A Triple Interfaces Secure Token -TIST- for Identity and Access Control in the Internet Of Things

Pascal Urien
Telecom ParisTech
23 avenue d'Italie, 75013 Paris, France
Pascal.Urien@telecom-paristech.fr

Michel Betirac
EtherTrust, Avenue Louis Philibert
Domaine Petit Arbois 13100 Aix en Provence
Michel.Betirac@EtherTrust.com

Abstract— **This paper introduces an innovative technology based on secure microcontrollers, such as smartcards, equipped with TLS stack. The secure microcontroller, identified by its X509 certificate, is embedded in a triple interfaces secure token (the *TIST*), supporting USB, NFC and Mifare connectivity. The *TIST* secures OPENID and electronic key delivery services and makes them available for multiple terminals such as mobiles, laptops or tablets.**

Keywords- Security, smartcard, NFC, OPENID, TLS

I. INTRODUCTION

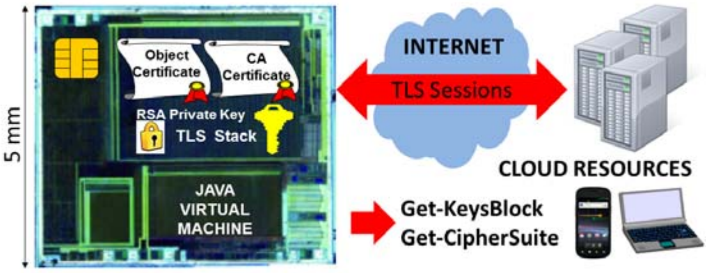

Figure 1. A TLS Stack for Secure Microcontroller

The Internet Of Things (IoT) is an architecture in which billions of devices with computing capacities and communication interfaces are connected to the internet and perform collaborative tasks [4]. In this context security is a major issue. In this paper we focus on access control services to logical (OPENID) and physical (electronic key delivery) resources, which are deployed in laptops, tablets, mobiles or embedded systems such as electronic locks. In previous works [1][2] we introduced a TLS stack with a small memory footprint of about 20 KB, which works in most of the secure microcontrollers of the market such as smartcards (defined by the ISO 7816 standards). The TLS protocol (RFC 2246) is the cornerstone of the internet security; the server is usually identified by its X509 certificate; the client is optionally authenticated with a certificate. Our TLS stack always works with certificates for both client and server, and therefore performs strong mutual authentication, based on PKI, between these entities. Secure microcontrollers [7] (see figure 1) are tamper resistant devices whose security is enforced by multiple hardware and software countermeasures; they include CPU (8, 16, 32 bits), nonvolatile memory (about 100 KB), ROM (about 200 KB) and RAM (about 10 KB). Most of them comprise a java virtual machine (JVM), and therefore run software written in javacard (a subset of the java language) such our TLS stack. As a consequence an object equipped with a secure microcontroller running a TLS stack (see figure 1), performs strong mutual authentication with internet server, and afterward establishes secure channel (i.e. the TLS Record Layer) with cloud computing services. In this paper we describe the integration of this TLS stack in a token that offers three communication interfaces: USB for laptops, NFC (Near Field Communication) for mobile phones, and ISO 14443A (MIFARE) for physical access control.

This paper is constructed according to the following outline. Section 2 introduces the Triple Interfaces Secure Token (or ***TIST***) architecture, and details some services such as OPENID (http://openid.net) authentication and electronic keys delivery [5] available for mobiles phones and laptops. Section 3 describes the concept of dual NFC interfaces, and the logical architecture of applications dealing with this concept. Section 4 details the services supported by the TLS stack embedded in the TIST. Section 5 presents the APIs dedicated to the TIST, available for laptops and smartphones environments. Finally section 6 concludes this paper.

This work was funded by the FEDER *OnDemand* project.

II. TRIPLE INTERFACES SECURE TOKEN - TIST

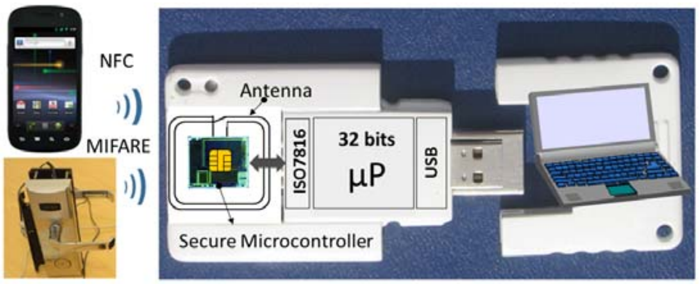

Figure 2. Triple Interface Secure Token (TIST) hardware architecture

The ***TIST*** architecture is depicted by figure 1, it comprises four main components

- The system core is a 32 bits microprocessor, which drives an USB port and an ISO7816 serial interface with the secure microcontroller (i.e. a smartcard)

- The USB port gives access to the smartcard, which usually exchanges packets with the terminal thanks to the well-known PC/SC API.

