A breakthrough for prepaid payment: End to end token exchange and management using secure SSL channels created by EAP-TLS smart cards

Pascal Urien, Marc Pasquet, Christophe Kiennert

A Breakthrough for Prepaid Payment: End to End Token Exchange and Management Using Secure SSL Channels Created by EAP-TLS Smart Cards

Pascal Urien
Telecom ParisTech
23 av. d'italie, Paris, France
pascal.urien@telecom-paristech.fr

Marc Pasquet
GREYC CNRS UMR 6072
laboratory, Higher Education National Engineering School of Caen, France
marc.pasquet@ensicaen.fr

Christophe Kiennert
EtherTrust
62bis rue Gay Lussac, Paris, France
christophe.kiennert@ethertrust.com

ABSTRACT

In this paper we present an innovative architecture for prepaid services. Digital tokens are securely exchanged between EAP-TLS smart cards used both by merchant and customer. We describe the global framework that comprises a back-office server delivering tokens, a front-office server collecting tokens from merchant terminal, merchants and customers equipped with smart cards. We detail data exchange choreography and discuss performances issues for the experimental platform built with commercial devices.

KEYWORDS: Security, EMV, TLS, WEB, Smart Card

1. INTRODUCTION

Prepaid facilities refer to services bought in advance; from a legal and practical point of view, they are very different from fiduciary money.

Fiduciary money (coming from the Latin *fiducia* meaning confidence or trust), is typically a paper certificate including some typographic authenticity proofs (banknote…) used to acquire any goods. As an illustration the US dollar could be converted in gold until 1971.

Prepaid services (see [1] for a more detailed analysis) target a specific class of consumer goods, which have been (pre) paid before their purchase.

We divide prepaid systems in two classes, centralized and distributed.

Centralized architectures, such as phone prepaid calls [14], deal with serial numbers, typically printed on vouchers, and hidden by a scratch area. The *serial number* lifecycle is managed by a database, whose interface may be a RADIUS server. The user is registered in the system; his account is credited with the value associated with the serial number, and afterwards is managed by on-line facilities.

In distributed infrastructures, prepaid transactions are performed off-line. The customer generally gets a voucher, with a serial number and some typographic authenticity proofs. The serial number may include cryptographic digits, typically computed with a signature algorithm. As an illustration *Cheque Dejeuner*, a French company specialized in prepaid services, delivered (in 2006) vouchers to 15 millions of customers, for an amount of 2,6 billions €.

In this paper we present a distributed architecture that aims at replacing printed tickets by digital tokens, securely exchanged [5] between smart cards. These tamper resistant devices are widely used for electronic payments, mainly in the EMV framework [4]; they are also deployed as electronic purse [15] in Europe, for micro-payments (a few €) operations.

The central idea of our proposal is to establish secure SSL tunnels between smart cards, or between smart cards and SSL servers. SSL [8] [9] is the de facto standard for the Internet security; it is the cornerstone of the e-Commerce. Digital tokens are provisioned from back office servers. They are exchanged via customers and merchant smart cards. They are cleared through sessions between merchant smart cards and front office servers.

This paper is organized according to the following plan. Section two introduces payment systems. Section three unveils our new prepaid framework. Section four presents EAP-TLS smart cards and performances issues. Finally section five concludes this paper.

2. ABOUT PAYMENT SYSTEMS

According to [1] "Commerce always involves a ***payer*** and a ***payee*** – who exchange money for goods of services – and at least one financial institution – who links *bits* to *money*". Generally two financial entities are involved in a transaction, an ***issuer*** acting for the payer, and an ***acquirer*** used by the payee (frequently refereed as the *merchant*).

2.1. About card transaction

A payment card transaction usually consists of two steps [2], *transaction flow* and *clearing and settlement*. Depending on the card type payer and payee may be credited / debited according to different policies.

