# SSL dans une carte à puce

[1]Pascal URIEN, [2]Hayder SALEH, [3]Adel TIZRAOUI

## I. INTRODUCTION

Ce papier présente *la carte à puce internet*, une nouvelle technologie (baptisée ***Oversoft***) permettant de renforcer la sécurité des applications multimédia distribuées. Par ses caractéristiques physiques et logiques, la carte à puce est considérée aujourd'hui comme le dispositif informatique portable le plus sure. Notre concept est d'exploiter cet avantage pour renforcer les modèles de sécurité réseaux actuels. Ces derniers se basent sur des éléments logiciels dont la faible résistance aux attaques et les configurations complexes sont les principaux inconvénients. La technologie Oversoft permet à une carte à puce de partager les ressources réseaux d'un terminal banalisé et ainsi, d'accéder à internet en mettant en œuvre des sessions client - serveur standards (c'est à dire définies par des RFCs). Dans ce modèle, la carte est positionnée entre l'application (par exemple un navigateur) et le réseau, elle peut donc ouvrir et gérer une session SSL avec un serveur distant. Il en résulte deux avantages majeurs , d'une part c'est la carte qui vérifie le certificat du serveur et non plus le code peu sure du navigateur, d'autre part elle peut loger un certificat X.509 permettant l'authentification du client.

## II. LA CARTE A PUCE

### A. *Un bref historique.*

La carte à puce à microprocesseur (SPOM) a été inventée au début des années 80 par Michel Ugon [2]. A travers le GIE cartes bancaires des dizaines de millions d'exemplaires ont été mis en circulation. La carte à puce classique c'est une tranche de silicium de 25 $mm^2$, qui comporte un microprocesseur 8 bits cadencé à 3.3 Mhz, une RAM de quelques centaines d'octets, des mémoires non volatiles ROM et EEPROM de l'ordre de quelques dizaines de Kilo-octet. Les puissances de traitement actuelles sont de l'ordre du MIPS, mais une nouvelle génération de composants à base de processeurs RISC 32 bits, disponible courant 2000, permettra des capacités de traitement supérieures à 33 MIPS [3].

[1] Pascal.Urien@Bull.net - Bull CP8 R&D, 68, route de Versailles 78431 Louveciennes Cedex.
[2] Hayder.Saleh@Bull.net - Doctorant Cifre.
[3] Adel.Tizraoui@Bull.net - Doctorant Cifre.

La puce réalise schématiquement deux types de services :

- Des algorithmes cryptographiques utilisés pour le chiffrement, l'authentification, la génération de certificat.
- Le stockage de données qui sont protégées en lecture ou écriture par différentes méthodes d'authentification.

La qualité principale d'une carte à puce est d'être *Tamper Resistant* [5], c'est à dire qu'elle résiste à des attaques extérieures menées dans le but de lire les données ou le code stocké dans la puce. Une particularité d'un tel objet informatique est l'absence de périphériques d'entrée sortie classiques tels que, clavier, écran ou carte de communication. Il est donc nécessaire d'associer à une carte d'une part un lecteur (qui réalise des services de bas nivaux tels que alimentation en énergie et en horloge), et d'autre part un terminal offrant des ressources d'entrées sorties et par exemple un accès réseau.

### B. *Les architectures à base de carte à puce.*

Les architectures à bases de cartes se sont historiquement développées dans un contexte bancaire, le terminal est réputé sûre, le couple terminal carte est identifié de manière visuelle, ainsi une carte bleue fonctionne avec un terminal carte bleue.

La norme ISO7816 [1] définit le format des ordres qui sont échangés entre une carte et le terminal, que l'on nomme encore dans l'appendice iso 7816-4, APDU. Le terminal émet une commande (.command), la carte répond (.response). De fait l'ISO a normalisé un format de message, mais pas leur sémantique, c'est à dire qu'il n'existe aucun moyen pour envoyer à priori les bonnes commandes à la bonne carte. De surcroît aucun procédé normalisé ne permet d'identifier une carte particulière, c'est donc une information connue par ailleurs qui permet de constituer un couple correct carte terminal.

### C. *Les architectures ouvertes Java, SCW.*

Les cartes traditionnelles sont mono-application, elles sont en fait conçues et fabriquées pour une fonction unique liée à un opérateur particulier (banques, opérateur de téléphonie mobile, ministère de la santé ...); de manière générale le porteur ne contrôle pas l'information qu'il transporte.

En 1997 est apparu la JavaCard [10] qui intègre une machine virtuelle Java, c'est le premier pas vers des architectures multi-applications écrites bien évidemment en Java.

