On Line Secure Elements: Deploying High Security Keystores and Personal HSMs

Pascal Urien

On Line Secure Elements: Deploying High Security Keystores and Personal HSMs

Pascal Urien
Telecom Paris
19 Place Marguerite Perey 91120 Palaiseau, France
Pascal.Urien@Telecom-Paris.fr

***Abstract*—This paper presents innovative approach to deploy secure elements providing cryptographic resources in TCP/IP environment. The main idea is to execute in secure element, TLS1.3 server, secured by 256 bits pre-shared-key. All cryptographic resources are protected by TLS-PSK sessions. In the user plane the secure element is a TLS server, what enables to define uniform resource identifier (URI) for embedded resources. The user is optionally equipped with access card (TLS identity module) that stores procedures working with PSK. The security level may be increased by the use of dedicated terminal, similar to payment terminal, which protects dual factor authentication. We present two open platforms: keystore devices hosting preconfigured TLS-SE secure elements, and personal HSM supporting on-demand TLS-SE applications. Finally we detail some performance elements.**

***Keywords*—** Secure Element, IOSE, Security, TLS

I. INTRODUCTION

Secure elements [1] are tamper resistant microcontrollers widely used in services such as payments (EMV card), mobile subscriber authentication (SIM card), or identity (electronic passport). Secure elements provide, according to *Common Criteria* standards, the highest *Evaluation Assurance Level* (EAL), up to EAL6+ the maximum level being EAL7, and the minimum level being EAL1. On-line secure elements enables individuals to have their own cost-effective personal on-line secure storage and computing service with the same trust level as the smartcards they carry with them. Remote use implies the establishment of secure channel. The Transport Layer Security (TLS 1.3, RFC 8446), protocol widely used for internet security seems a logical choice. In this spirit, secure element is a TLS server, and depending on the required security level, TLS credentials are stored in an access card optionally controlled by terminal similar to payment terminal.

Nevertheless secure element has no TCP/IP stack and network interface; therefore a front TLS server is needed. A common technique is to route TLS packets, received by TLS front server, to backend servers identified by their TLS server name (SN).

This paper presents on-line secure elements [7] based on open software and hardware technologies (see figure 1). We detail front servers based on PC boards, Raspberry Pi, or Arduino chips. TLS secure element (TLS-SE [3][6]) servers run TLS stacks written in javacard [2] language. On client side, standard software applications can be used (for example OPENSSL); we also introduced TLS Identity Module (TLS-IM [3][5]), and dedicated terminal (a kind of smartcard reader with touch screen [4]), based on open hardware (i.e. Arduino).

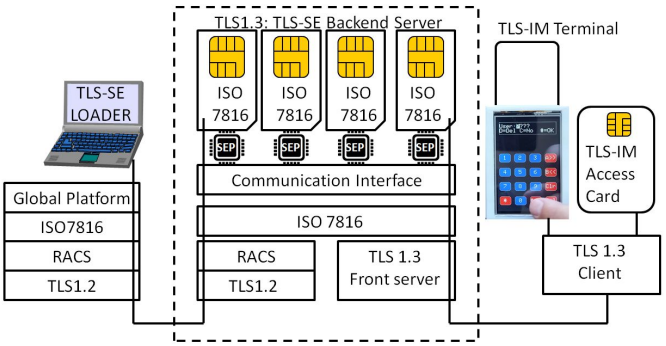

Fig. 1. Personal HSM architecture.

This paper is organized according to the following outline. Section 1 introduces the basic technological bricks: TLS1.3 for secure element (TLS-SE), TLS identity module (TLS-IM), performances for today secure elements, secure element *server name* (SEN), and *Uniform Resource Identifier* (URI) for embedded TLS-SE resources. Section 2 presents *Secure Element Processor* (SEP) required to plug secure elements in legacy computing systems. Section 3 describes TLS-IM access card and crypto terminal, similar to payment terminal, which can be used to increase security level. Section 4 introduces keystore devices, which enable to deploy TLS-SE secure elements with cryptographic resources, in TCP/IP environment. Section 5 describes Personal HSM managing grid of TLS-SE secure elements, which supports on-demand applications and parallel cryptographic resources. Finally section 6 concludes this paper.

