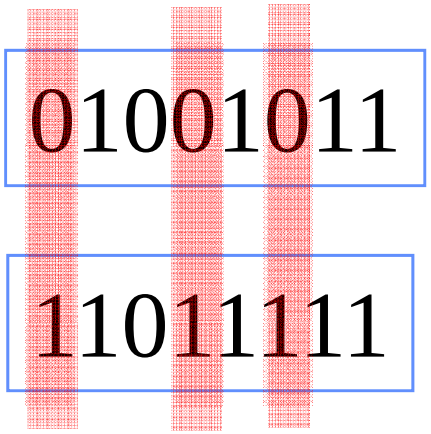
TELECOM ParisTech

Introduction: trusted computing platforms 1/5

- **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), $\rho_{X,Y} \neq 0$
    - $\langle X_j(k,t).Y_j(k) \rangle_k = \rho_{X,Y}\, \sigma(X)\, \sigma(Y)$

Introduction: trusted computing platforms 2/5

- ***Information leakage***
  - *Thomas S. Messerges and Ezzy A. Dabbish,* Investigations of Power Analysis Attacks on Smartcards, *USENIX Workshop on Smartcard Technology,* Chicago, Illinois, USA, May 10–11, 1999
  - Wo = ½ CV^2
  - Energy = ∑ Wo (bi(t+1) exor (bi(t))



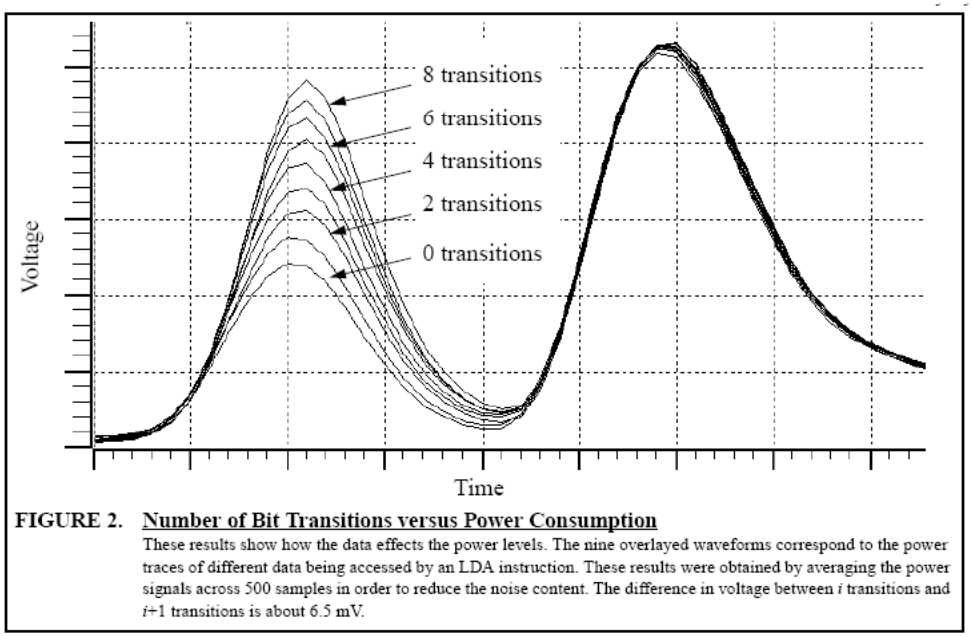


FIGURE 2. **Number of Bit Transitions versus Power Consumption**
These results show how the data effects the power levels. The nine overlayed waveforms correspond to the power traces of different data being accessed by an LDA instruction. These results were obtained by averaging the power signals across 500 samples in order to reduce the noise content. The difference in voltage between *i* transitions and *i*+1 transitions is about 6.5 mV.

Introduction: trusted computing platforms 3/5

- ***D. Boneh,* Twenty years of attacks on the RSA cryptosystem,*Notices* of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203-213, 1999**
- **Example, Bellcore attack**
  - N= pq
  - Chinese Remainder Theorem : $E= x^s \bmod pq = a\,E1 + b\,E2$
    - $a=1 \bmod p$, $a=0 \bmod q$
    - $b=1 \bmod q$, $b=0 \bmod p$
    - $E1 = x^s \bmod p$, $E2 = x^s \bmod q$
  - If a computing fault E1' is created in place of E,
    - If E1-E1' is not divisible by p, then
    - $\gcd(E'-E, N) = \gcd(a(E1'-E1), N) = q$

Introduction: trusted computing platforms 4/5

- **Security is the cornerstone for huge deployment of pervasive WEB applications, more and more dealing with radio technology.**
- **In the today landscape, the SSL protocol looks like the Holy Grail, whose golden padlock restricts service access to legitimate subscribers and enforces security policies.**
  - HTTPS, eMail, SIP, …
- **SSL/TLS robustness has been checked and confirmed by multiple security analysis, based on formal models**
  - *L. C. Paulson. "Inductive analysis of the Internet protocol TLS". ACM Transactions on Computer and System Security,2(3):332–351, 1999.*
  - *He, C., Sundararajan, M., Datta, A., Derek, A. and J. Mitchell, "A Modular Correctness Proof of IEEE 802.11i and TLS", CCS'05, November 7-11, 2005, Alexandria, Virginia, USA.*

Introduction: trusted computing platforms 5/5

- ***Branch Prediction Attacks*** **(2006) may recover an RSA key during a single calculation**
  - *Acucmez, O., Seifert, J.-P., Koc, C.K., "Micro-Architectural Cryptanalysis", Security & Privacy Magazine, IEEE Volume 5, Issue 4, July-Aug. 2007 Page(s):62 – 64.*
- ***Instruction Cache Attacks*** **(2005) may recover an AES key in 65 milliseconds**
  - *Dag Arne Osvik, Adi Shamir, Eran Tromer, Cache attacks and countermeasures: the case of AES, proc. RSA Conference Cryptographers Track (CT-RSA) 2006, LNCS 3860, 1--20, Springer, 2006.*
- **These attacks work with OpenSSL, which runs on more than 60 percent of the world's server installations.**
- **The issue is that SSL/TLS stacks are running in untrustworthy computers.**
  - RSA private key may be recovered (client's side)
  - TLS master secret may be recovered
    - Session hijacking
  - No plug and play architecture
    - Host computer must be configured with the CA Certificate

About Smart cards

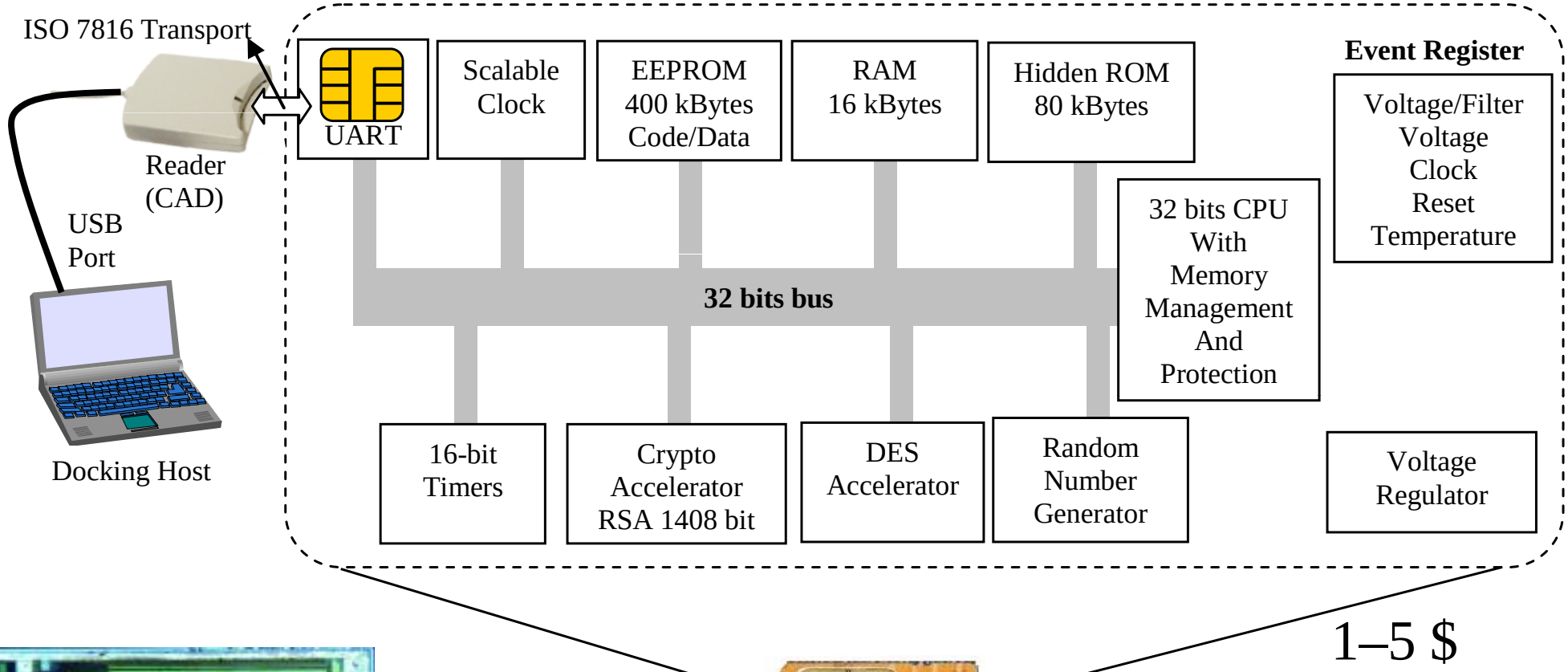
What is a smart card ?

- **A smart card is a SPOM**
  - Self Programmable One Chip Microcomputer, born in 1980
    - 4 billions smartcards were produced in 2008 !
  - Tamper resistant device
    - Security is enforced by physical and logical countermeasures
  - Typical chip size, 5mm x 5mm
- **Memories size**

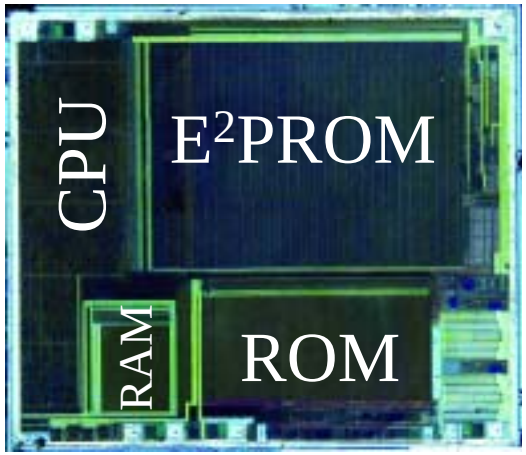
ROM	28 - 256 Kbytes	Area Factor 1
E²PROM	64 - 128 Kbytes	Area Factor 4
RAM	4 - 8 Kbytes	Area Factor 16

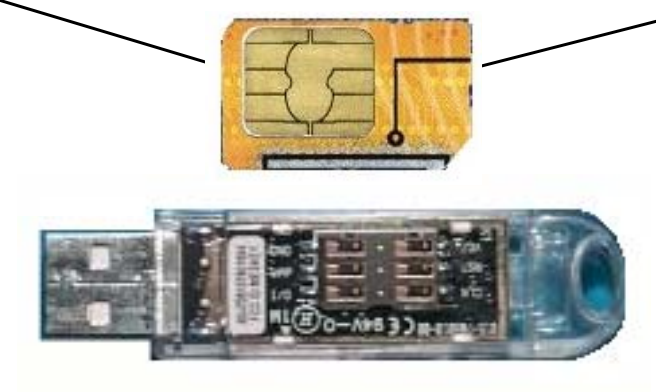
- **CPU**
  - Classical 8 bits processors, 1 - 3 MIPS (Clock 3.3 MHz)
  - 32 bits RISC processors
- **Communication port**
  - ISO7816 serial link 9600 to 230,400 bauds
  - USB (ISO7816-12), 10 Mbit/s
- **Binary Encoding rules**
  - A five bytes Header
    - CLA INS P1 P2 P3
  - An optional payload of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

What is a smart card ?



1–5 $





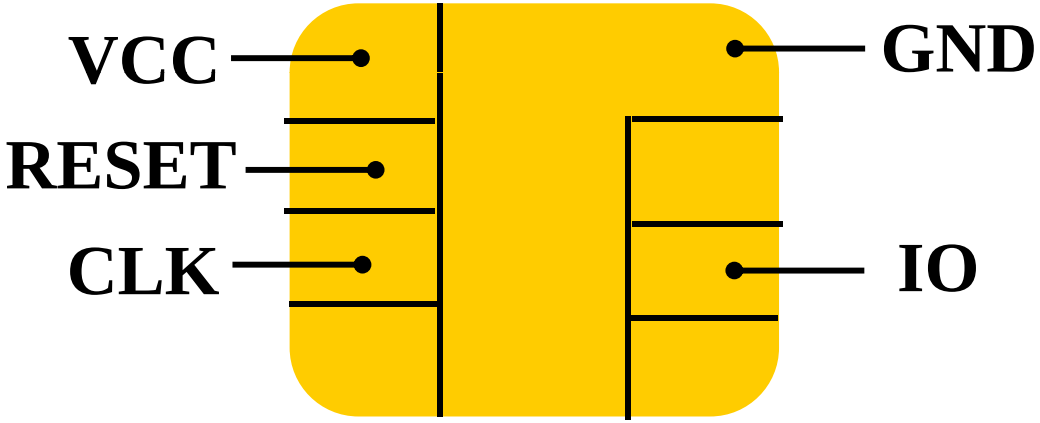
Performances example: The ST22 micro-controller

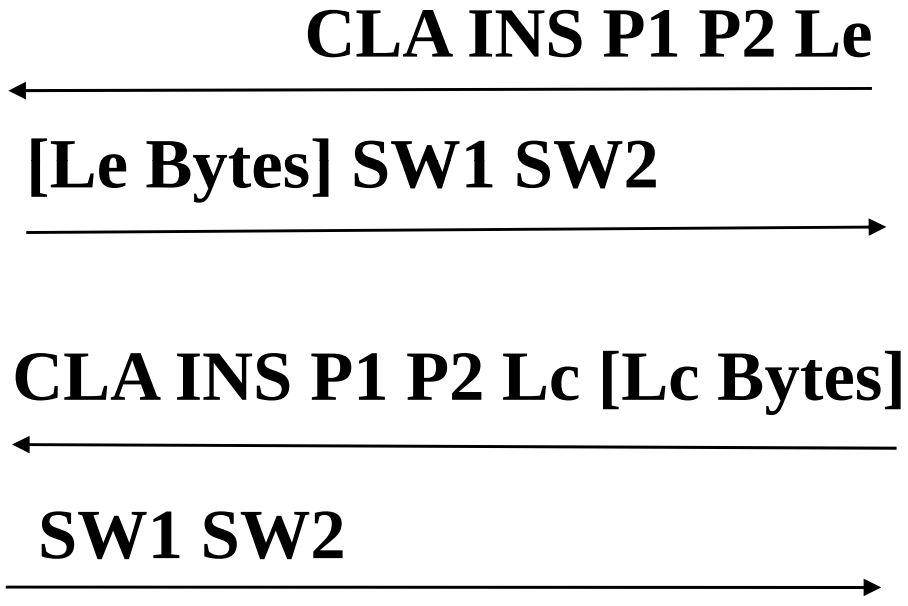
Algorithm	Function	Time(1)
RSA 1024 bits	Signature with CRT	79.0 ms
	Signature without CRT(2)	242.0 ms
	Verification (e=0x10001)	3.6 ms
RSA 2048 bits	Signature with CRT	485.0 ms
	Signature without CRT	1.7 s
	Verification (e=0x10001)	11.0 ms
DES	Triple	18 µs
	Single	8 µs
TDES(3)	Triple (with keys loaded)	1.8 us
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

1. Internal clock at 33 MHz

Basic smart card operations

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

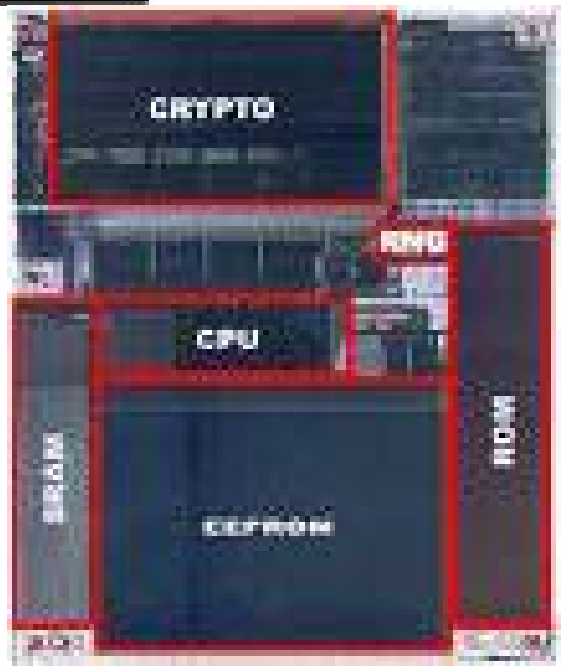




2008 shipments

Sectors	Telecom SIM USIM	Financial EMV	Government Healthcare passport	Transport	PayTV	Corporate Security - Other	Total
x 10^6	3200	650	140	30	100	65	4185

Smart card technological figures



CPU- 8bit data bus CPUs are dominating the microcontroller smart cards as in the industry globally.
Favorite 8bit CPUs are : 8051, 6805, HC05, AVR etc....
8bit CPU complexity is ranging from 1500 gates to 6000 gates.
Using state of the art 0.35 µm technology, 8bit CPU consumes 0.3-0.6 mm² of silicon.

32 bit CPU complexity is in the 100 000 gates range.

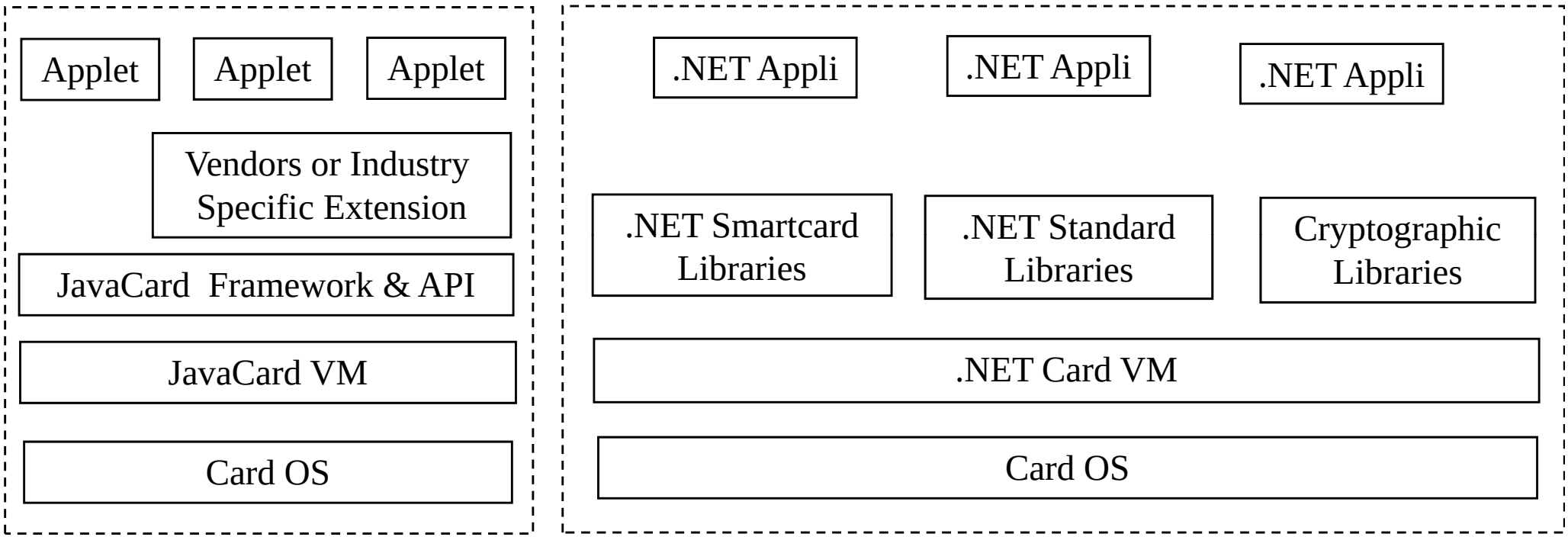
SRAM capacity ranges between 256 Bytes to 2k Bytes.
SRAM takes a lot of area on the IC since each memory cell consists of 6 transistors.
Using state of the art 0.35µm technology, 2kByte SRAM consumes 0.25-0.35mm² of silicon.

EEPROM- capacity is ranging today between 8kBytes **and 32kBytes.**
Using state of the art 0.35 µm technology, 32kByte EEPROM consumes 4-6 mm² of silicon.
EEPROM program/erase uses internally generated high voltage (15-20V) and low current (nA/cell) but takes about 2ms per cell. To cope with the slow program/erase operation, 64Bytes are usually programmed at once in a Page mode mechanism.
Main issue with the EEPROM functionality is its reliability expressed in data retention and program/erase cycling or endurance.
Access time to the data stored in the EEPROM is in the nanosecond range.

ROM capacity usually ranges between 8k Byte and 64k Bytes.
Since ROM unit memory cell is made of a single transistor, it is very dense.
Using state of the art 0.35 µm technology, a 64kByte ROM consumes 0.9-1.2mm² of silicon.
Access time to the Operating System microcode instruction is in the nanosecond range.
Multi application and needs for interoperability are requesting more complex operating system, and therefore larger ROM capacity (>64 k Bytes). .

The current **Flash-EEPROM** memories are guaranteed for a data retention time of at least 10 years or at least 100.000 write/erase cycles. **There is a considerable gain of writing time per memory access: to about 10µs with Flash, compare to 3-10 ms with normal EEPROM.**

Smartcards programming: Java VM and .NET VM



JAVA .CS

TELECOM ParisTech

Demonstration 1.