The *transaction flo*w is used for authorization purpose. Merchant-acquirer and issuer bank are connected via a secure *payment card network* (PCN) managed by card companies. Some of them (like American Express) may also play the role of acquirers. The merchant collects information about the card (number, date of expiry, type, etc.) via a terminal device, and sends it to the acquirer. This latter forwards these data (but this is not always mandatory) to the issuer financial institution (through the PCN) that checks the account status. A response is returned to the acquirer, which upon success, delivers an ***authorization*** code to the merchant terminal.

The *Clearing and Settlement* process deals with payment operations and funds transfer, typically occurring within a few days.

2.2. Security Issues

The payment card industry data security standard (or *PCI DSS* [3]), specifies security requirements dispatched in six main groups.

1- *Build and Maintain a Secure Network*, by deploying secure physical infrastructure (firewall…) and logical protections (*Virtual Private Networks*…).

2- *Protect Cardholder Data*, by using cryptographic means (strong encryption…) for data storage and transmission.

3- *Maintain a Vulnerability Management Program*, by deploying anti-virus software, updating operating systems with security patches, developing secure WEB applications.

4- *Implement Strong Access Monitor Control Measures*, by avoiding passwords for access control and using two factors authentications tokens or biometric means, especially for operations dealing with cardholder data.

5- *Regularly Monitor and Test Networks*, by tracking and monitoring accesses to the network and by regularly testing system security and processes.

6- *Maintain Information Security Policy*, which targets information security for both employees and contractors.

2.3. The EMV Use Case.

EMV acronym comes from *Europay MasterCard and Visa*, the three companies that in 1994 initiated this standard [4]. Today most of electronic payments are performed with EMV cards. In 2009 more than 944 million EMV compliant chip-based payment cards were deployed in the world.

In non electronic payment, card information (number, validity date…) is read from the magnetic strip. In the EMV technology a tamper resistant micro-controller (a smart card), stores cardholder data, but also supports additional security features.

The card is equipped with an issuer certificate, and contains a signature (called the SSAD, *Signed Static Application Data*) of the embedded information, computed with the associated private key.

The device optionally includes an individual certificate (delivered by the issuer), and a private key, which proves the electronic chip authenticity thanks to a challenge/response mechanism, called DDA (*dynamic authentication*).

The EMV card generates *application cryptograms* (AC) generally based on a triple DES algorithm working with a 112 bits secret key.

A transaction is initiated by the merchant terminal, via a GENERATE AC command (ARQC, *Authorization Request Cryptogram*), conveying various parameters such as the amount or the date. Upon success, the device returns a *cryptogram data information* (CDI).

The merchant forwards an authorization request to the acquirer that comprises the ARQC, the CDI, and

additional attributes. Depending on the response, the terminal indicates its decision to accept or decline the transaction in a second GENERATE AC command.

The card generates a *Transaction Certificate* (TC) for an approval or an *Application Authentication Cryptogram* (AAC) for a decline.

The merchant transmits this notification to the acquirer via a confirmation message.

The datagram collected by the first GENERATE AC command, indicates if a PIN was successfully given by the cardholder. This mechanism enforces a dual factor authentication, i.e. a cryptographic key stored in the chip and a PIN known by a human user.

2.4. Prepaid Systems

As mentioned by [1] in *prepaid systems*, "a certain amount of money is taken away from the payer, before purchases are made". This mode of operation works for electronic purse and bank checks.

3. A NEW PREPAID FRAMEWORK

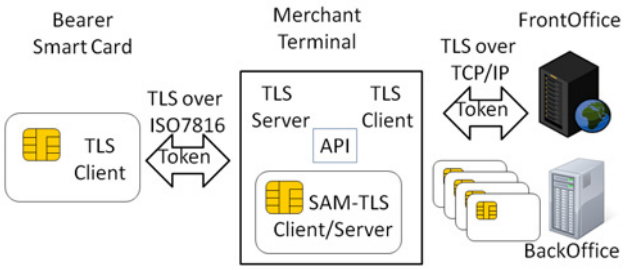

Figure 1. Main Components

3.1. Overview

The customer generally gets about twenty vouchers for a month. Each has a serial number, a value date, a value (e.g. 7.5 €) and some typographic authenticity proofs. The serial number may include cryptographic digits, typically computed with a signature algorithm. The Front End issues the vouchers by delegation of the Back End. These vouchers are sent to the card, by the way of a secure channel and are stored in the chip, through the terminal. Then the card is able to use these vouchers, but only one or two each time. The voucher(s) is (are) sent by the card to the Merchant terminal, by a secure channel, to be stored in the chip of the SAM (*Secure Access Module*). At the end of the day, the different vouchers stored during the day in the SAM are collected by the Front End by a secure channel, verified and destroyed. The value is used to credit the merchant through the Back End.

