CON9758

Building a Trusted Gateway with Java ME and a Secure Element

Pierre Girard, Security Solution Expert
San-Francisco, October 27, 2015

Agenda

- Gemalto introduction
- Bringing Trust to M2M with Secure Elements
- The Trusted gateway use case
- Developing the building block with Java ME and Java Card

gemalto

Our **purpose**

We enable our clients to bring
trusted and convenient digital services
to billions of people

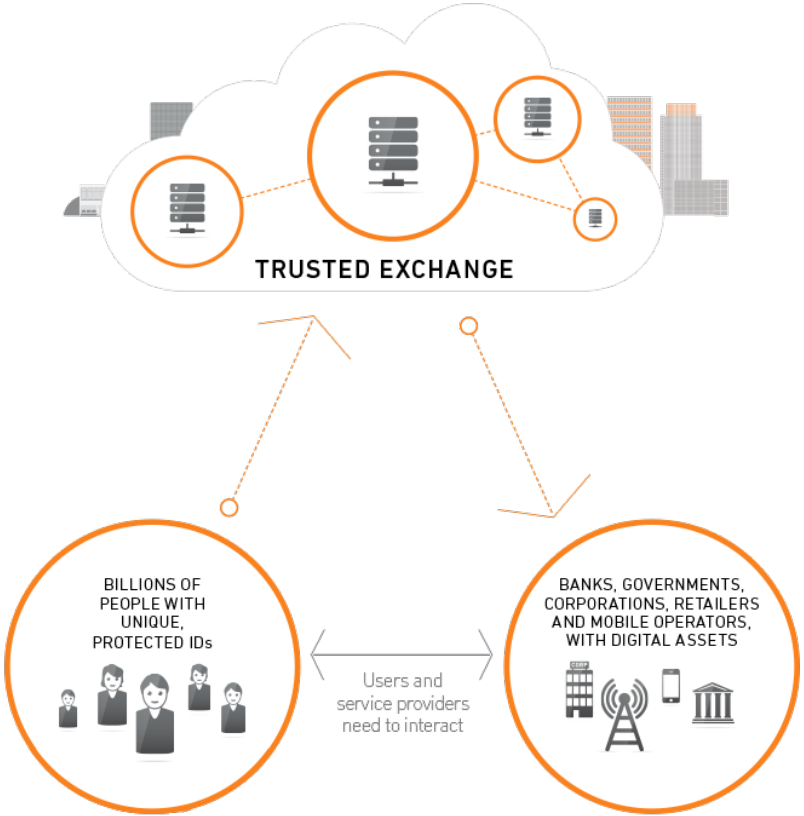
gemalto

We are the **world leader** in digital security

WE'RE UNIQUE. WE'RE GLOBAL. WE'RE INNOVATIVE

Digital security enables **trusted interactions**

PEOPLE
who want to access any service, with any device – and need to use a digital ID.



SERVICE PROVIDERS
who need to check the ID is valid – and to manage and protect the data in their care.

gemalto

We secure and manage **the entire trust chain**

Ensuring strong identities and securing data **from the edge to the core**

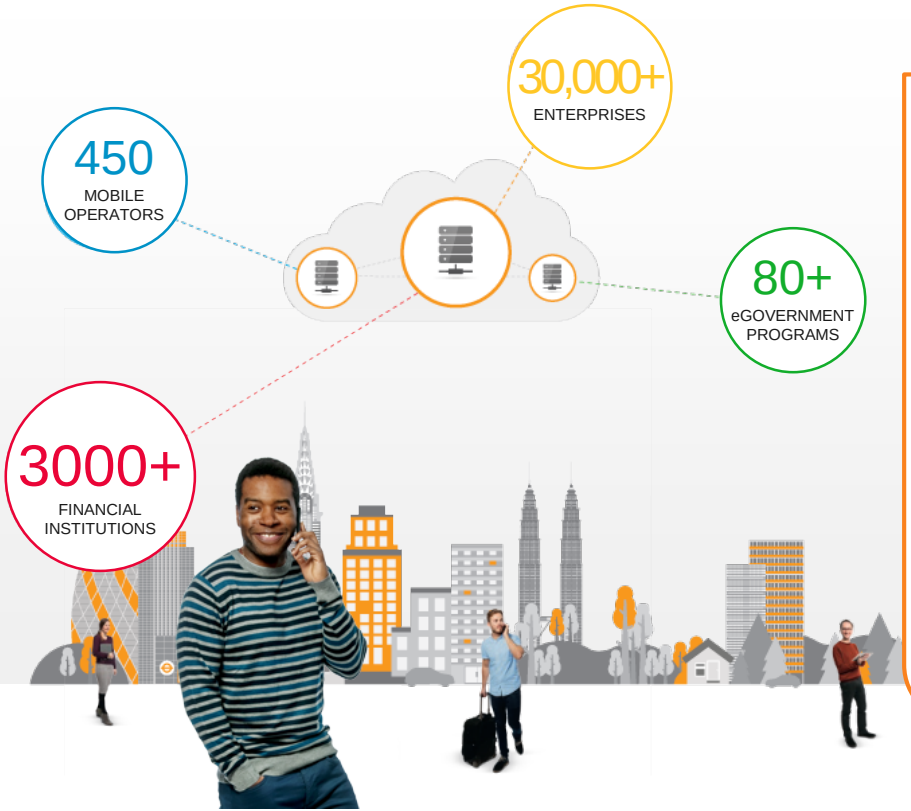
gemalto

Our **seamless chain** of software, products, platforms and services

gemalto

We enable our clients to deliver a vast range of services

Our clients are some of **the world's big brands**



Agenda

- Gemalto introduction
- Bringing Trust to M2M with Secure Elements
- The Trusted gateway use case
- Developing the building block with Java ME and Java Card

gemalto

Typical M2M domains with high security requirements

- M2M : the raise of the connected machines
  - Subject to traditional Internet security concerns
  - + massive deployments
  - + run unmanned, in the field, 24x7 ...
  - + long lifecycle