II. TLS 1.3 FOR SECURE ELEMENTS

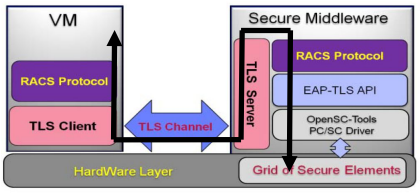

Fig. 2. Grid of secure element from [8]

In previous papers [8][9][10] we introduced the idea of a grid of secure elements (see figure 2) hosting TLS 1.0 servers. Nevertheless communication was based on ISO7816 packets; a front server running RACS (*Remote APDU Call Secure*) protocol [9], transports ISO7816 messages sent to EAP-TLS smartcards [10]. In this work we still use RACS for TLS-SE application deployment, but communication with TLS-SE smartcards only relies on TLS1.3.

A. TLS 1.3

TLS 1.3, the last version of the Transport Layer Security protocol was released in August 2018. It supports three modes (see figure 3) for key exchange:

- Diffie-Hellman exchange authenticated by asymmetric signature and X509 certificates (PKI mode).

- Diffie-Hellman exchange authenticated by pre-shared-key (PSK mode).

- Pre-share-key mode, without Diffie-Hellman exchange.

The secure channel works with *Authenticated Encryption with Associated Data* (AEAD) symmetric algorithm. A record layer packet comprises a five bytes header, which is authenticated, and a payload, which is encrypted and authenticated.

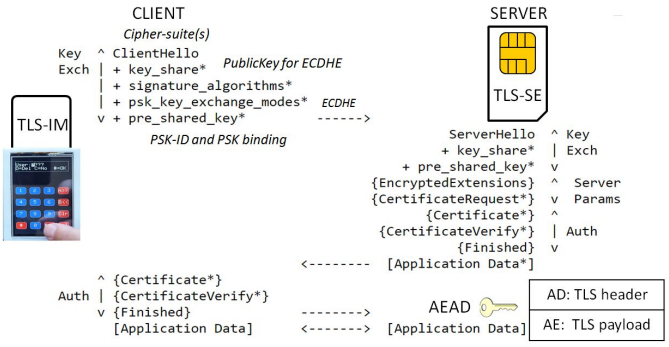

Fig. 3. TLS1.3 session establishment

Because our actual research work targets personal assets, we focus on TLS1.3 pre-shared-key mode, with Diffie-Hellman exchange.

B. TLS-IM

TLS identity module [3][5][11] computes on client side, procedures that are required for its authentication.

- For PSK mode two procedures are needed, first (*PSK Binding*) is used to authenticate the client hello message (i.e. it proves that the client knows PSK), second (*Derive Handshake Secret*) computes value from the Diffie-Hellman secret and PSK.

- For PKI mode only, one procedure is needed. An asymmetric signature (ECDSA) is generated thanks to a private key stored in the TLS-IM module

C. TLS-SE

TLS for secure element is an implementation of TLS1.3 with PSK mode for javacards [3][6][11]. The number of deployed javacards [2] is estimated to about 6 billion devices. Many secure elements embed a *Java Virtual Machine* (JVM) and run applications written in javacard (JC), a subset of the java language. The last javacard version is 3.1, and commercial versions are JC3.0.4 and JC3.0.5. Typical microcontrollers have 3KB RAM and 100KB FLASH. They provide cryptographic resources that are required by TLS 1.3 such as SHA256, public key generation and DH secret computing over the SECP256r1 elliptic curve, ECDSA signature, and AES algorithm.

D. TLS-SE Performances

Depending on the type of javacard used, the observed time required to open a TLS1.3 session ranges between 0,75 and 3,0 second. The AEAD channel, based on AES-CCM (128 bits) performs encryption/decryption of a 128 bytes payload according to time ranging from 100ms to 350ms.

E. Secure element name (SEN)

According to TLS the first message of a session (i.e. ClientHello, see figure 3) may comprise a server name (SN). This attribute is used to route TLS packet, from front server (server.com:port), to backend server identified by its SN parameter.