3.2. Bearer

The cardholder belongs to a company that offers him a value added service we can call “lunch voucher”. This voucher is paid half by the company, half by the bearer. It can be used to pay a part, or the totality, of the bearer’s lunch. The bearer’s company buys the vouchers to a specialized company as *Cheque Dejeuner,* in Europe. The former system delivered these prepaid vouchers in paper checks. They were sent to the company and given to the bearer who may use no more than two vouchers per day. The vouchers are accepted as lunch payment in a large number of European restaurants.
We have designed the new electronic architecture to be as close as possible to the former one, for two main reasons: in one hand to protect the value added chain of the specialized company, and in another hand not to confuse the bearers.
The bearer could use an Internet connection to know how many tokens are still available.

3.3. Smart Card

The smart card is issued by the specialized company from its Back End and send to the Bearer’s company which delivers it to the cardholder. This Smart Card is empty and has to be inserted in the Merchant terminal to be filled with tokens. A technical description of the card is given in § 4.1.

3.4. Tokens

Tokens are issued by the Front End, which gets a list of tokens to issue from the Back End. The life cycle of the token is: existence in the Back End, creation in the Front End, life in the bearer’s Smart card or in the Merchant’s SAM, destruction in the Front End and trace of existence in the Back End.
The token is made of: an X509 certificate and a record containing the voucher value (the bearer’s company can choose any values for the voucher from 1€ up to 20€) in clear and seal in the certificate. Thus, the value can be read by the merchant terminal and compared in the Front End after the daily collection.
Tokens are stored only in secure devices: Smart card, SAM, HSM (*Hardware Security Module*).

3.5. Merchant Terminal

The merchant terminal dialogs with the bearer's card to accept the payment with tokens. The terminal verifies the value date, the value of the token and gets one or two tokens as ordered by the terminal dialog with the bearer. The PIN code is not necessary at this level. This terminal is contactless for payments by tokens and uses its contact capability to fill the card with tokens (when the card is empty, the cardholder enters his PIN code, the terminal calls the Front End and lets the smart card create a secure channel to transfer the tokens from the Front End to the smartcard).
The merchant terminal assures the transmission flow: from the smartcard to the SAM plugged under the terminal, from this SAM to the Front End, from the Front End to the smartcard as shown in figure 2.

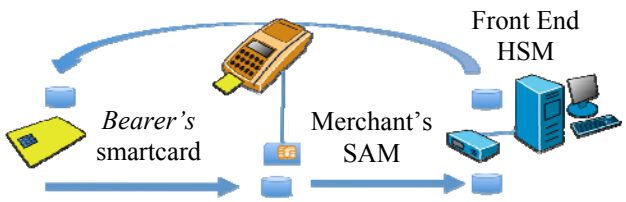

Figure 2. Token exchanges

A phone line, a GSM line or an Internet connection, connects the Merchant terminal to the Front End either.
Today the terminals are specific but in the future software will be included in EMV POS to limit the implementation costs.

3.6. Issuer/Acquirer Front End

The Front End is:

- Issuer of tokens. In fact, the Front End is connected to a Hardware Security Module. This HSM creates tokens, which are stored encrypted in the Database of the Front End (FE). Only the registered cards can be fulfilled by the corresponding Tokens (cards and tokens belong to the bearer's company).
- Acquirer of Merchant's Terminal data. The terminal phone calls one time each day the FE to perform the collection of tokens stored in the SAM. The FE can collect only registered terminals.

If the verification of the token is valid, the FE destroys the different tokens and advises the Back End to credit the merchant account.
Security software manages fraud prevention.

