Towards Trusted Operated Services in the Internet of Things

Pascal Urien
LTCI UMR 5141
Telecom ParisTech
23 av d'Italie, 75013, Paris, France

***Abstract*— This paper presents an innovative concept for the Internet of Things (IoT), in which objects work over TLS stacks running in secure elements. We notice that most of today IoT architectures are secured by the DTLS or TLS stack. Furthermore, tamper resistance, secure communications and storage are consensual requests for the emerging IoT frameworks. We demonstrate that it is possible to design cheap secured and trusted systems based on Javacards plugged in commercial nano-computers. Finally we detail the structure of an innovative JAVA framework able to provide trusted operated services, in a way similar to mobile network operators (MNO) managing smartphone fleets thanks to Subscriber Identity Modules (SIMs).**

Keywords-. IoT; Secure Elements; TLS; DTLS; Security.

I. INTRODUCTION

The Internet of Things (IoT) is a major topic for the development of the digital economy; in [8] it is defined as "*a network of connected things*". According to [1] about 50 billion of connected objects are forecasted by 2020, about 6.6 per human being. It is expected [2] that "today households, across the OECD (*Organisation for Economic Co-operation and Development*) area, have an estimated 1.8 billion connected devices, in 2017 this could be 5.8 billion and in 2022, 14 billion devices". More than 50 connected objects could be located in four people house, such as computers, smartphones, tv, cars, internet connected power sockets, energy consumption display, thermostat, camera, and connected locks.

According to [6] the digital industry roadmap for next decades will not be centered on Moore's law but will deploy networks with hundred trillions of devices [7]. In that context [7] "*Security is projected to become an even bigger challenge in the future as the number of interconnected devices increases... In fact, the Internet of Things can be viewed as the largest and most poorly defended cyber attack surface conceived by mankind*".

As an illustration [3] introduces an IoT service-oriented architecture (SoA), based on the following four layers (see Figure 1) :

- The Sensing layer that comprises hardware objects and acquisition protocols.
- The Network layer, which is the infrastructure needed for the information transport. New wireless networks such as SigFox [4] or Lora [5] have been recently designed for low throughout interactions with sensors.
- The Service layer that manages services needed by users or applications.
- The Interfaces layer that includes API (*Application Programming Interface*) and applications front ends.

This model suggests that some objects could be remotely managed by dedicated service providers. For example, in a smart grid context, connected plugs are remotely switched on in order to enable the battery recharge of electric cars. As underlined in [7] security is a major prerequisite and "*a short list of requirements includes tamper resistance and secure communications and storage*".

In this paper we propose a new perspective for the IoT security, based on cheap secure elements enforcing secure communications for connected devices. Most of legacy devices are monitored thanks to the HTTPS protocol. The IETF (*Internet Engineering Task Force*) comity is currently pushing a framework based on the CoAP (*Constrained Application Protocol*) protocol [12] whose security natively relies on the DTLS (*Datagram Transport Layer Security*) protocol and soon on TLS (*Transport Layer Security*) protocol [13]. We present an innovative concept in which TLS servers are fully running in secure elements. We describe a Java framework that enables the integration of such tamper resistant components in cheap boards, for example fuelled by nano-computer such as the popular *Raspberry Pi* (*www.raspberrypi.org*).

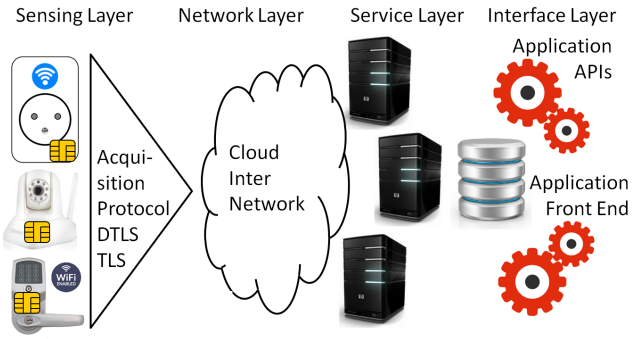

Figure 1. An IoT service-oriented architecture (SoA), based on four layers

The paper is constructed according to the following outline. Section 2 details some acquisition protocols in today IoT systems. Section 3 introduces the concept of TLS server

 ISBN: 978-1-61208-505-0

running in secure elements. Section 4 describes an experimental JAVA framework designed for nano-computers, such as the popular Raspberry Pi, whose communication security, is enforced by secure elements. Finally, section 5 concludes this paper.