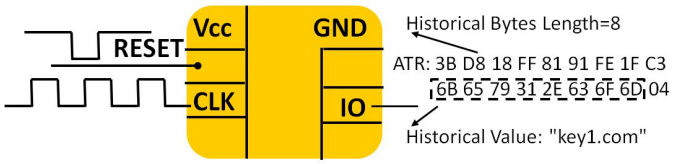

Fig. 4. Reading Secure Element Server Name (SEN) from ATR

When a secure element is reset, thanks to the physical ISO7816 [12] RESET pin, it returns (see figure 4) a set of bytes called *Answer To Reset* (ATR). ATR comprises a field (up to fifteen bytes) called *Historical Bytes*, which can be modified by standard javacard software. Each TLS-SE application is associated to *Secure Element Name* (SEN) found in the ATR. Furthermore TLS-SE is the default application in the secure element, what means that all requested information needed for communication is found in the ATR.

F. *Uniform Ressource Identifier (URI)*

When TLS-PSK is used, the secure element is identified by the following URI:

schemeS://sen:psk@server.com:port/?query, in which:

- server.com:port is the TCP/IP socket for TLS front server

- sen is the secure element name

- psk is the pre shared key value

- schemeS (S meaning secure by TLS) identifies the syntax used by application embedded in the secure element, and query a request encoded according to this syntax

In our actual work query is an ASCII command line ended by carriage return (CR) and line feed characters (LF). Therefore schemeS could be represented as shellS, and command line as *URL-encoded* query what leads to the URI:

shellS://sen:psk@server.com:port/?command_line

III. Secure Element Processor

Secure elements physical and logical interfaces are defined by ISO7816 standards [12][1]. ISO7816 physical interface (see figure 5) has five wires: ground, power (Vcc), reset, clock (a typical value is 4 MHz) and Input/Output. In contact mode, small messages, whose maximum size is about 256 bytes, are transported over a single IO pin. There are two transport protocols (noted T=x). The protocol T=0 is organized as byte stream in which a character comprises 12 bits (start, 8 data bits, parity, and 2 stop). The protocol T=1 organized with frames, in which a character comprises 11 bits (start, 8 data bits, parity, 1 stop), each frame has a three bytes header (NAD, PCB, LENGTH), a payload, and a trailer (one byte checksum or two bytes CRC).

Given a clock, with a frequency F_{CK}, the serial time bit, named ETU, is obtained according to the relation:

ETU = F/D $1/F_{CK}$, with default values F=372 and D=1, what leads with F= 4 MHZ to ETU= 93µs.

Parameter F, D and T=x can be modified thanks to a procedure called PTS (*Protocol Type Selection*).

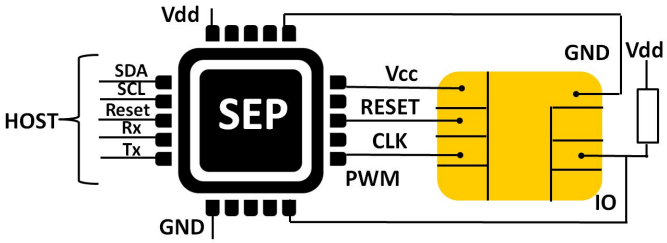

Fig. 5. Secure Element Processor (SEP)

Because ISO7816 legacy interface is not well fitted to common communication protocols, a *Secure Element Processor* (SEP) realizes a logical bridge between ISO7816 and protocols such as UART or I^2C (as illustrated by figure 6). We designed a library for Arduino environment that manages secure elements. The required functional hardware resources are the following:

- Clock generation, with *Pulse Width Modulation* facility.

- Vcc output pin for secure element.

- Reset output pin

- Digital Input/Ouput pin for serial IO operations.

The implementation requires a timer to sample the serial communication. We notice a limit of about 372x2 processor cycles, for T=x protocols processing. For example with 16 MHz processor clock, this leads to a limit ETU of about 46,5µs (F_{CK}= 4 MHz, F=372 D=2). SEP provides two host protocols: UART and I^2C. UART is adapted for laptop (Windows, Linux), and I^2C for embedded environment supporting grid of secure elements.