3.7. Issuer Back End

The Back End (BE):

- Issues the different cards: bearer's cards and SAM, as well as the PIN code (the cards are personalized with the files issued from the Back End)
- Manages the different flows: token creation, accounts credit and debit.

A specific Web access for the Bearer, the Bearer's Company and the Merchant is interfaced from the Back Office (the Back End is considered as a part of the Back Office of the specific company).

4. EAP-TLS smartcards

An EAP-TLS [11] [12] [13] device establishes a secure TLS tunnel between the bearer and the merchant terminal. The card located in the terminal is called the *Secure Access Module* (SAM). It supports server and client resources, is used first to collect tokens from the customer, and second to backup these elements to the front office server.

4.1. About Smartcards and EAP-TLS Devices.

A smart card [5] is a tamper resistant microcontroller, whose size is about 25 mm^2, which includes a CPU (8, 16, 32 bits), and various type of memories such as ROM (a few hundred KB), RAM (about 10 KB), and non volatile memory (FLASH or E^2PROM, about 128 KB). Security is enforced by multiple hardware and software countermeasures. About 5 billions of these electronics chips were produced in 2010, for telecommunication applications (SIM cards, 70% of the market) or financial transactions (EMV cards, 20% of the market). Most of these components run a *Java Virtual Machine* (JVM) [6], and therefore may execute applications written for Javacards with a subset of the Java language.

The SSL protocol (and TLS, a modified version standardized in 1999 by the IETF) is the cornerstone for secure exchanges over the Internet [8] [9]. Everyday billions of net surfers establish protected HTTP sessions for various purposes such as eMail reading, or electronic commerce transactions. This protocol has been deeply analyzed by the research community and enforces strong security; it supports a wide range of cryptographic algorithms such as asymmetric calculations for key exchanges (RSA, DH, ECC...), symmetric functions for data privacy (RC4, 3xDES, AES) and hash procedures (MD5, SHA1, SHA256) for ephemeral keys generation.

Although TLS messages are usually shuttled by the TCP transport, a new protocol (EAP-TLS) was invented in 1999 [10] in order to establish TLS sessions without TCP flavors, according to a datagram paradigm.

EAP-TLS smartcards [11] [12] [13] were initially designed for access control in Wi-Fi networks, in an IEEE 802.1x infrastructure. They may be used both on the client terminal, and on the authentication server side (typically a RADIUS server); they dialog through EAP packets and setup TLS tunnels.

EAP-TLS applications are written for javacards, the code sizes are respectively 20KB and 25 KB for client only and client/server applet.

4.2. Performances Issues

A TLS session is opened according to two different modes: full and resume.

A full session is opened via a four way handshake; it usually deals with PKI facilities. The server delivers his certificate (and those of the Certification Authority), the client computes a *PreMasterKey* and forwards it encrypted with the server public key (found in its certificate). Optionally, but in our case we always implement this facility, the client is equipped with a X509 certificate sent to the server and authenticated by a signature generated with the client private key.

A resume (or abbreviated session) re-uses a previously computed *MasterSecret* by a full session. It works according to a three ways handshake and only deals with symmetric cryptographic procedures.

We need to perform a transaction between the cardholder and the merchant terminal in an acceptable amount of time, five seconds seems a realistic limit.

EAP-TLS applications are written for javacards; these components are equipped with crypto-processors offering RSA facilities, but hash functions are provided by procedures executed by a modest CPU.

RSA Pub 1024	RSA Priv 1024	MD5 /bloc 64B	SHA1 /bloc 64B	3xDES /bloc 8B	AES /bloc 16B	RC4 /byte
25	550	0,50	0,90	2,10	2,60	0,50

Figure 3. Computing Performances (in ms)

Basic cryptographic performances are illustrated by figure 3, for the best Javacard available on the market that we could test. Furthermore we notice a data throughput of about 6000 Bytes/s.

