Support de cours : Carte à puce

Jean-Louis LANET

Version 1.0	Septembre 2005	Reprise habilitation
1.1	Octobre 2005	Intégration java card
1.2	Novembre 2005	Intégration biométrie (par Claude Barral)
1.3	Janvier 2006	Les applications & cryptographie
1.4	Octobre 2006	Glossaire + API crypto

SOMMAIRE

1. Cartes à puces et nouveaux défis

La carte à puce, dont le concept a été inventé par Roland Moreno puis repris quelques années plus tard par Michel Ugon, a fait une entrée remarquée dans le domaine des télécartes. L'idée originale est de rassembler sur une seule puce toutes les fonctionnalités et de mettre en œuvre les mesures visant à protéger les données contenues ou calculées. La carte à puce peut donc être vue comme un support électronique permettant d'intégrer des applications à haut niveau d'intégrité et de confidentialité dans des systèmes d'information. Pour ce faire, elle offre un espace sécurisé pour des applications "sensibles". Pendant plusieurs années, et ce depuis le début des cartes à microcontrôleur, les applications embarquées dans les cartes ont été développées sous forme monolithique, intégrant les fonctions du système d'exploitation et celles de l'application. Et même si des éléments des systèmes d'exploitation ont été développés de façon générique, c'est surtout grâce à l'avènement des cartes ouvertes que se propage aujourd'hui la notion de système d'exploitation pour cartes à puce. Le corollaire est donc une séparation entre la sécurité du système et celle des applications.

Les cartes ouvertes basées sur des technologies issues de l'informatique classique comme Java, améliorent grandement l'interopérabilité et permettent d'intégrer plusieurs applications. Elles offrent aussi la possibilité à des développeurs traditionnels de concevoir des applications sécuritaires sans avoir une connaissance approfondie du support d'exécution. Ces cartes deviennent un support générique permettant d'embarquer plusieurs applications potentiellement communicantes. La Java Card est un des vecteurs majeurs de ce changement, en fournissant une plate-forme d'exécution d'applications utilisant des technologies objet à l'intérieur de la carte. La complexité des applications qui y sont chargées devrait alors être prise en compte à toutes les étapes du développement. L'utilisation du langage Java a d'autres conséquences. L'utilisation d'un langage de haut niveau, conjuguée avec un cycle de production utilisant un chargement logiciel à la place d'une écriture en ROM a pour conséquence la création de nouveaux cycles de développements, plus courts mais aussi plus réactifs. De même, en conformité avec l'évolution de toutes les plates-formes communicantes, il est possible que le développeur n'ait bientôt plus à se préoccuper des protocoles de communication entre la carte et son environnement, en déléguant la génération des trames de communication à des outils dédiés.

Dès lors, la sécurité des cartes sort de son schéma traditionnel de contre-mesure contre des attaques matérielles et doit se préoccuper des aspects logiciels. Ces aspects logiciels concernent non seulement les défauts résiduels pouvant être exploités par une application malicieuse mais aussi plus simplement de la conception des applications de sécurité pouvant déléguées à des tiers n'étant pas toujours au faîte de l'état de l'art. Il convient donc de garantir au minimum le même niveau de sécurité qu'avant l'avènement des cartes ouvertes. Les outils à la disposition des concepteurs et des évaluateurs vont de l'audit traditionnel de code jusqu'à la modélisation mathématique.

1.1. Fonctionnalités de la carte à puce

La carte est un support électronique mobile et fiable pour conserver des secrets. Ces secrets peuvent être des clefs cryptographiques servant à coder ou décoder des messages

confidentiels, soit à base d'algorithmes à clefs publiques (RSA[1]) ou d'algorithmes à clefs secrètes (AES[2]). Seul le détenteur de la carte peut coder ou décoder ces messages. Le principe général est donc de conserver et de traiter ces secrets dans un environnement de calcul sécurisé : la carte. Les données ne devant jamais sortir de cet environnement.