- Most sensitive domains

Connected cars

Smart energy

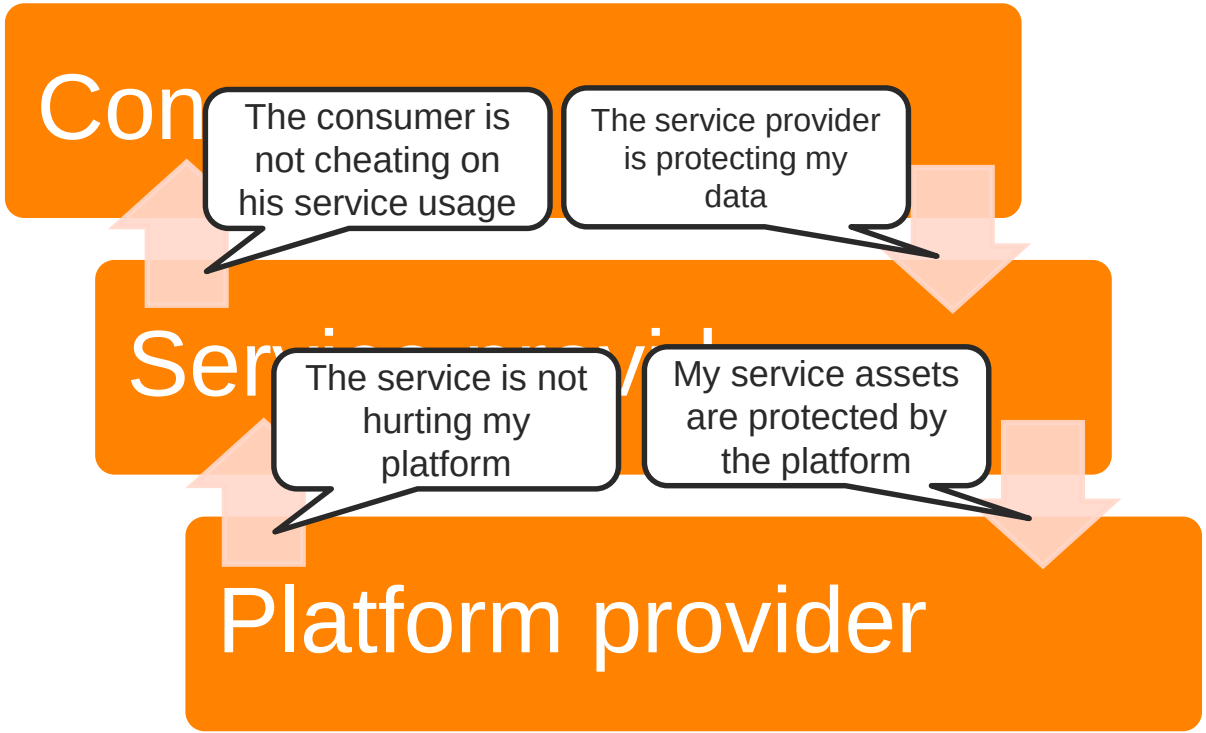
e-health

Smart home

Why do we need trust ?

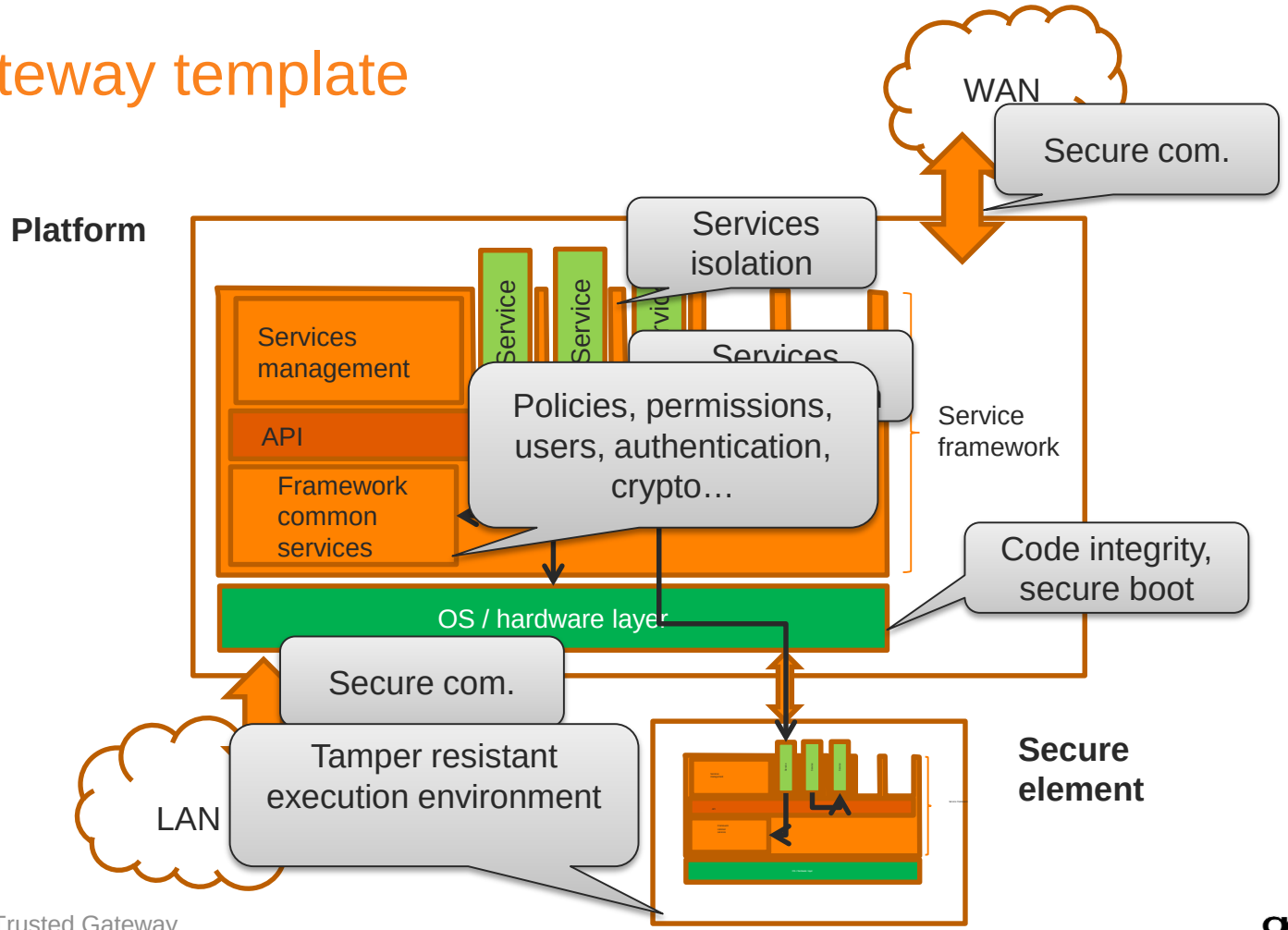
- Management of sensitive devices
  - Car engine, PV arrays, heat pump, home door, …
- Management of sensitive transactions
  - Energy: (not) producing, (not) consuming, storing …
  - X as a Service: mobility, temperature …
  - Peer-to-peer transactions
- Management of sensitive data
  - Location / presence, behavior / consumption patterns, live video / sound streaming, …