- The ISO 7816 interface is used for the contact mode, i.e. when the key is plugged to an USB terminal. It transports messages to/from the smartcard, over the ISO 7816 serial link.

- The secure microcontroller (see figure 1) is a smartcard equipped with two communication protocols, first is used in contact mode according to the ISO 7816 standards, and second in contactless mode. In this last case the chip is feed by an electromagnetic field at the frequency of 13,56 MHz; radio packets are exchanged according to the ISO14443 specifications, which have been endorsed by the NFC consortium (http://www.nfc-forum.org).

The TLS stack delivers two main facilities. First it established TLS sessions with remote TLS servers, via the EAP-TLS protocol (RFC 5216) that transports TLS without TCP/IP flavors. The session is transferred to the application (handling the secure microcontroller) upon the reception of the server finished message for TLS full mode, and after the transmission of the client finished message for the resume mode. The session transfer requires two parameters a set of ephemeral keys (the *KeysBlock*) and the associated cryptographic algorithms (the *CipherSuite*). According to a mechanism called container [5], the terminal may push data (such as keys values used by electronic locks) that are ciphered with the secure microcontroller public key (found in its certificate) and signed by a trusted authority. Containers are checked and decrypted by the secure microcontroller.

In summary the two services of TLS stack are first TLS session booting and second container delivery. The following sections presents applications based on these facilities such as OPENID [3][8] and key delivery [5][6], which are available with three communication interfaces, i.e. NFC, Mifare, and USB.

A. NFC Interface

The NFC technology covers an umbrella of standards working with the 13,56 MHz frequency, with throughput ranging from 106 Kbits/s to 424 Kbits/s. Android is an operating system originally created by the Android Inc company, bought in 2005 by Google. The gingerbread version (v2.3) endorses the NFC technology; more precisely the mobile may be used as a NFC reader that communicates with external NFC devices feed by its electromagnetic field.

When the ***TIST*** is tapped against the mobile, an Android application registered to the NFC service is started (see [5] for more details). This application realizes four main functionalities:

- It exchanges packets with the TLS stack running in the secure microcontroller;

- It boots the TLS session and transfers its control back to the mobile (via the *Get-KeysBlock* and *Get-CipherSuite* commands)

- It manages TCP/IP sockets resources, and realizes a proxy server in order to establish the logical glue between the browser, the external contactless smartcard and the remote internet server.

- It pushes toward the NFC device, container received over HTTPS session.

OPENID is a typical application for NFC services for mobile in which the embedded TLS stack is used for mutual authentication with the OPENID server (see [5][8] for more details).

B. Mifare Interface

Mifare devices introduced in 1994 by the NXP Company are very widely deployed for access control purposes, used by transport infrastructures or electronic locks. For example the *"Mifare Classic S50"* component is organized in sectors, divided in 16 bytes blocks, whose reading and writing operations are controlled by a couple of 48 bits keys (*KeyA* and *KeyB*). Our contactless smartcard provides Mifare emulation, and therefore is compatible with the Mifare ecosystem. Furthermore the secure microcontroller has access to Mifare blocks. As we demonstrated in [6] this feature enables the secure delivery of keys, transported in containers. TLS sessions with internet keys servers are booted from the secure microcontroller. The user experience is quite simple: the user taps the ***TIST*** against its mobile, starts the Android application that collects a key which is afterwards stored in the appropriate Mifare block.

C. USB Interface

The USB interface provides connectivity for laptops. The ***TIST*** is accessed via the PC/SC API which is supported by Windows and Linux operating systems. It works as a smartcard reader to which is internally plugged the secure microcontroller. A proxy server application (see section 2-A) establishes the logical glue between the PC browser, the secure microcontroller and a remote internet server. This architecture was previously detailed in [2], and is used for OPENID platform secured by EAP-TLS smartcard [1] It should be noticed that the key delivery service described in section 2-B is also provided for laptops.