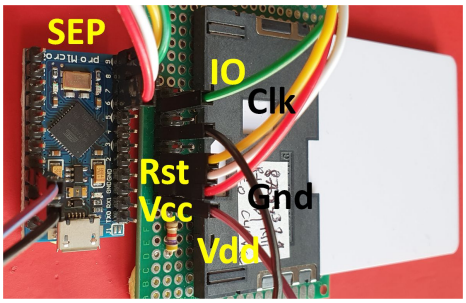

Fig. 6. Illustration of SEP based on ATMEGA 32U4 (Arduino Leonardo)

A SEP provides three types of command:

- SE-On(): Power on secure element and return SEN found in ATR.

- Send-APDU(request): Send an ISO7816 request to secure element an return response. Response may include an error status, or an indication for End-Of-TLS session

- SE-Off(): Power off secure element.

IV. TLS-IM Access Card & Crypto Terminal

A. *Access Card*

As mentioned before, TLS-IM provides credentials needed for PSK and PKI mode. As an illustration WolfSSL TLS1.3 library provides a call back for ECDSA signature, and a call back to read the PSK value. So we need to define two new callbacks one for the *PSK Binding* procedure, and another one for the *Derive Handshake Secret* procedure. The access card is protected by a PIN code. In order to increase the security level, a dedicated terminal (crypto terminal) may be used in order to enter the PIN.

B. *Crypto Terminal*

The crypto terminal (see figure 7) is a device designed in Arduino environment [4][13]. It includes a set of countermeasures in order to prove the firmware authenticity (remote attestation) and the hardware integrity (dynamic PUF). It acts as a firewall between the host, such as laptop or mobile, and the access card. In a way similar to payment terminal, the PIN code is typed on touch screen, in order to avoid Trojan horse attack. TLS-IM procedures may also be acknowledged by user before execution.

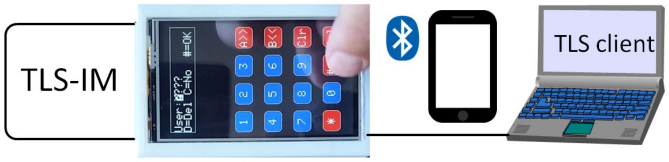

Fig. 7. Crypto Terminal with USB and Bluetooth connectivity.

The crypto terminal works over serial communication, with command lines. Three commands are used by TLS-IM :

- ***user***, trig the crypto terminal user to type the access card (TLS-IM) PIN code

- ***binder 32bytes_hexadecimal_value***, return the binder value

- ***derive 32bytes_hexadecimal_value***, return the TLS1.3 Handshake Secret.

C. *TLS-IM with SE050*

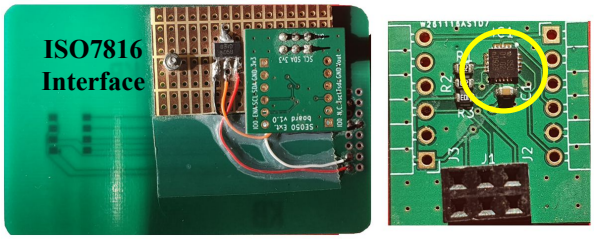

Fig. 8. SE050 secure element (right), and access card board (left) for ISO7816 form factor

The SE050 chip [15] is a secure element from NXP, running the JC3.0.5 javacard API. The SE050 is a turnkey solution, based on Javacard operating system. It is delivered with a proprietary applet that implements a cryptographic library. It is a surface mounted device (see figure 8), whose dimensions are 3mm x 3mm x 0,32mm. It has an I²C interface [16] that supports clock frequencies up to 3.4 MHz and uses T=1 over I²C protocol. SE050 provides secure objects that work with keys and associated cryptographic algorithms. They can realize TLS-IM procedures, such as. *PSK Binding*, *Derive Handshake Secret,* and *ECDSA signature*. For example an open source implementation is available in [11]; other cryptographic facilities like random number generator, public key generation and Diffie-Hellman computing are also provided. Figure 8 shows an ISO7816 adaptation board that can be plugged in the crypto terminal; five wires are needed: ground, Vcc, SDA, SCL, RESET. Thanks to its small size and low cost (about 1.5$), SE050 could be integrated in many devices used as TLS1.3 access card.