- **A simple programming example**
  - Application name is: A000000030 0002FFFFFFFF8931323800
  - A non volatile memory of 4KB is handled by the tamper resistant chip
  - Three operations
    - Application selection
      - **00A40400 10 A000000030 0002FFFFFFFF8931323800**
    - Reading operating
      - **00 B0 msb lsb LE**
      - **[LE bytes] 90 00**
    - Writing operation
      - **00 D0 msb lsb LC [LC bytes]**
      - **90 00**
  - Error managements
    - Out of range index
      - **Tx:00D0 2000 04 12 34 56 78, Rx:6D01**
      - **Tx:00B0 2000 04 Rx: 6D02**

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()  {  } }
```

Dotnet code

```
using System;
using System.IO;
using SmartCard;
using SmartCard.Runtime.InteropServices.ISO7816;

namespace MyCompany.MyOnCardApp
{
public class MyService : MarshalByRefObject
{ byte[] NVR = null;

public MyService()
{  NVR = new byte[4096];   }

 [APDU("SELECT")]
  public void Select(MemoryStream AID)
 {// Application is selected  }

public static short makeShort(byte msb, byte lsb)
{   ushort value, v;
    v = (ushort)lsb; v &= 0xFF;
    value = v;
    v = (ushort)msb; v &= 0xFF;
    value = (ushort)(value | (v << 8));
    return ((short)value);
 }
```

```
 [APDU("00B00000", Mask = "FC00FFFF")]
 [APDUException(typeof(ApplicationException), "6D01")]

public byte[] Read([APDUParam(APDUHeader.CLA)]byte cla,
[APDUParam(APDUHeader.INS)]byte ins,
[APDUParam(APDUHeader.P1)]byte p1,
[APDUParam(APDUHeader.P2)]byte p2,
[APDUParam(APDUHeader.P3)] byte p3)
 {  short len     =  makeShort((byte)0, p3);
    short offset  =  makeShort(p1, p2) ;
     if (offset < (short)0) offset = (short)(NVR.Length + offset);
     byte[] b = new byte[len];
     System.Array.Copy(NVR, offset, b, 0, len);  return b;   }

 [APDU("00D00000", Mask = "FC00FFFF")]
 [APDUException(typeof(ApplicationException), "6D02")]

 public void Write([APDUParam(APDUHeader.CLA)]byte cla,
[APDUParam(APDUHeader.INS)]byte ins,
[APDUParam(APDUHeader.P1)]byte p1,
[APDUParam(APDUHeader.P2)]byte p2,
[APDUParam(APDUHeader.P3)]byte p3, byte[] b)
 {  short len =       makeShort((byte)0, p3);
     short offset =  makeShort(p1, p2)  ;
    if (offset < (short)0) offset = (short)(NVR.Length +
    System.Array.Copy(b,0,NVR, offset, len);
} } }
```

Demonstration 1.

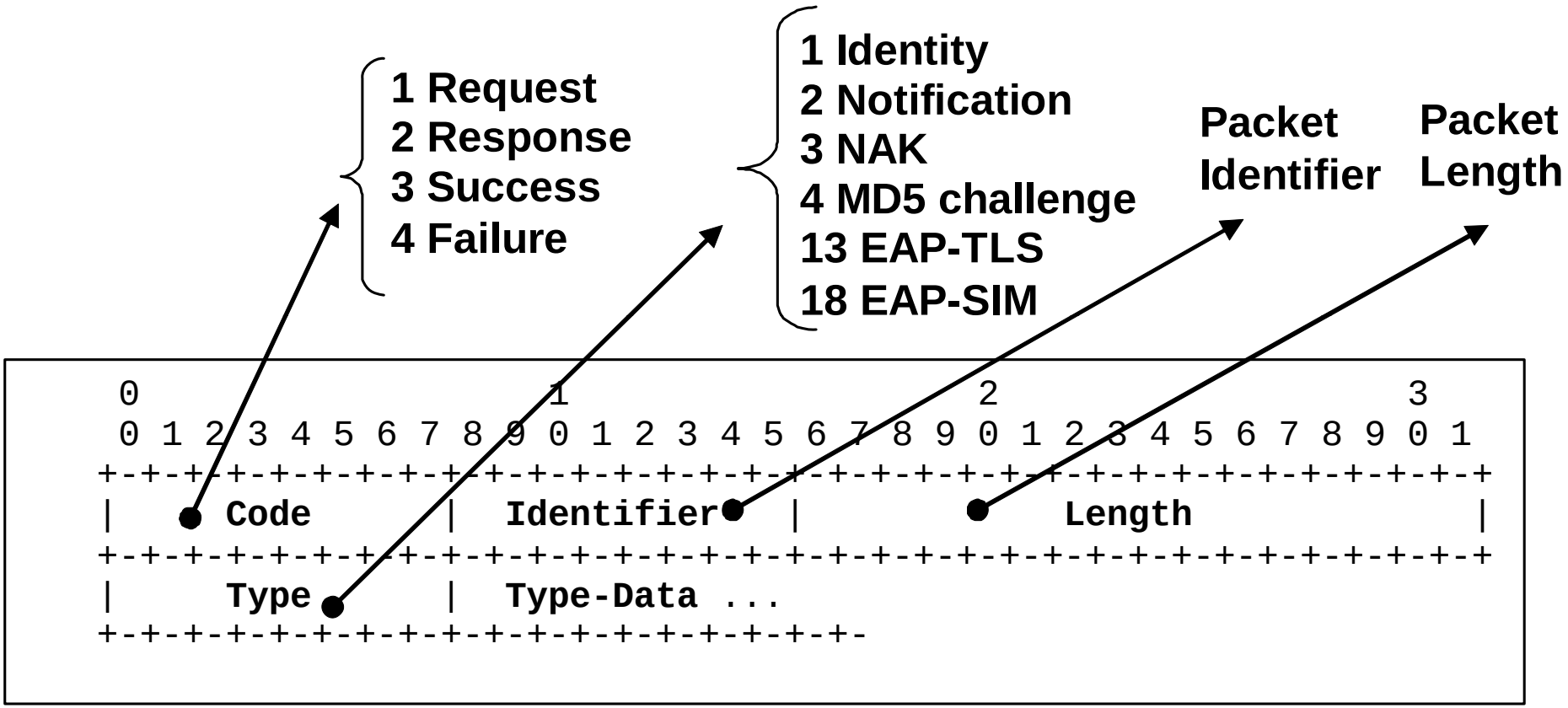
Javacard and donetcard simple application

EAP smart cards

EAP is an IETF standard

- **The Extensible Authentication Protocol (EAP) was introduced in 1999, in order to define a flexible authentication framework.**
  - ***EAP***, RFC 3748, "Extensible Authentication Protocol, (EAP)"
  - ***EAP-TLS***, RFC 2716, "PPP EAP TLS Authentication Protocol“
  - ***EAP-TLS***, RFC 5216, “The EAP-TLS Authentication Protocol”,
  - ***EAP-SIM***, RFC 4186, " Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) "
  - ***EAP-AKA***, RFC 4187, " Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA) “

EAP Message Format



An Esperanto for Access Control in IP infrastructures.

- ***Wireless LAN***
  - Wi-Fi, IEEE 802.1x
  - WiMAX mobile, IEEE 802.16e , PKM-EAP
- ***Wired LANs***
  - ETHERNET, IEEE 802.3
  - PPP, RFC 1661, “The Point-to-Point Protocol (PPP)”
- ***VPN (Virtual Private Network) technologies***
  - PPTP, RFC 2637 , " Point-to-Point Tunnelling Protocol "
  - L2TP, RFC 2661 , " Layer Two Tunnelling Protocol "
  - IKEv2, RFC 4306, "Internet Key Exchange Protocol“
- ***Authentication Server***
  - RADIUS, RFC 3559, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)”
  - DIAMETER, RFC 4072, "Diameter Extensible Authentication Protocol Application“
- ***Voice Over IP***
  - UMA, Unlicensed Mobile Access, http://www.umatechnology.org

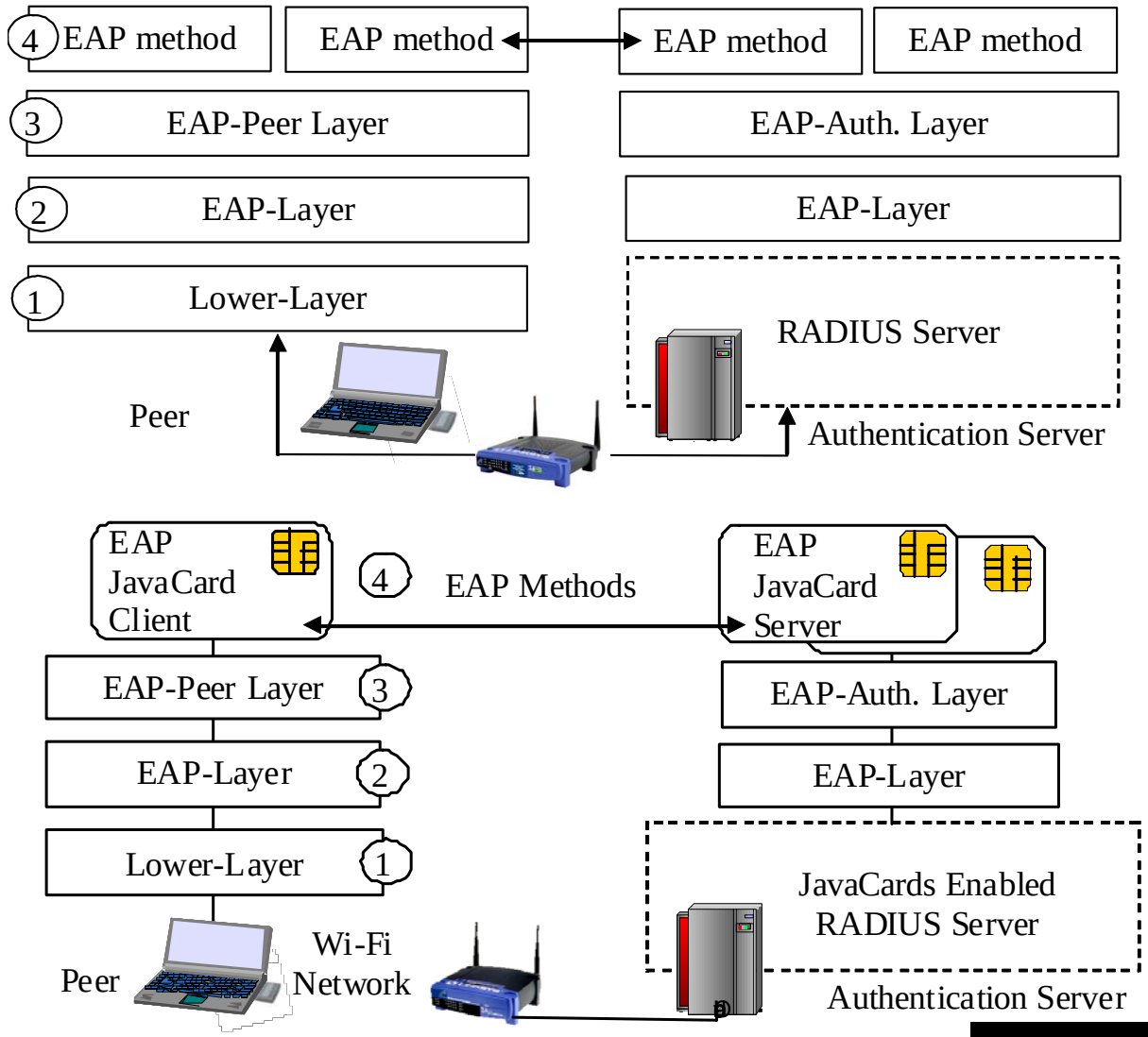
EAP components

- **According to RFC 3748, EAP implementations conceptually consist of the four following components:**
  - 1- The lower layer is responsible for transmitting and receiving EAP frames between the peer and authenticator.
  - 2- The EAP layer receives and transmits EAP packets via the lower layer, implements duplicate detection and retransmission, and delivers and receives EAP messages to and from EAP methods.
  - 3- EAP peer and authenticator layers. Based on the Code field, the EAP layer de-multiplexes incoming EAP packets to the EAP peer and authenticator layers.
  - 4- EAP methods implement the authentication algorithms, and receive and transmit EAP messages. **EAP methods can be implemented in smartcards.**

About EAP smartcards

Full Software Implementations

Partial Software Implementations + EAP smartcard



Internet (IETF) draft - EAP support in smartcard

- **The EAP smartcard is described in an internet draft, whose 18th version was issued in August 2009.**
- **It processes EAP requests or notifications and returns responses.**
- **Its logical interface, is a set of four services :**
  - **The IDENTITY service**. A smartcard may manage several network accounts.
  - **The NETWORK service**. EAP messages are processed by the smartcard. At the end of the authentication method, a session key is computed.
  - **The SECURITY service**. This service manages PIN codes (Personal Identification Number) that are needed for security purposes.
  - **The PERSONALIZATION service**. This service updates information stored in the smartcard.

- **Four components**
  - The ***EapEngine*** that implements the EAP core, and acts as a router that sends and receives packets to/from authentication methods.
    - It offers four services, Network interface, Identity management, Security management, Personalization management.
  - A ***Credential Object*** that stores information needed for method initialization.
  - One or more ***Methods*** that instantiate authentication scenari like EAP-TLS or EAP-AKA.
  - An ***Authentication Interface*** that defines all services offered by EAP methods.
    - The two main functions are ***Init(CredentialObject)*** and ***Process-Eap().***

A simple EAP Session

```
//(1) Select EAP application(AID= 11223344556601)
Select.request:  00 A4 04 00 07 11 22 33
                 44 55 66 01
Select.response: 90 00
```

Security Service

```
//(2) PIN code verification ("0000")
Verify.request:  A0 20 00 00 08 30 30 30 30
                 FF FF FF FF
Verify.response: 90 00
```

Identity Service

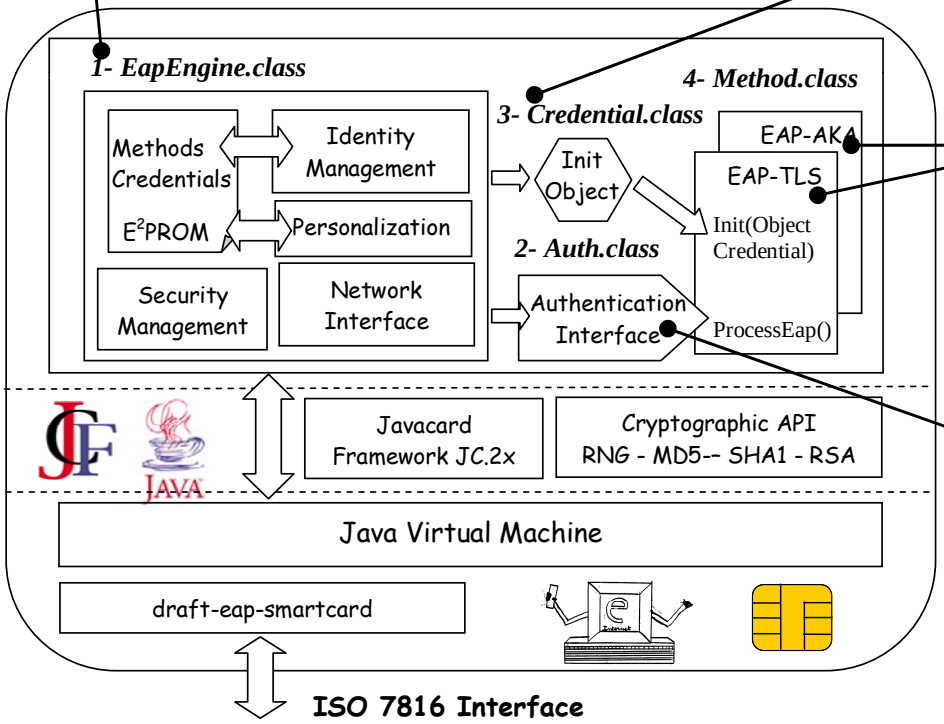
```
// (3) Get-Current-Identity()
Get-Current-Identity.request:   A0 18 00 00 00
Get-Current-Identity.response:  6C 04
Get-Current-Identity.request    A0 18 00 00 04
Get-Current-Identity.response:  61 62 63 64 90 00
                                   "abcd"

// (4) Get-Next-Identity()
Get-Next-Identity.request:  A0 17 00 01 00
Get-Next-Identity.response: 6C 04
Get-Next-Identity.request:  A0 17 00 01 04
Get-Next-Identity.response: 61 62 63 64 90 00
                               "abcd"

// (5) Get-Next-Identity("abcd")
Get-Next-Identity.request:  A0 17 00 01 00
Get-Next-Identity.response: 6C 04
Get-Next-Identity.request:  A0 17 00 01 04
Get-Next-Identity.response: 61 62 63 64 90 00
                               "abcd"

// (6) Set-Identity("abcd")
Set-Identity.request:  A0 16 00 80 04 61 62 63 64
Set-Identity.response: 90 00
```

Network Service

```
// (7) EAP Packets exchange
EAP-Packet.request:    A0 80 00 00 05 01 A4 00 05 01
EAP-Packet.response:   61 09
GetResponse.request:   A0 C0 00 00 09
GetResponse.response   02 A4 00 09 01 65 66 67 68
                       90 00                "efgh"

EAP-Packet.request:    A0 80 00 00 1A 01 A5 00 1A 07
                       14 83 D9 72 D1 01 F4 09 73 DE
                       C8 E3 20 68 B1 DE 58 16 41 EA
                       76
EAP-Packet.response:   61 1A
GetResponse.request:   A0 C0 00 00 1A
GetResponse.response:  02 A5 00 1A 07 14 BA 14 A8 09
                       C8 B0 D3 0E 55 DA D3 8A 00 93
                       E2 A4 80 BD DE 3B 90 00

EAP-Packet.request:    A0 80 00 00 04 03 A5 00 04
EAP-Packet.response    90 00

// (8) MSK reading
Get-MSK.request:       A0 A6 00 00 00
Get-MSK.response:      6C 40
GetResponse.request:   A0 A6 00 00 40
GetResponse.response:  CB B5 8C 1C 30 DB 13 C0 9C 97
                       6B B2 12 62 BA 46 26 F1 05 AB
                       F4 36 4A B4 52 6D 39 FA 05 3F
                       D4 D3 A9 0A 70 79 9B 22 84 89
                       AA 0E EB 8E 2F A8 81 A4 C2 3A
                       2A F7 C6 08 2A BC 98 53 18 90
                       E2 24 0E 04
                       90 00
```

Open EAPSmartcard.java

1- The EapEngine manages several methods and/or multiple instances of the same one. It implements the EAP core, and acts as a router that sends and receives packets to/from authentication methods. At the end of authentication process, each method computes a master cryptographic key (*AAA Key*) which is read by the terminal operating system.



3- The Credential objects, are used by to methods, and encapsulate all information required for processing a given authentication scenario.

4- The Methods are associated to various authentication scenari. Once initialized, the selected method analyses each incoming EAP request and delivers corresponding response.

2- The Authentication interface defines all mandatory services in EAP methods, in order to collaborate with the EapEngine. The two main functions are *Init*() and *Process-Eap*().

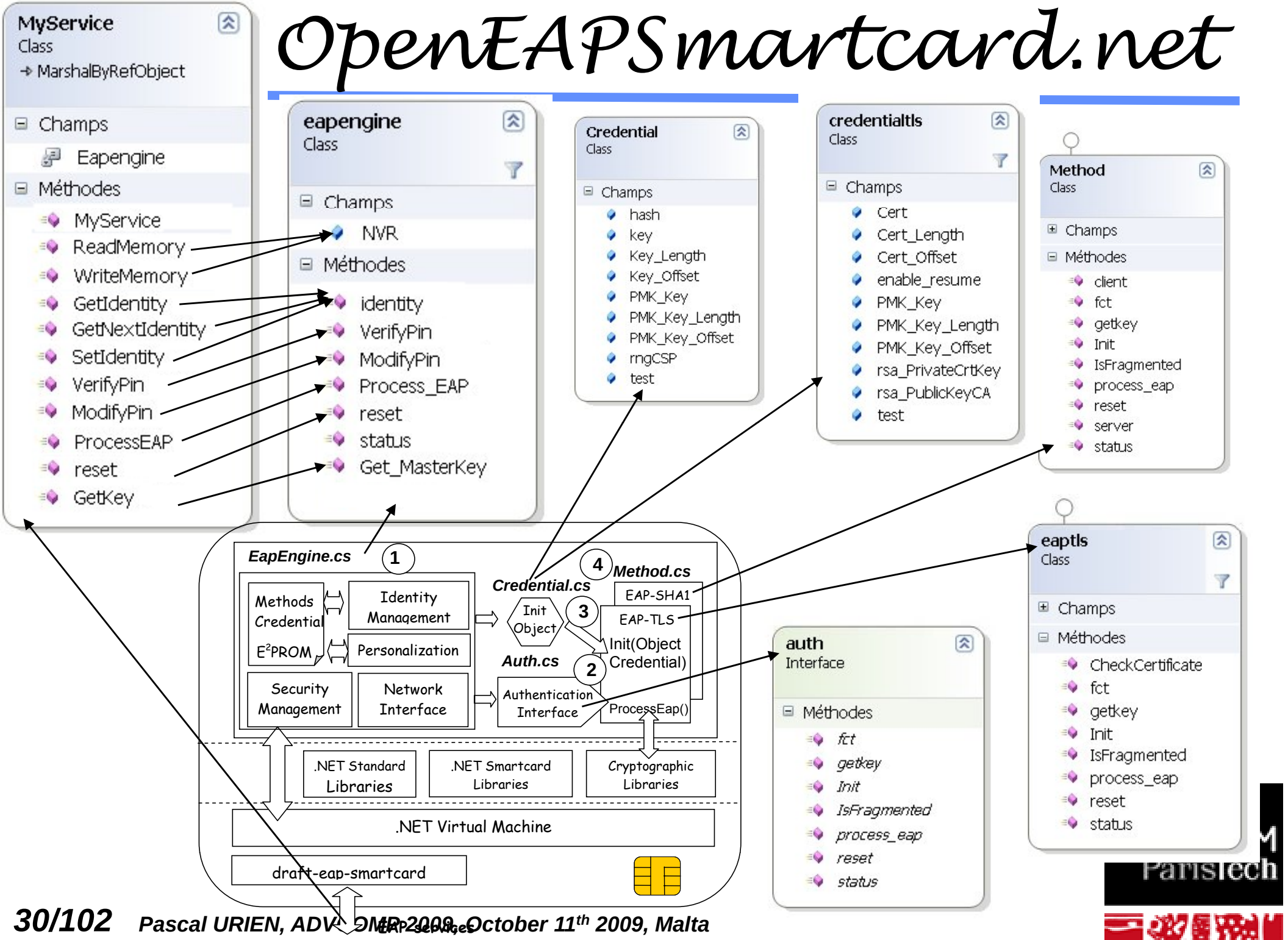
- First initializes method and returns an Authentication interface;
- Second processes incoming EAP packets. Methods may provide additional facilities dedicated to performances evaluations.

OpenEAPSmartcard.java: the Authentication Interface

Interface auth	
void	**fct**(javacard.framework.APDU apdu, byte[] in, short inlength) Method functions apdu: incoming APDU in: buffer associated to the incoming APDU inlength: P3 value
byte[]	**Get_Fct_Buffer**() Returns a function buffer
short	**Get_Fct_Length**() Returns a function buffer length
short	**Get_Fct_Offset**() Returns a function buffer offset
byte[]	**Get_Out_Buffer**() Returns the response buffer
short	**Get_Out_Length**() Returns the response buffer length
short	**Get_Out_Offset**() Returns the response buffer offset
auth	**Init**(java.lang.Object credentials) Method Initialization
boolean	**IsFragmented**() Fragmentation in progress
boolean	**IsLongFct**() Indicates that the response of a function is stored in a private buffer
boolean	**IsLongResponse**() Indicates that the response of the method is stored in a private buffer
short	**process_eap**(byte[] in, short inlength) Method Processing in: incoming APDU buffer inlength: length of the incoming APDU Returns -length of the response -negative value if an error occurred
void	**reset**() Resets the method
short	**status**()Gets the method status

TELECOM ParisTech

OpenEAPSmartcard.net



```
done = service.VerifyPin(s2b("0000"));
done = service.WriteMemory(-2, a2b("80"));
Console.WriteLine("WRITE (Offset=-2, value=x80)");

bin = service.ReadMemory(-3, 3);
Console.WriteLine("READ(Offset=-3, Length=3): " + b2s(bin));
for (i = 0; i < 7; i++)
{ id = service.GetNextIdentity();
  Console.WriteLine("GETNEXT: " + b2a(id));}

id = service.SetIdentity(s2b("marc"));
Console.WriteLine("SET: " + b2a(id)) ;

id = service.GetIdentity()          ;
Console.WriteLine("GET: " + b2a(id));
```

Microsoft .net

Console.exe

Server.exe

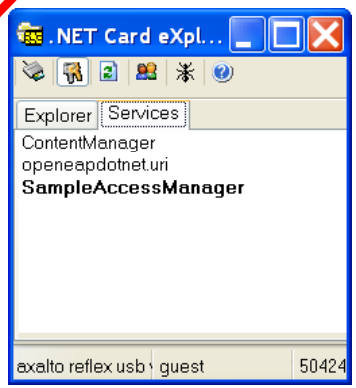
OpenEAP Smartcard.NET

```
//=======================================
//            Test EAP-SHA1
//=======================================
eapreq = a2b("01 A5 0005 01");
Console.WriteLine("REQ : " + b2s(eapreq));
eapresp = service.ProcessEAP(false, eapreq);
Console.WriteLine("RESP: " + b2s(eapresp));

eapreq = service.ProcessEAP(false, eapresp);
Console.WriteLine("REQ : " + b2s(eapreq));

eapresp = service.ProcessEAP(false, eapreq);
Console.WriteLine("RESP: " + b2s(eapresp));
mkey = service.GetKey();
Console.WriteLine("MSK_Client " + b2s(mkey));

eapreq = service.ProcessEAP(false, eapresp);
Console.WriteLine("REQ: " + b2s(eapreq));
mkey =  service.GetKey();
Console.WriteLine("MSK_Server " + b2s(mkey));

////////////////////////////////////////////
//              Test eap-tls
////////////////////////////////////////////
id = service.SetIdentity(s2b("abc"));
Console.WriteLine("SET: " + b2a(id));

eapreq = a2b(pkt1);
Console.WriteLine("REQ : " + b2s(eapreq));
eapresp = service.ProcessEAP(false, eapreq);
Console.WriteLine("RESP: " + b2s(eapresp)) ;
```

file:///D:/Documents and Settings/urien/Mes docume