III. About Dual NFC Interfaces Apllications

The ISO 14443A standard specifies the boot sequence of a NFC device, when it is feed by the electromagnetic field (whose frequency is 13,56 Mhz) generated by a NFC reader, providing a throughput of 106 Kbits/s. The reader uses an ASK 100% modulation with for packet transmission, and a subcarrier of 848 kHz for data reception. The booting process is illustrated by the figure 3,

Initially the card powered by the RF field is in the IDDLE state. The reader sends a REQA (Request Command of Type A) packet and waits for an ATQA (Answer To Request of Type A) message, whose two bytes content indicates the protocol to be used for the anti-collision process. Afterwards the reader performs an anti-collision loop (based on SELECT and ANTICOLLISION commands) whose goal is to collect the UID (Unique Identifier), attribute (4, 7 or 10 bytes). The anti-collision procedure ends by a SAK packet (SELECT Acknowledge), which indicates if the card works with the ISO7816-4 protocol or with the MIFARE protocol. If the card is ISO7816-4 compliant the reader sends the RATS (Request for Answer To Select), which is acknowledged by an ATS (Answer To Select); otherwise the reader and the card exchange MIFARE messages.

The availability of both MIFARE and ISO7816-4 protocols is usually referred as a dual NFC interface. Despite the fact that MIFARE protocols are proprietary (i.e. designed by the NXP company), they are widely used for access control and transport (about 70% of the transport market).

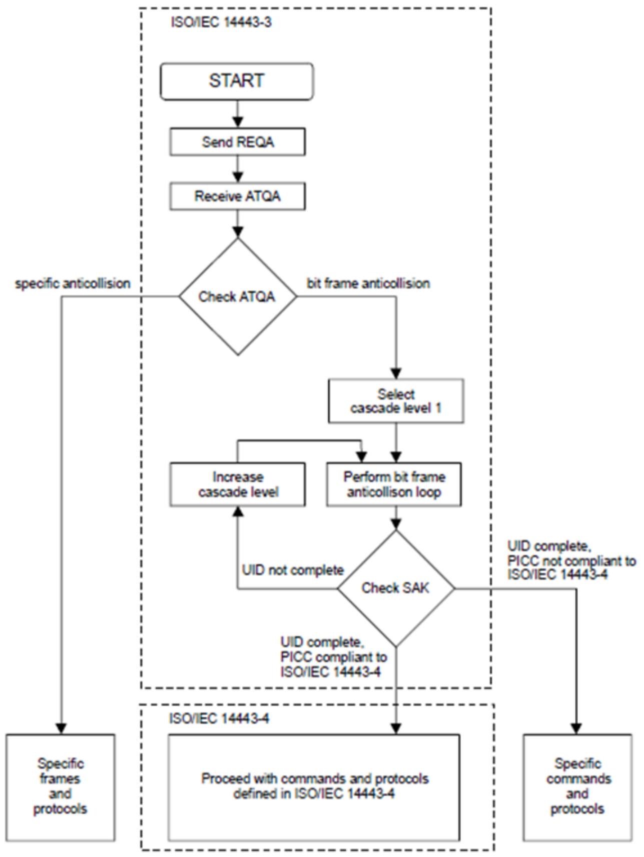
ISO/IEC 14443-3
START
Send REQA
Receive ATQA
specific anticollision
Check ATQA
bit frame anticollision
Select cascade level 1
Increase cascade level
Perform bit frame anticollison loop
UID not complete
Check SAK
UID complete, PICC not compliant to ISO/IEC 14443-4
UID complete, PICC compliant to ISO/IEC 14443-4
ISO/IEC 14443-4
Specific frames and protocols
Proceed with commands and protocols defined in ISO/IEC 14443-4
Specific commands and protocols

Figure 3. The ISO14443-A state machine (from the ISO14443 standard)

The ISO7816-4 standard provides a communication interface both for contactless (NFC) and contact smartcards that usually comprise a java virtual machine (JVM) and therefore run programs written in the JAVACARD language (a subset of JAVA).

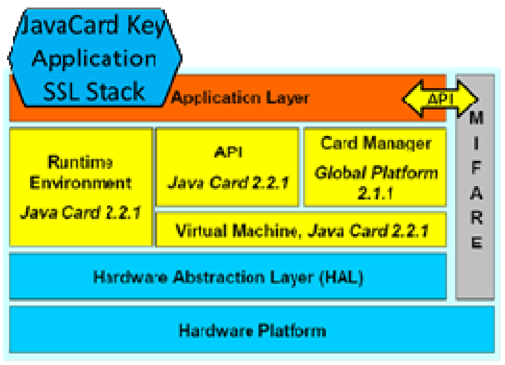
JavaCard Key Application
SSL Stack
Application Layer
API
MIFARE
Runtime Environment Java Card 2.2.1
API Java Card 2.2.1
Card Manager Global Platform 2.1.1
Virtual Machine, Java Card 2.2.1
Hardware Abstraction Layer (HAL)
Hardware Platform

Figure 4. Illustration of a dual interfaces apllication

The JCOP operating system, designed by the IBM company and afterwards bought by the NXP company supports a JAVACARD API (*JZSystem.readWriteMifare*) that enables read and write operations in MIFARE blocks from a JAVACARD application.