V. KEYSTORE WITH PRECONFIGURED TLS-SE CARDS

As previously mentioned, TLS-SE is a TLS1.3 stack in ISO7816 secure element. The SEP enables connectivity to host system, supporting TCP/IP resources, thanks to protocol such as UART or I²C. ESP8266 is a low cost Wi-Fi SoC, with 4 MB FLASH, 64KB SRAM, and 80MHz clock, which provides TCP/IP wireless connectivity. This system is not suitable for multithreading applications; nevertheless it is an efficient and cost effective way, to put TLS-SE secure element on-line. Upon booting, the SoC resets one or several SEPs and collects SEN values. A TCP front server waits for incoming connections. Because each TLS packet includes a length field, it is able to detect message limit, and therefore to forward them to secure element. The first TLS message (*ClientHello*) contains a server name (SN) attribute that must match one of available SENs. A TLS session implicitly begins by the reception of a *ClientHello* message and ends either by specific responses returned by secure element or TCP/IP socket closing.

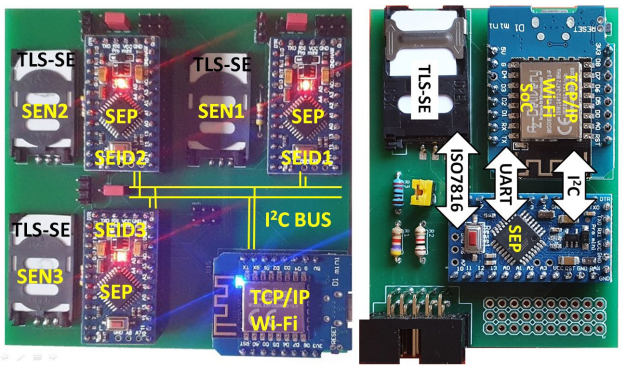

Fig. 9. Single TLS-SE bord (right), and multiple TLS-SE board (left)

A. *UART protocol*

UART protocol targets board with only one SEP (see figure 9). Depending on SEP microcontroller, data throughput ranges between 19200 and 115200 bauds. In embedded systems serial software is a very common feature; therefore secure element integration is straightforward.

B. *I²C protocol*

I²C protocol [15] manages a two wires (SDA, SCL) bus (see figure 9) comprising a master node, and slave nodes identified by their address (usually 7 bits). We use frequency clock of 200 KHz; given 10 bits per character this leads to a maximum throughput of about 20KB/s. A table establishes the relation between SEN and I²C address. Nevertheless, due to single-tasking architecture only one TLS-PSK session is available at a given time.

C. *Keystore APP*

A keystore APP [16] is an application that provides cryptographic procedures protected by TLS-PSK secure session, executed in a TLS-SE secure element. A command line is for example a set of ASCII characters with a prefix (one character), an index (two hexadecimal digits), and a data payload (usually hexadecimal digits); it ends by carriage return (CR) and line feed (LF) characters.

D. *OPENSSL Client.*

A keystore session can be opened thanks to the popular OPENSSL software with the following command line:

openssl s_client -tls1_3 -connect server:port -servername SEN -groups P-256 -cipher DHE -ciphersuites TLS_AES_128_CCM_SHA256 -no_ticket -psk [PSK-VALUE]

Thereafter ASCII command lines are used for interactions with the *keystore APP*.

VI. PERSONAL HSM

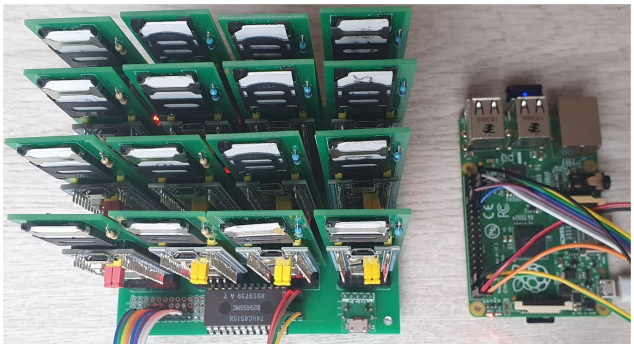

Fig. 10. A personal HSM. Grid of secure element with I^2C conectivity (left) and host system (Raspberry Pi, right)

The goal of personal HSM is to enable on-demand TLS-SE application, and to efficiently share a grid of secure elements thanks to multitasking computing environment. Two TCP daemons are used, one for RACS and another for TLS front server. The server is powered by processor with rich operating system, such as LINUX. The open *Internet Of Secure Element* project (IOSE [7]) is dedicated to TLS-IM and TLS-SE trusted applications. The IOSEv5 open software [16] works with Windows, Ubuntu, and Raspberry Pi platforms. It supports two communication interfaces for secure element PC/SC (Personal Computer / Smart Card API) and I^2C.