Trust relationships



gemalto

A gateway template



gemalto

Software security

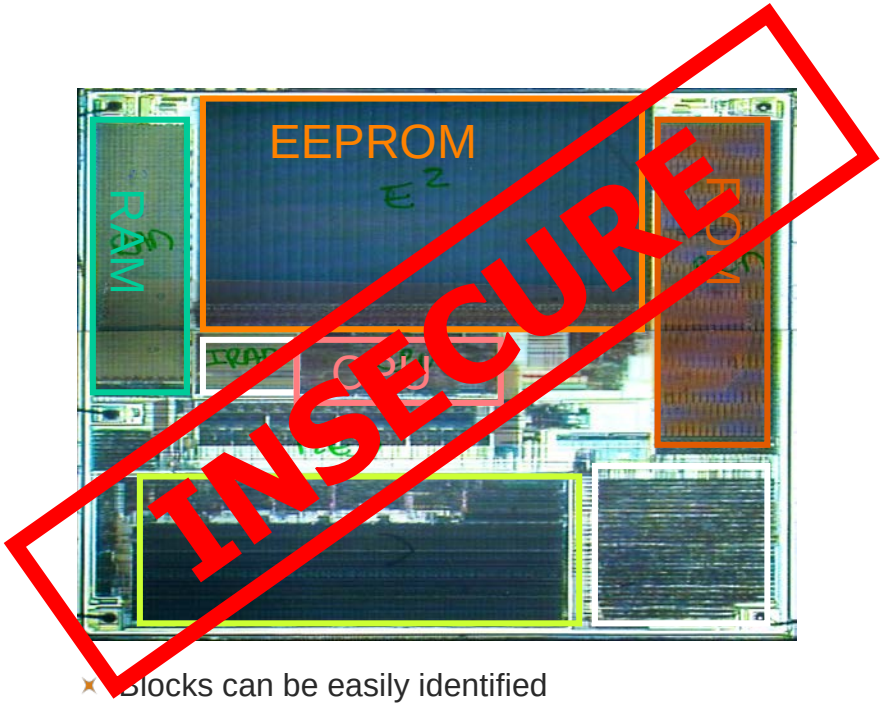
- Protected environment
- Trusted users
- Direct access to data

Hardware security

- Unprotected environment
- Non trusted users
- No direct access to data
- **Tamper resistant devices**

gemalto

Tamper resistance at chip level

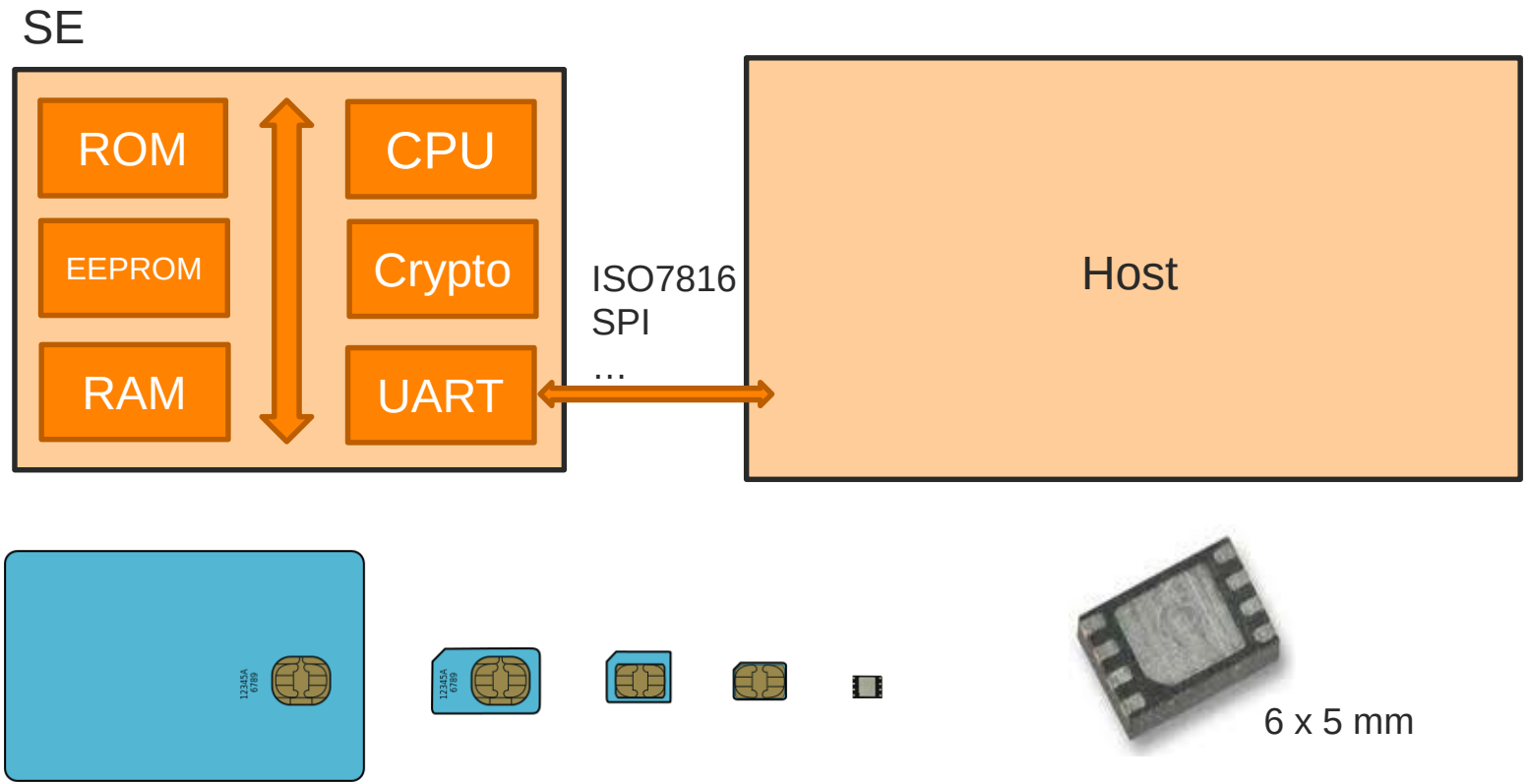


- Blocks can be easily identified
- No shield
- No glue logic
- Buses clearly visible

- Shield
- Glue logic
- No Buses visible
- Memories and buses encryption
- Sensors