II. ACQUISITION PROTOCOLS

Today many connected devices are using HTTPS, i.e. communication with web servers secured by the TLS protocol. As an illustration the popular Nest thermostats work [7] with JSON (*JavaScript Object Notation*) formatted data POSTed to their web servers; some connected plugs [8] use *Home Network Administration Protocol* (HNAP) a proprietary network protocol based on SOAP (*Simple Object Access Protocol*), invented by Pure Networks, Inc. and acquired by Cisco Systems, which allows identification, configuration, and management of network devices.

The *Constrained Application Protocol* (CoAP) [12] is designed according to the *Representational State Transfer* (REST) architecture [11], which encompasses the following six features: 1) Client-Server architecture; 2) Stateless interaction; 3) Cache operation on the client side; 4) Uniform interface ; 5) Layered system ; 6) Code On Demand.

CoAP is an efficient RESTfull protocol easy to proxy to/from HTTP, but which is not understood in an IoT context as a general replacement of HTTP. It is natively secured by DTLS (the datagram adaptation of TLS), and works over a DTLS/UDP/IP stack. Nerveless the IETF is currently working [13] on a CoAP version compatible with a TLS/TCP/IP stack.

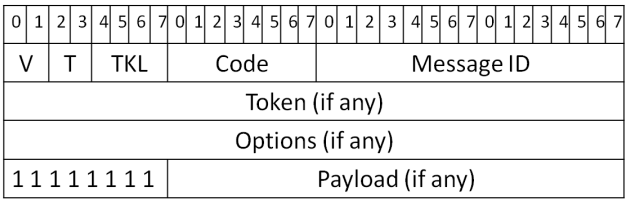

Figure 2. The CoAP Header

The CoAP header is illustrated by Figure 2. Version (V) is the protocol version (01). Type (T) indicates if the message is of type Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) or Reset. Token Length (TKL) is the length of the Token field (0-8 bytes). The Code field identifies the method and is split in two parts a 3-bit class and a 5-bit detail documented "c.dd" where "c" is a digit from 0 to 7 and "dd" are two digits from 00 to 31. For example methods named GET, POST, PUT and DELETE are respectively noted 0.01, 0.02, 0.03, and 0.04. The attribute Message ID matches messages ACK/Reset to messages CON/NON previously sent; it is usually noted inside two brackets ([0xMessageID]). The Token (0 to 8 bytes) is used to match a response with a request. Options give additional information such as *Content-Format* dealing with proxy operations.

According to the CoAP model objects act as "servers". Clients deliver requests to servers that return responses and may proxy HTTP requests.

Some IoT frameworks (for example the ARM® mbed™ IoT Device Platform, see Figure 3) are supporting the MQTT (*MQ Telemetry Transport*) protocol [14], a Client Server publish/subscribe messaging transport protocol that is secured by TLS.

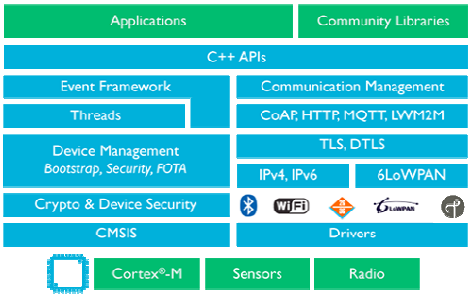

Figure 3. MBED stack from the ARM company

From the above lines it appears that TLS and DTLS are the security cornerstones of many IoT stacks. We believe that the integration of TLS servers in cheap tamper resistant chips enforces secure communications and storage. It could enable trusted and operated IoT infrastructure. Furthermore TLS/DTLS servers perform strong mutual authentication with clients when both entities are equipped with private keys and certificates, used for object identities.

III. TLS SERVERS FOR SECURE ELEMENTS

A secure element [19] is a secure micro controller, whose area is about 5x5 mm^2. ISO7816 standards specify electrical and logical interfaces; small messages whose size is less than 256 bytes are exchanged over serial or USB interfaces. It is usually glued in PVC rectangular supports referred as smartcards. Nerveless these tamper resistant devices can be shrunk in other electronic dies such as NFC controllers or SD memories. Most of secure elements include a Java Virtual Machine (JVM) and therefore run applications written in the javacard language [18], a subset of JAVA. They include cryptographic libraries providing symmetric procedures (3xDES, AES), asymmetric algorithms (RSA, ECC) and hash functions (MD5, SHA1, SHA2..).