```
WRITE (Offset=-2, value=x80)
READ(Offset=-3, Length=3): ff8054
GETNEXT: a
GETNEXT: test
GETNEXT: c
GETNEXT: marc
GETNEXT: abc
GETNEXT: aaa
GETNEXT: a
SET: marc
GET: marc
REQ : 01a5000501
RESP: 02a50009016d617263
REQ : 01a6001a071483d972d101f40973dec8e32068b1de581641ea76
RESP: 02a6001a0714f23eedcfcf2eca7d5390435769d625a35624612e
MSK_Client f98d3ad3b6de88f21679523837fba230825c19220de75d5e01b3c116931593206214c
187e73fe159cff5c821de0bf47b4f7f3f4a07c9dd606b35aecd42c4b48d
REQ: 03a60004
MSK_Server f98d3ad3b6de88f21679523837fba230825c19220de75d5e01b3c116931593206214c
187e73fe159cff5c821de0bf47b4f7f3f4a07c9dd606b35aecd42c4b48d
SET: abcd
REQ : 011400060d20
RESP: 021400500d800000004616030100410100003d03013faa2b6a08bdd285b43d1f3bc9715fc9
f85fc453fe58f3a9e07ff397cd6539220000160004000500a0009006400620003000600130012 00
630100
```

Standalone dialog between EAP Client and Server entities



.NET APIs

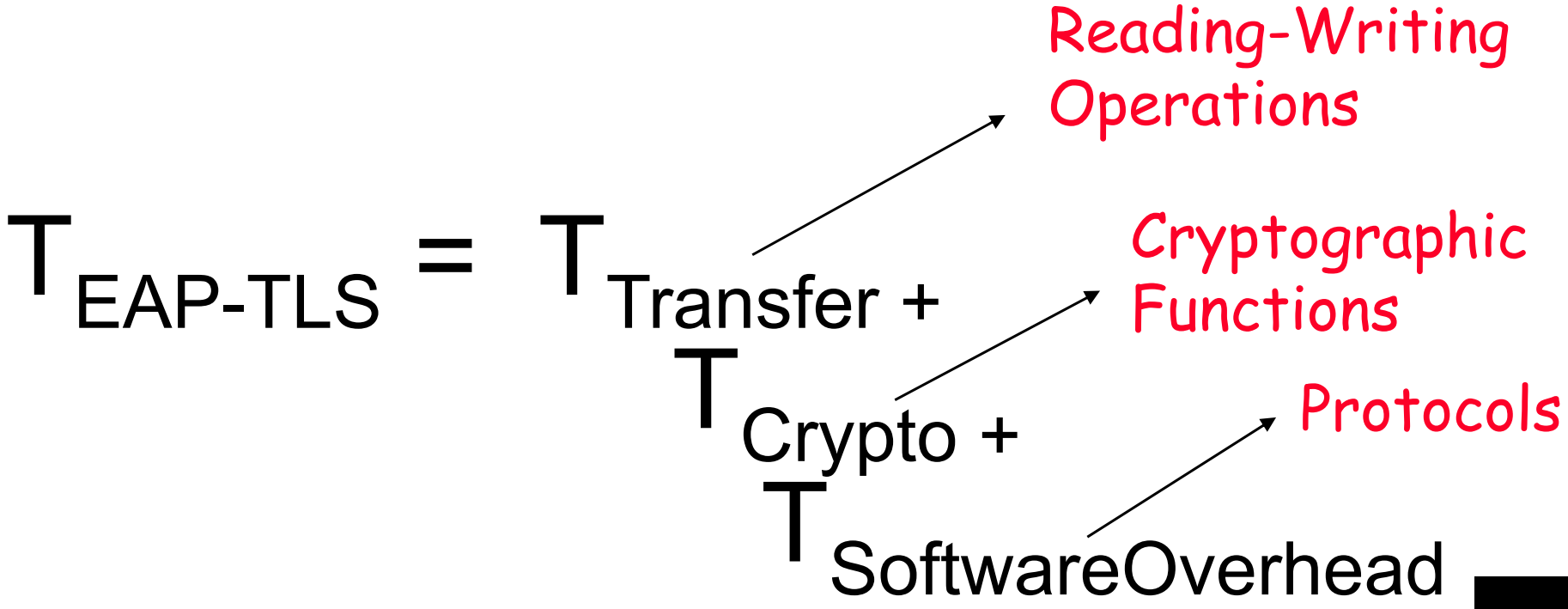
TELECOM ParisTech

OpenEapSmartcard & Performances Issues

Are you serious ?

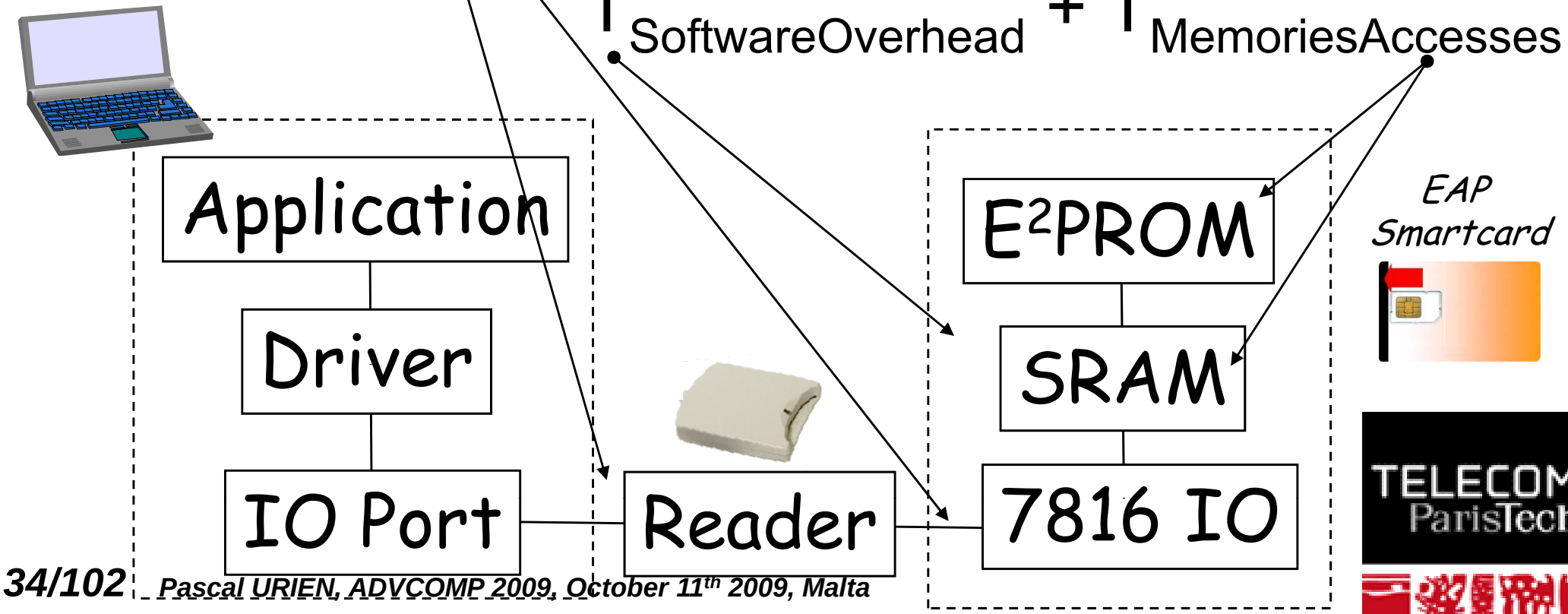
Performances Analysis

- **3xT analysis**
  - Data Transfer
  - Cryptographic Operations
  - Software Overhead

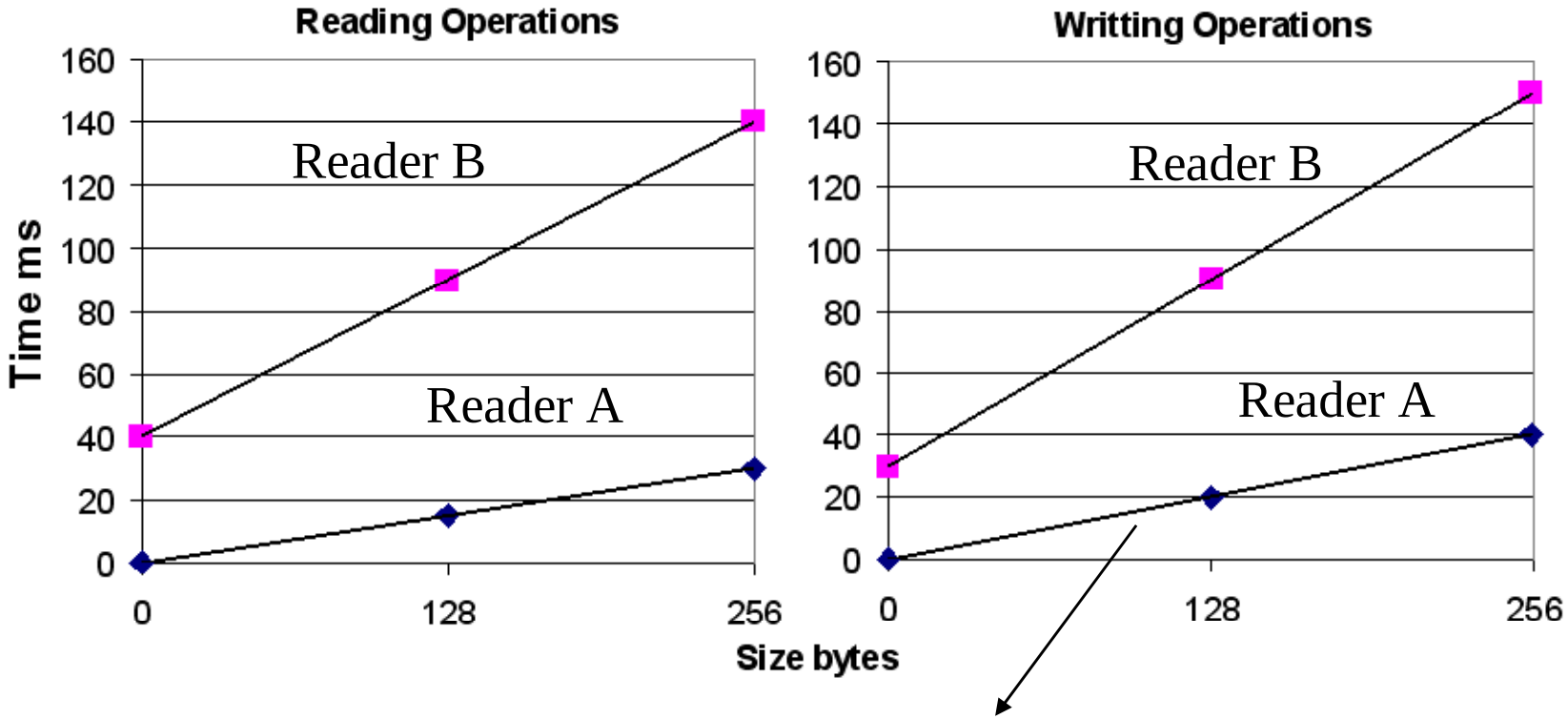


- **In protocols dealing with X.509 certificates like EAP-TLS, several kilobytes (typically 3600 bytes) of data are sent/received to/from the smartcard. Due to the lack of RAM memory, these information are written or read in the non-volatile memory (E^2PROM, flash memory,…)**

$$T_{Transfer} = T_{TransferReader} + T_{TransferSmartcard} + T_{SoftwareOverhead} + T_{MemoriesAccesses}$$



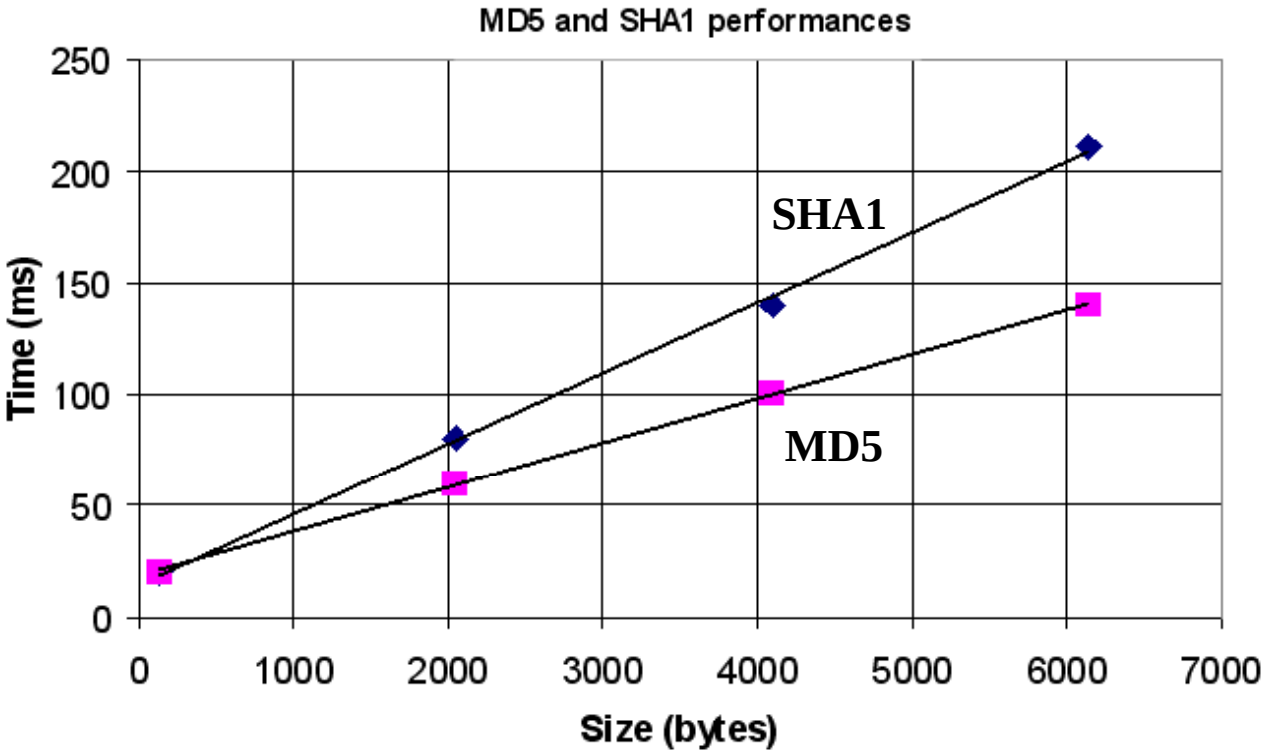
Examples of Reading-Writing Operations



0,15 ms/byte (50 Kbits/s)

$T_{Transfer}$ = 2600 x 0,15 = 390 ms

Cryptographic Operations



$T_{Digest} = \frac{1}{2}(T_{MD5} + T_{SHA1})$
=11,8 ms/bloc)

Private Key Encryption	Public Key Decryption	Public Key Encryption	Private Key Decryption
750ms	70ms	60ms	760ms

$T_{RSA} = T_{PubKD} + T_{PubKE} + T_{PrivKD} = 890$ ms

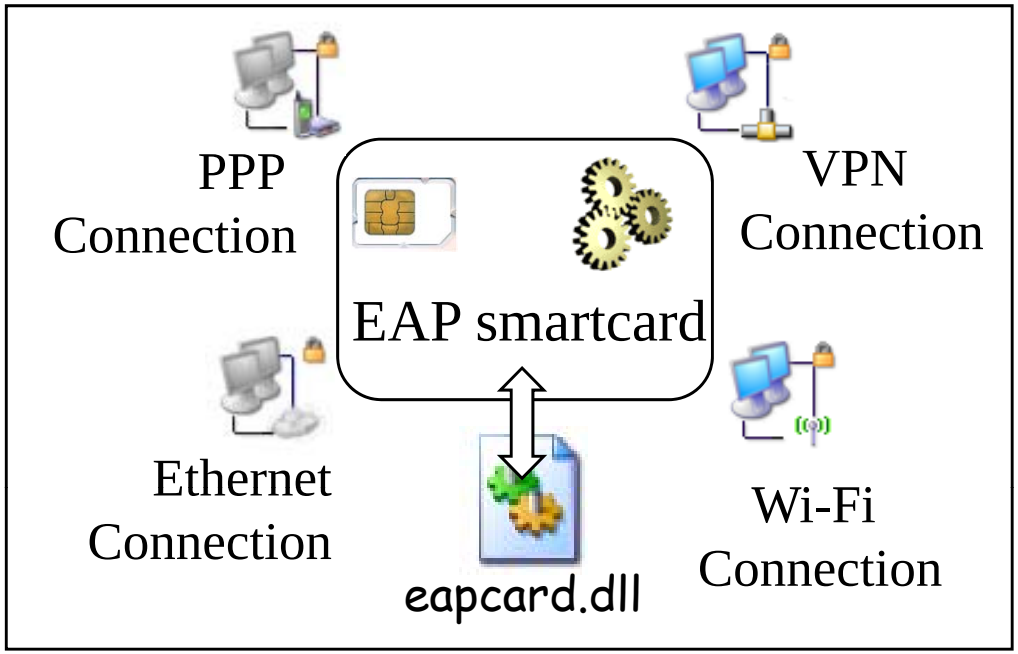
$$T_{Crypto} = T_{RSA} + 532 \times T_{Digest} = 1850 \text{ ms}$$

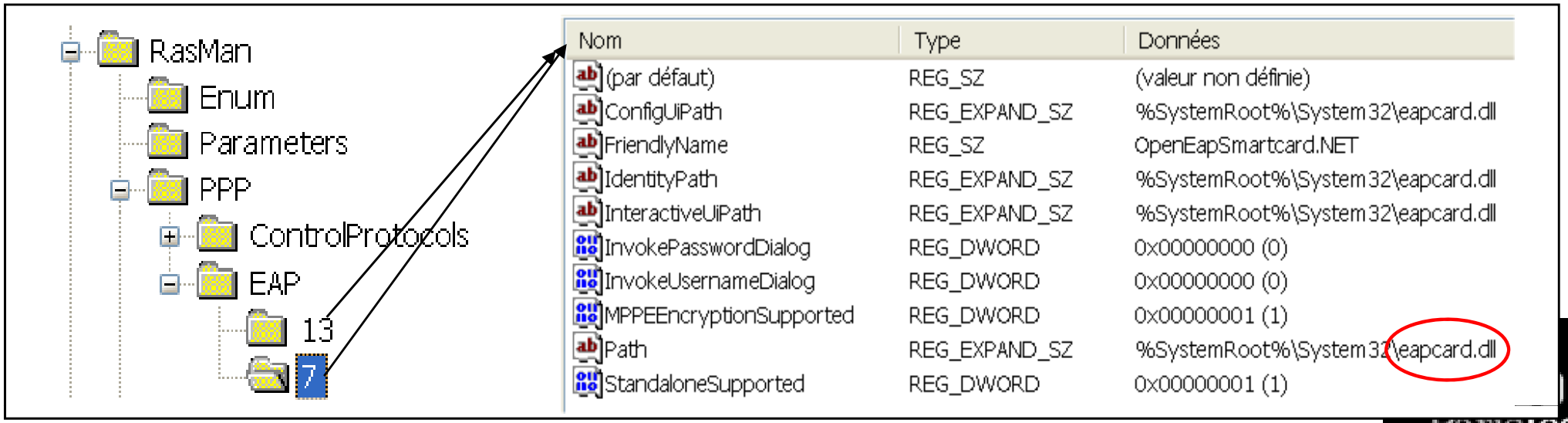
Summary of performances

- $T_{EAP-TLS}$ = 5300 ms
  - $T_{Other} = T_{EAP-TLS} - T_{Transfer} - T_{Crypto}$
    $= 5300–400–1850$
    $= 3050ms$
- **As a conclusion**
  - 0,4s (08%) in data exchange with the docking station.
  - 1,9s (35%) in cryptographic APIs,
  - 3,0s (57%) in other operations realized by Java software.

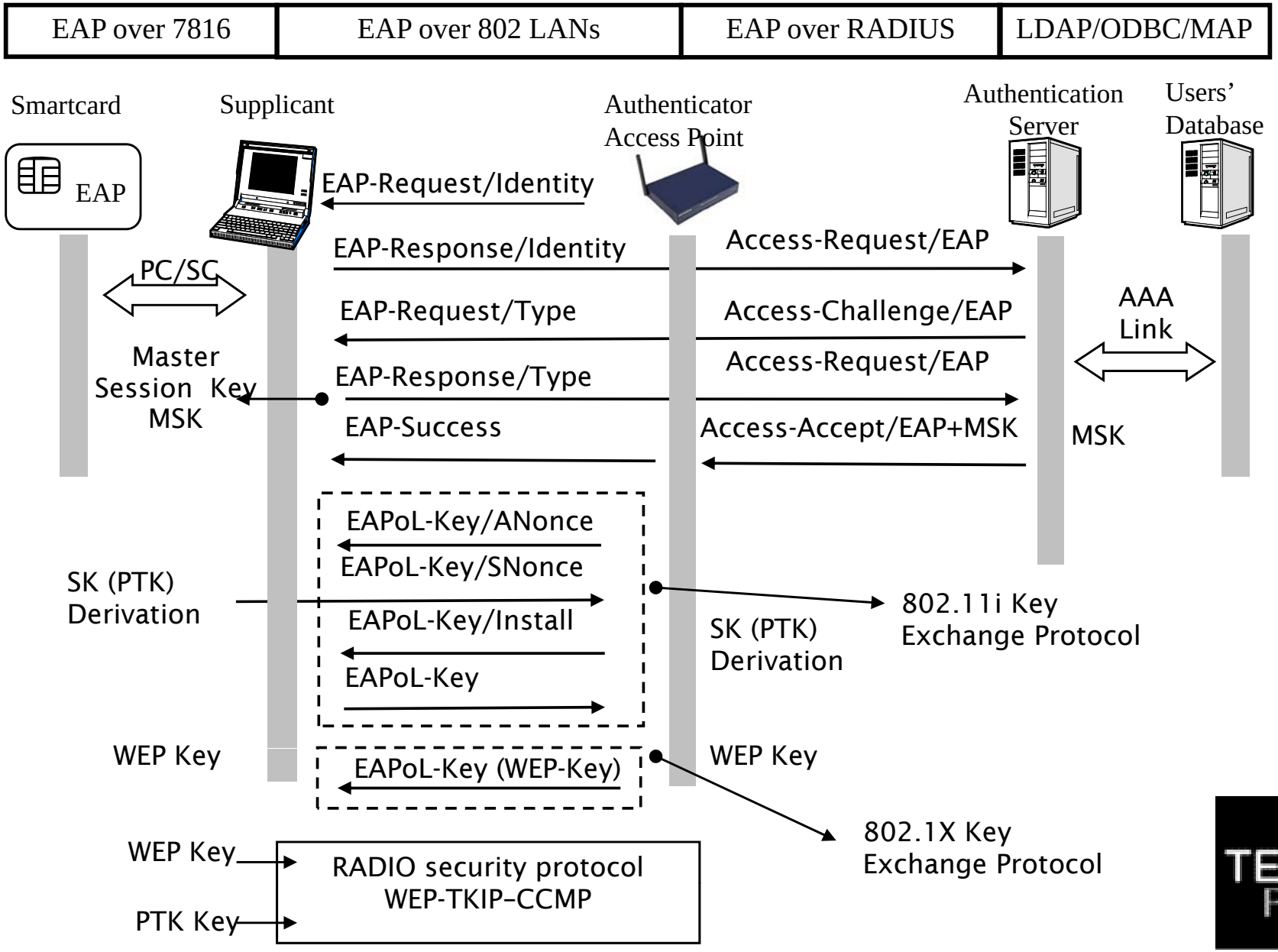
Use Case

EAP in Windows Operating System





Wi-Fi Security Model



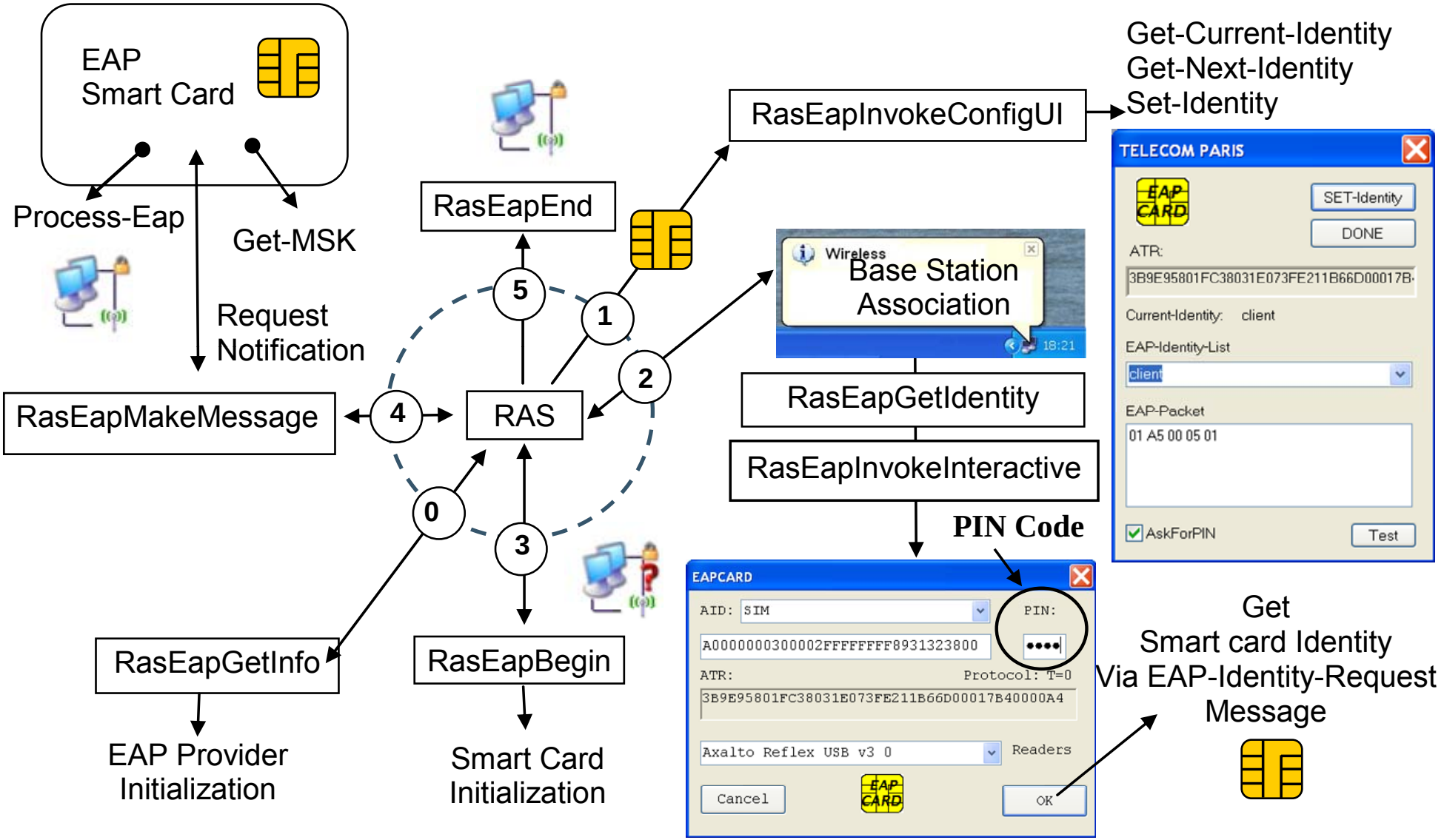
MS Platform EAP - Management

- **One EAP DLL per authentication type. The same DLL may support more than one authentication type.**
- **DLL CORE**
  - Ras-Eap-GetInfo
  - Ras-Eap-Initialize
  - Ras-Eap-Begin
  - Ras-Eap-MakeMessage
  - Ras-Eap-End
  - Ras-Eap-FreeMemory

A standard is needed for smartcard interface

- **Vendor Specific Services, may be implemented in an other DLL.**
  - Ras-Eap-InvokeConfigUI
    - User parameter setup (pin code, …)
  - Ras-Eap-InvokeInteractiveUI
    - Additive protocol information

User's Experience



Demonstration 2.

Wi-Fi Environment

Electronics Identity Packing

Friendly Name

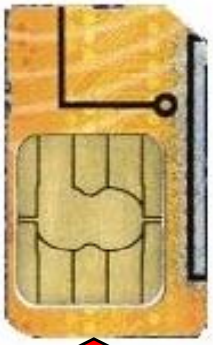
EAP-ID

CA's Certificate

User's Certificate