Cependant, la multiplication des services disponibles implique un nombre encore plus important de logiciels spécifiques (nommés Application Protocol dans la terminologie OCF [9]) qui doivent être présent sur chaque terminal ou la carte sera utilisée.
En 1999 Microsoft [11] a annoncé Smart Card for Windows (SCW), une carte à puce qui sera disponible et intégré avec son système d'exploitation, courant 2000. La compagnie justifie son choix par sa conviction que les cartes sont une composante clé de la sécurité (et donc du commerce électronique), et que pour compenser l'absence d'un standard de haut niveau pour leur mise en œuvre, une des solutions est d'offrir une interface familière à un utilisateur (le *look and feel* windows).
Cette architecture a l'inconvénient de n'être réellement ouverte que pour un système d'exploitation unique.
Par contre, notre modèle s'applique à un utilisateur mobile qui utilise des terminaux variés accédant à internet, (Les ordinateurs personnels, les téléphones mobiles, les assistants personnels, les consoles de jeu...). Ces terminaux banalisés accèdent à des services divers qui nécessitent l'authentification d'un client et la réalisation de paiements, et pour lesquels la carte à puce est un vecteur idéal. Nous pensons donc que la mise en œuvre ouverte d'une carte (c'est à dire sans avoir à imposer à l'utilisateur un système d'exploitation ou une configuration particulière) est la clé du succès des futurs services qui seront créés et disponibles au moyen du réseau internet.

## III. LA CARTE A PUCE INTERNET

La technologie carte à puce internet [15,16,17,18,19], a pour objectif de transformer cette dernière (associée à un terminal) en un nœud internet. En conséquence la puce se comporte comme un serveur et/ou un client TCP.
L'intérêt primordial d'un serveur web logé dans une carte est de permettre à un navigateur de communiquer avec la puce. Par exemple l'url http://127.0.0.1:8080 donne accès à la page html de garde (home page) d'une carte à puce, qui présente une liste (associée à des hyperliens) des ressources hébergées. Les ressources embarquées sont des objets (éventuellement multimédia) tels que page html, images, applets ou connexion à un serveur distant. Certaines d'entre elles peuvent être associées à un *pin code* (l'équivalent du code d'une carte bleue ou d'une carte SIM) qui sera demandé à l'aide d'un formulaire html.
L'association simultanée d'un serveur et d'un client localisés dans une carte permet d'introduire la notion de *trusted proxy*. Un tel objet associe à une url carte (par exemple http://127.0.0.1:8080/eMail) une procédure de connexion avec un serveur web externe (par exemple un serveur de courrier gratuit). On introduit ainsi la notion de *fichier virtuel*, c'est à dire un objet qui est associé à une url carte, mais qui en fait est logé sur un serveur distant, quelque part dans le web. La carte se comporte comme un interface d'accès à cet objet, elle sera par exemple responsable de la procédure d'authentification.

La carte à puce internet est *url centric*, toutes les ressources qui lui sont associées sont identifiées par une url spécifique. Elle partage en fait les couches TCP/IP présentes sur le terminal auquel elle est reliée, en particulier elle hérite de l'adresse IP de ce dernier, et de toute sa configuration réseau. Pour réaliser cet objectif nous avons défini une nouvelle pile protocolaire dénommée Smart Layer, qui est organisée autour d'un protocole original SmartTP (Smart Transfer Protocol [19]), une version allégée du protocole TCP.

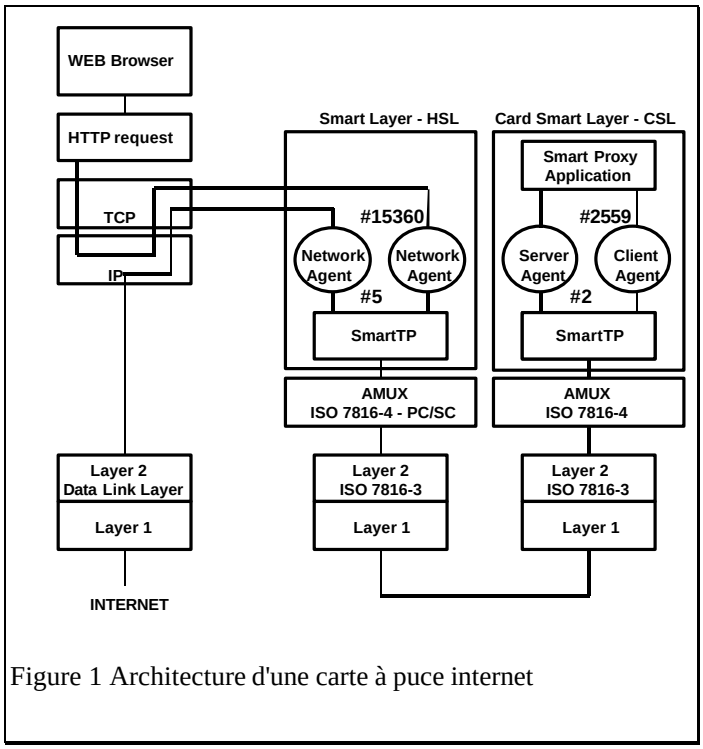


Figure 1 Architecture d'une carte à puce internet

## IV ARCHITECTURE

L'architecture (Cf figure 1) comporte deux piles protocolaires (Smart Layer) une logée sur le terminal (Host Smart Layer HSL) et l'autre dans la puce (Card Smart Layer CSL). Chaque pile s'appuie sur des couches OSI 1 et 2, conformes aux normes ISO 7816 , et comporte deux sous ensembles:

- Un entité smartTP, qui route des SmartTP_pdu
- Des SmartTP agents, qui échangent des SmartTP_pdu

Typiquement deux agents situés respectivement dans la puce et le terminal dialoguent au moyen du protocole SmartTP, transporté par des APDUs spécifiques. Une couche [4]AMUX assure l'aiguillage des données protocolaires (SmartTP pdu) vers une entité SmartTP.