The opening of a full session based on 1024 bits RSA cryptography, requests 2,0s. It exchanges 2,500 bytes, whose transfer costs 0,40s. About 230 MD5 and SHA1 calculations are performed (dealing with 64 bytes blocs), which consumes 0,32s. One RSA operation with the private key, and two RSA procedures with public keys, are computed in 0,60s. The sum of the previously cited operations is 1,32s; the remaining time (0,68s = 2,00 - 1,32) is burnt by the java code execution.

The opening of a resume session costs 0,50s. It requires the exchange of 250 bytes (0,04s), and the calculation of 75 MD5 and SHA1 that consumes 0,11 s. The remaining time (0,35s) is needed by the java code execution.

Because both the customer and the merchant terminal use an EAP-TLS device, a session opening costs 4,0s or 1,0s depending on the operating mode either full or resume.

Once the secure TLS channel has been established, messages exchanged by the client and the server are encrypted with the selected algorithm and integrity is enforced by a HMAC procedure. According to figure 3, we observe that the computing time is mostly consumed by encryption and decryption operations (in other word hash calculations have a second order effect). As an illustration, for a classical RC4, HMAC-MD5 cipher suite, the transfer of 1000 bytes burnt 500ms for encryption, 16ms for MD5 hash, and 170ms for data transmission.

4.3. A Transaction Scenario

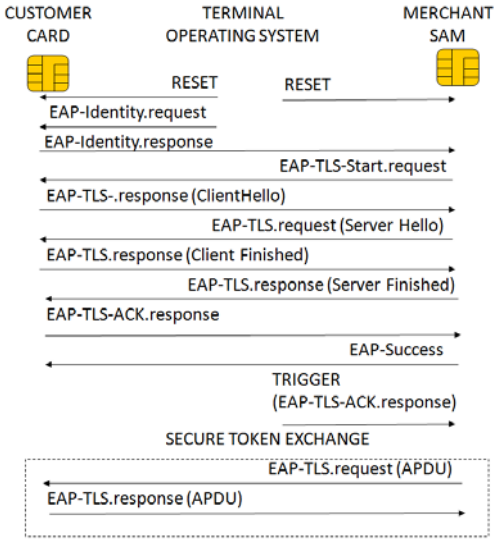

Figure 4. A Transaction Scenario

As described in [11], EAP messages are transported by ISO7816 [5] packets whose maximum size is about 256 bytes; consequently a segmentation/reassembly process is managed by the ISO7816 driver.

When the merchant terminal starts a new transaction (see annex I for packets details) it performs a *reset* both on client and SAM cards (see figure 4).

Afterwards according to the EAP standard it sends an *EAP-Identity.request* message to the client, which returns an *EAP-Identity.response*, forwarded to the server.

This later produces an *EAP-TLS-Start.request* packet that notifies the beginning of an EAP-TLS session.

Thereafter TLS messages are shuttled in EAP-TLS packets, the client receives requests and transmits responses, while the server works in a symmetric way.

Upon success, the server forwards a last packet, called EAP-Success, which indicates that a new SSL session has been opened.

At this step client and server devices are ready to exchange encrypted information in EAP-TLS packets.

The terminal delivers a *ServerTrigger* message, expressed as an *EAP-TLS-ACK.response*. The SAM builds a command encrypted according to the negotiated CipherSuite, and encapsulated in an *EAP-TLS.request* packet.

Thereafter commands used for token exchange are encapsulated by EAP-TLS requests (delivered by the server) and responses (produced by the client), transparently conveying TLS messages.

4.4. Connection to the Front End server

This scenario (see figure 5) occurs when the SAM needs to backup previously collected tokens to the front office. Packets are detailed in annex II.

The terminal manages the TCI/IP connectivity thanks to the well know sockets API. It opens a TCP session with the remote Front Office server on the port 443.

It sends a *Reset* command to the SAM, and an *EAP-TLS-Start.request* packet indicating the beginning of the EAP-TLS session. The SAM returns an *EAP-TLS.response* packet transporting the first TLS message (the *TLS Client Hello Message*).