```
\MakeId>eaptools.exe enst pascal.urien@enst.fr id.bin 1024 ca.der 1024

identity:..............enst
eapid:.................pascal.urien@enst.fr
output file:...........id.bin
CA key size(bits):.....1024
CA_certificate:........ca.der
Client key size(bits):...1024
Client certificate:.....cert.der
Client private key file:.key.der

D:\Wi-Fi\javacard\OPENEAP\tlstools\MakeId>bin2script id.bin id.txt 0
Infile:...id.bin
Outfile:..id.txt
Adr:.....0
D:\Wi-Fi\javacard\OPENEAP\tlstools\MakeId>
```

Smart Card Personalization



Script Generation

Fichier Edition Format Affichage ?

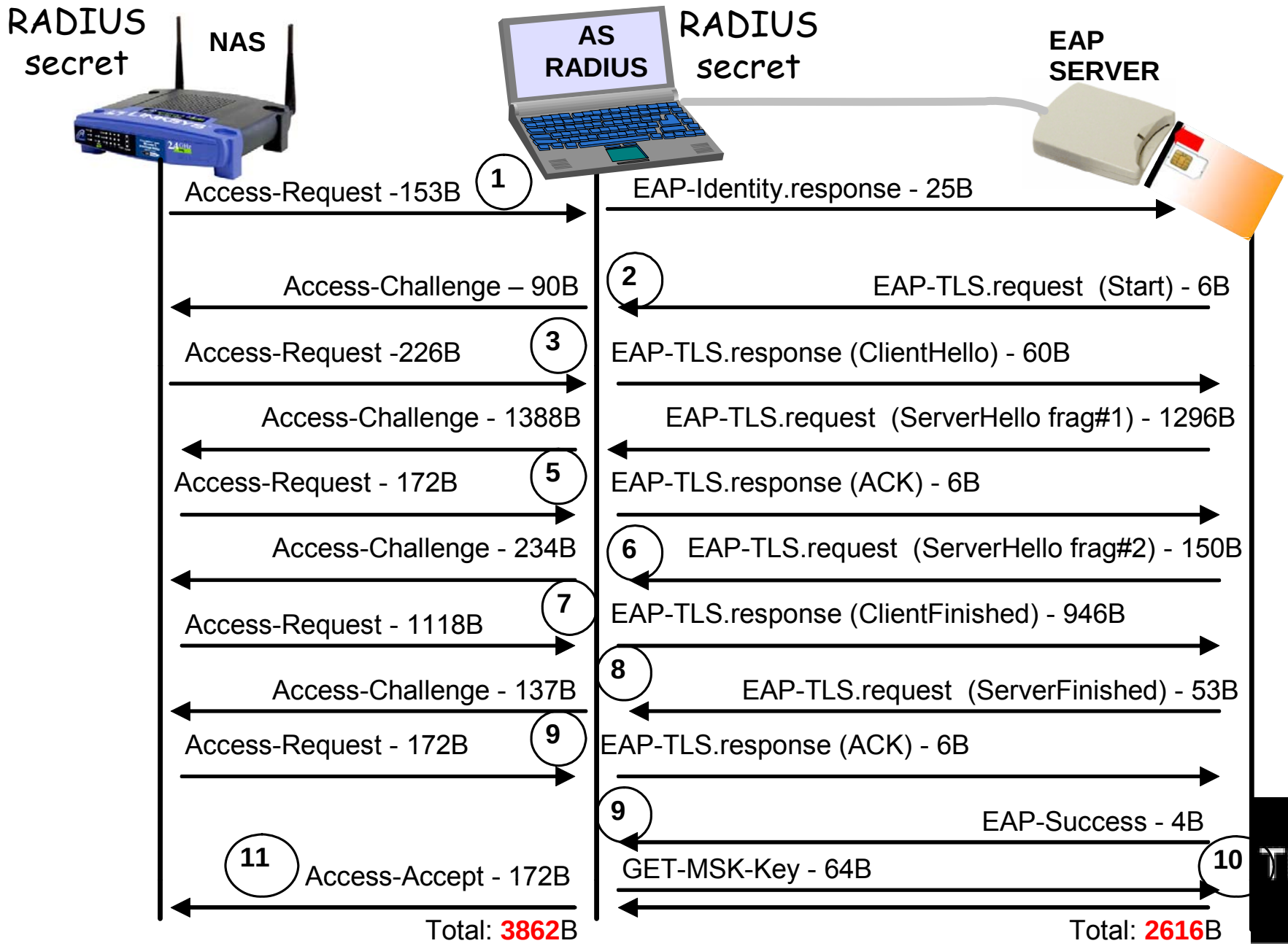
```
// select EAP Application                                    Application name
00A40400 10 A0 00 00 00 30 00 02 FF FF FF FF 89 31 32 38 00

// Verify Operator PIN = 00000000                            Administrator PIN code
00 20 00 01 08 30 30 30 30 30 30 30 30                       = "00000000"
//90 00

00 D0 00 00 FF 0006636C69656E74147061736163C2E757269656E40656E73742E66720D04010
00 D0 00 FF FF CA181D33CFCB3E8A162F7CC1A90C9DF02565AD118ACA5C6C408B730A3DCB143B9
00 D0 01 FE FF 6C96B7B848AE89B5B8A6BB3BB6401DDC69911BD4128452AAC0A81A7ECAA0AF6E0
00 D0 02 FD FF 0B496C6544654672616E6365310E300C060355040713055061726973311030 0E0
00 D0 03 FC 93 0D06092A864886F70D0101050500038181004075 7A6B07426A14710E03167A1B8

// Set Max EAP Size=1300                                     Electronic ID data
A082C600 02 05 14