[4] Nous désignons par AMUX (Apdu MUltipleXeur) la couche logicielle logée sur le terminal ou la carte, qui réalise la commutation des APDUs vers les applications utilisatrices. En 1996 un ensemble d'industriels de la carte à puce et de systèmes informatiques ont adopté le standard PC/SC[8] qui permet l'intégration des lecteurs de cartes à puce dans les machines informatiques

Une unité de donnée SmartTP (SmartTP pdu, protocol data unit) comporte une référence de destination (2 octets), une référence source (2 octets), un champ flag de 8 bits (1 octet), et de manière optionnelle des octets d'information (240 octets au plus). Une référence est l'équivalent d'un port TCP ou UDP. Un jeton est un pdu de 5 octets, qui ne comporte aucun octet d'information.
L'entité SmartTP (associée à une référence nulle) se comporte comme un commutateur qui dirige les SmartTP pdu depuis/vers des entités logicielles autonomes nommées Agents. Deux Agents sont mis en relation par une session, et échangent des SmartTP pdu. Un flag OPEN définit l'ouverture d'une session, un flag CLOSE indique la fermeture d'une session. Une agent peut se bloquer c'est à dire émettre un Smart_TP pdu avec un flag BLOCK positionné, et suspendre toute activité jusqu'à la réception d'un nouveau pdu. Deux flags additionnels (READ et WRITE) permettent de construire le paradigme OPEN-READ-WRITE-CLOSE (bloquant ou non) utilisé dans les systèmes Unix pour la manipulation des fichiers.
Côté terminal un agent réseau accède à un [5]SAP (typiquement de niveau 4, c'est à dire relié à la couche TCP ou UDP du terminal), il se comporte comme un traducteur de protocole TCP/SmartTP ou UDP/SmartTP.

### *A. HTTP.*

Une session entre un agent carte web (qui implémente le protocole http) et un agent terminal réseau (par exemple un serveur TCP sur le port 8080) permet de transformer une carte à puce en un serveur web qui utilise la pile TCP/IP du terminal auquel la puce est reliée mais qui réalise le protocole HTTP.

### *B. Trusted Proxy*

Une application logée dans une puce et activée au moyen d'une URL (c'est une notion similaire au CGI), peut accéder au web à l'aide d'une session supplémentaire avec un agent réseau se comportant comme un client TCP. L'association de deux sessions simultanées permet de réaliser un proxy qui relie par exemple un navigateur à un serveur externe. C'est la notion de *trusted proxy* (Cf figure 1).
Un fichier virtuel associe à une URL carte une procédure de connexion à un serveur externe. Les données reçues par la puce sont ensuite traitées et retransmises vers le navigateur.

[5] Un *Service Access Point* [12] est une interface logicielle avec une couche réseau. Ainsi un pilote de carte réseaux (NDIS…) est un SAP de niveau 2. De façon analogue une application utilise une pile TCP/IP au moyen d'une bibliothèque réseau fournie par le système hôte, elle utilise cette dernière au moyen d'un SAP de niveau 3 (SAP3) ou 4 (SAP4). Par exemple winsock.dll est la bibliothèque dynamique qui permet d'utiliser TCP/IP sur une machine windows.

Voici par exemple un scénario possible:

1. Une session est établie entre un agent réseau (serveur TCP) #15360, et un agent #2 serveur web, logé dans la carte.
2. Des données sont échangées entre le client TCP (navigateur) et le serveur de la carte. A partir de ces informations l'application proxy est capable de déterminer l'adresse d'un serveur internet.
3. L'application carte ouvre une connexion avec un serveur internet. Une nouvelle session est crée entre un agent carte et un agent réseau. L'agent carte obtient une référence éphémère (#2559) et envoie un pdu *Open+Block* à un agent réseau (client TCP) dont la référence est prédéterminée (#5). Les données associées à ce pdu précisent l'adresse et le port du serveur distant. L'agent réseau (#5) ouvre une connexion TCP, et en cas de succès émet un jeton à destination de l'agent #2559.
4. L'agent carte #2559 échange des données avec le serveur internet distant, grâce à la session ouverte avec l'agent réseau #5.
5. Les données reçues depuis un des agents réseaux sont traitées et la cas échéant retransmises vers le deuxième agent réseau par le proxy.
6. Un des deux agents réseaux referme la session associée, parce que la connexion TCP qu'il gère a pris fin.