A. *RACS protocol*

The Remote APDU Call Secure (RACS) protocol works over TLS (more precisely TLS1.2 with IOSEv5 software). Client and server are mutually authenticated by X509 certificates and associated private keys. A security policy is enforced from the client identity, found in its certificate common name (CN). Secure element are identified by a *Secure Element IDentifier* attribute (SEID), for example deduced from SEP I^2C address.

Once the TLS secure channel is opened, the following commands are available:

- List secure elements that can be managed by client
- Power on secure element
- Send ISO7816 request and return response
- Power off secure element
- Associate a secure element to a SEN parameter

The Global Platform (GP) standards define protocols, based on ISO7816 APDU transport, for downloading application in javacards. These procedures require mutual authentication and used encrypted secure channel.

B. *IOSE Server*

An IOSE server is made of two electronics parts (see figure 10): a secure element grid and processor with rich operating system and internet connectivity.We designed a grid of 16xSEPs; each of them has five wires interface: Gnd,Vcc, SDA, SCL and RESET. Reset pins are controlled by 4-to-16 line decoder. Therefore each SEP has a reset address and an I^2C address. The I^2C bus is clocked at 200 KHz (i.e. a bandwidth of about 20KB/s). SEP work with a 6 MHz clock with F=372 and D=2, what leads to an ISO7816 throughput of about 2,7 KB/s. The IOSEv5 software runs in raspberry pi. The grid is controlled by 10 lines: Gnd, Vcc, SDA, SCL plus 6 wires driving reset pin decoder.

An I^2C packet are similar to T=1 message. It comprises a 3 bytes header (NAD, PCB, Length), a payload, and a two bytes CRC. The grid is managed by three main commands

- Power on SEP and return its SEN
- Send ISO7816 request to SEP and return response
- Power off SEP

When the host system boots, it resets SEPs and collects the list of SEN. Each SEN is associated to a SEID.

The TLS host daemon is the front server. As mentioned before it routes TLS messages to SEPs acting as TLS back end server, and identified by the SEN.

C. *On-Demand Software & Attestation*

The main enhancement between keystore and personal HSM, is the support off on-demand software service, and the definition of an attestation procedure that transfers exclusive secure element content control to user. The attestation procedure relies on two claims: a secure element cannot be cloned, and it is only able to manage a single TLS-PSK session at a given time. These processes are illustrated by figure 11.

(1) A TLS-SE App (embedding a provider PSK and a SEN) is downloaded thanks to the RACS protocol, in a secure element. Upon installation the TLS-SE App generates a pair of asymmetric public and private key.

(2) The application provider binds SEID to SEN thanks to a dedicated RACS command. At this step the TLS-SE application is ready to use.

(3) The provider opens a TLS session with the secure element, thanks to its knowledge of PSK. It reads the public key.

(4) The provider generates a certificate for the public key and writes it in the secure element over a TLS session.

(5) The provider sends SEN and PSK-Provider to user, who is going to perform attestation procedure.

(6) The user opens a TLS-PSK session with the secure element.

(7) The user reads the secure element public key

(8) The user reads the certificate, and verifies its signature with the *Certification Authority* public key

(9) The user sends a random value to the secure element, which concatenates this value to the TLS handshake secret (HS computed from the DH exchange value). The secure element computes an ECDSA signature with its private key. We assume that only a genuine secure element can know both PSK-Provider and private key.

(10) The user modified the PSK value thanks to a dedicated command. At this step he has an exclusive control on the secure element application data.