// Set-Identity (client)                                     Activation of the eID identfied by
A0 16 00 80 06 63 6C 69 65 6E 74                             the friendly name "client"
```

Smart card enabled RADIUS server

Classical RADIUS Server

- **RFC 2865, "Remote Authentication Dial In User Service (RADIUS)", 2000**
  - Two entities
    - The *Network Access Server* (NAS) .
    - The *Authentication Server* (AS).
  - In a telephony context the NAS is running in a *Point Of Presence* (POP) , while in Wi-Fi applications it runs in *Access Points* (AP), and blocks all frames that are sent/received by unauthenticated users.
- **RADIUS works over an UDP/IP stack, and therefore RADIUS messages are routable through the Internet.**
- **Mainly four types of messages**
  - Access-Request, Access-Challenge, Access-Reject, Access-Success
- **RADIUS in IEEE 802.1x context**
  - Clients (called supplicants) are authenticated before allocations of their IP addresses.
  - Authentication messages (EAP) are exchanged between user and NAS over PPP or LAN frames. These messages are encapsulated in RADIUS packets exchanged between NAS and AS entities.
- **RADIUS security is based on a shared secret (the *RADIUS secret*) between the NAS and the AS**
  - Cryptographic procedures use MD5 and HMAC-MD5

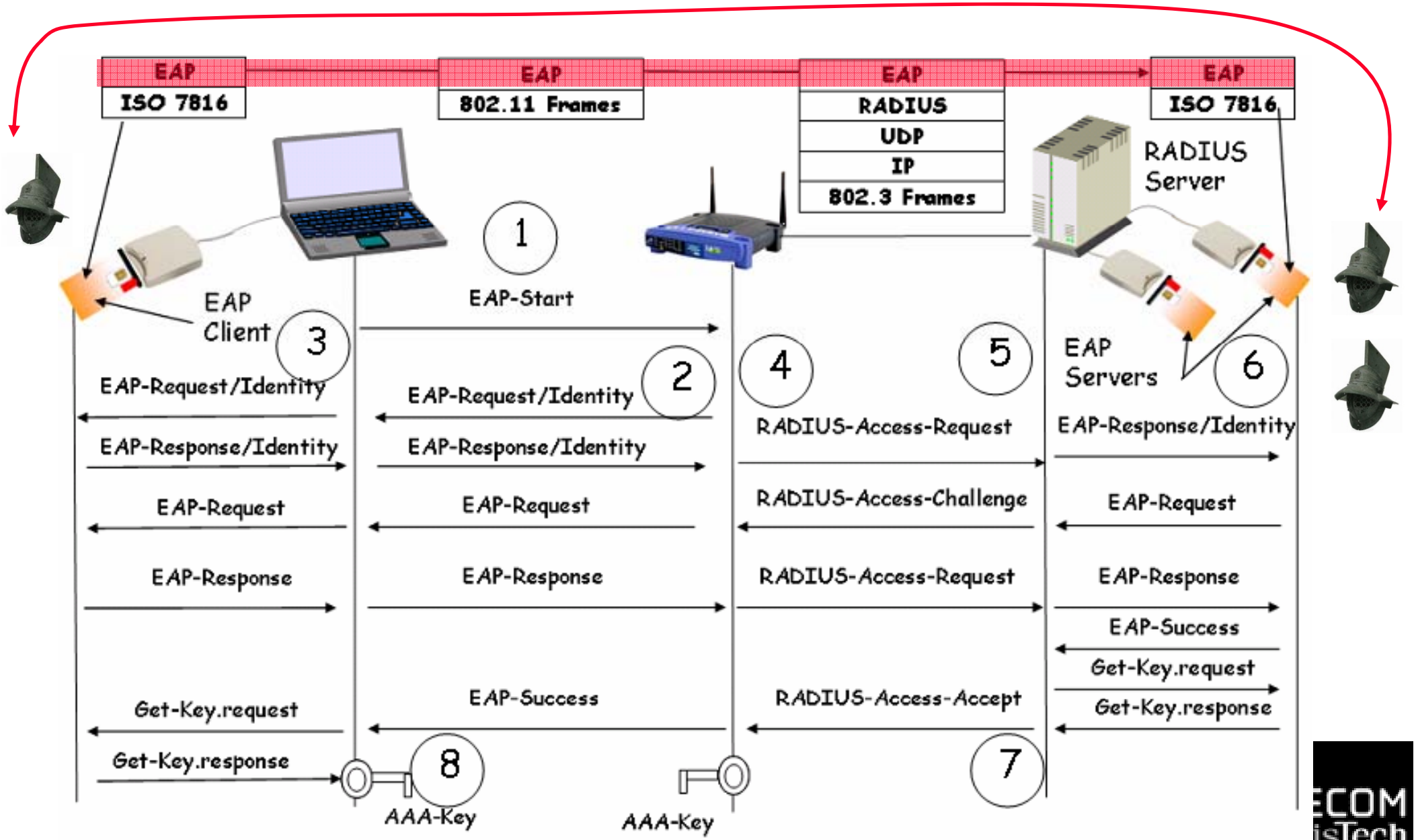
Overview of RADIUS Sessions



Benefits of smartcard enabled RADIUS server

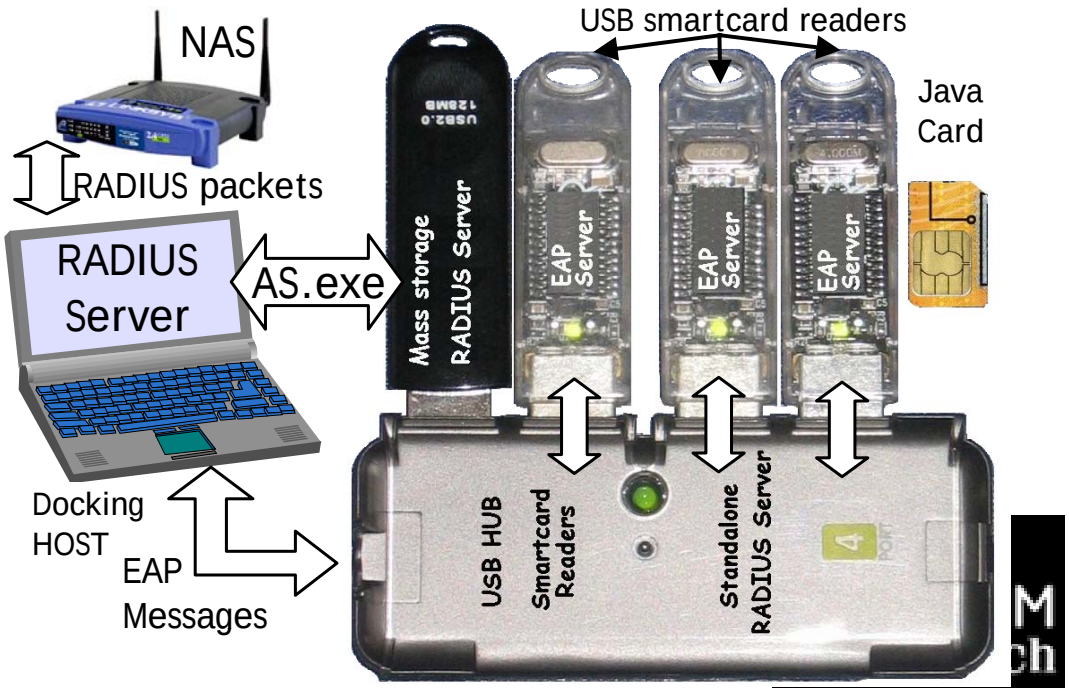
- **We believe that EAP server smartcards enhance the RADIUS security, specially in EAP-TLS case for the following reasons,**
  - The server private key is securely stored and used by the smartcard.
  - The client's certificate is autonomously checked by the EAP server.
  - If the EAP client also runs in a smartcard, the EAP session is then fully processed by a couple of tamper resistant devices, working as ***Secure Access Module* (SAM),** a classical paradigm deployed in highly trusted architectures.

Secure Access Module (SAM) concept

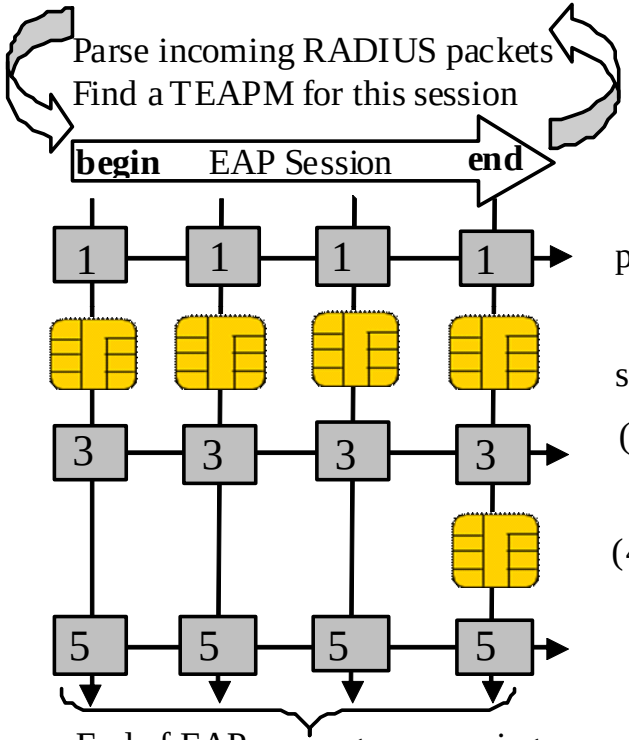


Smartcard Enabled RADIUS server

- **Two components**
- **A RADIUS authentication server, running in a docking host.**
  - It offers the Ethernet connectivity and IP services. It receives and sends RADIUS packets over UDP sockets.
  - It builds or parses RADIUS messages, handles the RADIUS secret, checks or generates authentication attributes. EAP messages, transported by RADIUS payloads are forwarded to smartcards, running EAP-Servers.
- **EAP servers.**
  - Each smartcard runs an EAP-server, and fully handles an EAP-TLS authentication procedure.
  - Each component stores an unique X509 certificate and its associated RSA private key.
  - It computes EAP responses and produces EAP requests.
  - At the end of a successful authentication session, a MSK is calculated and delivered to the RADIUS entity.
- **EAP sessions**
  - An EAP session is a set of messages associated to an unique **Session-Id** value, which is obtained by the concatenation of two values, the NAS-Identifier (RADIUS attribute n°32) and the Calling-Station-Id (the client's MAC address, corresponding to RADIUS attribute n°31) as follows:
  - Session-Id = NAS-Identifier | Calling-Station-Id



Implementation Details



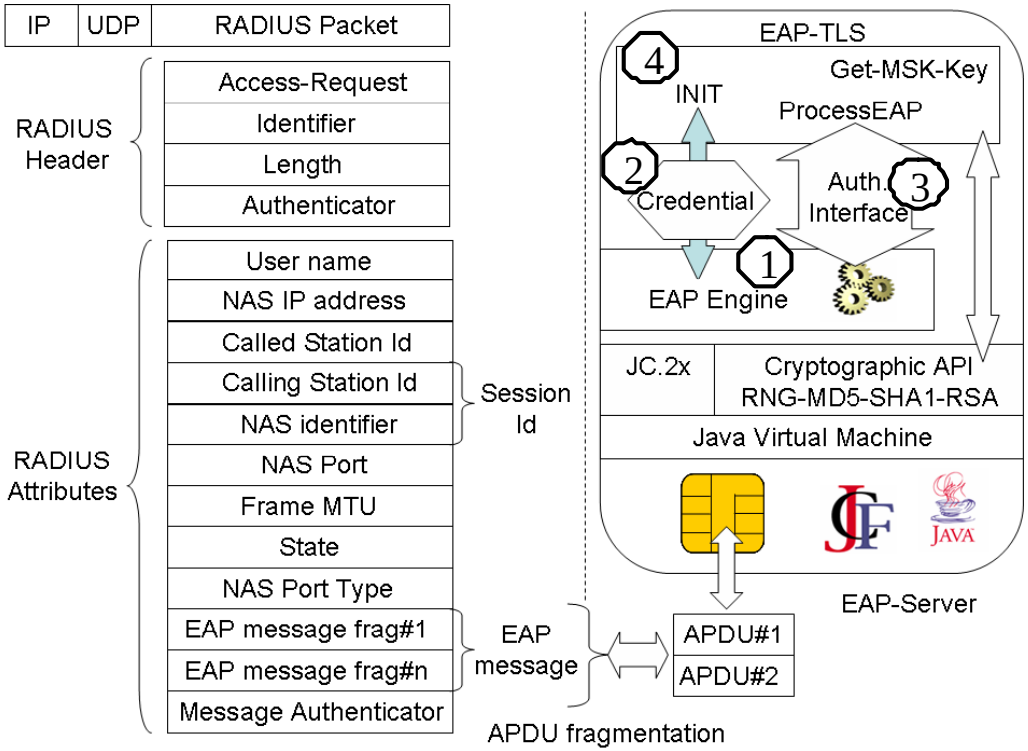
(1) Forward EAP packet to smartcard

(2) Wait for smartcard response

(3) Build RADIUS packet

(4) Get MSK key

(5) Transmit RADIUS packet



Scalability, privation versus the Erlang B law

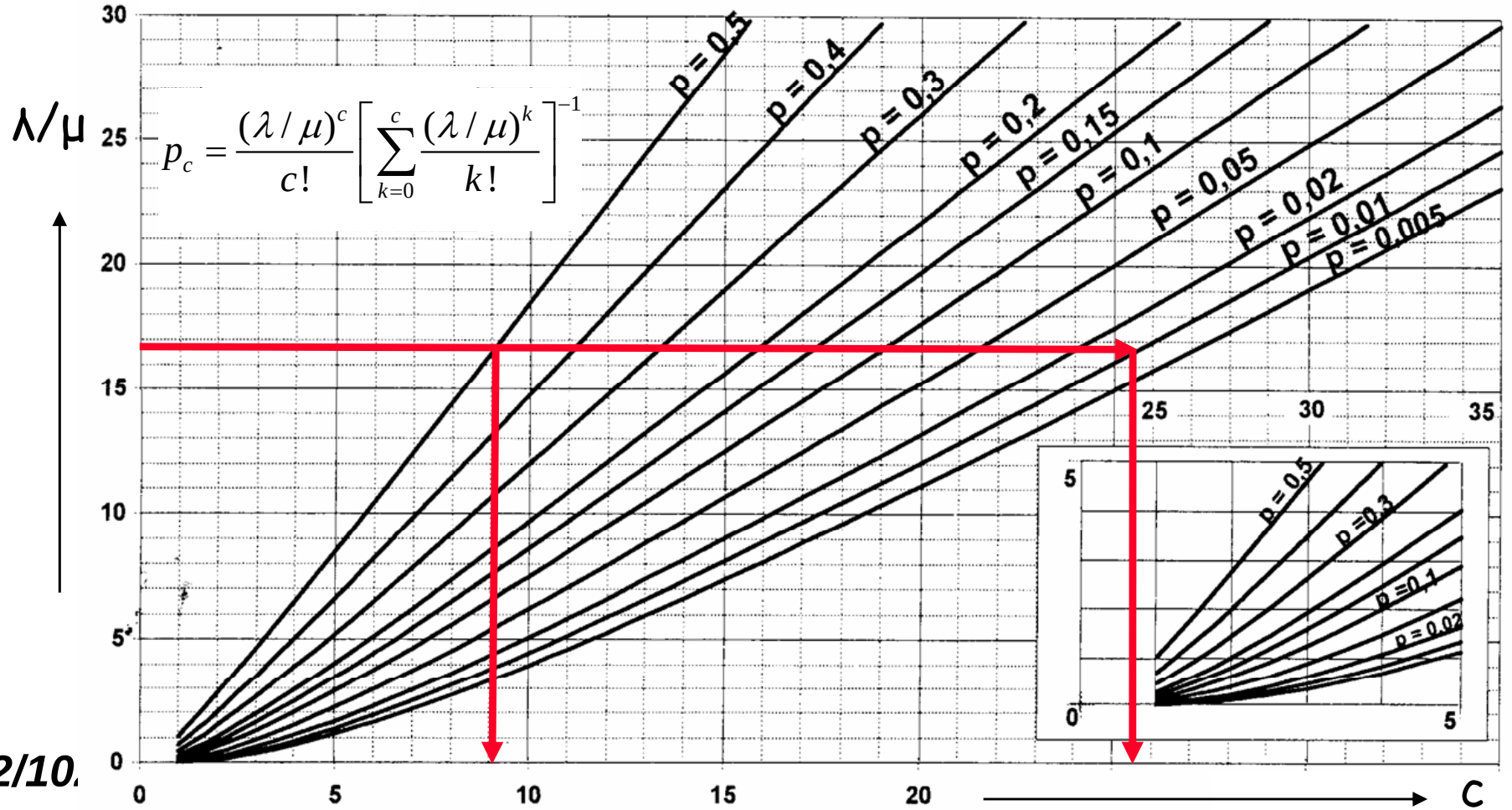
Pc is the probability of blocking (e.g. a RADIUS packet is silently discarded),

c is the number of EAP servers,

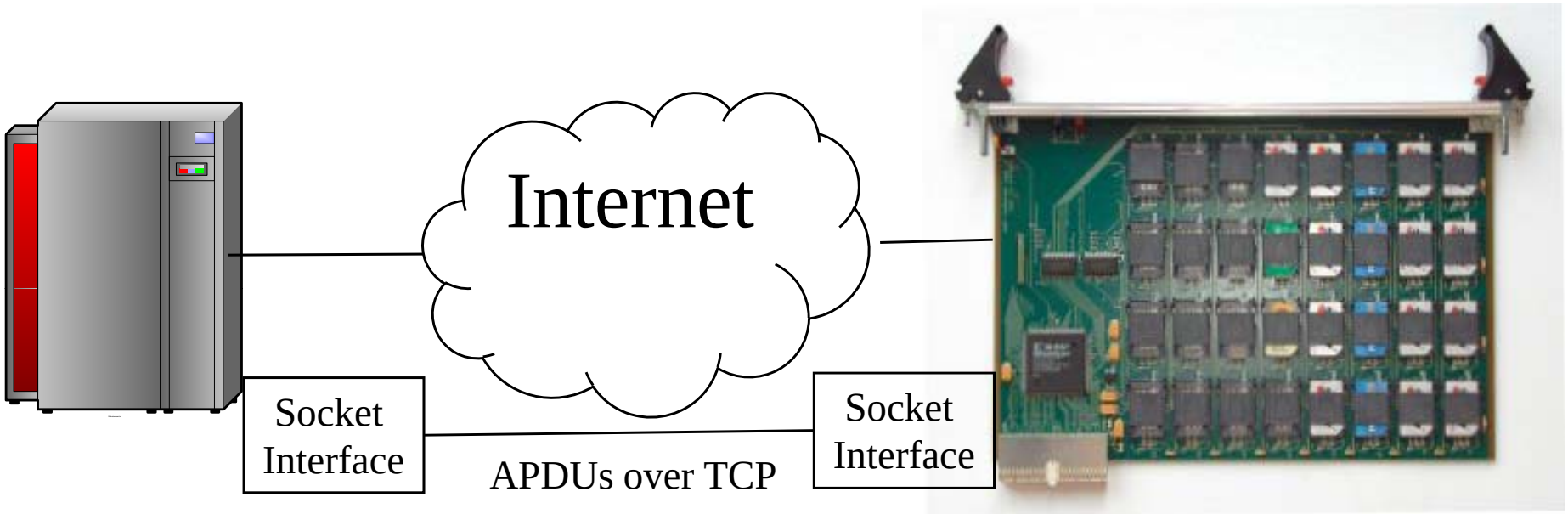
λ is the rate of authentication sessions, and

1/μ the mean time of an authentication session (10s = 5s + 5s)

Let's assume a network with 1000 users, authenticated every 10mn, then λ = 6x1000/3600=1,7 and so λ/μ = 60,000/3600 = 16,7. The probability of blocking (pc) is about 50% with 9 smartcards (c = 9) and only 1% with 21 smartcards (c = 21).



Work in progress : smartcards grids

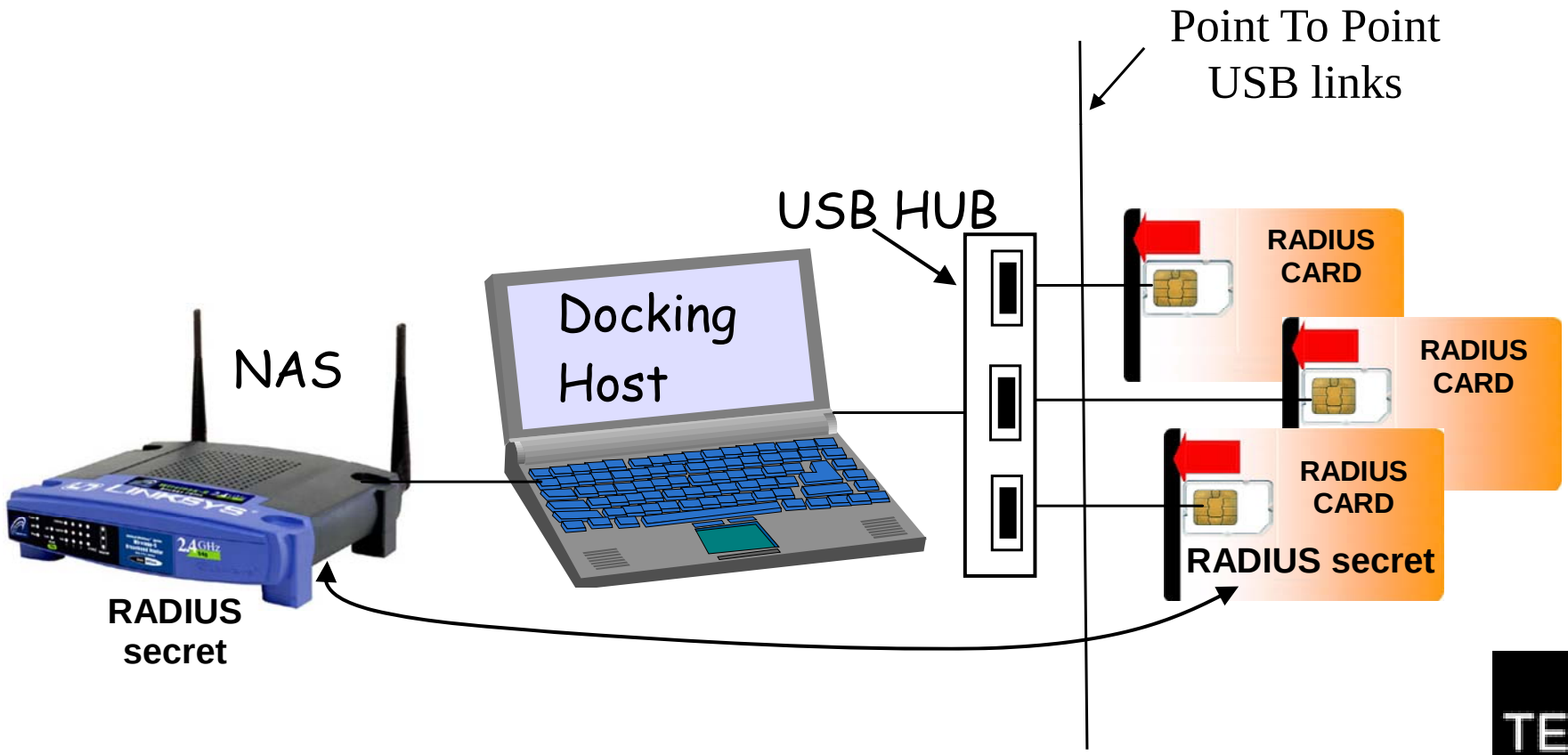


RADIUS Server

EAP-TLS smartcards grid

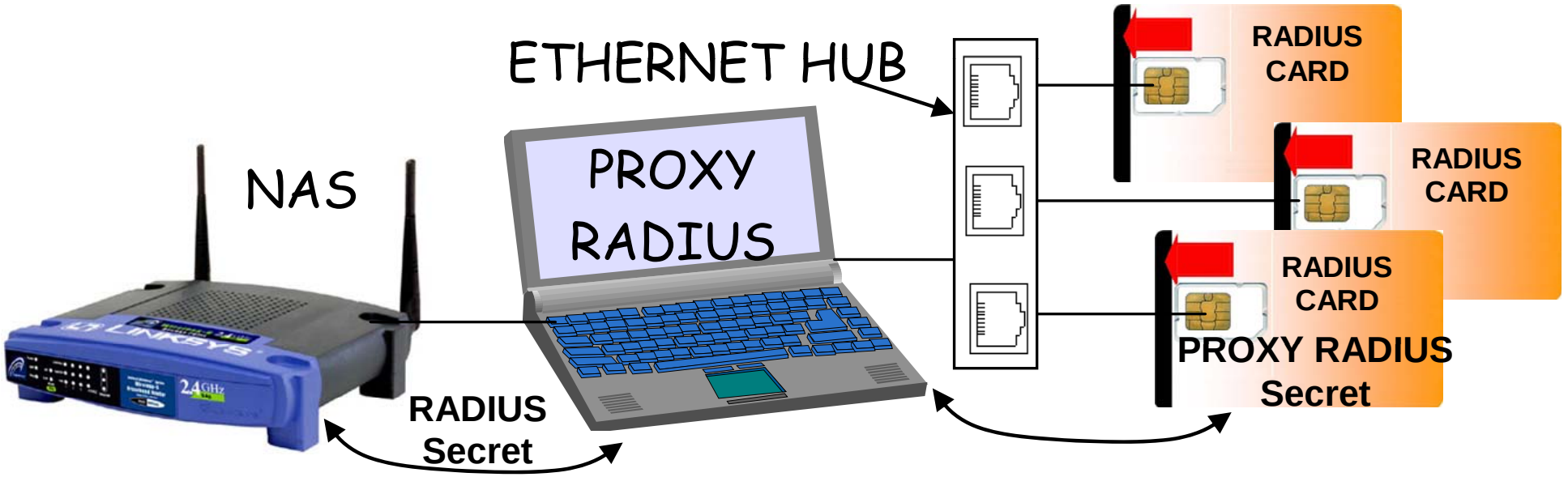
Ideas 1/3 - Multiple RADIUS smartcards

- The docking station handles timeout management (packets retransmission, sessions release…) and physical accesses to Ethernet network.



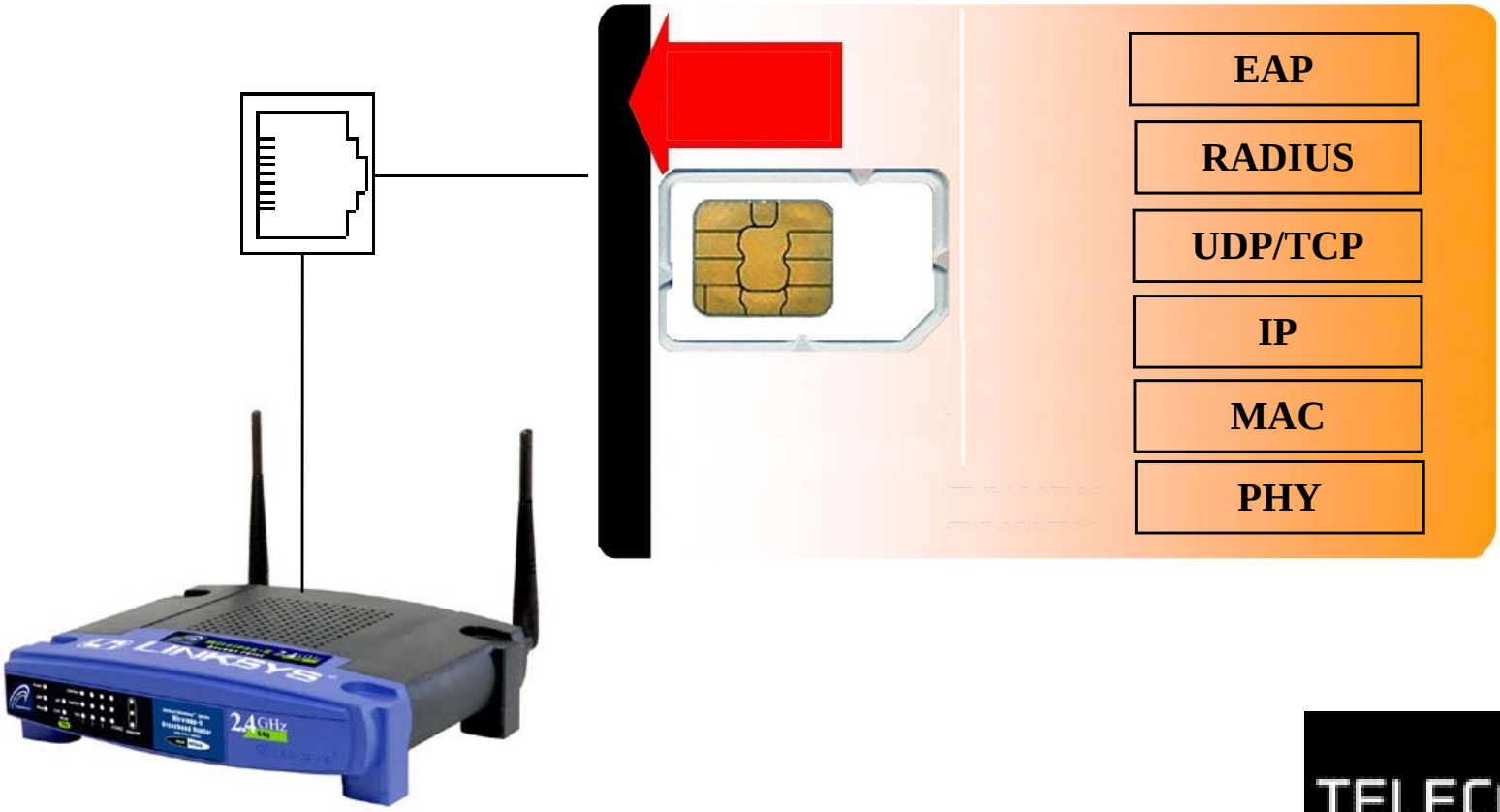
Ideas 2/3. Smartcards Cluster

- Smartcard handles the RADIUS protocol.
- Multiples RADIUS secrets are used for parallel processing.



Ideas 3/3. Full RADIUS Smartcard

- Network connectivity
- High computing capabilities



Demonstration 3.

Smartcard Enabled RADIUS Server

Identity Protection

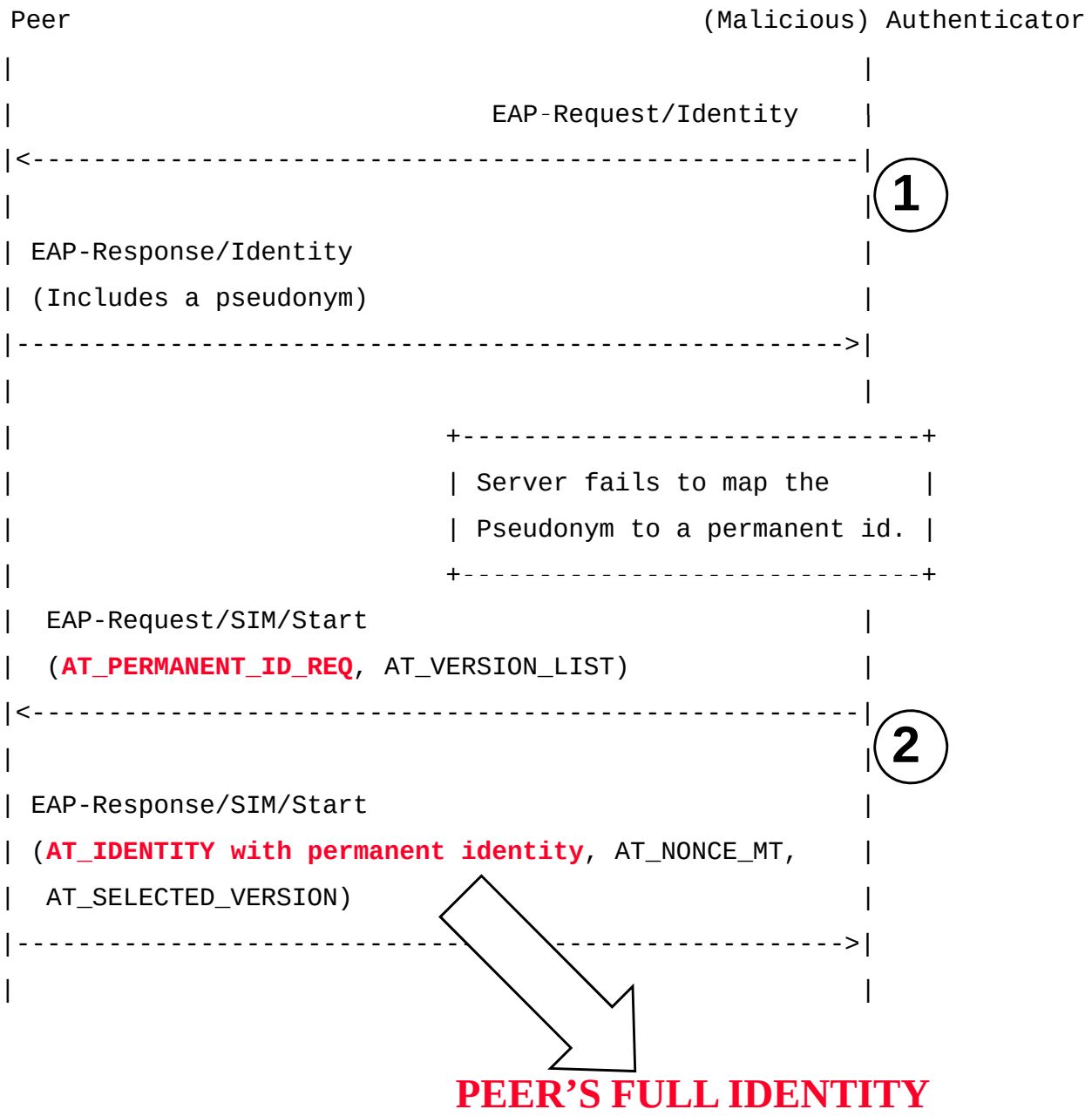
"Privacy is the right to informational self-determination, i.e., individuals must be able to determine for themselves when, how, to what extent and for what purpose information about them is communicated to others".

***S Kent and L Millet, "Who goes there? Authentication through the lens of privacy", The National Academies Press, Washington, D.C., 2003*.**

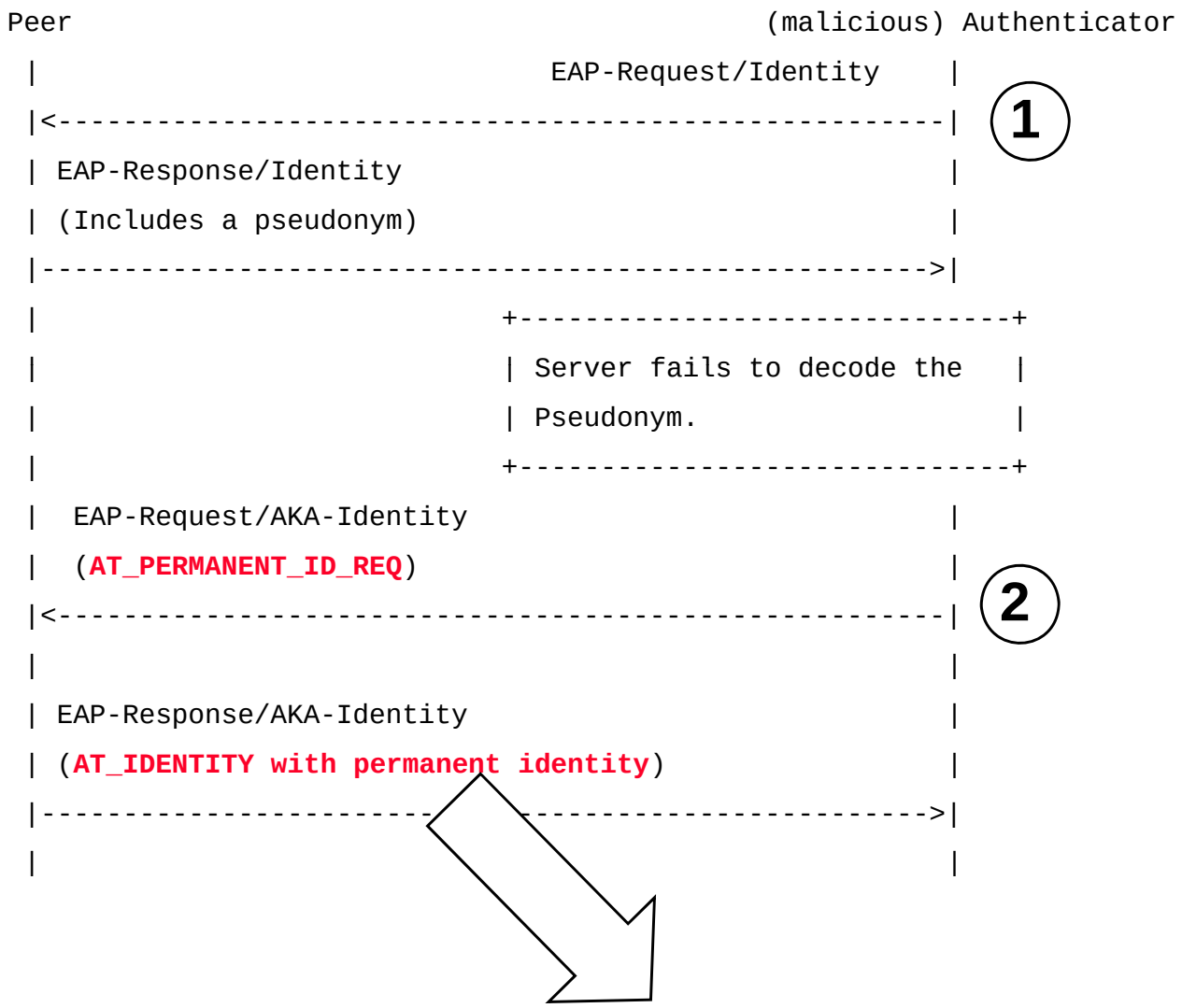
EAP-METHODS as RFIDs: Identity Leakage

- **The hacker aims at collecting the peer's identity, *over the air***
  - Passive attack, simple eavesdropping
  - Active attack, EAP packets generation from a malicious Access Point.
- **Number of EAP packets needed for active attacks**
  - EAP-SIM, RFC 4186, 2x requests, without previous knowledge
  - EAP-AKA, RFC 4187, 2x requests, without previous knowledge
  - EAP-TLS, RFC 2716, 3x requests. The knowledge of a *valid* authenticator's certificate is required.

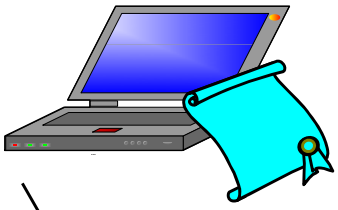
RFC 4186, EAP-SIM Identity Attack

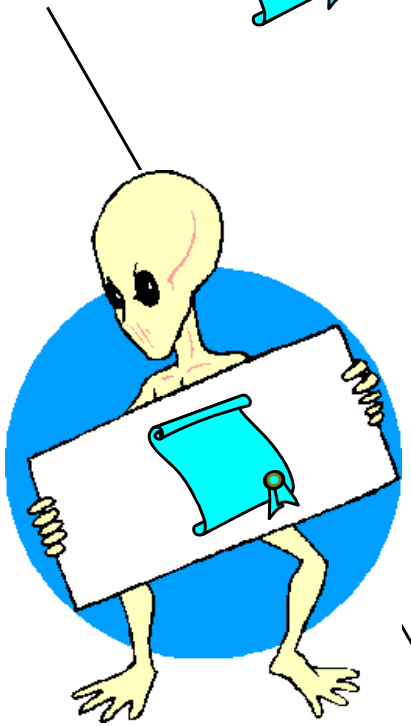


EAP-AKA, RFC 4187, Identity Attack



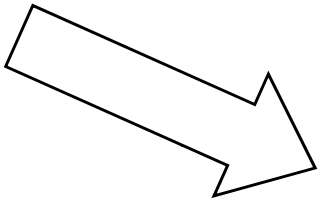
EAP-TLS (RFC 2716) Identity Attack





Electronic Identity

```
Authenticating Peer    (Malicious) Authenticator
==================     =========================
                           <- PPP EAP-Request/Identity      1
  PPP EAP-Response/
  Identity (MyID) ->
                           <- PPP EAP-Request/              2
                           EAP-Type=EAP-TLS
                           (TLS Start)
  PPP EAP-Response/
  EAP-Type=EAP-TLS
  (TLS client_hello)->
                           <- PPP EAP-Request/              3
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                            TLS certificate,
                       [TLS server_key_exchange,]
                       [TLS certificate_request,]
                           TLS server_hello_done)
  PPP EAP-Response/
  EAP-Type=EAP-TLS
  (TLS certificate,
   TLS client_key_exchange,
  [TLS certificate_verify,]
   TLS change_cipher_spec,
   TLS finished) ->
```



PEER'S FULL IDENTITY

A X509 certificate

Subject Name

Public RSA Key

Signature