Ce scénario produit les SmartTP_pdu [6]suivantes:
**T**[s=15360,d=2,open+block], **C**[s=2,d=15360].
Le navigateur envoie une requête http au serveur web de la carte.
**T**[s=15360,d=2,write+block,data], **C**[s=2,d=15360,ack].
La dernière partie de la requête http est émise.
**T**[s=15360,d=2,write+block,data].
Le proxy carte ouvre une deuxième connexion avec un serveur internet distant.
**C**[s=2559,d=5,open+block,data], **T**[s=5,d=2559,block].
Le proxy carte émet une requête http vers le serveur internet.
**C**[s=2559,d=5,write+block,data], **T**[s=5,d=2559,block].
La dernière partie de la requête est émise.
**C**[s=2559,d=5,write,data].
Le proxy carte relaie les informations reçues par les agents réseaux.
**T**[s=5,d=2559,block], **C**[s=2,d=15360,block],
**T**[s=15360,d=2], **C**[s=2559,d=5].
Des données sont reçues depuis internet.
**T**[s=5,d=2559,write+block,data],
**C**[s=2,d=15360,write+block,data].
**T**[s=15360,d=2,block], **C**[s=2559,d=5,block],
**T**[s=5,d=2559] **C**[s=2,d=15360].
L'agent #5 est à nouveau prêt à recevoir des données du réseau.

[6] **T**=Terminal, **C**=Carte, s=source, d=destination

*Fin de la session:* le proxy carte retransmet des données.
**T**[s=5,d=2599,write+block], **C**[s=2,d=15360,write+block]
L'agent #15360 émet un jeton (bloquant)
**T**[s=15360,d=2,block], **C**[s=2559,d=5,block].
Le navigateur met fin à sa connexion TCP, l'agent #15360 referme sa session.
**T**[s=15360,d=2,close], **C**[s=2559,d=5,close].
L'agent #5 ferme session avec l'agent #2559, mais deux pdu produits avant cet événement sont envoyés par HSL vers CSL et entraînent l'émission de deux jetons par CSL.
**T**[s=5,d=2559], **C**[s=0,d=0].
**T**[s=5,d=2559,write+block,data], **C**[s=0,d=0].

## IV. SSL DANS UNE CARTE A PUCE.

L'organisation d'une puce autour d'un serveur web permet de révolutionner son usage, d'une part un protocole unique (SmartTP) permet d'accéder aux cartes, d'autre part tout utilisateur d'un navigateur connaît à priori le mode d'emploi associé.
Cette nouvelle technologie améliore également la sécurité, par exemple un point faible de SSL (Secure Socket Layer) qui est le protocole le plus utilisé pour le commerce électronique, est la vérification du certificat émis par le serveur, par le navigateur (c'est à dire un logiciel écrit par Netscape ou Microsoft).Un proxy SSL logé dans une puce permet d'éviter cet inconvénient, ainsi une url (carte) http://127.0.0.1:8080/?ssl=www.bank.fr/urienp permet d'ouvrir au moyen d'un *trusted proxy* une session SSL. Dès lors la session SSL est entièrement gérée par la carte, qui en particulier réalise la vérification du certificat. Typiquement un internaute rempli un formulaire qui comporte un login et un mot de passe pour s'authentifier, ces données sont protégées par une session SSL. L'intérêt de cette opération menée dans une puce est que le terminal ne peut en aucune manière avoir accès à cette information. Une fois que l'utilisateur à été authentifié, son identification sera assuré par un cookie présent dans chacune de ses requêtes http. Les sessions SSL qui ne comportent pas de données sensibles pourront être gérées depuis le terminal, sans l'intervention de la puce.

## V UNE SESSION SSL

Nous allons à présent détailler le déroulement d'une session SSL afin de cerner les difficultés d'intégration de ce protocole dans une carte à puce.
1- Le client (logé dans la puce) débute une session sécurisé par un message *ClientHello* (Cf figure 2) qui comporte un *Session_ID* (identificateur de session) dont la valeur est égale à zéro pour une nouvelle session et non nulle pour la reprise d'une session préalablement ouverte. Il propose un choix d'options disponibles, nommé *Cipher_Suites* et qui comporte :

- une liste de mécanismes d'échange de clés, en règle générale à base de clés publiques RSA. La législation américaine limite les clés de chiffrement RSA *export* à 512 bits.
- un ensemble d'algorithmes de chiffrement tels que RC2, RC4, DES. Les clés sont généralement limitées à 40 bits, bien que depuis peu certains éditeurs supportent des clés (RC4) de 128 bits.
- une liste de fonctions de digests (usuellement MD5)

```
8025      SSL v2 - Length = 37
01        Client Hello
0300      Version                 major=3 Minor=0
0000      Session_Id_Length       = 0
0010      Random_Length           = 16 octets
000003    SSL_RSA_EXPORT_WITH_RC4_40_MD5
000006    SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
020080    SSL_RSA_RC4_128_EXPORT40_WITH_MD5
040080    SSL_RSA_RC2_CBC_128_CBC_EXPORT40_WITH_MD5
842B32EF2B0CAE9DFF33A7AB03578C38 //Client.Random (16 octets)
```

Figure 2 - Un message *Client Hello* (émis par IE5). Seul un chiffrement avec une clé de 40 bits est proposé.