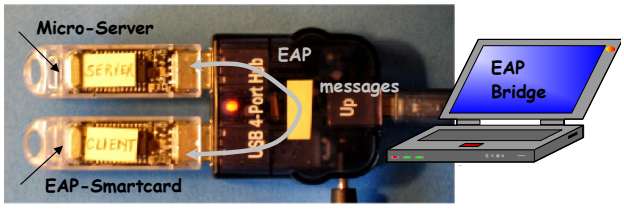

Figure 4. The first TLS micro-authentication server [15], 2005

The first micro-authentication server, illustrated by Figure 4 was introduced ten years ago in [15]. It was running in a javacard , including a TLS stack embedded application. This former TLS stack booted a TLS session in about 30s.

 ISBN: 978-1-61208-505-0

Ten years later, secure elements perform this task in about 1 to 10s for TLS full sessions and about 0,5 to 5s for TLS resume sessions.

According to [16] embedded TLS server interface is based on the EAP-TLS protocol, whose packets are transported over ISO 7816 messages. Figure 5 illustrates the choreography of these exchanges.

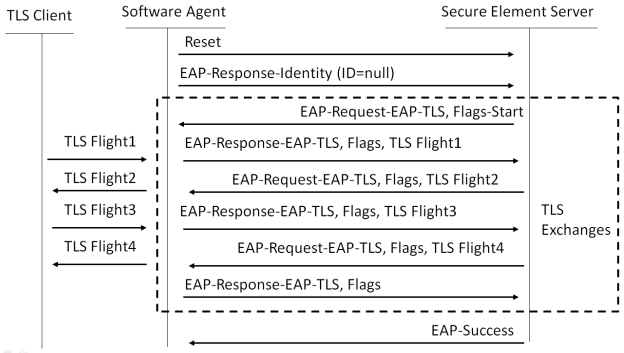

Figure 5. Booting of a TLS session from a secure element with an interface based on EAP-TLS over ISO 7816 [16][17]

A recent IETF draft [17] introduces TLS/DTLS security modules dedicated to secure elements. TLS/DTLS sessions are booted according to [16], but afterwards the TLS secure channel can be used for the decryption of data sent by the client or the encryption of information to be transmitted to the client (see Figure 6). Software agents mentioned in Figure 5 and 6 are logical entities that drive the secure elements. They act as a logical bridge between the network transporting TLS packets over TCP/IP and the secure element dealing with EAP-TLS messages shuttled in ISO7816 requests/responses.

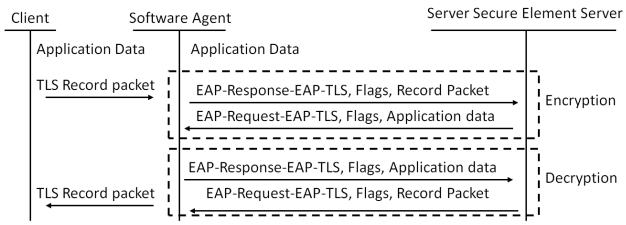

Figure 6. Application data encryption and decryption performed by a TLS server (i.e. a RecordLayer) running in a secure element [17].

It should be noted than in a previous work [20] we suggested to export TLS sessions from secure elements according to a technology named *TLS-Tandem*. Therefore, upon opening a TLS session two choices are possible : 1) processing TLS packet ciphering/deciphering in the secure element OR 2) performing this task in the application that drives the tamper resistant chip.

The TLS server javacard application is designed according to the *OpenEapSmartCard* framework previously detailed in [21] and illustrated by Figure 7.

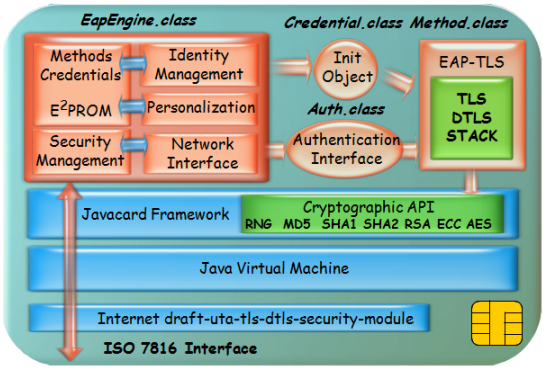

Figure 7. Javacard TLS/DTLS stack framework according to the OpenEapSmartcard [21] framework.

IV. JAVA FRAMEWORK

Today many objects are working in a LINUX software environment. For example the popular Raspberry Pi nano computer is powered by a DEBIAN operating system (see Figure 8). It supports the PCSC-Lite (*Personal Computer/Smart Card*) middleware developed by the M.U.S.C.L.E (*Movement for the Use of Smart Cards in a Linux Environment*) organization. Furthermore the JAVA framework (up to the 1.5 version) is also available. PCSC-Lite can be easily linked with the javax.smartcardio JAVA package, which provides a set of smartcard I/O APIs.