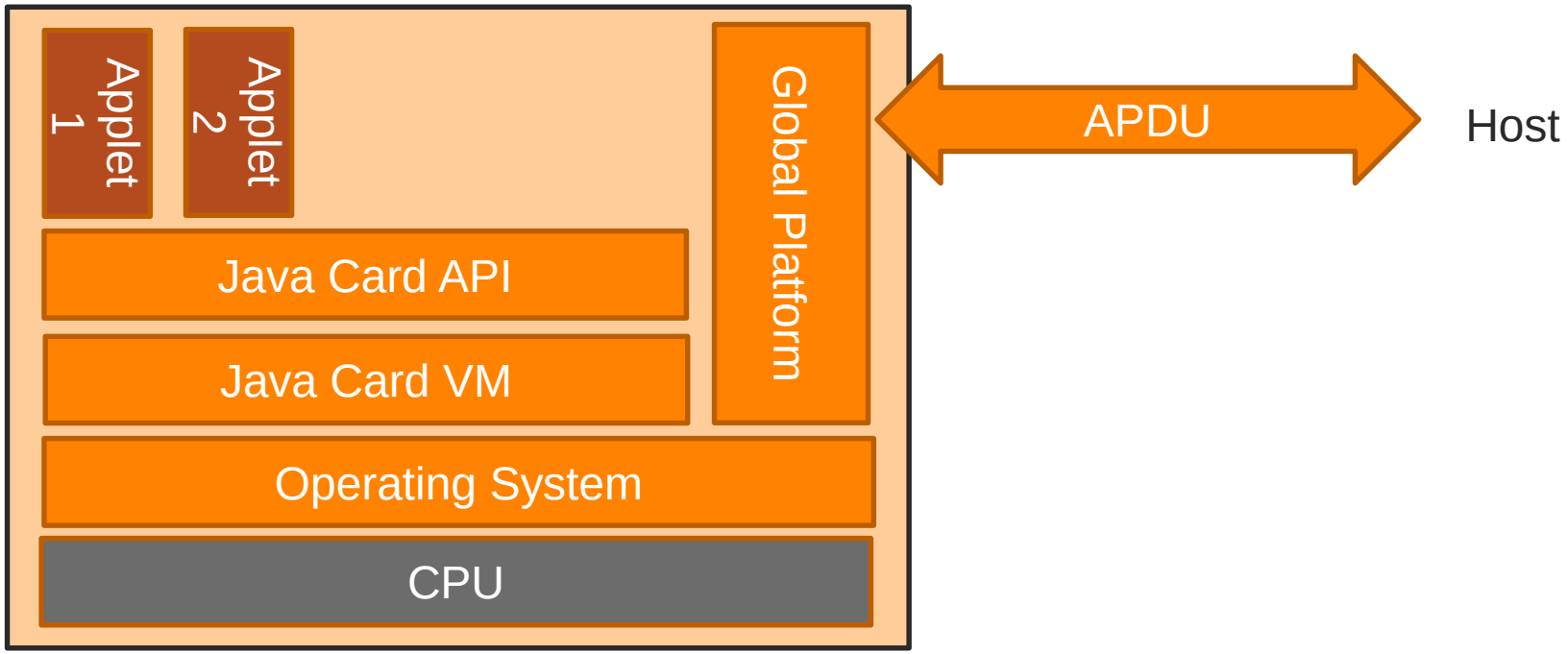
gemalto

HW architecture of a Secure Element



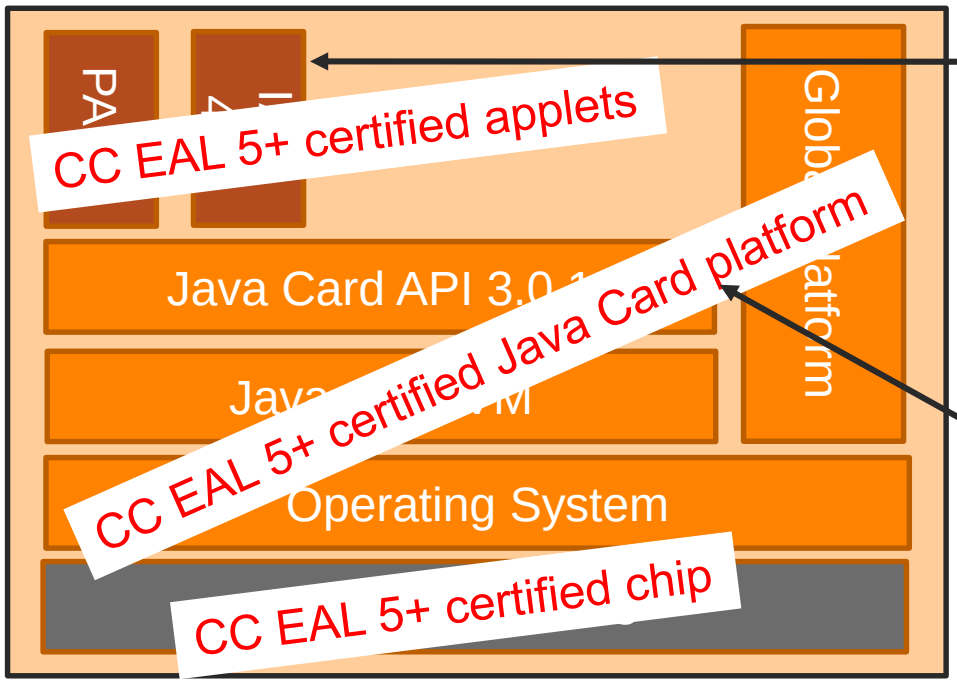
gemalto

SW Architecture of a Secure Element



gemalto

Example of a SE product



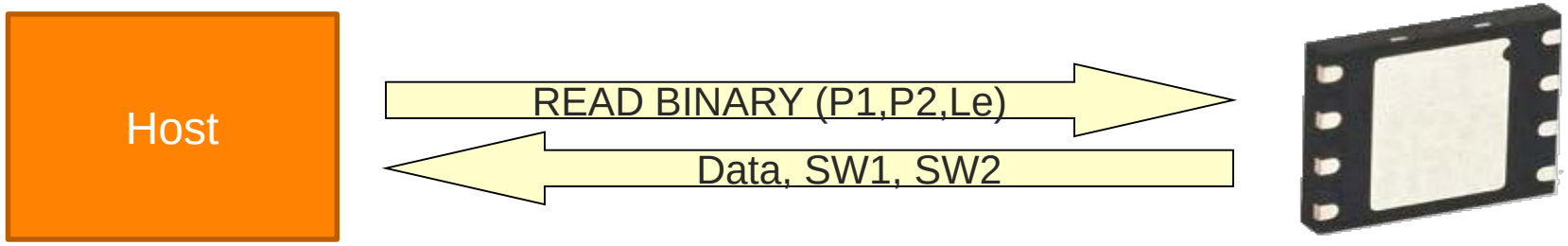
TLS 1.2 support
Data signature / verification
Data encryption / decryption
Certificates storage