Le message comporte également un nombre aléatoire *ClientHello.random.* utilisé pour le calcul des clés de session.
Un premier point sensible du protocole SSL est la liste des possibles algorithmes de chiffrement supportés par le client. Généralement un navigateur propose l'algorithme RC4 (dont la largeur de clé peut atteindre 128 bits) associé à une clé de 40 bits. A raison de 1 million de clés par seconde il faut 12.7 jours pour générer toutes les clés. En juillet 1995, Damien Doligez, doctorant à l'INRIA a cassé un clé RC4 de 40 bits en une semaine, à l'aide d'une centaine de machines générant chacune de l'ordre de 10,000 clés/seconde. Un pentium 3, 600 Mhz est capable de produire 200,000 clés par secondes, une demi dizaine de PCs suffisent aujourd'hui pour le test d'un million de clés par secondes. Remarquons également que certain navigateurs supportent l'absence de chiffrement (*NULL Cipher*) tolérée par le protocole SSL.

```
16          // Handshake
0300        // Version = 3.0
0546        // Length = 1350 octets
02          // ServerHello
000046      // Length = 70 octets
0300        // Version 3.0
02A0D24C0D12D69A2F8CC9E59E5561DA//Server.Random  32 octets
CD90CD9C3F59FFDCE8371D4F6A2C91D1
20          // sessionID_Length = 32 octets
4A1A9837AEB91E40FA66BB6815B1DA28
B58665A459A65EC6A61491E019F338EB
0003        // Cipher Suite SSL_RSA_EXPORT_WITH_RC4_40_MD5
00          // Null Compression
0B          // Server Certificat(s)
0004F4                          1268 octets
 0004 F1                        1265 octets
  0002BE// 1st certificat       702 octets
  X509 certificat
  00022D  // 2nd certificat     557 octets
  X509 certificat
0E000000 // Server Hello Done
```

Figure 3 - Un message *Server Hello*, le serveur accepte une clé de chiffrement de 40 bits.

*Certificate ::= SEQUENCE {*
**30 82 02 BA** X 698
*certificateInfo CertificateInfo.*
**30 82 02 27** X.1 551

*SerialNumber CertificateSerialNumber.*
**02 10** X.1.1 16
**02 92 61 CF BF 5B 85 C4 66 9B 59 14 BD 51 92 73**

*Signature Algorithm Identifier.*
**30 0D** X.1.2 13
**06 09** X.1.2.1 9
**2A 86 48 86 F7 0D 01 01 04**. pkcs 1.4 - md5+rsa
**05 00** X.1.2.2 0

*Issuer Name*
**30 5F** X.1.3 95
**31 0B** X.1.3.1 11
**30 09** X.1.3.1.1 9
**06 03** X.1.3.1.1.1 3
**55 04 06**
**13 02** X.1.3.1.1.2 2
**55 53** US
**31 20** X.1.3.2 32
**30 1E** X.1.3.2.1 30
**06 03** X.1.3.2.1.1 3
**55 04 0A** *organizationName*
**13 17** X.1.3.2.1.2 23
**52 53 41 20 44 61 74 61 20 53 65 63 75 72 69 74 79 2C 20 49 6E 63 2E** RSA Data Security, Inc.
**31 2E** X.1.3.3 46
**30 2C** X.1.3.3.1 44
**06 03** X.1.3.3.1.1 3
**55 04 0B**
**13 25** X.1.3.3.1.2 37
**53 65 63 75 72 65 20 53 65 72 76 65 72 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79** Secure Server Certification Authority

*validity Validity*
**30 1E** X.1.4 30
**17 0D** X.1.4.1 - 13
**30 30 30 36 31 33 30 30 30 30 30 5A** 2000/06/13 00:00:00 GMT
**17 0D** X.1.4.2 13
**30 31 30 36 32 37 32 33 35 39 35 39 5A** 2001/06/27 23:59:59 GMT

*subject Name*
**30 82 01 25** X.1.5 293
**31 0B** X.1.5.1 11
**30 09** X.1.5.1.1 9
**06 03** X.1.5.1.1.1 3
**55 04 06**
**13 02** X.1.5.1.1.2 2
**46 52** FR
**31 11** X.1.5.2 17
**30 0F** X.1.5.2.1 15
**06 03** X.1.5.2.1.1 3
**55 04 08**
**13 08** X.1.5.2.1.2 8
**59 56 45 4C 49 4E 45 53** YVELINES
**31 13** X.1.5.3 19
**30 11** X.1.5.3.1 17
**06 03** X.1.5.3.1.1 3
**55 04 07**.
**14 0A** X.1.5.3.1.2 10
**47 75 79 61 6E 63 6F 75 72 74** Guyancourt
**31 3B** X.1.5.4 59
**30 39** X.1.5.4.1 57
**06 03** X.1.5.4.1.1 3
**55 04 0B**
**13 32** X.1.5.4.1.2 50
**77 77 77 2E 63 65 72 74 70 6C 75 73 2E 63 6F 6D** www.certplus.com
**2F 52 50 41 20 49 6E 63 6F 72 70 2E 20 62 79 20 52 65 66 2E 2C 4C 49 41 42 2E 4C 54 44 28 63 29 39 38** /RPA Incorp. by Ref.,LIAB.LTD(c) 98
**31 25** X.1.5.5 37
**30 23** X.1.5.5.1 35
**06 03** X.1.5.5.1.1 3
**55 04 0B**
**13 1C** X.1.5.5.1.2 28
**41 75 74 68 65 6E 74 69 63 61 74 65 64 20 62 79 20 43 65 72 74 70 6C 75 73 20 53 41** Authenticated by Certplus SA
**31 27** X.1.5.6 39
**30 25** X.1.5.6.1 37
**06 03** X.1.5.6.1.1 3
**55 04 0B**
**13 1E** X.1.5.6.1.2 30
**4D 65 6D 62 65 72 2C 20 56 65 72 69 53 69 67 6E 20 54 72 75 73 74 20 4E 65 74 77 6F 72 6B** Member, VeriSign Trust Network
**31 2C** X.1.5.7 44
**30 2A** X.1.5.7.1 42
**06 03** X.1.5.7.1.1 3
**55 04 0A** *organizationName*
**14 23** X.1.5.7.1.2 35
**43 61 69 73 73 65 20 4E 61 74 69 6F 6E 61 6C 65 20 64 65 20 43 72 65 64 69 74 20 41 67 72 69 63 6F 6C 65** Caisse Nationale de Credit Agricole
**31 0C** X.1.5.8 12
**30 0A** X.1.5.8.1 10
**06 03** X.1.5.8.1.1 3
**55 04 0B**.
**14 03** X.1.5.8.1.2 3
**47 54 49** GTI
**31 25** X.1.5.9 37
**30 23** X.1.5.9.1 35
**06 03** X.1.5.9.1.1 3
**55 04 03**.
**14 1C** X.1.5.9.1.2 28
**77 77 77 2E 70 61 72 69 73 2E 63 72 65 64 69 74 2D 61 67 72 69 63 6F 6C 65 2E 66 72** www.paris.credit-agricole.fr