La carte à puce apparaît le plus souvent sous son format carte de crédit mais d'autre facteurs de forme sont apparu comme le format GSM ou la carte USB. La puce elle-même est définie par une norme ISO, la norme ISO-7816. Cette norme rassemble l'ensemble des caractéristiques de la carte à puce:

- les caractéristiques physiques,
- les dimensions et les positions des contacts,
- les signaux électroniques et les protocoles de transmissions,
- les commandes inter-industries pour l'échange,
- les identificateurs d'applications,
- les éléments de données inter-industries,
- les commandes inter-industries pour les SCQL[3] ,
- la sécurité de l'architecture et des commandes inter-industries.

La carte à puce jouit d'un avantage considérable d'abord par sa taille qui lui permet d'affronter les tests de torsion sans être détériorée, puis par le fait que tout sur une carte à puce est rassemblé au même endroit. En effet, le cœur de la puce, le microprocesseur est un bloc monolithique. Il contient à la fois les unités de calcul tels le processeur et le co-processeur cryptographique, les mémoires contenant le code de la carte dans la ROM[4], les mémoires de travail, persistantes comme l'EEPROM[5], ou temporaires telle la RAM[6], et les éléments de communication. La surface maximum allouée au microprocesseur par la norme ISO est de 25 millimètres carrés. Ainsi, il est plus difficile pour un attaquant d'isoler facilement telle ou telle partie de la carte. Pour compliquer le tout, elle dispose de capteurs permettant de détecter une perte de puissance électrique, une surchauffe des composants ou encore la mise à nu des circuits (capteur de lumière). Part sa conception, la carte à puce apparaît comme un élément hautement sécurisé d'un système pouvant garder et protéger des secrets.

Cependant à cause de ces contraintes en taille, les capacités de la carte à puce n'évoluent pas aussi vite que ce que les développeurs pourraient espérer. Les cartes actuelles de haut de gamme proposent 64 ko[7] de ROM, 64 ko de EEPROM et 4 ko de RAM pour ce qui concerne la mémoire. Il faut donc tenir compte de ces restrictions qui touchent la

[1] inventé en 1977 par Ron Rivest, Adi Shamir et Len Adleman

[2] Advanced Encryption Standard

[3] Structured Card Query Language

[4] Read Only Memory

[5] Electronic Erasable Programmable Read Only Memory

[6] Random Access Memory

[7] kilo octets

mémoire mais aussi les performances des processeurs. Le tableau suivant (Tableau 1-1) donne l'évolution des processeurs de cartes à puce du début des années 1980 à nos jours. L'apparition des nouvelles technologies mémoires comme la FéRAM[8], ainsi que la gravure de plus en plus fine des puces, permet d'espérer une constante croissance des capacités des cartes. Cependant, malgré cette évolution, les ressources de la carte sont toujours limitées, surtout lorsque nous les comparons aux ordinateurs de bureau ou aux autres objets nomades tels les téléphones mobiles ou les assistants numériques.

Année	Taille du bus	Fréquence	RAM	Mémoire persistante
1981	8 bits	4,77 MHz	36 octets	1 ko d'EPROM
1985	8 bits	4,77 MHz	128 octets	2 ko d'EEPROM
1990	8 à 16 bits	4,77 MHz	256 octets	8 ko d'EEPROM
1996	8 à 32 bits	4,77 à 28,16 MHz	512 octets	32 ko de FLASH
2000	8 à 32 bits	4,77 à 28,16 MHz	1536 octets	32+32 ko de FLASH et d'EEPROM
2002	8 à 32 bits	4,77 à 28,16 MHz	3 à 4 ko	64+64 ko de FLASH et d'EEPROM

Tableau 1-1 Evolution des caractéristiques des microprocesseurs dédiés aux cartes à puce

1.2. Les applications

La première application des cartes à puce a été la télécarte. La télécarte n'est pas encore une carte à microprocesseur, mais une carte à mémoire qui contient un compteur. Ce compteur représente les unités disponibles sur la carte et est décrémenté lors de l'utilisation dans une cabine publique. Les clients des opérateurs téléphoniques sont satisfaits car il devient plus simple de posséder une carte que de rechercher de la monnaie. Les opérateurs téléphoniques quant à eux sont les grands gagnants : les clients payent d'avance leur consommation (d'où un apport important de trésorerie) et la dégradation des cabines est diminuée puisqu'elles ne contiennent plus de monnaie. Le plus des télécartes a été de proposer aux industriels une surface publicitaire leur permettant de diffuser leurs messages.