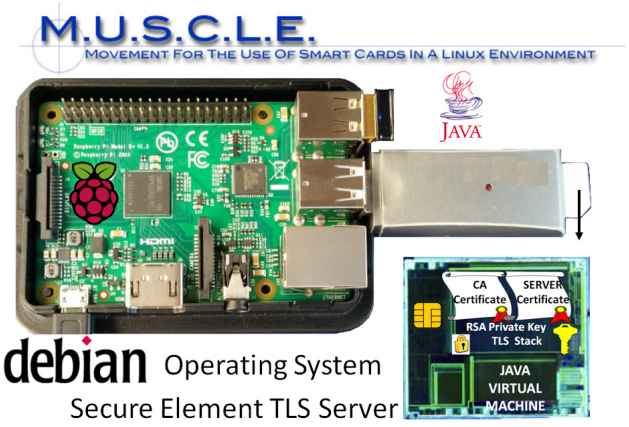

Figure 8. TLS Server satck embeded in secure element, in a Raspberry Pi environnment.

Therefore it is possible to deploy TLS servers running in secure elements for this class of objects and to control these chips, thanks to a dedicated API (SE-TLS API) described below. This approach facilitates the design of secure communications and storage.

 ISBN: 978-1-61208-505-0

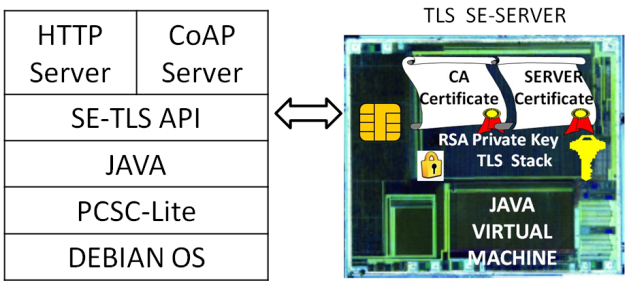

Figure 9. Software stack for SE based TLS server, such as HTTP or CoAP

The software for HTTP and CoAP servers based on secure elements is depicted by Figure 9. The main JAVA package needed by a server daemon is the SE-TLS API, which comprises three logical components: the core implementation, the ServerTLS thread, and the GenericServer class.

The *core implementation* of the server daemon is illustrated by Figure 10. It begins by the instantiation of a tls-tandem object (named *myserver*) that initializes a TLS socket server (on the 443 port) and resets the secure element. The secure element embedded TLS stack, written in javacard, is identified by the parameter AID. It requires a PIN value, and is associated to an identity attribute "server" defining a set of cryptographic attributes such as certificates and private key.

```
public static void ServerDaemon()
{
 tls-tandem myserver= new
 tls-tandem(tls_tandem.SERVER, readername,
            AID, PIN , "server");
 myserver.ServerTLS.rdv    = false;
 myserver.ServerTLS.InUse = false;

 while(true)
 { //myserver.ServerTLS.rdv = true;
   recordlayer RecordLayer =
   myserver.OpenSession();
   myserver.ServerTLS.rdv =    false;
   myserver.ServerTLS.InUse = false;
 if (RecordLayer == null);
 else
 { GenericServer myGS= new
   GenericServer(myserver,RecordLayer);
  // myserver.CloseSession(RecordLayer);
  }
 }
```

Figure 10. Core implementation of a SE based TLS server

During the instantiation of the tls-tandem class, the ServerTLS thread (see Figure 11) is created. It deals with two control (boolean) flags *rdv* and *InUse*.

The Rendez Vous mechanism is a fundamental paradigm for a SE based server. An incoming TCP connection is denied if no logical entity is ready for its processing; this availability is indicated by the *rdv* flag. Afterwards the *InUse* flag is set and the ServerTLS thread will remain idle until its resetting. The *InUse* variable means that the secure element is currently computing a TLS session, implying that no new incoming client can be processed. In that case incoming TCP sessions are stored in the backlog queue, whose size is fixed by the JAVA constructor of the ServerSocket class (see Figure 11). According to the JAVA documentation the default value is set to 50. The backlog queue size tunes the number of TLS sessions that can be delayed before being processed by the secure element.

The OpenSession() method (see Figure 10) provided by the tls-tandem object initializes a Rendez Vous (rdv= true) with a TCP client. The secure element is thereafter in use (InUse =true) and a software agent (as illustrated by Figure 5) boots the TLS session. Upon success a RecordLayer object is created (see Figure 10) and returned by the OpenSession() procedure.