*subjectPublicKeyInfo SubjectPublicKeyInfo*
**30 5A** X.1.6 90
**30 0D** X.1.6.1 13
**06 09** X.1.6.1.1 9
**2A 86 48 86 F7 0D 01 01 01** pkcs 1.1 - rsa
**05 00** X.1.6.1.2 0
**03 49** X.1.6.2 73
**00 30 46 02 41** Modulus 512 bits
**00 C7 04 38 E7 07 52 3D 4C F9 A7 4C 72 77 65 2A 47 40 85 FD DD 0A 82 C3 B4 69 0B.0B E5 72 E7 C4 70 32 40 4F 22 07 05 38 31 DD 6A 2E E1 CB 81 27 26 E2 39 49 2E AB 9C A3 C1 3A 3E F4 43 9E A4 35 01**
**02 01 03** Public Key (3)

*SignatureAlgorithm AlgorithmIdentifier*
**30 0D** X.2 13
**06 09** X.2.1 9
**2A 86 48 86 F7 0D 01 01 04** pkcs 1.4 - md5+rsa
**05 00** X.2.2 0

*signature BIT STRING*
**03 7E** X.3 126
**00 42 90 89 83 B1 A3 07 FD AC 51 47 97 E8 4C 11 6C A4 77 00 0F 1E 56 71 29 E0 BA B4 0C A2 E2 04 62 CF 76 C4 60 7F 7C 2B C4 38 33 FB EF C2 AD B2 53 30 C4 BE BF 52 07 DC CE 10 D0 EE 15 8A 1E C6 DB BF F8 C9 35 25 CC 82 97 12 4F BE C1 09 A8 0D 08 68 BB D4 AF F5 03 C0 F0 ED A8 7E 3B DD 64 B9 3C 2D C9 04 FE E1 80 74 9B 0A 5B F2 DB F4 3A 04 DF 60 54 F6 7E 41 2D CE B4 CE CE 5E 90 50**

Figure 6: - Un certificat X509. La clé publique est 3, le modulo un nombre de 512 bits.

2- Le serveur répond par un ensemble de messages (Cf figure 3), qui typiquement comporte un nombre aléatoire (*ServerHello.random)*, le choix d'un *Cipher_Suite*, une liste de certificats des clés publiques du serveur, et de manière optionnelle (et rarement utilisée) une demande d'authentification du client.
Le serveur sélectionne l'algorithme de chiffrement. Par exemple si un choix est possible entre une clé de 128 bits et 40 bits, il peut opter pour la deuxième solution.

```
16          // Handshake
0300        // v3.0
0044        // length 68 octets
10          // Client Key Exchange, PreMasterSecret PublicKey
000040      // length 64 octets
89 21 AD 4E 2A 06 E3 CC FB 39 96 16 1B C7 35 3D 6C C4 8A C6 5B 83 20 90 1C
CF 65 31 2D 0C AF 69 C5 18 F9 72 B6 8F B1 32 6F 29 B6 87 64 8A EB AD A9
6C A8 4F 3C 01 E9 2C 0C 3A 25 B0 B9 02 F7 75

14          // Change Cipher Spec
03 00       // v3.0
00 01       // Length 1 octet
01          // Change_cipher_spec(1)

16          // Handshake – Début de chiffrement
03 00       // v3.0
00 38       // Length 56 octets
5B 28 4C 53             // RC4(Finished 14 00 00 24)
C5 F7 CD 41 69 B1 A6 A7 C9 5A 83 8E BD F1 E5 A6     // MD5
C5 FD 75 F9 F9 58 6F E6 28 FA E3 60 D8 36 FF 13 80 0F B0 D7 //SHA1
B2 C4 3D FD C5 F5 00 BF 0F EB D1 23 42 FF 08 46                  // MD5
```

Figure 4 - Echange du PreMasterSecret, et début du chiffrement par le client.