Ce succès a été poursuivi par la suite avec l'ouverture de nouveaux marchés et donc de nouvelles applications, notamment dans le domaine bancaire. La création du GIE carte bancaire favorise l'utilisation de cartes à puce dans les transactions commerciales. Deux objectifs sont affichés : réduire les fraudes (fausse monnaie, chèque non endossable) et faciliter les transactions commerciales. De fait, la carte de crédit est aujourd'hui très utilisée. Elle est de plus en plus souvent associée à un porte monnaie électronique de type Monéo dont le principe est identique à la télécarte. Il consiste à pré charger électroniquement le porte monnaie et son utilisation n'est pas soumise à une protection. Cette évolution de la carte à puce passe par l'utilisation de cartes à microprocesseur, capables d'exécuter des applications.

Après le domaine bancaire, la carte à puce est utilisée dans la reconnaissance d'identité. En effet, l'un de ses atouts majeurs est sa personnalisation et son inviolabilité. Nous pouvons donc donner un secret à la carte qui permet à son possesseur de s'identifier auprès de certains services. Cela peut être par exemple une carte d'identité qui contiendrait nos informations personnelles, une carte similaire pour l'identification

[8] Ferroelectric Random Access Memory

professionnelle. Cette carte peut non seulement permettre l'accès aux locaux mais peut aussi, donner l'accès aux systèmes d'information de l'entreprise. De nombreux travaux sont en cours pour justement sécuriser ces cartes et en particulier le secret qu'elles doivent contenir pour assurer l'identification. Cela peut être par exemple un mot de passe ou un PIN[9]. Tel est le cas aujourd'hui. Mais cela peut également être une empreinte digitale, une empreinte rétinienne ou pourquoi pas, une empreinte vocale. Le but est de trouver un secret que seuls la carte et l'utilisateur peuvent partager afin de rendre le système plus sûr.

Dans le marché de la téléphonie mobile, où la technologie évolue très vite, il est très dur, voire très dangereux, de se lier commercialement avec un partenaire particulier. Ainsi, les opérateurs de téléphonie et les fabricants de téléphone n'ont pas d'accords commerciaux exclusifs. Les opérateurs veulent pouvoir s'approvisionner chez plusieurs fabricants et ne pas être dépendant d'un seul, ce qui pourrait les affaiblir dans un marché fortement concurrentiel. Ils se sont donc mis d'accord sur la norme GSM[10], permettant à la carte à puce au format SIM[11] d'être le lien physique entre le téléphone, l'opérateur de téléphonie mobile et le client final. La carte SIM permet à un client de s'affranchir du téléphone en stockant toutes ses données sur la carte. Ainsi, il peut « personnaliser » chaque téléphone grâce à sa carte SIM. Afin de se différencier de la concurrence, de nombreuses applications sont désormais développées par les opérateurs afin de rendre leur offre plus attrayante. Ces applications sont stockées et exécutées par la carte SIM qui représente l'opérateur dans le téléphone.

De la simple télécarte aux actuelles Java Card, la carte à puce propose de nouvelles fonctionnalités tout en gardant ses qualités qui ont fait sa réputation : un objet portable, personnalisé et sécurisé. Elle est devenue une plate-forme d'exécution à part entière. De plus, les cartes multi-applicatives, c'est-à-dire capables d'abriter plusieurs applications, font leur apparition. Ces cartes permettent de faire cohabiter plusieurs applications. Ces applications peuvent être soit indépendantes les unes des autres soit partager des informations et des services entre elles. L'idée est de simplifier le rôle du client en lui proposant toujours plus de services basés sur un objet sécurisé.