```
000000  30 82 02 37 30 82 01 A0  A0 03 02 01 02 02 02 00  0..70.........
000010  FB 30 0D 06 09 2A 86 48  86 F7 0D 01 01 05 05 00  .0...*.H........
000020  30 81 84 31 0B 30 09 06  03 55 04 06 13 02 46 52  0..1.0...U....FR
000030  31 0B 30 09 06 03 55 04  08 13 02 37 35 31 0E 30  1.0...U....751.0
000040  0C 06 03 55 04 07 13 05  50 61 72 69 73 31 0D 30  ...U....Paris1.0
000050  0B 06 03 55 04 0A 13 04  45 4E 53 54 31 0F 30 0D  ...U....ENST1.0.
000060  06 03 55 04 0B 13 06 49  6E 66 72 65 73 31 12 30  ..U....Infres1.0
000070  10 06 03 55 04 03 13 09  49 6E 66 72 61 64 69 6F  ...U....Infradio
000080  31 31 24 30 22 06 09 2A  86 48 86 F7 0D 01 09 01  11$0"..*.H......
000090  16 15 72 6F 6F 74 40 69  6E 66 72 61 64 69 6F 2E  ..root@infradio.
0000a0  65 6E 73 74 2E 66 72 30  1E 17 0D 30 36 30 39 30  enst.fr0...06090
0000b0  31 30 39 34 38 31 39 5A  17 0D 30 39 31 32 31 34  1094819Z..091214
0000c0  30 39 34 38 31 39 5A 30  24 31 22 30 20 06 03 55  094819Z0$1"0 ..U
0000d0  04 03 14 19 70 75 72 69  65 6E 31 40 69 6E 66 72  ....purien1@infr
0000e0  61 64 69 6F 31 2E 65 6E  73 74 2E 66 72 30 81 9F  adio1.enst.fr0..
0000f0  30 0D 06 09 2A 86 48 86  F7 0D 01 01 01 05 00 03  0...*.H.........
000100  81 8D 00 30 81 89 02 81  81 00 BF F4 CE FC 83 44  ...0...........D
000110  03 DD 1F E2 30 5C 4D 6D  E6 E7 41 EF 58 CF 94 46  ....0\Mm..A.X..F
000120  05 B5 76 A5 16 62 6E 15  11 CD 12 B6 35 7D 5F F2  ..v..bn.....5}_.
000130  31 BB 3E 43 39 1F 02 F9  A8 B1 CC 25 D5 81 07 9D  1.>C9......%....
000140  29 12 D7 6B 49 B6 E1 FC  4F EA 63 B9 D1 36 5C 98  )..kI...O.c..6\.
000150  3B 52 0B 78 87 56 29 4A  84 B0 FD 12 F2 A2 7C 10  ;R.x.V)J......|.
000160  66 72 48 68 D3 07 8C 8A  77 C4 43 00 9B AD 88 AB  frHh....w.C.....
000170  02 EB 14 33 C9 47 B9 D1  CE 3E D7 77 82 B8 DF 12  ...3.G...>.w....
000180  27 25 4E 3F C7 3D 8E 8B  5C 71 02 03 01 00 01 A3  '%N?.=..\q......
000190  17 30 15 30 13 06 03 55  1D 25 04 0C 30 0A 06 08  .0.0...U.%..0...
0001a0  2B 06 01 05 05 07 03 02  30 0D 06 09 2A 86 48 86  +.......0...*.H.
0001b0  F7 0D 01 01 05 05 00 03  81 81 00 34 FC D5 7B 61  ...........4..{a
0001c0  8C 08 C4 21 63 F8 B6 22  D7 2A E8 46 F9 16 40 0B  ...!c..".*.F..@.
0001d0  C8 06 AC D1 01 6B 13 2F  89 06 69 F2 ED 2F 3E 1F  .....k./..i../>.
0001e0  E4 E3 FA 74 71 79 DD 31  6F D7 2F 73 CD 7C 0A BC  ...tqy.1o./s.|..
0001f0  44 81 BD 6C 92 55 9D 52  A8 AC 58 3D BF 9B 16 10  D..l.U.R..X=....
000200  63 E7 A1 FA 3C 31 F8 C1  5A 8D CB 2E 58 66 2F 78  c...<1..Z...Xf/x
000210  85 2C 94 3E 83 6F 7A 81  A9 0E 87 0B C9 0C AE 71  .,.>.oz........q
000220  2A 7E 68 34 CA F5 BA A4  DF 8C 03 58 C0 32 AA DF  *~h4.......X.2..
000230  8B 6F E5 08 D0 93 C3 7D  E5 F0 70                 .o.....}..p
```

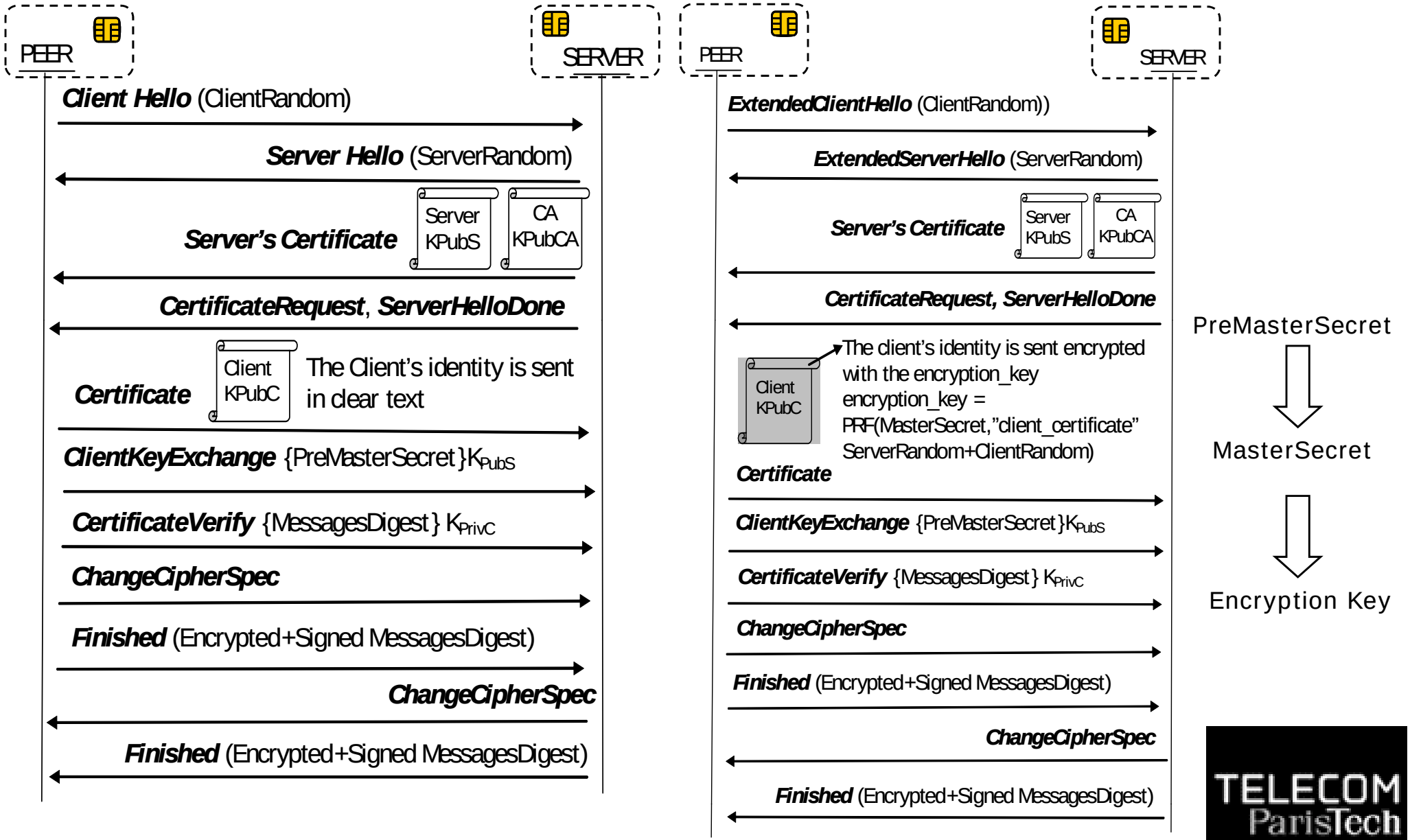
Classical solutions, but not standardized

- **Establishment of a first protected channel, that secures the peer's identity**
  - Asymmetric protected channel
    - Protected EAP Protocol (PEAP) Version 2, daft-josefsson-pppext-eap-tls-eap-10.txt (2004, *expired*)
    - EAP Tunneled TLS Authentication Protocol Version (EAP-TTLSv1), draft-funk-eap-ttls-v1-01.txt, (2006, *active*)
  - Symmetric protected channel
    - EAP-Double-TLS Authentication Protocol, draft-badra-eap-double-tls-05.txt (2006, *active*)

Proposed solution for EAP-TLS

- **Main idea**
  - The peer's certificate is sent encrypted, the encryption key is deduced from the master_secret.
    - **encryption_key = PRF(master_secret, "client_certificate",client_random+server_random);**
  - In order to allow an EAP-TLS peer to request identity protection exchange, a new extension type is added (TBD) to the Extended Client and Server Hello messages.
  - The 'extension_data' field of this extension contains a list of encryption algorithms supported by the client, ordered by preference.
  - If the server is willing to accept using the extension, the client and the server negotiate the symmetric algorithm that will be used to encrypt/decrypt the client certificate.
  - At the end of the hello phase, the client generates the pre_master_secret, encrypts it under the server's public key, and sends the result to the server.
- **Encryption of the peer's certificate**
  - If a stream cipher is chosen, then the peer's certificate is encrypted with the enc_key, without any padding byte.
  - If a block cipher is selected, then padding bytes are added to force the length of the certificate message to be an integral multiple of the b cipher's length.

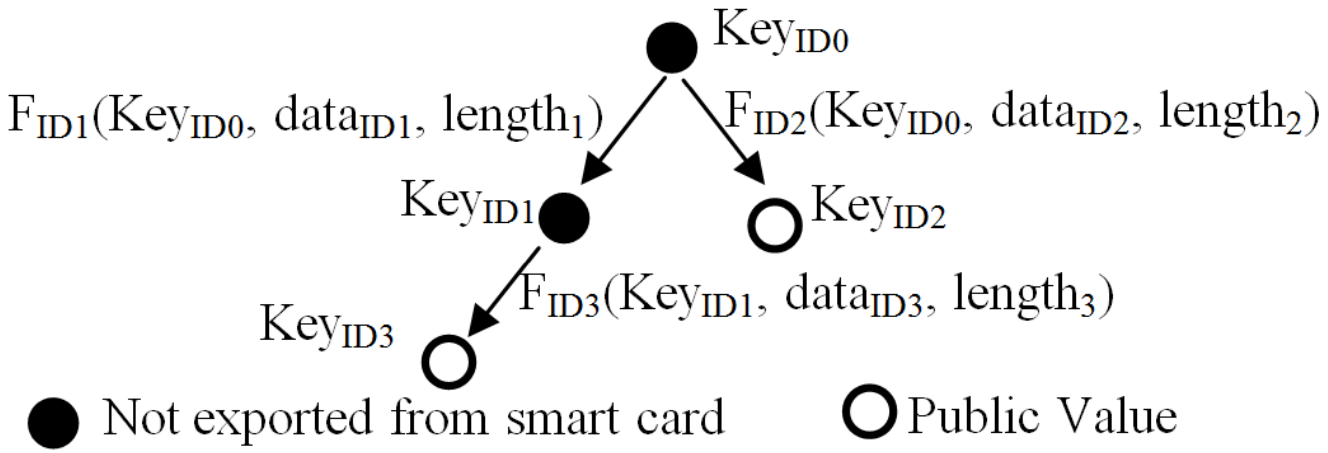
Identity Protection Dialog



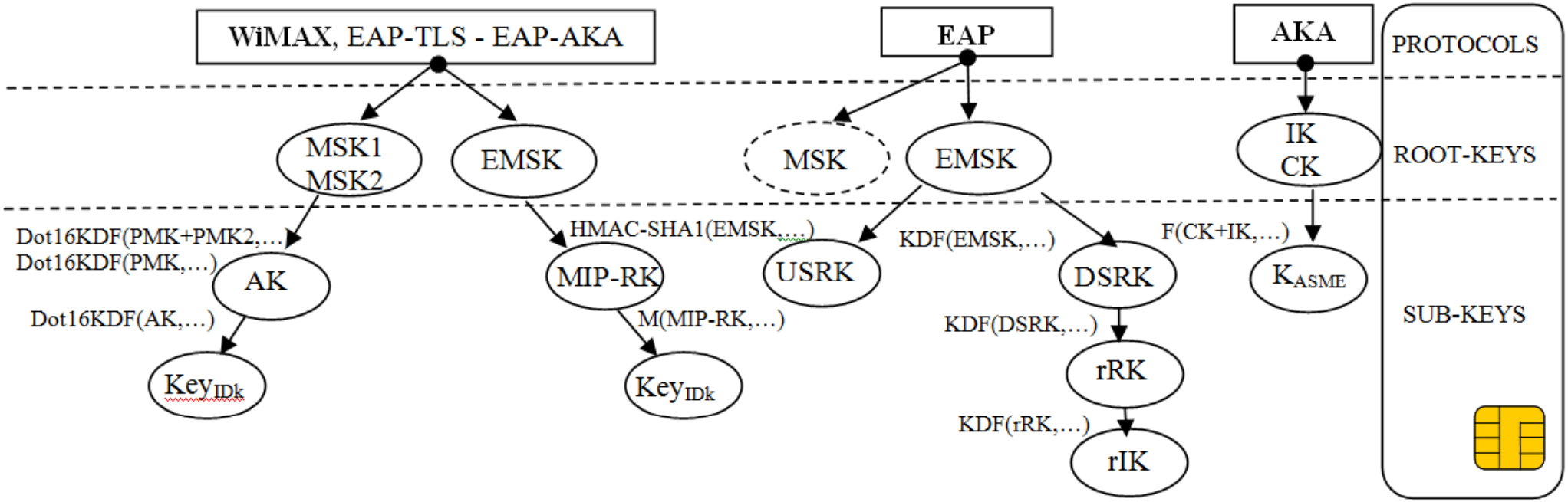
Keys-Tree computing

What is a keys-tree

- **A keys-tree is a graph whose nodes are key values (Key_{IDi}).**
- **Leafs are public values.**
- **Branches represent procedures (F_{IDj}), such as KDF (*Key Derivation Function*) or PRF (*Pseudo Random Function).***
  - **$Key_{IDj} = F_{IDj}(Key_{IDk}, public\text{-}data_{IDj}, length\text{-}of\text{-}key\text{-}j)$**
- **Two classes of keys-tree**
  - Pre-defined
  - Dynamic, i.e. working with scripts



Network Keys -Tree



ETSI TR 102 469 V1.1.1 (2006-05) Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Architecture

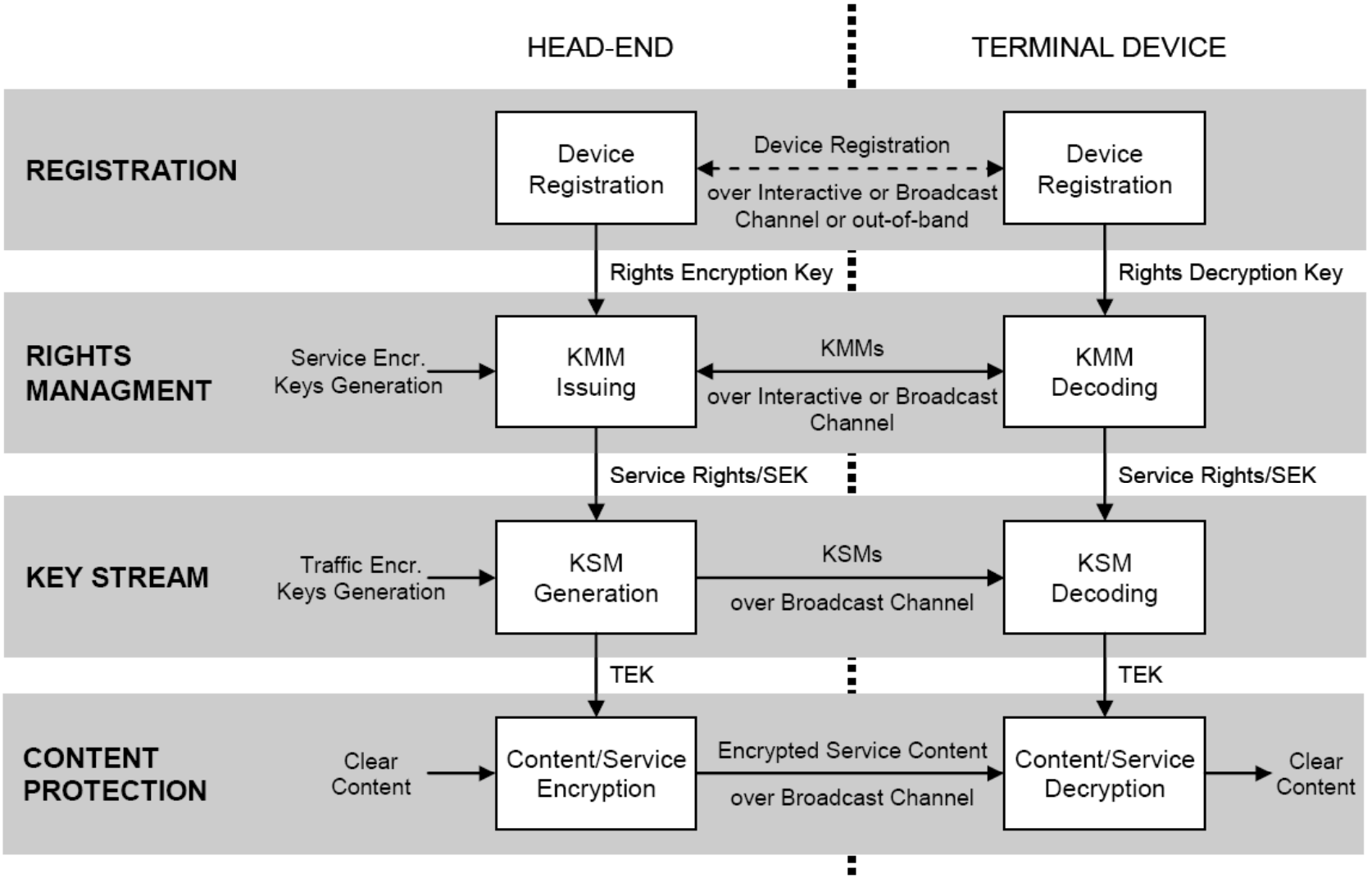


Figure 13: Hierarchical Model for Service Protection

Example

rRK = KDF(**EMSK**, key-label | "\0" | optional-data | length)

key-label= "EAP Re-authentication Root Key@ietf.org »

rIK = KDF(**rRK**, key-label | "\0" | crypto-suite | key-length)

Key-label= "Re-authentication Integrity Key@ietf.org"

rMSK= KDF(**rRK**, key-label | "\0" | SEQ | key-length)

Key-label= "Re-authentication Master Session Key@ietf.org »

TELECOM ParisTech

Keys-tree declaration in ASN.1

KeyTree :: = SEQUENCE OF SubKeyTree

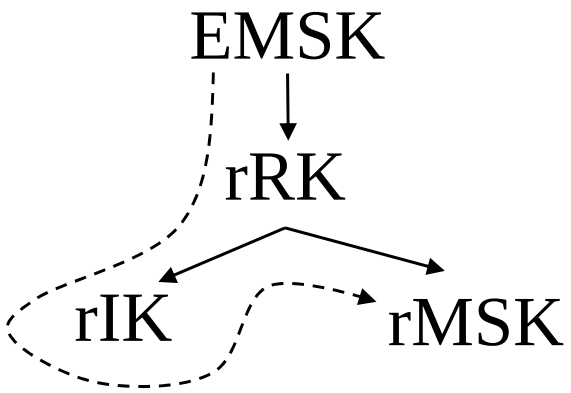
SubKeytree ::= CHOICE {
end [0] NULL
kdf [1] IMPLICIT SEQUENCE
{KeyOut KeyReference,
KeyIn KeyReference,
KeyLabel OCTETSTRING,
OptionalData OCTETSTRING,
Length INTEGER ,
SubTree SEQUENCE OF **SubKeyTree**
} }

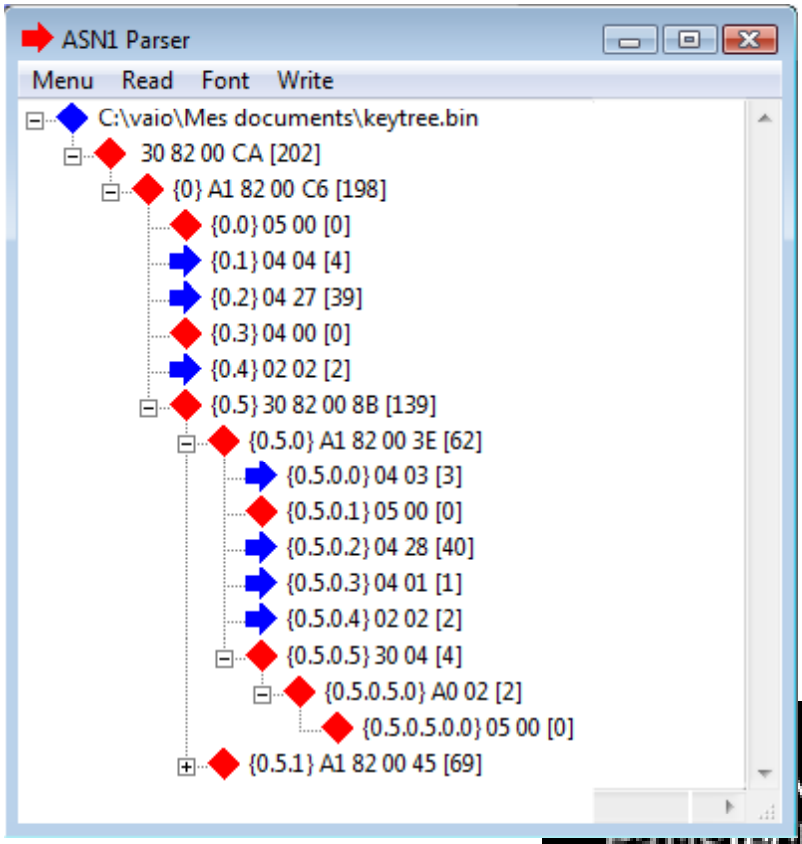
KeyReference ::= CHOICE {
Implicit NULL,
Reference OCTETSTRING }

Result ::= SEQUENCE OF KeyValue
KeyValue ::= SEQUENCE OF
{ reference KeyReference,
value OCTETSTRING }

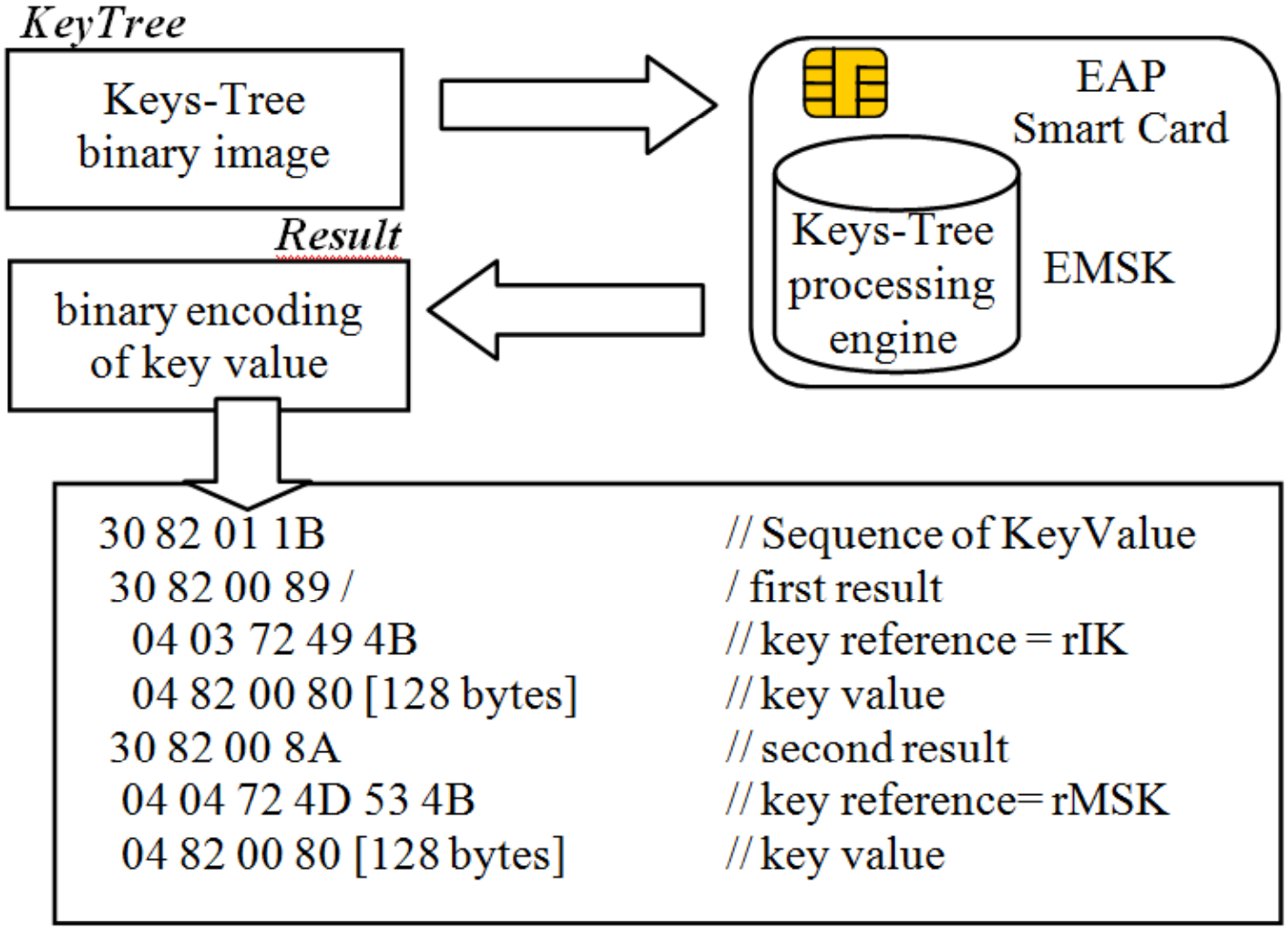
Keys-Tree, binary encoding