A software bridge converts *EAP-TLS.response* packets in outgoing TLS messages (forwarded to the remote server), and ingoing TLS messages (received from the remote server) in *EAP-TLS.request packets* (sent to the SAM)

If a TLS session is successfully opened, the terminal sends a *ClientTrigger* message, expressed as an *EAP-TLS-Ack.request* packet. The SAM builds a command, encrypted according to the negotiated *CipherSuite*, and encapsulated in an *EAP-TLS.response*.

Afterwards the software bridge transforms *EAP-TLS response* in a TLS message, and a secure dialog occurs between the SAM and the Front Office in order to extract tokens.

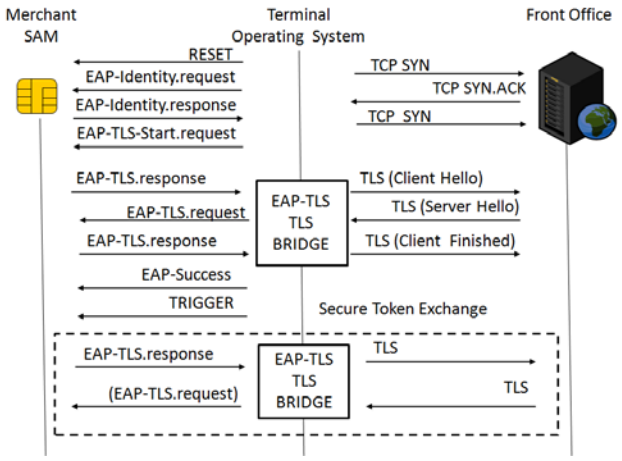

Figure 5. Connection to the Front Office server

5. CONCLUSION

In this paper we have presented an innovative architecture for prepaid services. Experimental platform demonstrated that performances are compatible with real time constraints. The next step targets the deployment of this system for a set of beta customers.

We strongly thank *Cheque Dejeuner Group* for their help. The deployment will be realized on a set of their customers.

ANNEX I

This annex illustrates an EAP-TLS dialog, in the resume mode between the client and the SAM. EAP-TLS header is underlined, TLS message is in bold.

Commands sent to smart cards are called APDUs, and are defined by the ISO7816 standard. Request messages have a prefix, whose length is five bytes, respectively noted

CLA INS P1 P2 P3. The P3 attribute is the size of the body, or the size of the requested information. Response messages have an optional body, and end by a two bytes status; the hexadecimal value 9000 indicates a successful operation. EAP-TLS messages [10] begin by a four bytes prefix noted, *code*, *identifier*, and *length*. Code defines the class of the message: 01 for request, 02 for response and 03 for success Identifier is a label used both for requests and responses. The length attribute is the EAP message size. The fifth byte indicates the type of operation, 01 is used for Identity and 0B for EAP-TLS. The sixth byte (the flag attribute) gives more details about the EAP-TLS packet structure. The hexadecimal value 80 is followed by the content size coded by a 32 bits number; a null value means that no options are used. The content of an EAP-TLS packet is a TLS message whose format is precisely detailed in [9].

The Terminal sends *Reset* to the Client.
Tx: A0 19 10 00 00
Rx: 90 00

The Terminal sends *Reset* to the SAM.
Tx: A0 19 10 01 00
Rx: 90 00

The Terminal sends *EAP-Identity.request* to the Client.
Tx: A0 80 00 00 05 01 A5 00 05 01

The Terminal sends *EAP-Identity.response* to the SAM.
Tx: A0 80 00 00 0B 02 A5 00 0B 01 63 6C 69 65 6E 74

The SAM sends *EAP-TLS-Start.request* to the Terminal.
Rx: 01 A6 00 06 0D 20 90 00

The Terminal sends *EAP-TLS.response* to the SAM.
Tx: A0 80 00 00 5C 02 A6 00 5C 0D 80 00 00 00 52 **16 03 01 00 4D 01 00 00 49 03 01 4D 08 FA F0 06 D4 E5 C2 80 74 D6 54 54 3D 96 04 6D D1 A9 4D 8B 98 AF 5C 49 8D EB 1A C8 96 5A 42 20 57 25 5C 81 71 17 79 C8 E1 44 1A 51 6E FD 53 21 32 EC 2D 07 F8 75 96 36 90 DA BC 37 45 B1 9B C0 00 02 00 04 01 00**