Avec l'évolution de la carte et sa complexité grandissante, se pose le problème de garantir le même niveau de qualité. Effectivement si jusqu'à présent les attaques contre les cartes portaient essentiellement sur le matériel, l'apparition d'attaque exploitant des vulnérabilités du code n'est plus à exclure. Le processus de développement actuel comprend une phase de test de plus en plus longue due à l'augmentation de la complexité des applications et des systèmes. Dans le même temps, les délais de développement, dans un marché de haute technologie très concurrentiel, tendent justement vers une réduction. Cet antagonisme pousse à expérimenter de nouvelles solutions et à étudier leur impact technologique et économique sur un processus de développement industriel, dans le but de garantir le même niveau de qualité des produits.

[9] Personal Identification Number

[10] Global System for Mobile Communication

[11] Subscriber Identification Module

2. Technologies de la carte à puce

Les cartes à puce se divisent en deux grandes catégories suivant leur architecture matérielle : les cartes à mémoire et les cartes à microprocesseurs. On distingue également deux grandes catégories suivant la technologie utilisée pour communiquer avec le monde extérieur : les cartes à contact et les cartes sans-contact.

Dans tous les cas, les cartes à puce sont mono-composant et celui-ci est directement connecté à l'interface physique (contacts visibles à la surface de la carte ou une antenne noyée dans le corps plastique de la carte ou éventuellement aux deux). Il existe des cartes intégrant un composant accessible par une interface contact et un deuxième accessible par une interface sans contact, mais les deux composants ne communiquent pas entre eux et il s'agit en fait de deux cartes à puces juxtaposées dans le même corps plastique. Le reste de ce chapitre traite spécifiquement des cartes à contact, même si la plupart des concepts sont transposables directement aux cartes sans contact.

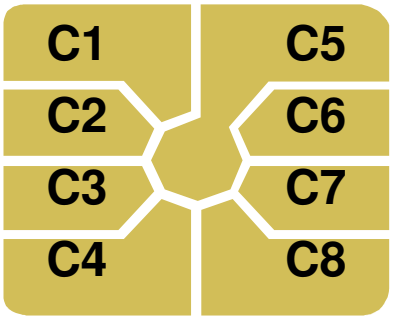
2.1. Cartes à mémoire et cartes à microprocesseurs

Les cartes à mémoire réunissent sur un même circuit intégré de la mémoire non volatile permettant de stocker quelques centaines de bits ainsi qu'un peu de logique pour effectuer les opérations de lecture écriture et exécuter des algorithmes d'authentification rudimentaires. Il s'agit typiquement de télécartes contenant un compteur d'unités, un numéro de série et une donnée secrète permettant d'authentifier la carte. Dans la suite nous ne revenons pas sur ce type de cartes et nous traitons uniquement des cartes à microprocesseurs.

Ces dernières contiennent en fait un microcontrôleur qui comporte, non seulement un microprocesseur mais également de la mémoire morte, de la mémoire vive, ainsi que des périphériques permettant d'effectuer des opérations spécialisées comme la gestion de la ligne série, opérations arithmétiques sur les grands nombres, génération de nombres aléatoires, etc. Une carte à microprocesseur est un fait un véritable système sur composant intégrant tous les éléments matériels et logiciels d'un ordinateur sur un seul composant.

2.2. Communication carte / monde extérieur

Les cartes disposent de huit contacts dont l'utilisation est standardisée par la norme ISO 7816-2.



Contact	Utilisation
C1	Tension d'alimentation
C2	Remise à zéro
C3	Horloge
C4	USB+
C5	Masse
C6	Tension de programmation (obsolète)
C7	Données
C8	USB-

Figure 1 : Numérotation et position des contacts selon la nome ISO 7816-1

Comme indiqué sur la Figure 1, la communication entre une carte et le monde l'extérieur se fait par l'intermédiaire d'une ligne série disponible sur le contact C7. En 2004, l'utilisation du protocole USB à été normalisé, mais il s'agit toujours d'un protocole de type série même si les données transmises sont codées de façon différentielle sur les deux contacts C4 et C8.

Une autre manière de communiquer est d'utiliser une interface radio on parle alors de carte sans contact. Elle fonctionne par induction grâce à une antenne imprimée ou intégrée dans la carte à puce. Cette carte n'utilise pas de contact physique avec le lecteur. Plusieurs technologies existent, mais de façon générale, la carte sans contact contient une puce électronique avec un émetteur hyperfréquence et une antenne intégrée dans le plastique..