```
30 82 00 CA
 A1 82 00 C6
   05 00                                  // KeyOut reference is implicit
   04 04 45 4D 53 4B                      // KeyIn  reference=  “EMSK"
   04 27 "EAP Re-authentication Root Key@ietf.org"
   04 00                                  // No Optional Data
   02 02 0080                             // Length = 128 bits

 30 82 00 8B                              // Sequence of SubKeyTree
    A1 82 00 3E
      04 03 72 49 4B                      // KeyOut reference = "rIK"
      05 00                               // KeyIn reference = implicit
      04 28 "Re-authentication Integrity Key@ietf.org"
      04 01 01                            // optional-data= crypto-suite=01
      02 02 0080                          // Length= 128 bits
      30 04                               // sequence of SubKeyTree
       A0 02 05 00                        // null SubKeyTree
    A1 82 00 45
      04 04 72 4D 53 4B                   // KeyOut reference = rMSK
      05 00                               // KeyIn reference = implicit
      04 2D "Re-authentication Master Session Key@ietf.org"
      04 02 0001                          // / optional-data= SEQ= 0001
      02 02 0080                          // Length= 128 bits
      30 04                               // sequence of SubKeyTree
       A0 02 05 00                        // null SubKeyTree
```





Keys-Tree processing with an EAP Smart card



EAP-TLS smart card for WEB applications

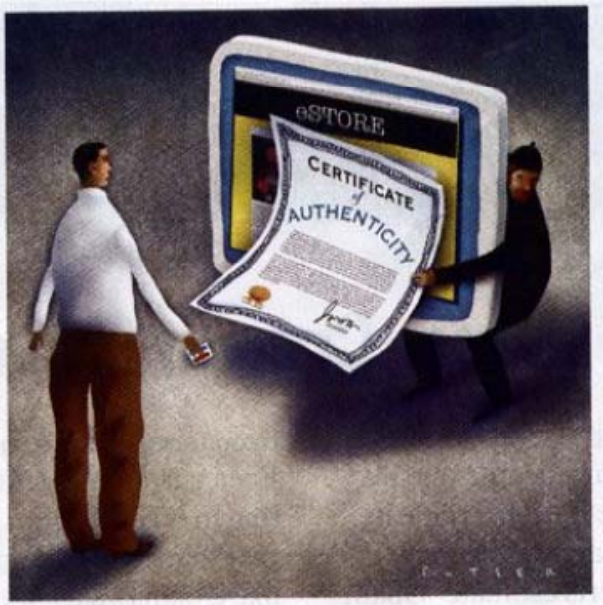
TELECOM ParisTech

About the Ubiquitous Internet

- **The dream of ubiquitous internet access is about to appear in real life**
- **In this context, security issues, such as *identity management* for WEB applications appear as very critical and sensitive topics.**
- **Today tens of login and password items are required by WEB sites, in order to :**
  - recover subscriber's information in their databases,
  - trace users' requests via session management facilities

How to suppress passwords

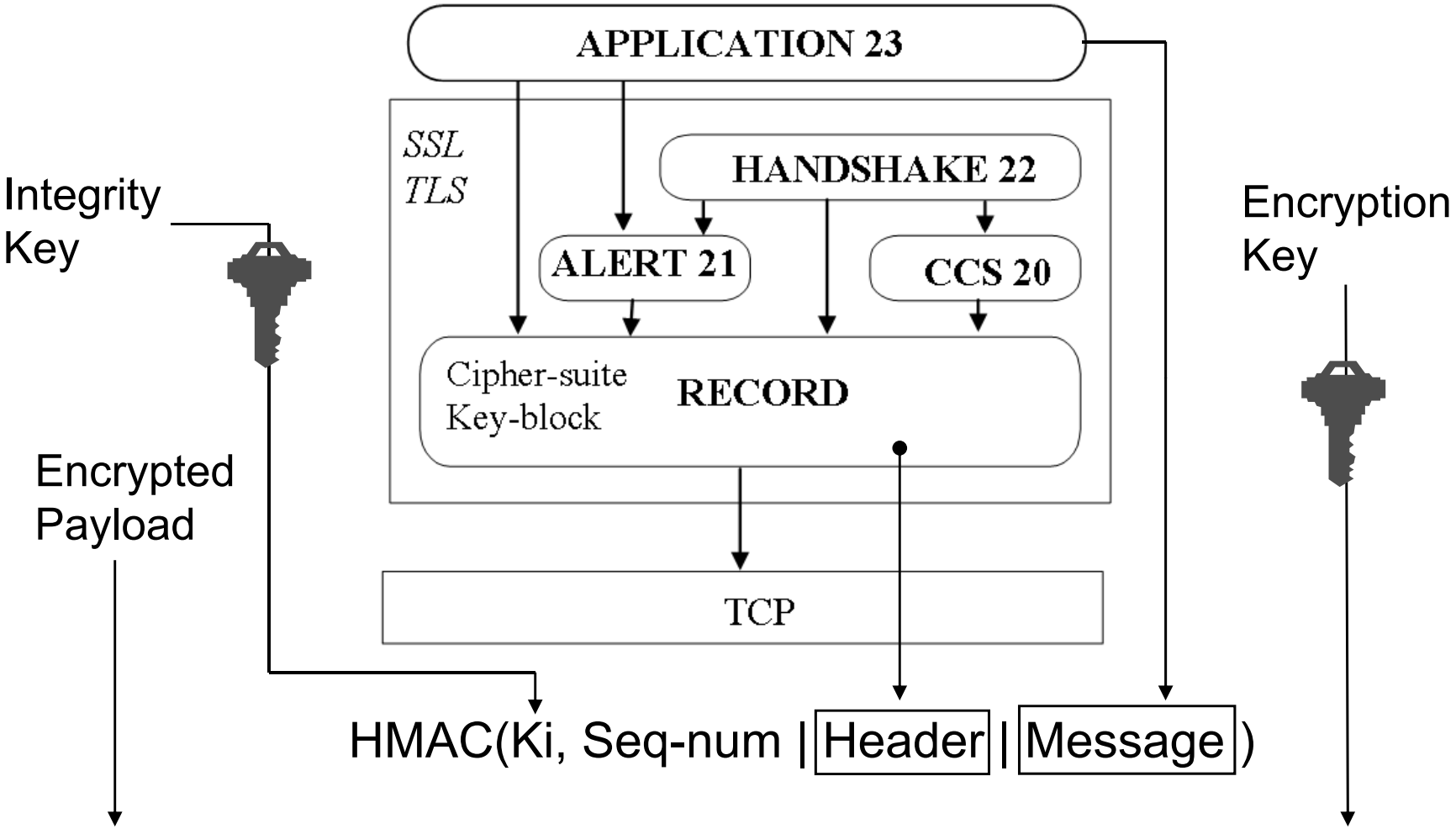
- **We suggest to banish passwords and to replace them by X509 certificates.**
- **More precisely we propose to deploy innovative tamper resistant devices (such as smart cards) autonomously processing the SSL/TLS protocol, and securely storing all associated credentials (including certificates and cryptographic keys), packed in identity containers.**
- **The *"SSL in the box"* concept avoids the use of certificates based on obsolete hash functions**
  - Stevens et al («*Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*». MD5 certificates may be closed in 1-2 days with 200 play-stations
  - During the Eurocrypt'2009 conference, Cameron McDonald, Philip Hawkes and Josef Pieprzyk, unveiled a collision algorithm working for SHA1 with a complexity of 2^{52}.



A cybercriminal holding a rogue certificate could convince any browser that a fake Web site is authentic.

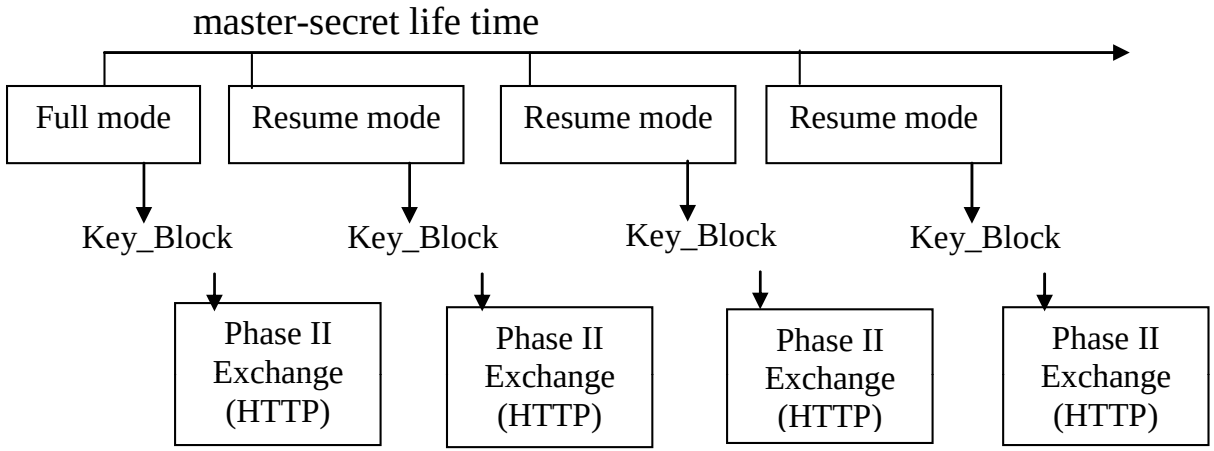
Newsweek, February 2009

TLS Stack Structure



Basic keys calculations

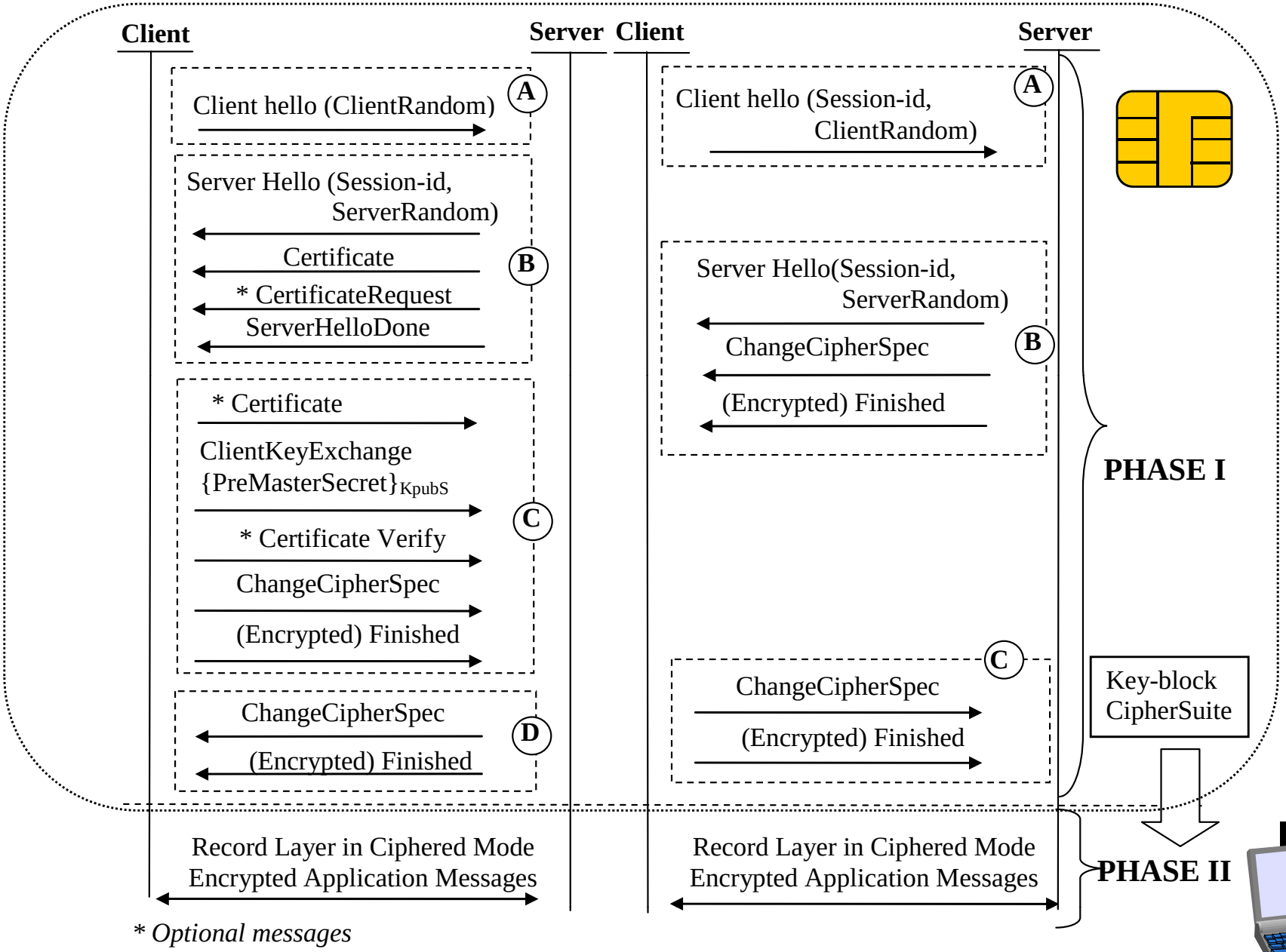
- **For Full Sessions**
  - A *PreMasterSecret* is sent by the client encrypted with the server public key
    - { PreMasterSecret } KPubS
  - master-secret = PRF(PreMasterSecret, "master secret", ClientRandom | ServerRandom)
  - If a client certificate is required (mutual authentication) it is forwarded to the server, and authenticated by a signature generated with the client private key.
- **For Resume Sessions**
  - A master secret has already been computed, it is identified by an index, named the Session-ID
- **Negotiated cryptographic algorithms are identified by a two bytes value labeled CipherSuite.**
  - key-block = PRF(master-secret, "key expansion", ServerRandom | ClientRandom)



TLS-Tandem main idea

- **A TLS session is split in two phases.**
  - First (*Phase I*) deals with authentication and cryptographic key calculations
  - Second (*Phase II*) takes advantage of the previously created secure channel, in order to exchange information between applications in a safe context.

TLS Dual Stack

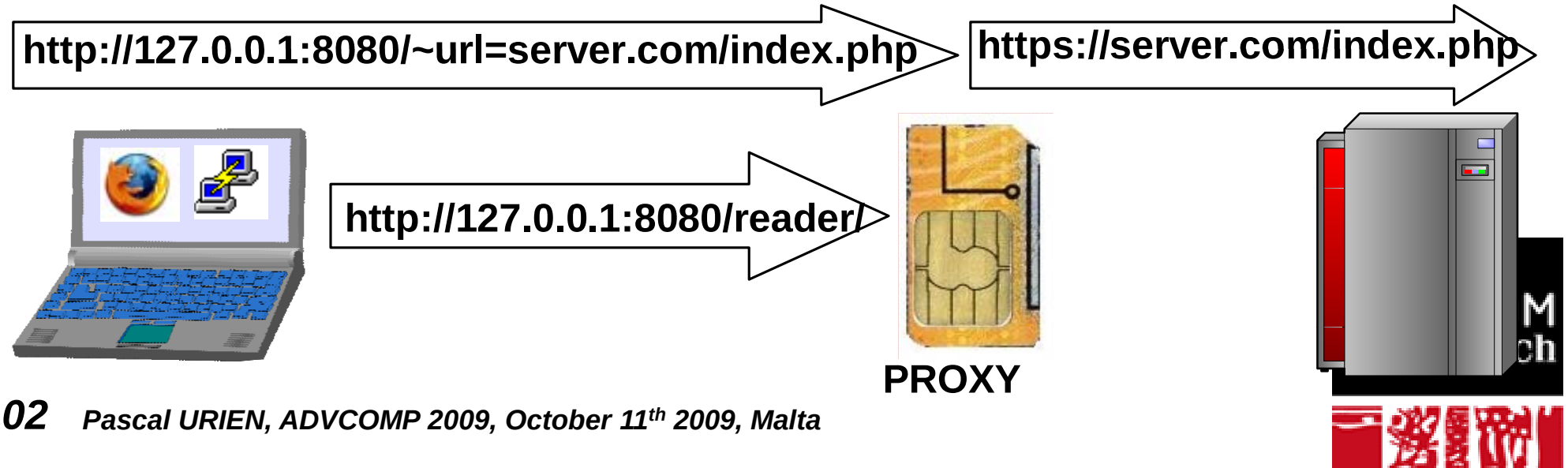


TLS-Tandem benefits

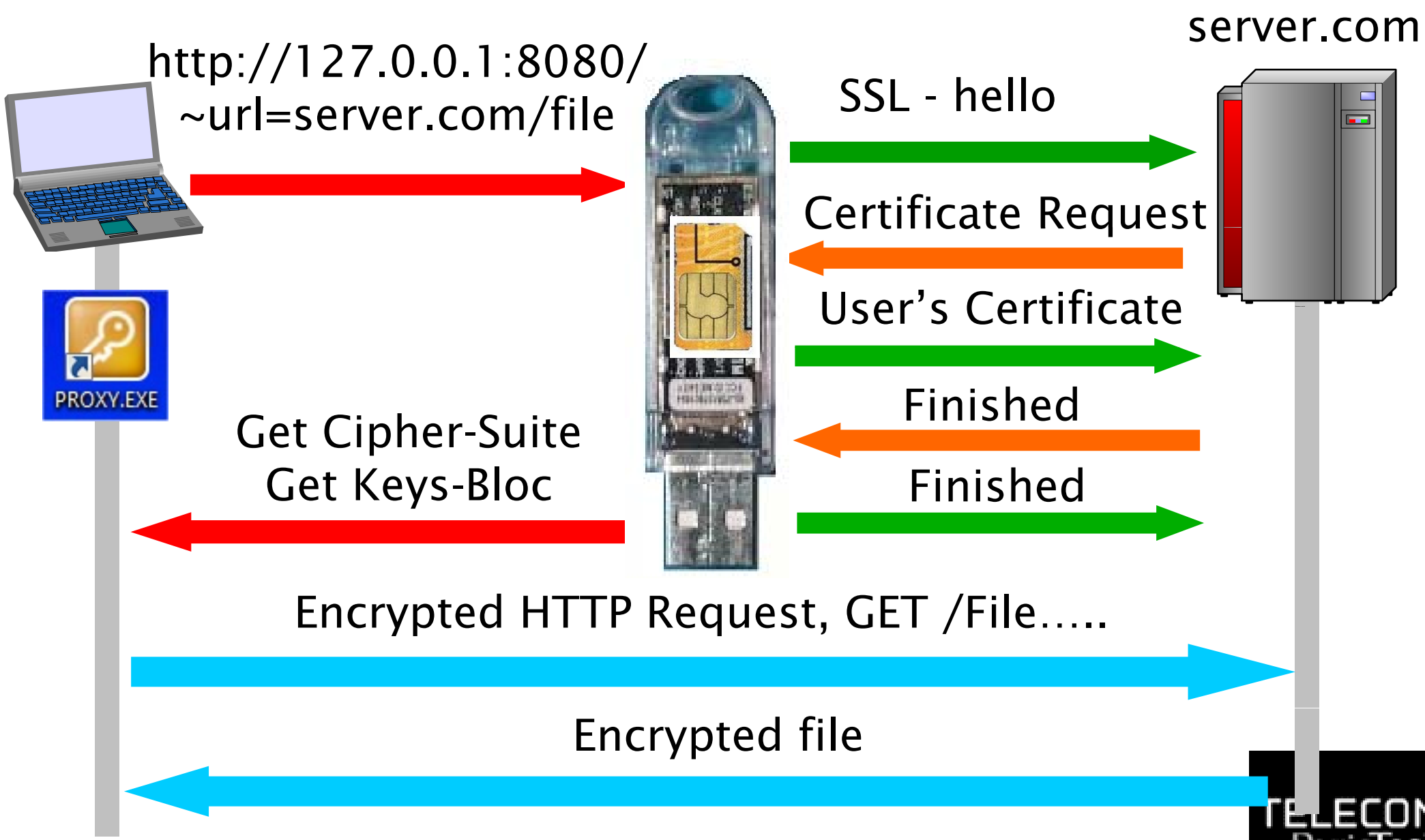
- ***Phase I* is fully computed in a tamper resistant device**
  - Checking of server certificate. This is the heart of SSL trust.
  - Generation of client signature, what requires a secure storage and use of its RSA private key.
  - Secure storage of master-secret, which is often used for session resumption, and therefore gives the same level of privilege than the knowledge of a private key.
  - A WEB session starts by a full session, and afterwards resume sessions are used during about 10 minutes.
- ***Phase II* runs in an untrustworthy computer**
  - Untrustworthy computers realizes the translation between clear and ciphered data.

The Platform

- **A Browser**
- **A Proxy Software**
  - Runs the SSL record layer
  - http://127.0.0.1:8080/~url=server.com/index.php
  - http://127.0.0.1:8080/reader/apdu?=data
- **A Smart Card**
  - Runs the SSL stack
- **A WEB Server**
  - Configured for SSL session with mutual authentication



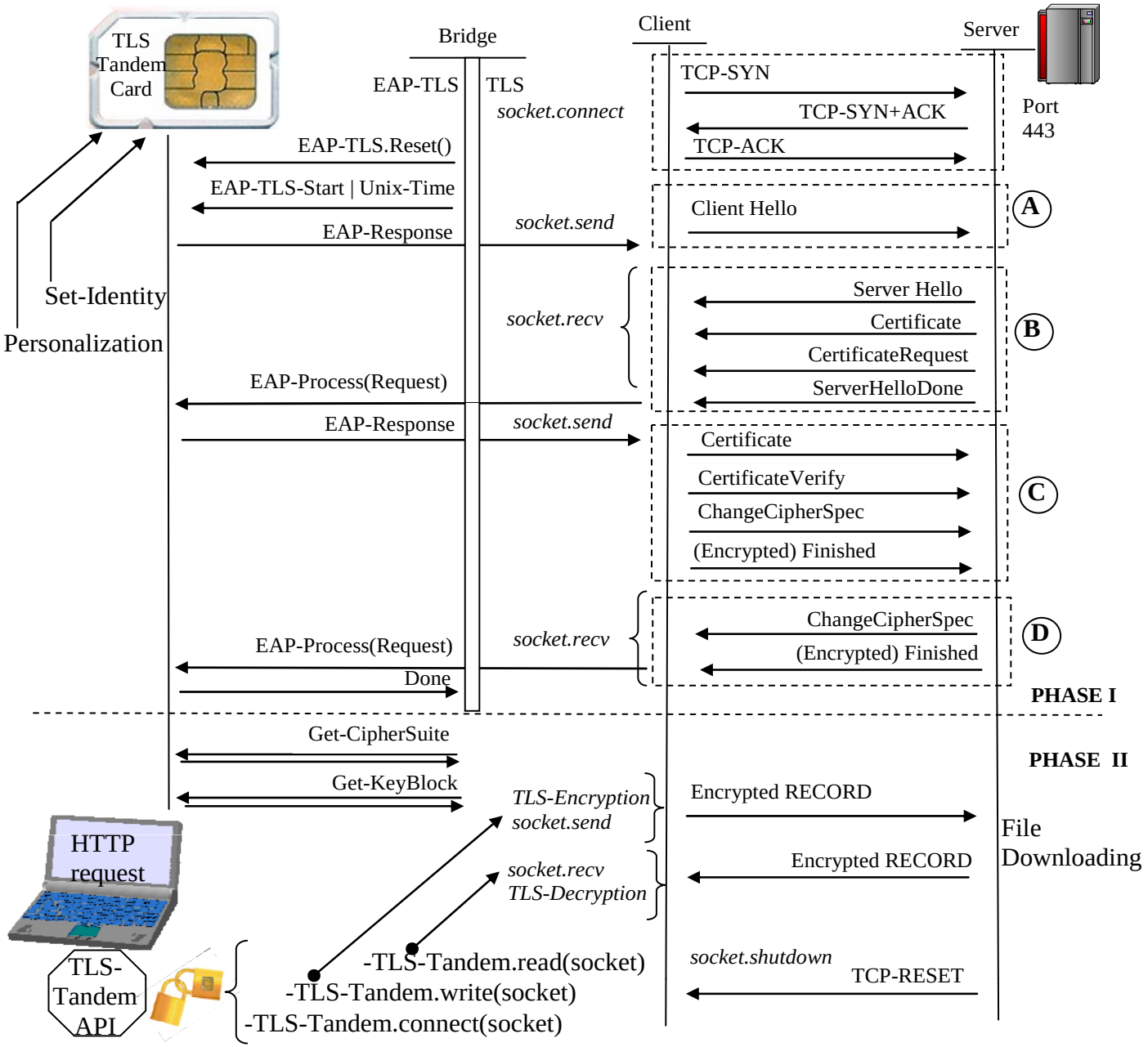
Tandem Overview



High Level API

- ***int error = OpenSSLCard(socket s),*** **establishes a SSL session via the smart card.**
- ***int length = ReadSSLCard(socket s, char buffer, int size),*** **receives and decrypts SSL packets from the remote site, and returns HTTP messages.**
- ***int length = WriteSSLCard(socket s, char buffer, int size),*** **performs SSL encryption and sends the content to the remote server.**

Choreography



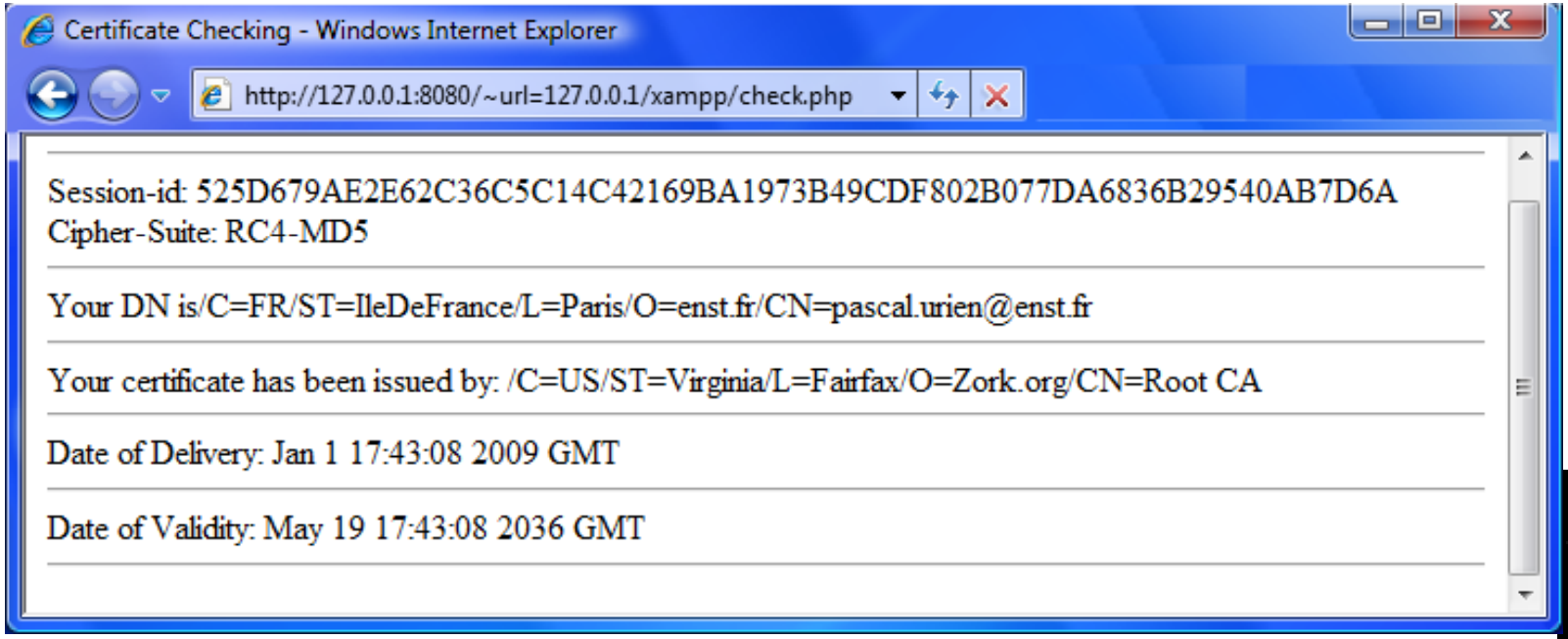
Performances analysis

WEB site setting

- The *SSLCACertificateFile*, stores a concatenation of supported certification authorities.
- The *SSLCertificateFile*, stores the server certificate.
- The SSL *CertificateKeyFile*, stores the server RSA private key.
- It is possible, thanks to directives such as *SSLRequireSSL* and *SSLVerifyClient,* to enforce SSL use with mutual authentication

PHP facilities