The Java Card Forum (JCF) API supports since the JC2.2 release a dedicated class (*javacardx.external.Memory*) providing access to memory subsystems such as MIFARE.

The figure 4 illustrates the architecture of an electronic key service dedicated to legacy MIFARE lock readers. A TLS-Stack running in the javacard is used to securely transfer a key value from a WEB server. This attribute is afterwards written in the MIFARE memory.

IV. THE TLS APLLICATION

	CLA	INS	P1	P2	P3	LE
Select	00	A4	04	00	LC bytes	0 byte
Verify PIN	00/A0	20	00	00 or 01	LC bytes	0 byte
Set Identity	00/A0	16	80	00	LC bytes	0 byte
Reset	00/A0	19	10	00	0	0 byte
Process EAP	00/A0	80	0.last 1.more	00	LC bytes	LE bytes
Get CipherSuite	00/A0	82	CC	00	LE=0x03 or 0x24	0x40
Get KeyBlock	00/A0	82	CA	00	LE=0x40 or 00	0x40
Get Result	00/A0	C0	00	00	LE	#0
Fetch Result	00/A0	12	00	00	LE	#0
Write Container	00/A0	D0	0.first 1.more 2.last	00	LC bytes	0 byte

Figure 5. ISO7816-4 Interfcae for theEAP-TLS smartcard

The TLS embedded application is based on the EAP-TLS smartcard IETF draft specification [1]. The software interface is made of ten (Select, Verify-PIN, Set-Identity, Reset, Process-EAP, Get-CipherSuite, Get-KeyBlock, Get-

Result, Fetch-Result, Write-Container) ISO7816-4 commands, illustrated by figure 5.

Select activates the EAP-TLS application, which is according to the ISO7816 standard identified by a 16 bytes number, the AID (application identifier).

Verify-PIN unlocks the application; two PINs are available one for the user and the other for the administrator, which are associated to different privileges. The administrator manages all the application resources.

Set-Identity binds TLS sessions to a set of cryptographic credentials that form the user's identity. An identity is a set of attributes such as the CA (Certification Authority) certificate, the TLS client or server certificate, and a private key; this collection of data is identified by an alias name that may be a well-known value if the device only hold a unique identity.

Reset resets the EAP-TLS state machine.

Process-EAP forwards EAP-TLS packets to the secure microcontroller, which processes its contents according to the current state machine and returns an EAP-TLS packet. The EAP-TLS standard (see RFC 5126) specifies the transport of TLS over the EAP framework (Extensible Authentication Protocol, RFC 3748). A double segmentation/reassembly procedure performed both for EAP and ISO7816 messages, realizes the transfer of TLS packets (up to 16384 bytes) to and from secure microcontrollers whose (ISO7816) command size is limited to 256 bytes.

Get-CipherSuite collects the cipher suite (algorithms used for cipher and HMAC purposes) negotiated during the TLS session establishment; if available this command returns also the Session-Id of the current TLS session.

Get-KeysBlock returns a couple (for data transmission and reception) of cryptographic keys, needed by the record layer entity for encryption and integrity procedures, Due to performances issues (cryptographic operations performed by secure microcontrollers are not fast), the TLS session may be exported from the secure microcontroller if a great amount of exchanged data is expected.

Get-Result and *Fetch-Result* are service commands used to collect EAP packets that are produced by the secure microcontroller.

Write-Container pushes a set of data (the container) that are ciphered with the secure microcontroller public key (found in its certificate) and signed by a trusted authority.

The TLS application is downloaded in the TIST token during the manufacturing process. The administrator PIN is set by default to eight zeros. It may be thereafter modified, and its knowledge gives access to personalization operations, which imply the generation of identity attributes and their downloading in the secure microcontroller.

V. USING TIST FROM APIS

The TIST works with two class of computing platforms, laptops equipped with USB connectivity, and mobile phones supporting NFC interface. Two kinds of APIs are available written in C language for PC running the Windows operating system, and JAVA for Android smartphones.

A. APIs for C environment

The glue with PC/SC Windows environment is realized by five functions performing smartcards detection and ISO7816-4 commands exchanges.

- int DetectAllCard(char *Aid), returns the number of detected TLS modules.

- int StartFirstCard(int index,char *aid), starts a session with a smartcard identified by its index (first index is set to zero).