La carte doit être assez près du lecteur, entre 3 et 10 centimètres. Certaines cartes sans contact utilisent les ondes radio et peuvent donc fonctionner à plus grande distance. Le standard des cartes sans contact est le standard ISO 10536. La carte est un peu plus complexe car elle doit intégrer un régulateur de tension, un modulateur/démodulateur ainsi qu'un mécanisme d'anti-collision, un générateur d'horloge et bien entendu une antenne.

Les principaux avantages des cartes sans contact sont :

- Absence de contacts susceptibles d'être salis ou endommagés
- Elles ne doivent pas être insérées dans un lecteur :
- Moins de possibilités de vandalisme sur les lecteurs
- Plus rapide à manipuler, d'où un débit d'utilisateur potentiellement plus élevé

Mais ce système a aussi des désavantages :

- Transfert assez lent
- Coût de fabrication plus élevé

- Les cartes peuvent être endommagées quand elles sont pliées (contiennent plusieurs composants répartis dans la carte
- la sécurité de transmission n'est pas assurée puisque susceptible d'être "entendue"

Un domaine privilégié des cartes sans contact est le transport. Ce marché est en train de connaître des évolutions importantes. En France, notamment où la RATP s'est déjà engagée, pour remplacer les tickets magnétiques et les cartes Orange de transport dans un programme de déploiement d'un *« pass »* sans contact baptisé Navigo. Ce *« pass »* , en partie déjà déployé en région parisienne, est proposé sous plusieurs formes : cartes à microcontrôleur pouvant offrir des services autres que les seules applications billettiques (paiement par porte-monnaie électronique notamment), ou ticket électronique jetable (en papier). La carte sans contact répond embarque des algorithmes de cryptographie de type 3DES. Le ticket est doté d'une mémoire de 256 bits (il s'agit d'un circuit mémoire de 1,2 mm ^{2}), qui offre des temps de transactions de l'ordre de 100 ms. Il met en oeuvre aussi un algorithme 3DES.

2.3. La réponse au Reset

L'Answer To Reset (ATR) est la séquence de par laquelle la carte et le lecteur se synchronisent pour les échanges futurs. Il est défini dans la norme 7816-3 et définit de nombreux paramètres indispensables au bon établissement de la communication entre la carte et le terminal. Il est composé d'au moins deux octets obligatoire et d'au plus trente trois octets. Le premier octet (TS) définit la convention de transfert l inverse (0x3f) le niveau logique bas correspond à un et le niveau logique haut correspond à un zéro ou la convention direct (0x3b).

Le second caractère est le caractère de format ou T0. La partie basse constituée des bits b0 à b3 code le nombre de caractères d'historique. Ces caractères sont laissés à la discrétion des fabricants de carte et permettent souvent d'identifier les applications de la carte. Les autres bits définissent la présence ou non d'autre octets. Ainsi b4 =1 implique que TA1 sera transmis ensuite.

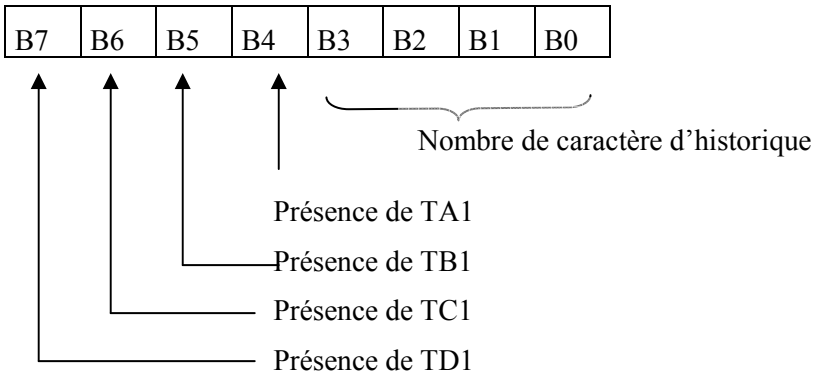


Figure 2 : Réponse au Reset

La signification des autres bits non obligatoires est donnée en annexe.