The SAM sends *EAP-TLS.request* to the Terminal,
Rx: 01 A7 00 84 0D 80 00 00 00 7A **16 03 01 00 4A 02 00 00 46 03 01 4D 08 FA F0 6C AC 27 C3 26 C8 0E 46 F7 84 08 A6 45 B8 EC 0D 57 37 93 4C 53 E5 01 19 83 17 25 E3 20 57 25 5C 81 71 17 79 C8 E1 44 1A 51 6E FD 53 21 32 EC 2D 07 F8 75 96 36 90 DA BC 37 45 B1 9B C0 00 04 00 14 03 01 00 01 01 16 03 01 00 20 A7 3D 21 C8 98 95 A1 B6 ED 74 C2 F0 D1 9D 04 E0 75 DC C1 CF 67 0A 0E B3 9B 90 87 CD A6 64 BF AD** 90 00

The Terminal sends *EAP-TLS.response* to the SAM.
Tx: A0 80 00 00 35 02 A7 00 35 0D 80 00 00 00 2B **14 03 01 00 01 01 16 03 01 00 20 B9 3C 03 09 E1 F9 A 60 C0 D9 F3 28 51 DF D5 6F 66 29 60 23 5A 10 0A E1 A1 67 A2 F9 70 73 81 C7**

The SAM sends *EAP-Success* to the Terminal.
Rx: 03 A7 00 04 90 00

The Terminal sends the *Server Trigger* to the SAM.
Tx: A0 80 00 00 06 02 A8 00 06 0D 00

The SAM sends *EAP-TLS.request* to the Terminal. One data byte is sent encrypted and HMACed by the SAM.
Rx: 01 A9 04 1F 0D 80 00 00 00 16 **17 03 01 00 11 AB 6C B3 96 B4 D9 59 2B 03 94 9F C6 13 F2 E4 D1 81** 9000

ANNEX II

This annex illustrates a TLS connection, in the resume mode, between the SAM and the *Front Office* server. EAP-TLS header is underlined, TLS message is in bold.

The Terminal sends *Reset* to the SAM.
Tx: A0 19 10 00 00
Rx: 90 00

The Terminal sends *EAP-Identity.request* to the SAM.
Tx: A0 80 00 00 05 01 A5 00 05 01

The SAM sends *EAP-Identity.response* to the Terminal.
Rx: 02 A5 00 0B 01 63 6C 69 65 6E 74

The Terminal sends *EAP-TLS-Start.request* to the SAM.
Tx: A0 80 00 00 0A 01 A6 00 06 0D 20 6E 07 16 4D

The SAM sends *EAP-TLS.response* to the Terminal.
Rx: 02 A6 00 5C 0D 80 00 00 00 52 **16 03 01 00 4D 01 00 00 49 03 01 6E 07 16 4D 0F FE F3 77 29 E7 B2 75 02 F7 BE 89 6C CB CB 52 15 9A CB EC 9B 38 C7 F9 3E 45 E8 32 20 FC F1 46 8E 74 CF F7 44 17 78 9B A2 C5 C9 A1 97 A9 2D BC D1 58 7B 0B BC 63 73 AA 2A EA CE 97 88 00 02 00 04 01 00** 90 00

The Terminal sends *EAP-TLS.request* to the SAM.
Tx: A0 80 00 00 80 01 A7 00 80 0D 00 **16 03 01 00 4A 02 00 00 46 03 01 4D 16 07 6E E7 AB C0 25 F6 E5 0B DA 97 49 7D 31 0F 6E 9F 3F EA FD 68 75 CB E8 50 86 A5 27 8B 0E 20 FC F1 46 8E 74 CF F7 44 17 78 9B A2 C5 C9 A1 97 A9 2D BC D1 58 7B 0B BC 63 73 AA 2A EA CE 97 88 00 04 00 14 03 01 00 01 01 16 03 01 00 20 C6 A3 AF 49 D2 8E 06 97 2A 94 6C 4C 9D 69 A5 DA 9D 29 41 DB 57 31 47 23 72 42 C4 0F 0A 8A FD 66**

The SAM sends *EAP-TLS.response* to the Terminal.