```
<html> <head> <title> Certificate Checking </title> </head> <body>
<?php
if ($_SERVER['SSL_CLIENT_VERIFY'] != "SUCCESS")
{ Header("HTTP/1.0 401 Unauthorized");  exit; }
echo "<hr>Session-id:  "     ; echo $_SERVER['SSL_SESSION_ID'];
echo "<br>Cipher-Suite: "    ; echo $_SERVER['SSL_CIPHER'];
echo "<br><hr>Your DN is";  echo $_SERVER['SSL_CLIENT_S_DN'] ;
echo "<br><hr>Your certificate has been issued by: ";
echo $_SERVER['SSL_CLIENT_I_DN'];
echo "<br><hr>Date of Delivery: " ;
echo $_SERVER['SSL_CLIENT_V_START'] ;
echo "<br><hr>Date of Validity: ";
echo $_SERVER['SSL_CLIENT_V_END'] ;
echo "<br><hr>"; ?>  </body> </html>
```



WEB/WEB2 Interface

- **A set of URLs for:**
  - Collecting the readers list
    - URI= /reader/list
  - Setting of a working reader
    - URI=/reader/select/?=index
  - Getting the current reader
    - URI= /reader/selected
  - Sending a cold reset to the smart card plugged into the current reader,
    - URI= /reader/atr
  - Sending a warm reset to the smart card plugged into the current reader,
    - URI= /reader/warm
  - Sending on or several APDUs to the smart card plugged to the current reader,
    - URI= /reader/apdu?=$request_1$&=$request_2$&=$request_i$

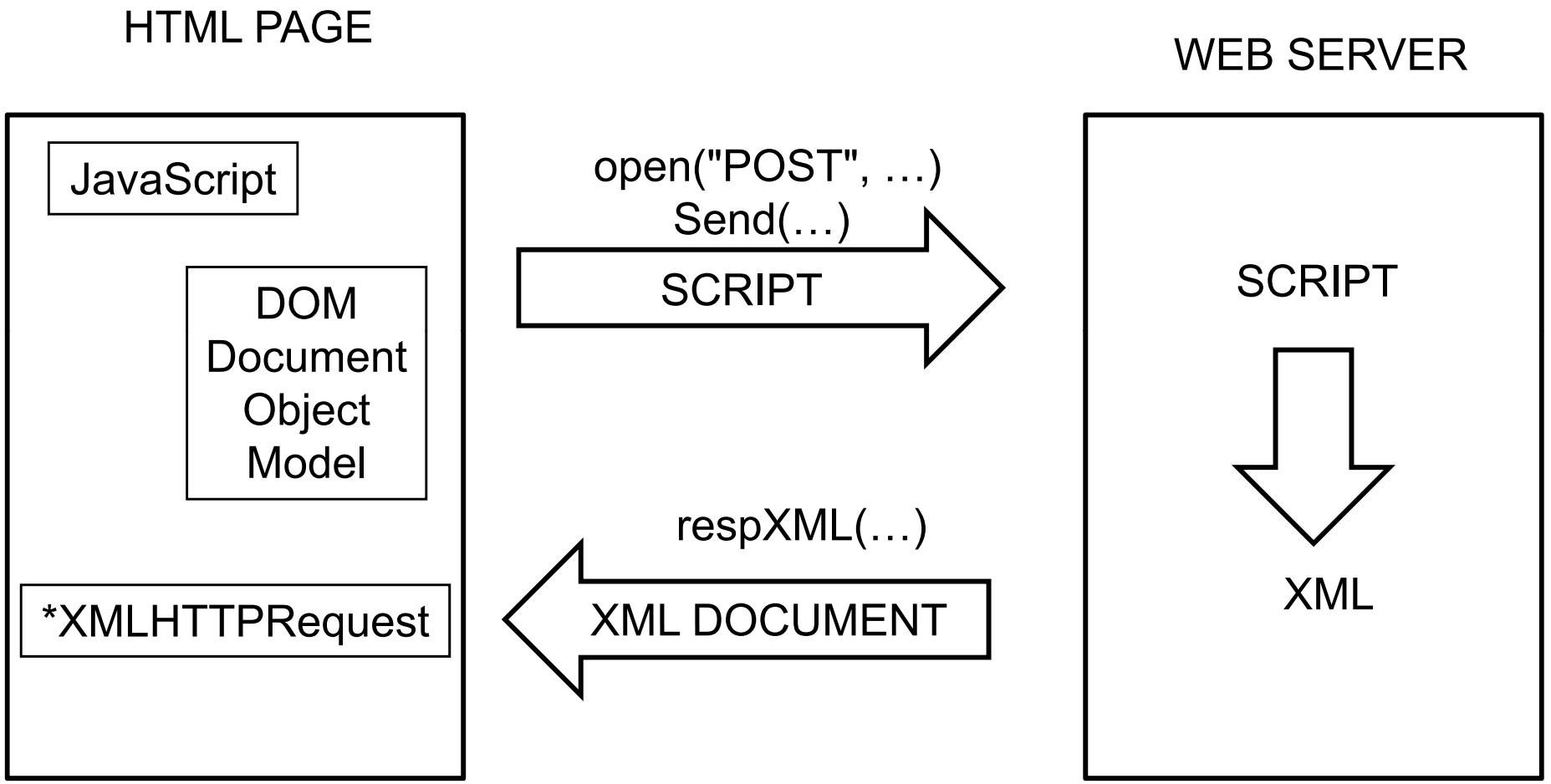
XML Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<coldatr>
    - <item>
        <title>coldatr</title>
        - <content>
            <![CDATA[ 3B9E95801FC38031E073FE21 ]]>
          </content>
      </item>
</coldatr>
```

XML Coding

URI	First tag of the XML response	Number of <item> tags	Meaning of the <title> element	Meaning of the <content> element
/reader/ list	<list>	>=1	Reader name	Reader index
/reader/ select	<select>	1	Reader name	Reader index
/reader/ selected	<selected>	1	Reader name	Reader index
/reader/ atr	<coldatr>	1	constant “coldatr”	ATR value
/reader/ warm	<warmatr>	1	constant “warmatr”	ATR value
/reader/ apdu	<apdu>	>=1	Request value	Response value

AJAX: Principle



*The XMLHttpRequest Object, W3C Working Draft 15 April 2008.

AJAX Programming example

```
<html><head><TITLE></TITLE>
<script type="text/javascript">
var req;

function processReqChange()
{  if (req.readyState == 4)
  {   if (req.status == 200) display();
      else alert("Error"); }}

function doit()
{ req = new XMLHttpRequest();
req.onreadystatechange = processReqChange;
req.open("POST", document.mycmd.action, true);
req.send("=" + document.mycmd.apdu.value);
return false ; }

function display()
{ var itemArray = req.responseXML.getElementsByTagName("item");
var content="";
 content = itemArray.length + " items has been found in the response" +"<br/>";

for (var i = 0; i < itemArray.length; i++)
{ content +=  "Title: "
+ itemArray[i].getElementsByTagName("title")[0].firstChild.nodeValue + "<br/>";
content += "Content: "
+ itemArray[i].getElementsByTagName("content")[0].firstChild.nodeValue + "<br/>";
}

var div = document.getElementById("details");
div.innerHTML = ""; div.innerHTML = content;}
</script></head>

<body><h2>AJAX Example</h2>

<form action="/reader/apdu"  method="post" name="mycmd" onsubmit=
"return doit()">

apdu <INPUT id="apdu" NAME="value"  value="00A4040010
A0000000300002FFFFFFFF8931323800" SIZE="60"><br>
<input TYPE="submit" value="Send" ><input TYPE="reset"
value="Reset">
</form>

<div id="details"></div>                    </body></html>
```

AJAX Example

apdu 00A4040010 A0000000300002FFFFFFFF8931323800

Send Reset

AJAX Example

apdu 00A4040010 A0000000300002FFFFFFFF8931323800

Send Reset

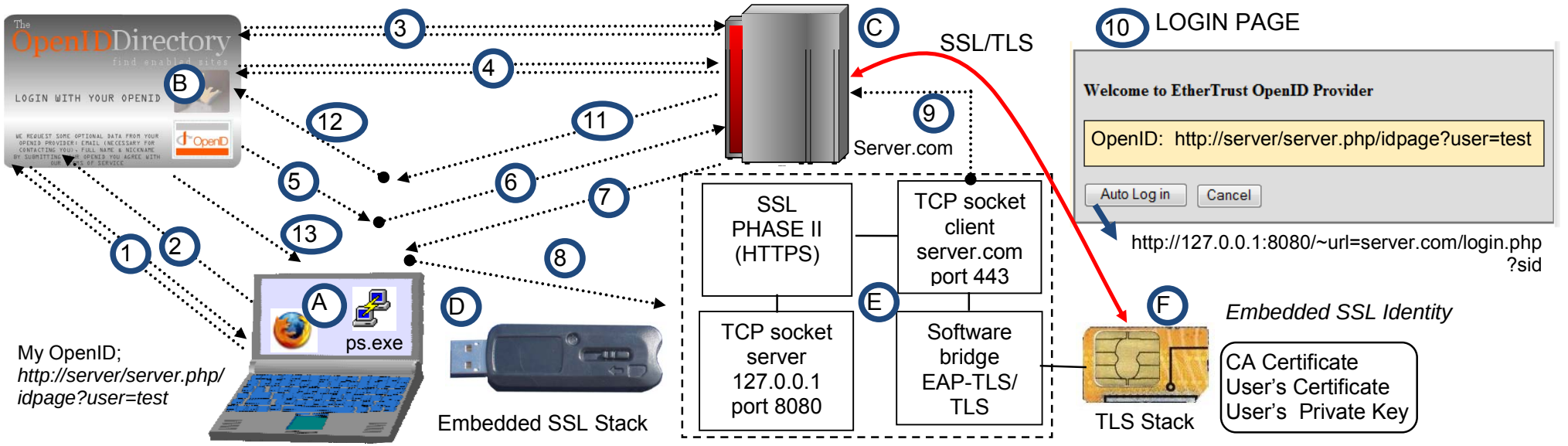
1 items has been found in the response
Title: 00A4040010A0000000300002FFFFFFFF8931323800
Content: 9000

Demonstration 4.

SSL Access Control & Trusted Downloading

The OpenID platform

- **In OpenID, the *consumer* WEB site (or *Relying Party*) is an application that needs to authenticate a net surfer.**
- **The user ‘s identity is an URL**
  - http://192.168.2.44/openid/examples/server/server.php/user=pascal
- **The OpenID provider is the identity server that authenticates the user.**



OpenID 1/4

- **When the browser opens an HTTP session with the *consumer* (1) it gets a form that aims at collecting its OpenID identifier (OPID).**
- **An *OPID* is an URL, pointing an OpenID *Provider,* i.e. a WEB site managing the authentication process.**
  - The user fills this form with its *OPID* and returns it to the *consumer* (2).
- **The consumer starts a discovery (XRI) procedure (3) with the provider and collects the XRDS server address**
  - http://192.168.2.44/openid/examples/server/server.php/user=pascal
  - Return information
    - XRDS Server Location
    - http://192.168.2.44/openid/examples/server/server.php/userXrds?user=pascal
- **The consumer collects (3') information with the XRDS server**
  - http://192.168.2.44/openid/examples/server/server.php/userXrds?user=pascal
  - Return information
    - Openid version, OpenId Server
    - http://192.168.2.44/openid/examples/server/server.php
- **It establishes a security association (4) identified by a handle and producing a *mac-key* used for messages authentication.**
  - http://192.168.2.44/openid/examples/server/server.php
    - It includes the consumer DH public key g^x
  - Return information
    - The Server DH public key g^y
    - The type of assocation (HMAC-SHA1)
    - The association handle
    - The mac encryption key, encrypted key = $H(g^{xy} \bmod p)$ exor mac-key
    - The type of session (DH-SHA1) used for mac-key encryption

- An *authentication request* (5) is then sent by the consumer to the browser
- Thanks to the HTTP redirect mechanism this message is forwarded (6) to the OP *provider*.

```
<html><head><title>OpenId transaction in progress</title></head>
<body onload='document.forms[0].submit();'>
<form accept-charset="UTF-8" enctype="application/x-www-form-urlencoded" id="openid_message"
action="http://192.168.2.44/openid/examples/server/server.php" method="post">
<input type="hidden" name="openid.ns" value="http://specs.openid.net/auth/2.0" />
<input type="hidden" name="openid.ns.sreg" value="http://openid.net/extensions/sreg/1.1" />
<input type="hidden" name="openid.ns.pape" value="http://specs.openid.net/extensions/pape/1.0" />
<input type="hidden" name="openid.sreg.required" value="nickname" />
<input type="hidden" name="openid.sreg.optional" value="fullname,email" />
<input type="hidden" name="openid.pape.preferred_auth_policies" value="" />
<input type="hidden" name="openid.realm" value="http://192.168.2.35:80/openid/examples/consumer/" />
<input type="hidden" name="openid.mode" value="checkid_setup" />
<input type="hidden" name="openid.return_to"
value="http://192.168.2.35:80/openid/examples/consumer/finish_auth.php?janrain_nonce=2009-03-17T09%3A07%3A14ZGoN7wY" />
<input type="hidden" name="openid.identity" value="http://192.168.2.44/openid/examples/server/server.php/idpage?user=moi" />
<input type="hidden" name="openid.claimed_id" value="http://192.168.2.44/openid/examples/server/server.php/idpage?user=moi" />
<input type="hidden" name="openid.assoc_handle" value="{HMAC-SHA1}{49bf5e64}{Va0OCQ==}" />
<input type="submit" value="Continue" />
</form>
<script>var elements = document.forms[0].elements;for (var i = 0; i < elements.length; i++) {  elements[i].style.display = "none";}
</script></body></html>
```

OpenID 3/4

- **In a classical provider implementation, the user is identified by a password collected by a form, the *login page* (10) and protected via a SSL session.**
  - In our platform we suppress this password, and we replace it by an SSL session with mutual authentication (i.e. both sides hold X509 certificate and RSA private key
- **The login page (7) is associated to an HTTP header including a cookie (whose value is *sid*), and the associated URL (8) also comprises this value, i.e. looks like**
  - http://127.0.0.1:8080/~url=server.com/login.php?sid
- **When the user clicks on the *AutoLogin* button it opens via its token the *https//server.com/login.php?sid* link (9) and is authenticated by its X509 certificate.**
- **The HTTP response to this request will set the cookie (sid) for the proxy address (127.0.0.1:8080).**
- **A second exchange occurs between the provider and the browser (secured by the token), in order to confirm the association with the consumer site.**

```
<form method="post"
action="http://127.0.0.1:8080/~url=192.168.2.44/openid/exa
mples/server/server.php/login?
8f52be58280899a770ce37a1df46e4b2 ">
<input type="hidden" name="openid_url" value= "pascal" >
<input type="submit" value="Log in" />
<input type="submit" name="cancel" value="Cancel" />
</form>
```

```
POST /openid/examples/server/server.php/login ?
F52be58280899a770ce37a1df46e4b2 HTTP/1.1
Referer:
http://192.168.2.44/openid/examples/server/server.php
Host: 192.168.2.44
Content-Length: 17
Connection: Keep-Alive
Cache-Control: no-cache

openid_url=pascal
```

- **Upon success the authentication response is returned to the browser (11), redirected to the consumer site (12), which finally delivers a *welcome* page (13).**

HTTP/1.1 302 Found
Date: Tue, 17 Mar 2009 09:07:28 GMT
Server: Apache/2.2.6 (Win32) DAV/2 mod_ssl/2.2.6 OpenSSL/0.9.8g mod_autoindex_color PHP/5.2.5
X-Powered-By: PHP/5.2.5
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Thu, 19 Nov 1981 08:52:00 GMT
location: http://192.168.2.35:80/openid/examples/consumer/finish_auth.php?janrain_nonce=**2009-03-17T09%3A07%3A14ZGoN7wY**&openid.assoc_handle=%7BHMAC-SHA1%7D%7B**49bf5e64**%7D%7B**Va0OCQ%3D%3D**%7D&openid.claimed_id=http%3A%2F%2F192.168.2.44%2Fopenid%2Fexamples%2Fserver%2Fserver.php%2Fidpage%3Fuser%3Dmoi&openid.identity=http%3A%2F%2F192.168.2.44%2Fopenid%2Fexamples%2Fserver%2Fserver.php%2Fidpage%3Fuser%3Dmoi&openid.mode=id_res&openid.ns=http%3A%2F%2Fspecs.openid.net%2Fauth%2F2.0&openid.ns.sreg=http%3A%2F%2Fopenid.net%2Fextensions%2Fsreg%2F1.1&openid.op_endpoint=http%3A%2F%2F192.168.2.44%2Fopenid%2Fexamples%2Fserver%2Fserver.php&openid.response_nonce=2009-03-17T09%3A07%3A28Z2Z6LCL&openid.return_to=http%3A%2F%2F192.168.2.35%3A80%2Fopenid%2Fexamples%2Fconsumer%2Ffinish_auth.php%3Fjanrain_nonce%3D2009-03-17T09%253A07%253A14ZGoN7wY&openid.sig=HUBr4K7Ff51FHCywZFux2IcZ9Xg%3D&openid.signed=assoc_handle%2Cclaimed_id%2Cidentity%2Cmode%2Cns%2Cns.sreg%2Cop_endpoint%2Cresponse_nonce%2Creturn_to%2Csigned%2Csreg.email%2Csreg.fullname%2Csreg.nickname&openid.sreg.email=invalid%40example.com&openid.sreg.fullname=Example+User&openid.sreg.nickname=example
Connection: close
Content-Length: 0
Content-Type: text/html

Demonstration 5

The OpenID platform

TELECOM ParisTech

QUESTIONS ?

- EAP Support in Smartcard draft-urien-eap-smartcard-17.txt
- Urien P, Dandjinou M “Designing smartcards for emerging wireless networks”, Seventh Smart Card Research and Advanced Application IFIP Conference, CARDIS 2006, Tarragona-Catalonia, April 19-21 2006.
- Urien, P, Dandjinou, M, “Introducing Smartcard Enabled RADIUS Server”, The 2006 International Symposium on Collaborative Technologies and Systems (CTS 2006) May 14-17, 2006, Las Vegas, Nevada, USA
- Badra, Mohamad, Urien, Pascal, “Adding Identity Protection to EAP-TLS Smartcards”, in proceedings of Wireless Communications and Networking Conference 2007, WCNC 2007, pp 2951-2956, March 11-15 2007, Hong Kong, China
- Pascal Urien, Guy Pujolle, “JavaCard for Emerging WLAN Environments”, JavaOne Technical Session 2007 (TS -0285), JavaOne 2007, May 8-11, 2007, San Francisco, California, USA
- Pascal Urien, Guy Pujolle, "Security and Privacy for the next Wireless Generation", International Journal of Network Management, IJNM, Volume 18 Issue 2 (March/April 2008), WILEY
- Urien, P.; “Smart Card Benefits for Trusted Processing of Keys-Tree in WLANs”, AICT '09. Fifth Advanced International Conference on Telecommunications, 24-28 May 2009 Page(s):375 - 380
- Urien, P.; “Collaboration of SSL smart cards within the WEB2 landscape”, Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on 18-22 May 2009 Page(s):187 – 194