At this step the Rendez Vous is cancelled (rdv =false). The RecordLayer class manages a TCP socket and provides the procedures (see Figure 12) needed for exchanging TLS record packets over TCP/IP. If the TLS session is exported from the secure element then the InUse flag may be reset. Otherwise the secure element remains busy (inUse= true) and incoming TCP connections are delayed.

```
// ServerTLS Thread

ServerSocket soq = new
ServerSocket(443,backlog,
InetAddress.getByName("0.0.0.0"));
InUse=false;
while(true)
{ client = null ;
  try {Socket client= soq.accept();}
  catch(IOException e){client=null;}
  if (!rdv)
  { try {client.close();client=null;}
    catch(IOException f){}
  }

  if (client != null) InUse=true;
  while(InUse)
  {try  { Thread.sleep((long)100);}
   catch (InterruptedException e){};
  }
  if (client != null)
  { try { client.close();}
    catch(IOException e){};
  }
}
// End of ServerTLS
```

Figure 11. The ServerTLS thread

The GenericServer class (see Figure 10) is an implementation of an HTTP or CoAP server; as illustrated by Figure 6 it is a functional subset of the Software Agents. It uses services provided by the RecordLayer object (see

 ISBN: 978-1-61208-505-0

Figure 12) dealing with TLS packets reception and decryption, or TLS packets encryption and transmission. These operations may be performed in the secure element or by pure software means if the TLS session has been previously exported.

```
byte[] buf = RecordLayer.recv();
if (buf !=null);
buf = RecordLayer.decrypt(buf);

byte[] buf=RecordLayer.encrypt(buf;
int err = RecordLayer.send(buf) ;
```

Figure 12. TLS packets processing by the RecordLayer object.

V. CONCLUSION

In this paper, we have demonstrated a TLS server running over a secure element in an object environment, i.e. a cheap nano-computer with an LINUX operating system, similar to many devices already available on the IoT. Thanks to this innovation we claim trusted and secured communications booted from a strong mutual authentication. In a way similar to SIM modules managed by mobile network operator (MNO) we believe that this paradigm is a step towards IoT infrastructure remotely controlled by IoT operators.

REFERENCES

[1] D. Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", Cisco, White Paper, 2011

[2] OECD, "Building Blocks for Smart Networks", OECD Digital Economy Papers, No. 215, OECD Publishing, 2013

[3] L. D. X. Shancang Li and S. Zhao, "The internet of things: a survey", Information Systems Frontiers, vol. 17, pp. 243–259, 2015.

[4] LoRa Alliance, "LoRaWAN™ Specification", Version: V1.0, January 2015

[5] SigFox, "One network A billion dreams", M2M and IoT redefined through cost effective and energy optimized connectivity", white paper, 2015

[6] M. Waldrop, "More Than Moore", Nature, February 2016 Vol 530

[7] SIA/SRC, "Rebooting the IT Revolution: A Call to Action", 2015

[8] K. Pretz, "The Next Evolution of the Internet", March 2013, http://theinstitute.ieee.org/technology-focus/technology-topic/the-next-evolution-of-the-internet

[9] http://stackoverflow.com/questions/15482257/how-nest-thermostat-communicates, seen June 2016.

[10] http://www.devttys0.com/2014/05/hacking-the-d-link-dsp-w215-smart-plug/, seen June 2016

[11] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", 2000,

[12] Z. Shelby, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014

[13] C. Bormann ET al, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", IETF draft April 2016

[14] A. Banks and R. Gupta, "MQTT Version 3.1.1", OASIS Standard September 2014.

[15] P. Urien, P, M. Dandjinou, M.Badra, "Introducing micro-authentication servers in emerging pervasive environments", IADIS International Conference WWW/Internet 2005, Lisbon, Portugal, October 2005.

[16] P. Urien, "EAP Support in Smartcard", draft-urien-eap-smartcard-29.txt, July 2015

[17] P.Urien, "TLS and DTLS Security Modules", draft-urien-uta-tls-dtls-security-module-00.txt, June 2015

[18] Z. Chen, "Java Card™ Technology for Smart Cards: Architecture and Programmer's (The Java Series)", 2002, Addison-Wesley, ISBN 020170329

[19] T.M. Jurgensen ET. al., "Smart Cards: The Developer's Toolkit", Prentice Hall PTR, 2002,, ISBN 0130937304.

[20] P. Urien, "TLS-Tandem: A Smart Card for WEB Applications", Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, January 2009.

[21] P. Urien, P and M. Dandjinou, M, "The OpenEapSmartcard platform", Fourth IFIP International Conference on Network Control and Engineering for QoS, Security and Mobility, Lannion, France, November 2005, Springer 2007.

 ISBN: 978-1-61208-505-0