RSA 4096
AES 256
SHA 512
ECC 521

Secure Element

gemalto

ISO7816-4 communication

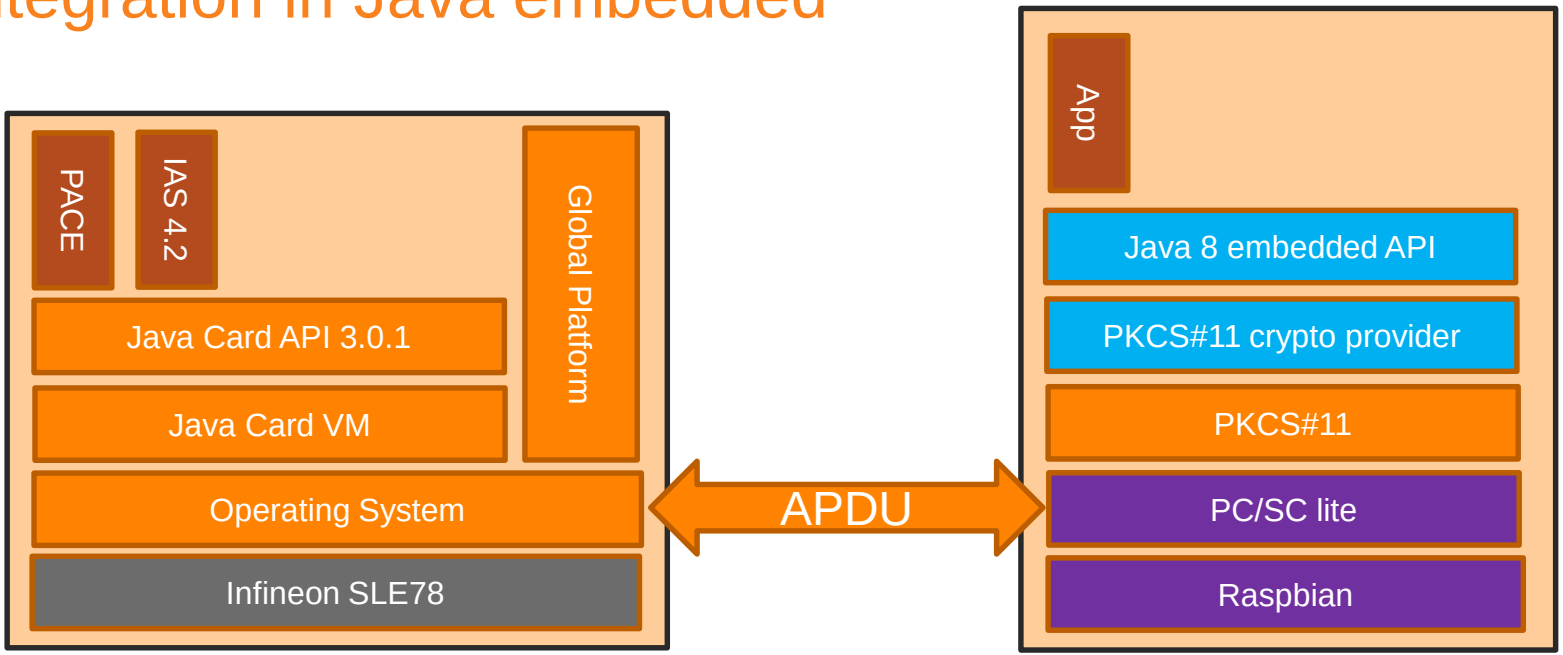


Example:

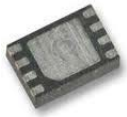
CLA	INS	P1	P2	Le
A0	B0	xx	yy	Le

P1, P2 : specify the data to be retrieved
Le : length of data to retrieve

Integration in Java embedded



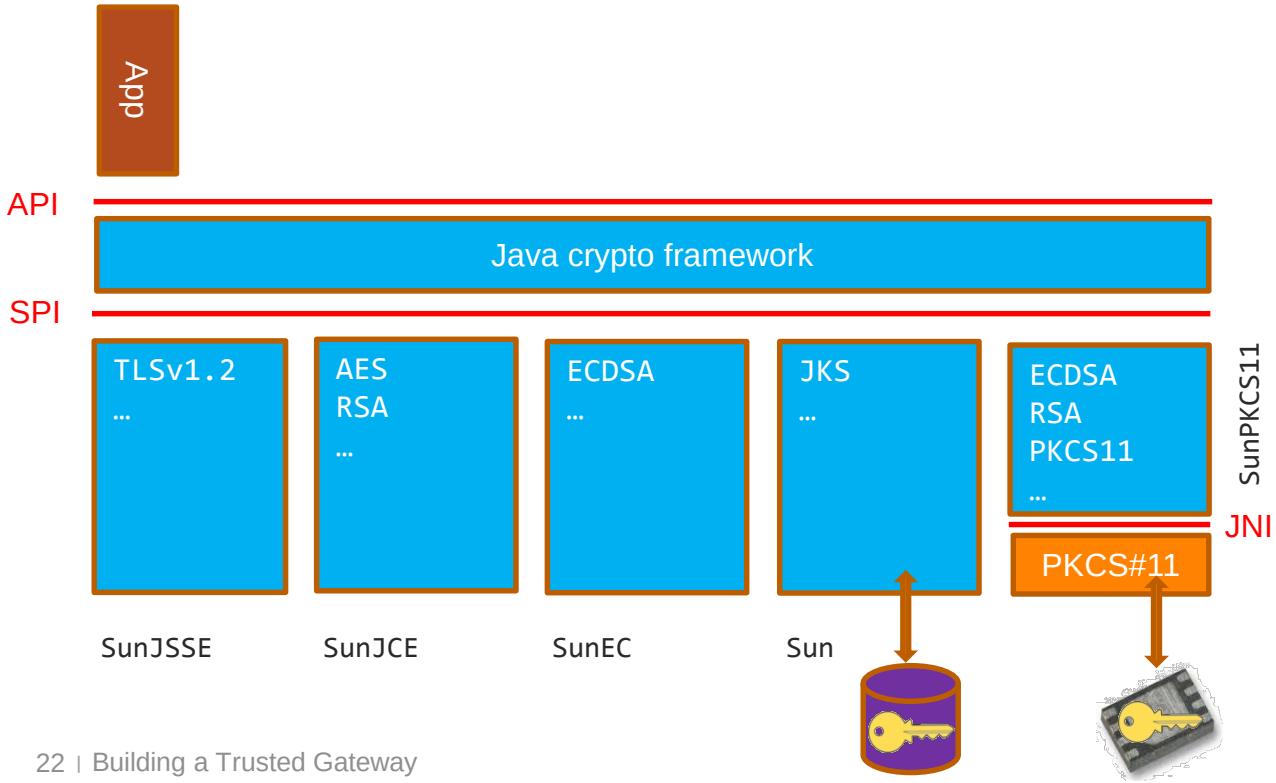
Secure Element



Raspberry Pi

gemalto

Some actual crypto providers



gemalto

Instantiating the Secure Element provider

```
// PKCS#11 configuration file for the Secure Element
private static final String PKCS11_CONFIG = "/home/pi/NetBeansProjects/TestPKCS11onPi/dist/GemaltoPKCS11.cfg";

// Create a PKCS#11 cryptographic provider which uses the Secure Element
Provider myPKCS11Provider = new sun.security.pkcs11.SunPKCS11(PKCS11_CONFIG);

// The PIN code protecting the Security Element
char [] myPIN = {'0','0','0','0'};

// Create a KeyStore corresponding to the Secure Element
KeyStore.PasswordProtection pinProtection = new KeyStore.PasswordProtection(myPIN);
KeyStore.Builder  ksb =  KeyStore.Builder.newInstance("PKCS11", myPKCS11Provider, pinProtection);
KeyStore ks = ksb.getKeyStore();

// Add the SE as a cryptographic provider (useful when it is not possible to pass a provider explicitly)
Security.addProvider(myPKCS11Provider);
…
```

Switching to hardware security is easy: EC signature

- Signing with software provider

```
char [] myPassword = {'1','2','3','4'};

// Let's sign a message
String s1 = "Les hommes naissent et demeurent libres et égaux en droits.";

// We sign with ECDSA
Signature ecSign = Signature.getInstance("SHA256withECDSA");

// Retreive the signature key in keystore by it’s alias
PrivateKey privKey = (PrivateKey) ks.getKey("SignKey", myPassword);

// And we sign !
ecSign.initSign(privKey);
ecSign.update(s1.getBytes());
byte[] signature = ecSignCard.sign();
```

Switching to hardware security is easy: EC signature

- Signing with Secure Element provider

```
char [] myPIN = {'0','0','0','0'};

// Let's sign a message
String s1 = "Les hommes naissent et demeurent libres et égaux en droits.";

// We sign with ECDSA
Signature ecSign = Signature.getInstance("SHA256withECDSA", myPKCS11Provider);

// Retreive the signature key in keystore by it’s alias
PrivateKey privKey = (PrivateKey) ks.getKey("SignKey", myPIN);

// And we sign !
ecSign.initSign(privKey);
ecSign.update(s1.getBytes());
byte[] signature = ecSignCard.sign();
```

How about Java ME ?

Agenda

- Gemalto introduction
- Bringing trust to M2M with Secure Elements
- The trusted gateway use case
- Developing the building block with Java ME and Java Card

Use case: connected home / alarm gateway

- Multiple industries are fighting to become the connected hub in the home: example in US

MNO

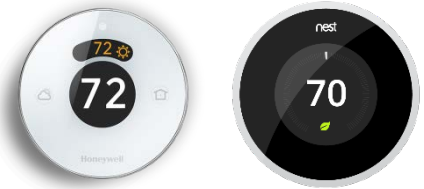
AT&T Digital Life®

Broadband

xfinity

Alarm System

Home Automation



Increased volumes and adoption will increase attractiveness of hacking

gemalto

Risks in today's devices

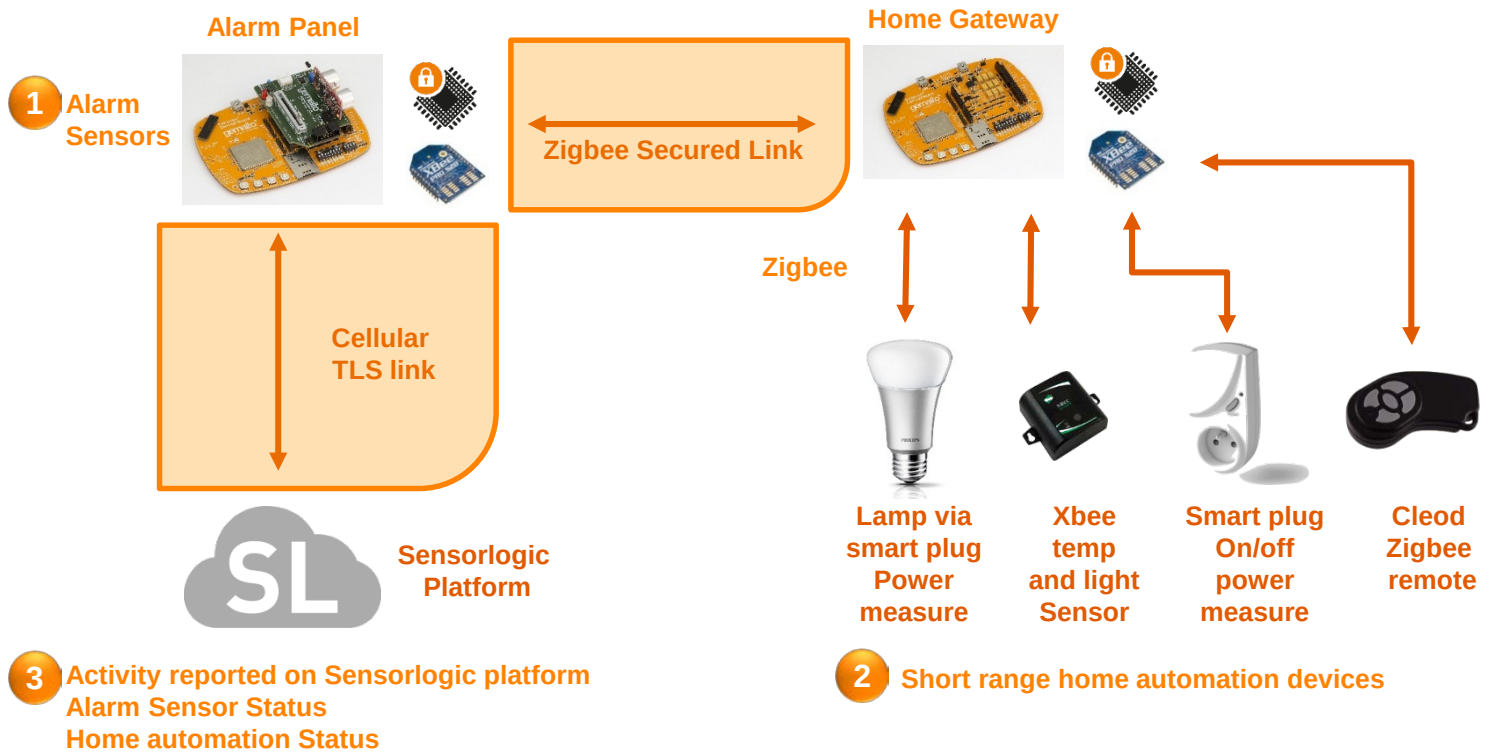
- Lack of strong authentication of the device
- Fake devices can be introduced in the system and interact with Service provider backend
- Basic ID Diversity scheme can be uncovered through brute force or social engineering
- Data is typically not strongly encrypted / authenticated
- Lack of Hardware tamper resistance will allow motivated hackers to enter the system either locally or remote