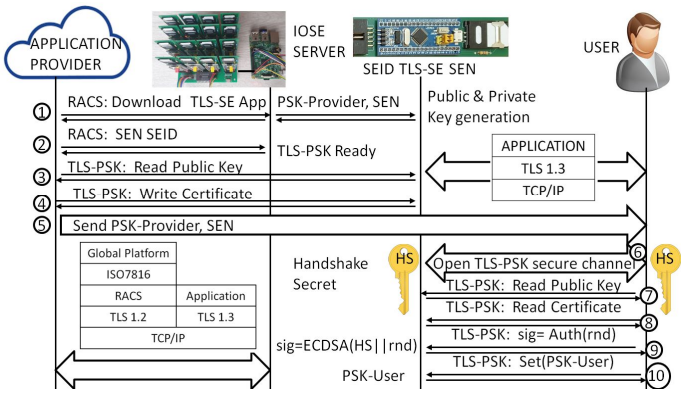

Fig. 11. On-demand TLS-SE App (left), and attestaion procedure (right)

D. *Performances*

The performances are measured thanks to a TLS-SE application that performs ECDSA signature over the SECP256k1 elliptic curve. First a TLS-PSK session is opened with a secure element, second a command line with an argument of 32 bytes (the value to be signed) is sent to the secure element, which returns the signature. Depending on the javacard, this operation required between two and four seconds.

According to the Amdahl 'law, the computing time for n simultaneous TLS-SE sessions can be written as:

$Tn/T_1 = 1 + \rho.(n-1), \ \rho \in]0,1]$

What leads to a speeding factor (Sf) factor

$Sf(n) = nT_1/Tn = 1/(\rho + (1-\rho)/n)$,

with $1/\rho$ the limit speed up factor

For IOSEv5 software and Raspberry Pi, we observe $1/\rho$ value in the range 50 to 80, what means that the speedup factor increases quite linearly with the number of used secure elements ($Sf \sim n$)

VII. Conclusion

In this paper we presented high security keystore and personal HSM devices. We are currently working on enhanced versions for open hardware devices and open source software.

Reference

[1] Jurgensen T.M., Guthery, S.B., "Smart Cards: The Developer's Toolkit", O'Reilly

[2] Chen, Z., "Java Card™ Technology for Smart Cards, Architecture and Programmer's Guide", ADDISON-WESLEY, 2000

[3] P. Urien, "A New IoT Trust Model Based on TLS-SE and TLS-IM Secure Elements: A Blockchain Use Case," 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), 2021, pp. 1-2, doi: 10.1109/CCNC49032.2021.9369485

[4] Urien, P., "Innovative Wallet Using Trusted On-Line Keystore," 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), 2021, pp. 12-14, doi: 10.1109/BRAINS52497.2021.9569783.

[5] IETF Draft, "Identity Module for TLS Version 1.3", draft-urien-tls-im-06.txt, January 2022

[6] IETF Draft, " Secure Element for TLS Version 1.3", draft-urien-tls-se-04.txt, March 2022

[7] IETF Draft "Internet of Secure Elements", draft-urien-coinrg-iose-05.txt, IETF Draft, April 2022

[8] H. Aissaoui-Mehrez, P. Urien and G. Pujolle, "Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements," 2014 IEEE Military Communications Conference, 2014, pp. 282-287, doi: 10.1109/MILCOM.2014.51.

[9] P. Urien, "RACS: Remote call secure creating trust for the internet," 2015 International Conference on Collaboration Technologies and Systems (CTS), 2015, pp. 351-357, doi: 10.1109/CTS.2015.7210448.

[10] Urien, P., Pujolle,G., "Security and privacy for the next wireless generation". Int. J. Netw. Manag. 18(2): 129-145 (2008)

[11] https://github.com/purien/TLS-SE

[12] ISO7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

[13] P. Urien, "High Security Bare Metal Bluetooth Blockchain Payment Terminal For Trusted Ethereum Transaction," 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), 2020, pp. 1-2, doi: 10.1109/CCNC46108.2020.9045146.

[14] NXP Semiconductors, I^2C-bus specification and user manual, UM10204, Rev. 7.0 October 2021

[15] NXP Semiconductors,"SE050 Plug & Trust Secure Element", Rev. 3.3, July 2021

[16] Urien, P., "The IOSE project", https://github.com/purien/IoSE