3- Le client analyse la validité d'un certificat X509 délivré par le serveur. L'interprétation de la syntaxe ASN.1 est réalisée par exemple en parcourant un arbre dont chaque noeud est associé à un type, une longueur et une valeur. Type est encodé par un octet, le bit b5 précise si ce dernier est construit, c'est à dire si le noeud possède des fils. Longueur est codée sur un ou plusieurs octets et précise la taille en octets d'une donné ou d'un fragment d'arbre. La figure 6 présente un certificat X509 dont les différents noeuds sont marqués à l'aide du préfixe X. Les paramètres importants d'un certificat sont, le nom de son émetteur (organizationName - X.1.3.3.1.2), ses dates de validité (validity, X.1.4.1 et X.1.4.2), le nom du titulaire (X.1.5.7.1.2) la clé publique du titulaire et le modulo utilisé (subjectPublicKeyInfo X.1.6), enfin la signature numérique de ses informations (X.3).
La carte à puce doit être en mesure d'établir une correspondance entre le nom de l'émetteur et sa clé publique. La vérification de la date de validité est un point délicat car les puces n'intègrent pas d'horloge, dans un premier temps on peut par exemple imposer que la date de validité du certificat soit compatible avec la date d'expiration de la carte. Dans notre exemple la taille du modulo (X.1.6.2) est de 512 bits, la puce peut garantir à son porteur le rejet d'une clé jugée peu sûre. Pour mémoire, en Août 1999 le laboratoire hollandais CWI a réalisé la factorisation d'une clé RSA 512 bits à l'aide d'une puissance de calcul de l'ordre de 8400 MIPS-an, fournie par 300 ordinateurs et stations de travail en 7 mois. Notre certificat daté de juin 2000, propose une clé RSA de 512 bits, qui est cassable avant la date d'expiration.
L'analyse de la signature d'un certificat X509 implique la présence d'une bibliothèque cryptographique dans la puce, supportant les algorithmes RSA, MD5 (MD4 et MD2 sont également souhaitables).
Un nombre de 48 octets (le *pre_master_secret*) est généré et chiffré avec la clé publique du serveur,

$$\text{pre\_master\_secret}^{\text{ServerPublicKey}}.$$

De manière optionnelle (et rarement utilisée aujourd'hui) le client fournit un certificat pour sa clé publique, à l'aide de laquelle il réalise la signature du *pre_master_secret*. Une carte à puce est l'emplacement idéal pour stocker une clé privé, qui permettrait donc de réaliser une authentification mutuelle entre client et serveur.
Les clés de chiffrement et divers paramètres (IV, MAC Secret...) sont déduits du *master_secret* (obtenu à partir du pre_master_secret) et des nombres aléatoires,

Key(s) = F(master_secret, ClientHello.random, ServerHello.random).

Le client notifie son passage en mode chiffré par un premier message chiffré *Finished* qui comporte l'empreinte (MD5 + HASH) des données préalablement échangées (Cf figure 4)

4- Le serveur calcule les clés et divers paramètres, la session SSL assure dès lors la confidentialité et l'intégrité des données échangées (Cf figure 5).

```
14          // Change Cipher Spec
0300        // v3.0
0001        // Length 1 octet
01

16          // Handshake – Début de chiffrement
0300        // v3.0
0038        // Length 54 octets
1D B9 3C 14             // RC4(Finished 14 00 00 24)
DE CA A3 02 E2 D1 22 FA 70 F4 3A CD 10 32 52 60     // MD5 hash
BA D4 D2 C4 35 6E 66 BC 15 A8 48 65 F5 7B 29 C9 BC 6F 04 58// SHA1 hash
7B 79 B8 41 14 1C 95 06 3D EE 47 42 56 89 A1 82                  // MD5 hash
```

Figure 5 - Fin de négociation des paramètres de chiffrement, et début de chiffrement par le serveur.

5- Echange de données et fin de session
La puce chiffre une requête http, par exemple le contenu d'un formulaire qui contient un login et un mot de passe. La puce déchiffre et contrôle l'intégrité de la réponse du serveur (Cf figure 6). Cette réponse peut contenir une indication de re-direction associée à un cookie, et est transmise en clair au navigateur. L'avantage de cette technique est que les données sensibles (login et mot de passe) ont été transmises dans une session SSL contrôlée par la carte à puce, l'internaute est par la suite identifié par un cookie.