Secure Element added value in the gateway

- Tamper resistance
- Personalization unique to each device: strong authentication of the field devices to the server
- Client authentication of the WAN client to the backend
- Strong applicative encryption portable on various short range technologies

gemalto

Demo view



gemalto

Hardware set-up for fast prototyping

Gemalto concept board

- 2G/3G wireless module, Java ME
- Arduino compatible extension

Arduino XBee

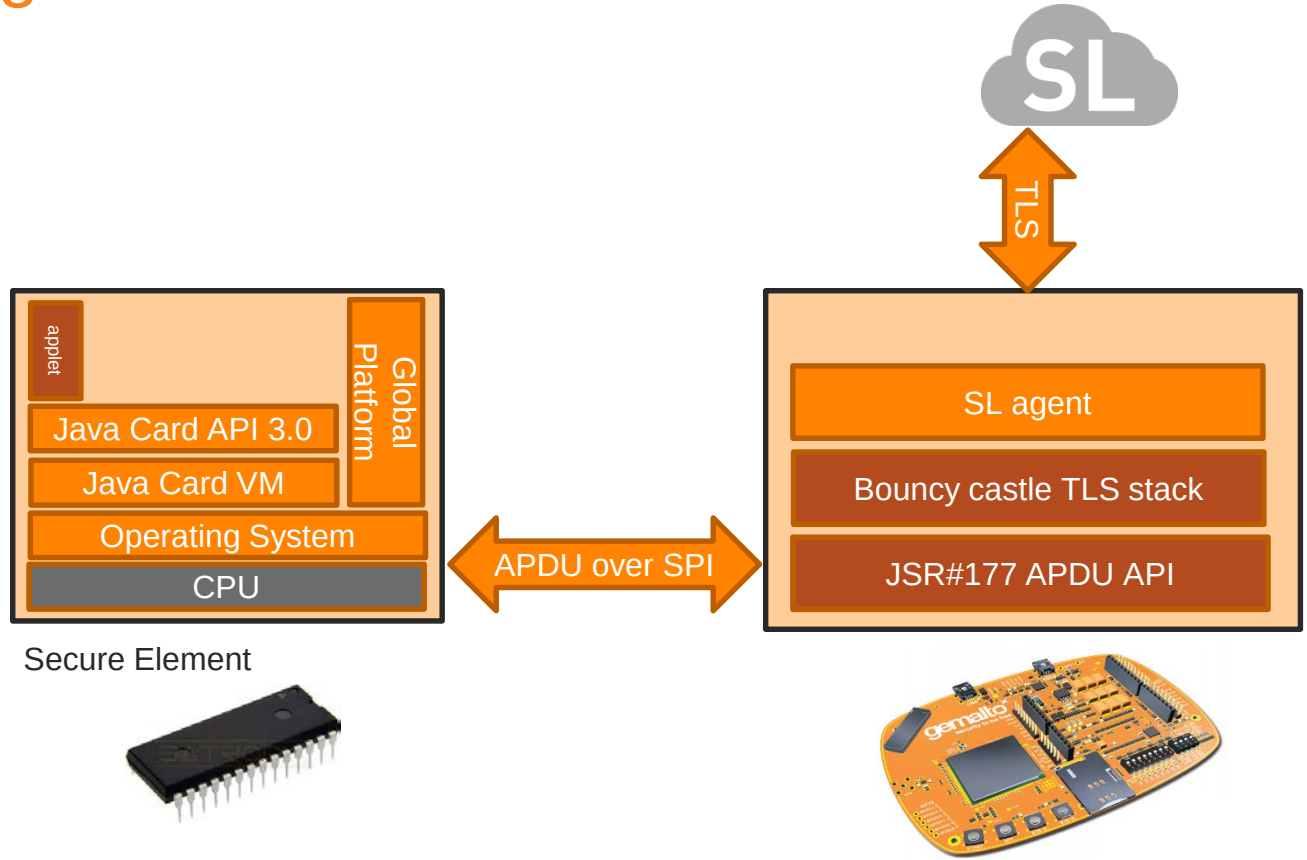
- ZigBee

SE shield

- DIL prototypes
- SPI communication

gemalto

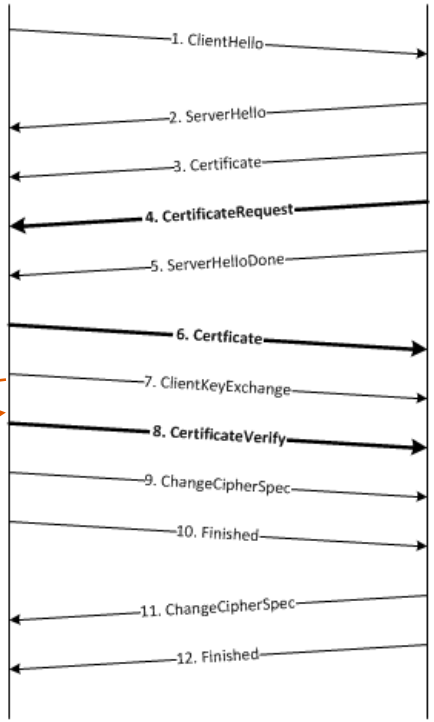
Using SE for TLS client authentication



TLS handshake

RSA key pair for client authentication

RSA sign signature



gemalto

Agenda

- Gemalto introduction
- Bringing trust to M2M with Secure Elements
- The trusted gateway use case
- Developing the building block with Java ME and Java Card

Developing the missing parts

- Generate the signature on the SE
  - Standard applets are available, but let’s build a demo one
- Communication with the Secure Element
  - Provide JSR#177 standard library for APDU communication
- Extension of Bouncy Castle to support an SE
  - Native Java ME TLS stack cannot use the SE

A simple Java Card applet for signing demo

4.1.3 RSA SIGN

This command perform a raw RSA 1024 signature on the submitted data. The card response length is 0x80 bytes if the signature process succeeds, 0x02 otherwise.