- int GetPtcol(int index), returns the T-Protocol (ISO7816 transport protocol) used by a smartcard identified by its index.

- int CloseCard(int index) closes a session with a smartcard identified by its index.

- int send_apdu(int index, char *request, int offset, int length, char *response, int response_offset) sends an ISO7816 command to a smartcard.

TLS sessions are opened by secure microcontrollers. Upon success, a set of ephemeral cryptographic keys (the KeysBlock) and the list of negotiated algorithms (the CipherSuite parameter) are read from the smartcard. They are handled by six procedures.

- BOOLEAN OpenSSLClient(int index, SOCKET s), opens a TLS session with a client EAP-TLS device identified by its index and a TCP/IP socket.

- BOOLEAN OpenSSLServer(int index, SOCKET s), opens a TLS session with a server EAP-TLS device identified by its index and a TCP/IP socket.

- int GetKey(int index, short *cs, char *key), collects the negotiated CipherSuite and the KeysBlock from a EAP-TLS device identified by its index and returns the keys size.

- int recordlayer(short cs, char* keybloc,CTX* ctx, int mode), creates a record layer (either CLIENT or SERVER side), working with a CipherSuite (cs) and a KeysBlock.

- int SSLRead(SOCKET s, char *buf, CTX *ctx), reads information over TLS associated to a TCP/IP socket and returns the data size.

- int SSLWrite(SOCKET s, char *buf, int len, CTX *ctx), writes information over TLS associated to a TCP/IP socket, returns the number of sent bytes.

B. APIs for JAVA environment

Two main java objects are required, the ***tls-tandem*** class building the core framework for the management of TLS session with an EAP-TLS RFID, and the ***recordlayer*** class created at the end of the TLS handshake, which performs all encryption decryption/operations.

The tls-tandem class is made of three main methods:

- public tls_tandem(int mode, ReaderSC reader, String aid, String pin, String identity)

- public recordlayer OpenSession(String ServerName, short Port)

- public void CloseSession(recordlayer RecordLayer)

The constructor of the tls-tandem class setups the software environment for TLS operations with external RFID, according to the following parameters,

- mode is the role of the RFID (either TLS CLIENT or TLS SERVER)

- reader is an abstract representation of the RFID reader, based on the NFC Android model. This object detects the presence of an external RFID feed by the reader, and provides support for IO operations.

- aid, is the Application Identifier for the application store in the RFID. According to the ISO7816 standard, this identifier size ranges between 5 to 16 bytes.

- pin is the optional PIN required for the RFID activation.

- identity is an optional alias that identifies the set of parameters (client's certificate, private key, CA Certificate) to be used by the TLS stack.

The OpenSession method performs the handshake with a remote TLS server identified by its name and its port (usually 443). It returns a recordlayer object initialized with the appropriate cryptographic parameters.

The CloseSession method deletes the TLS framework and the associated resources.

The recordlayer object is created upon a successful TLS handshake. It supports five main methods.

- public byte [] encrypt(byte[] msg). Perform encryption and integrity operations, and return a TLS formatted packet.

- public byte[] send(). Transmit a TLS packet over the TCP socket.

- public byte[] recv(). Receive a TLS packet from the TCP socket.

- public byte [] decrypt(byte[] msg). Decrypt a TLS packet and check its integrity.

- public void Close(). Close the TLS session.

VI. CONCLUSION

In this paper we introduced a new technology dealing with NFC, MIFARE and USB connectivity, which performs identity and access control services, compatible with both the OPENID standard and the MIFARE ecosystem deployed for electronics locks.

REFERENCES

[1] IETF Draft, "EAP support in smartcards", 2002-2012

[2] Urien, P., "Collaboration of SSL smart cards within the WEB2 landscape", CTS'09, 2009

[3] "An OpenID Provider based on SSL Smart Cards", Consumer Communications and Networking Conference, IEEE CCNC 2010, *Best demonstration Award*

[4] IETF Workshop, "Interconnecting Smart Objects with the Internet Workshop", Prague, March 2011.

[5] Urien, P., Kiennert, C. "A new key delivering platform based on NFC enabled Android phone and Dual Interfaces EAP-TLS contactless smartcards", in proceedings of MOBICASE 2011.

[6] Urien, P., Kiennert, C. "A New Keying System for RFID Lock Based on SSL Dual Interface NFC Chips and Android Mobiles", IEEE CCNC 2012

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

[8] P.Urien, "Convergent Identity: Seamless OPENID services for 3G dongles using SSL enabled USIM smart cards", IEEE CCNC 2011.