Dans la mesure ou toutes les données sont émises ou reçues en clair depuis/vers le terminal, de nouvelles sessions SSL pourront être conduites sans le contrôle de la puce.

```
Client
17          // Application Data (RC4 + MD5 hash-16 bytes)
0300        // v3.0
0103        // Length 259 octets
07 1D 66 12 74 45 D5 E0 65 06 74
Données chiffrées..…
68 7B 62 B4 1E A6 34 78 1A 69 70 E5 EE B3 38          // empreinte MD5

Serveur
17          // Application Data
0300        // v3.0
0070        // Length 112 octets
30 F0 6B B8 90 C5 28 D0 5F B7 3D 95 67 9F 04 A0
Données chiffrées…
59 84 D6 EB 3E 83 95 54 C0 05 0A C7 41 91 9E 2E       // empreinte MD5

15          // Alert Message
0300        // v3.0
0012        // length 18 octets
AF B3       // warning(1)close_notify(0) RC4(01 00)
49 6E 92 EB A1 61 53 E0 AC A6 23 FA AB 62 6E 2E       // MD5
```

Figure 7 - Echange de données chiffrées et fin de session

6- Lorsque le Session_ID est non nul, les deux extrémités calculent de nouvelles clés de session à partir du Master_Secret précédent et des nouveaux paramètres ClientHello.random, ServerHello.random, L'intégrité des messages est garantie par la conservation des valeurs courantes des fonctions de Hash. Remarquons que la puce peut conserver secret le *master secret*, mais transférer au terminal le contexte de la nouvelle session, c'est à dire la valeur courante des fonctions de HASH ainsi que les clés de chiffrement.

Grâce à la technologie carte internet, une puce peut gérer une session SSL, initialisée à partir d'une URL. Les processeurs actuels sont déjà en mesure de produire des calculs RSA (512 bits), la gestion de ce protocole nécessite l'ajout de fonctions cryptographiques aux cartes à puces telles que SHA1, MD5 ou RC4, ainsi que de *parsers* ASN1.

## VI CONCLUSION

Nous avons montré qu'il est possible d'intégrer un serveur web aux cartes à puces actuelles et de loger dans celles ci des applications client serveur issues du monde internet. Nous pensons que la mise en œuvre de processeurs RISC 32 bits, couplée à l'évolution des capacités mémoires embarquées permettra aux cartes à puces de se comporter comme un véritable processeur de sécurité, supportant des protocoles complexes, tels que SSL.

## VII REFERENCES


[1] International Organization for Standardization, "Identification Cards - Integrated Circuit(s) Cards with Contacts", ISO 7816.

[2] Louis Claude Guillou, Michel Ugon, and Jean-Jacques Quisquater, "The Smart Card: A Standardized Security Device Dedicated to Public Cryptology,", Contemporary Cryptology. The Science of Information Integrity, ed. Gustavus J.Simmons, IEEE Press 1992, pp. 561-613

[3] Jean Pierre Tual "MASSC: A generic Architecture for Multi application Smart Cards", IEEE Micro Journal, N°0272-1739/99, 1999.

[4] Klaus Vedder, Franz Weikmann "Smart Cards, Requirements, Properties and Applications" Chipkarten Grundlagen, Realisierungen, Sicherheitsaspekte, Anwendungen - Verlag Vieweg - 1998 - pp 2-23, ISBN 3-528-05667-3

[5] R.Anderson, M.Kuhn " Tamper Resistance - a Cautionary Note", USENIX Workshop on Electronic Commerce Proceeding, November 1996, pp 1-11.

[6] P.Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", Advances in Cryptology: Proceedings of EUROCRYPT'97, Springer-Verlag, August 1997, pp 513-525.

[7] Paul Kocher, Joshua Jaffe, and Benjamin Jun "Differential Power Analysis" Proceedings of CRYPTO'99, Springer-Verlag, August 1999, pp 388-397.

[8] "Interoperability Specification for ICCs and Personal Computer Systems", PC/SC, © 1996 CP8 Transac, HP, Microsoft, Schlumberger, Siemens Nixdorf.

[9] Open Card Framework, OCF, available at http://www.opencard.org/

[10] Schlumberger "Using a high level programming langage with a microcontroler" Patent WO 98/19237

[11] Microsoft "SmartCard White paper" http://www.microsoft.com/ - 1999

[12] International Organization for Standardization - (ISO) "Information Processing Systems - Open Systems Interconnection - Basic Reference Model" ISO 7498.

[13] Postel, J., "Transmission Control Protocol", RFC 793, September 1981.

[14] T Berners Lee & All "Hypertext Transfer Protocol - HTTP/1.1" - RFC 2068 - January 1997.

[15] Pascal Urien - "Procédé de communication entre une station d'utilisateur et un réseau, notamment de type internet, et architecture de mise en œuvre" brevet déposé le 13 août 1988, N° d'enregistrement 98 10401, N° de publication 2 782 435

[16] Pascal Urien, Hayder Saleh - "Une nouvelle approche de la carte à puce réseau" . Journées Nationales Réseaux JRES99 – décembre 1999 Montpellier.

[17] Pascal Urien, Hayder Saleh, Adel Tizraoui "La puce Internet, une architecture ouverte adaptée aux applications distribuées multimédia sécurisées". Infosec'Com 2000 - 7,8 juin 2000 La défense.

[18] Pascal Urien, "Carte à puce internet & Objets mobiles embarqués" OCM'2000 Ecole des Mines de Nantes, 18 mai 2000.

[19] Pascal Urien " Internet Card, a smart card as a true Internet node", Computer Communication, to appear.