Rx: 02 A7 00 35 0D 80 00 00 00 2B **14 03 01 00 01 01 16 03 01 00 20 40 F9 AF FF D9 E9 4F 8D 58 24 D1 A9 67 36 D1 34 C2 F3 D7 EC 56 D3 46 F7 9E D7 5B 32 43 7F D9 E0** 90 00

The Terminal sends the *trigger* command (*EAP-TLS-ACK.request*) to the SAM.
Tx: A0 80 00 00 06 01 A8 00 06 0D 00

The SAM sends *EAP-TLS-ACK.response* to the Terminal, indicating that no data are available.
Rx: 02 A8 00 0A 0D 80 00 00 00 00 90 00

The Terminal sends *EAP-TLS.request* to the SAM, including four bytes of data, HMACed by the TLS server.
Tx: A0 80 00 00 21 01 A9 00 21 0D 00 **17 03 01 00 16 E2 F1 90 92 A8 87 D3 0E CC 9E 41 DD 39 77 0E 7A 39 D6 E8 ED 30 4A**

The SAM sends EAP-TLS.response to the TLS server, including four bytes of data, HMACed by the SAM.
Rx: 02 A9 00 25 0D 80 00 00 00 1B **17 03 01 00 16 CA 2C B8 9F 8A 43 A7 33 D0 2E DA 76 0D 27 A8 0A 14 A6 02 25 3E 19** 90 00

REFERENCES

[1] Asokan, N.; Janson, P.A.; Steiner, M.; Waidner, M.; "The state of the art in electronic payment systems", Computer Volume: 30, Issue: 9, 1997

[2] Jing Liu; Yang Xiao; Hui Chen; Ozdemir, S.; Dodle, S.; Singh, V.; "A Survey of Payment Card Industry Data Security Standard", Communications Surveys & Tutorials, IEEE Volume: 12, Issue: 3, 2010

[3] PCI Security Standard Council, "Payment Card Industry Data Security Standard", http://www.pcisecuritystandards.org. September 2006.

[4] *EMV Books, 1 - Application Independent ICC to Terminal Interface Requirement and Application Selection, Book 2 - Security and Key Management, Book 3 - Application Specification, Book 4 - Cardholder, Attendant, and Acquirer Interface Require*, www.emvco.org

[5] T.M. Jurgensen, et al., SMART CARDS: THE DEVELOPER'S TOOLKIT, Prentice Hall, 2002.

[6] Z. Chen, JAVACARD™ TECHNOLOGY FOR SMART CARDS: ARCHITECTURE AND PROGRAMMER'S, The Java Series, Addison-Wesley, 2002

[7] M. Pasquet, "La sécurisation d'un système informatique complexe: le cas de la monétique", Habilitation à Diriger des Recherches, France, November 2008.

[8] E. Rescorla, SSL AND TLS DESIGNING AND BUILDING SECURE SYSTEMS, Addison-Wesley, 2001

[9] RFC 2246*, The TLS Protocol Version 1.0*, IETF, 1999

[10] RFC 2716 "PPP EAP TLS Authentication Protocol ", 1999

[11] P.Urien, G.Pujolle, "Security and Privacy for the next Wireless Generation", International Journal of Network Management, IJNM, Volume 18 Issue 2 (March/April 2008), WILEY

[12] P. Urien, "Collaboration of SSL smart cards within the WEB2 landscape", Collaborative Technologies and Systems, 2009. CTS'09. International Symposium, May 2009

[13] P.Urien, "EAP Support in Smartcard", draft-urien-eap-smartcard-20.txt, IETF DRAFT, 2011.

[14] Cakaj, S.; Shefkiu, M.; Haxha, S.; "Implementation of prepaid services in Kosovo's fixed network", ELMAR '09. International Symposium

[15] CEPSCO, Common Electronic Purse Specification, Technical Specification version 2.2, 2000