Command:

CLA	INS	P1	P2	Lc	Data
00	03	00	00	80	Data to be signed

Response:

RSA signature or error code	SW1	SW2

If the response length is 0x02 it contains an error code:

Data byte 1	Data byte 2	Reason
00	01	Illegal value
00	02	Uninitialized key
00	03	No such algorithm
00	04	Invalid initialisation
00	05	Illegal use

A simple Java Card applet for signing demo

```
public void process(APDU apdu) {

   // get the APDU buffer
   byte[] apduBuffer = apdu.getBuffer();

   switch (apduBuffer[ISO7816.OFFSET_INS]) {

      // Sign raw RSA
      case 0x03:

        // P1, P2 and Lc checks here

        // Receive the data
        apdu.setIncomingAndReceive();

        try {
           rsa.init(privKey, Cipher.MODE_DECRYPT);
           rsa.doFinal(apduBuffer, ISO7816.OFFSET_CDATA, (short) 128, result, (short) 0);
           Util.arrayCopy(result, (short) 0, apduBuffer, (short) 0, (short) result.length);
           apdu.setOutgoingAndSend((short)0 , (short) 128);
        } catch (CryptoException e) {
           Util.setShort(apduBuffer, (short) 0, e.getReason());
           apdu.setOutgoingAndSend((short)0 , (short) 2);
        }

        ISOException.throwIt(ISO7816.SW_NO_ERROR);
      break;
```

We are using raw RSA for demo purpose, real applet process padding internally

JSR#177 SATSA: Security And Trust Services API

- Provides 4 independent and optional packages for Java ME
- SATSA – APDU: API for APDU communication
  - No security as such, pure communication layer
- SATSA –JCRMI: API for RMI on Java Card objects
  - Obsolete (JCRMI didn't take off)
- SATSA – PKI: API for data signature
- SATSA – CRYTO
  - Provides crypto algorithms for encryption, hash, signature verification

JSR#177 on concept board

- Only SATSA-CRYTO is natively present
  - Software implementation only
  - No signature in the API

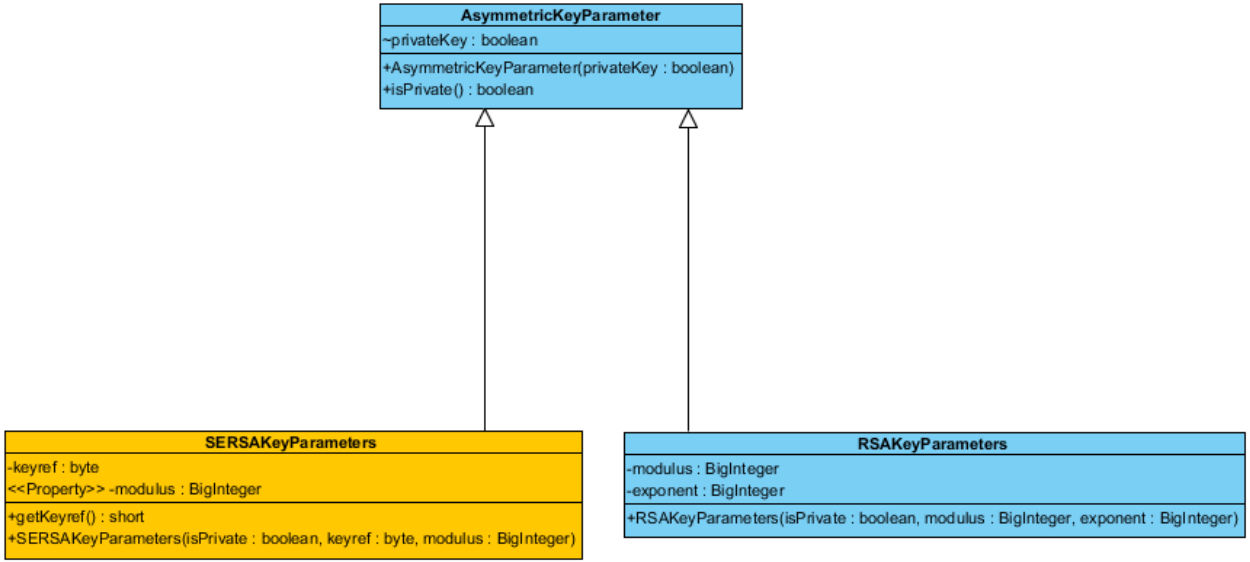
- We decided to implement a SATSA-APDU in Java
  - Based on GPIO API
  - Provides `javax.microedition.apdu.APDUConnection` interface

Using Bouncy Castle TLS stack

```
SocketConnection sc = (SocketConnection) Connector.open("socket://myserver.com:443");
DataOutputStream os = sc.openDataOutputStream();
DataInputStream  is = sc.openDataInputStream();
SecureRandom mySecureRandom =  new SecureRandom();

TlsClientProtocol tlscp = new TlsClientProtocol(is, os, mySecureRandom);

TlsClient tlsc = new DefaultTlsClient(){
   public TlsAuthentication getAuthentication() throws IOException {
      return new TlsAuthentication() {

         // Callback when receiving server certificate
         public void notifyServerCertificate(Certificate serverCertificate) throws IOException {
                // Check the certificate chain wrt to the Root CA certificate.
                …
         }

         // Callback when client CertificateVerify shall be sent
         public TlsCredentials getClientCredentials(CertificateRequest certificateRequest) throws IOException{
            // Factory class can decide which credential to use based on TLS context (e.g. the server address)
            // Can instatiate DefaultTlsSignerCredential or our SETlsSignerCredential
            return TlsSignerCredentialsFactory.getInstance(context);
         }

      };
   }
 };

tlscp.connect(tlsc);
```

Bouncy Castle extension

- Seven classes added to Bouncy Castle
  - New SE RSA key parameters and the associated signature classes



gemalto

Core RSA signature engine (simplified)

- Remove all actual RSA computations and delegate to SE

```
public BigInteger processBlock(BigInteger input)
   {
       byte[] signature = new byte[128];
       byte[] toBeSigned = input.toByteArray();

       // We check make sure that the input length is 128 bytes padded with leading 0 bytes
       byte[] data = new byte[128];
       System.arraycopy(toBeSigned, 0, data, 128 - toBeSigned.length, toBeSigned.length);

       final String AID ="A0.0.0.0.18.50.0.0.0.0.0.0.52.41.44.41";
       try {

           APDU apdu = new APDU(AID);

           // Sending the APDU to the SE: CLA=0x00, INS=0x03, P1=P2=0x00
           byte[] response = apdu.sendAPDU((byte)0x00, (byte) 0x03, (byte) 0x00, (byte) 0x00, new byte[] {(byte)128}, new byte[] {(byte)128}, data);

           // Copy result
           System.arraycopy(response, 0, signature, 0, 128);

        } catch (Exception e) {
           System.out.println("Cannot connect to the SE");
           e.printStackTrace();
       }

       return new BigInteger(signature);

   }
```

Take away

Gemalto has the techno bricks to prototype quickly a secure home gateway

Secure Element can be used to secure heterogeneous networks

SensorLogic Cloud Platform allows you to quickly and securely deploy your prototypes

gemalto

Thank you