2.4. Les protocoles de communication

Il existe deux protocoles de communication très répandus utilisés pour transporter des données sur la ligne d'entrée/sortie. Il s'agit des protocoles T=0 et T=1 tous les deux

décrit dans la norme ISO 7816-3. Ces protocoles sont asynchrones, semi-duplex et transportent des blocs de données appelés APDU (pour Application Protocol Data Unit). En fait il existe deux catégories d'APDU : les commandes et les réponses.

La pile de protocoles de communication s'en trouve extrêmement réduite par rapport à un système de télécommunication classique : les protocoles T=0 et T=1 correspondent à la couche 2 du modèle OSI et les APDU à la couche 7 (ou couche applicative). Le niveau 1 (ou couche physique) est également spécifié dans la norme ISO 7816-3.

Nous avons vu que juste après le signal de remise à zéro, la carte envoie systématiquement un paquet de données appelé ATR contenant une description des protocoles de communication supportés. Il peut suivre une commande de sélection de protocole si la carte en supporte plusieurs, puis, aussi longtemps que la carte sera alimentée et ne sera pas remise à zéro, elle attendra un bloc APDU de commande, le traitera puis émettra un bloc APDU de réponse avant de se remettre en attente d'une commande. La structure des blocs APDU est indiquée Figure 3. L'octet de classe (CLA) caractérise un groupe de commandes correspondant généralement à un domaine d'application (par exemple les commandes de carte SIM utilisent la classe A0h). Puis l'octet d'instruction (INS) identifie la requête qui est faite à la carte et peut être raffinée à l'aide de deux octets de paramètres (P1 et P2). Enfin s'il y a des données dans la commande, il est nécessaire de préciser leur longueur (Lc). De même, si des données sont attendues dans la réponse, leur longueur doit être précisée (Le). La réponse contient optionnellement des données, puis un code sur deux octets (SW1 et SW2) indiquant si la commande s'est correctement déroulée (dans ce cas SW1=90h et SW2=00h) ou non (SW1 et SW2 servent alors à coder le type d'erreur qui s'est produite).

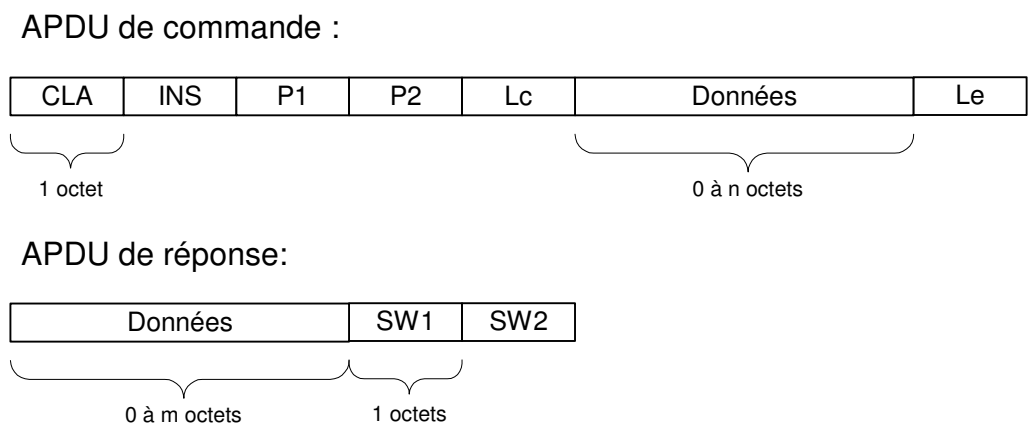


Figure 3 :Structure des APDU

Pour échanger des informations avec une carte, l'application doit envoyer un bloc APDU qui peut revêtir plusieurs cas :

- Dans le premier cas, le lecteur n'envoie aucune donnée à la carte et n'attend en retour aucune donnée. Sont APDU se résume donc au quatre premiers octets, la réponse de la carte ne sera formé que par les deux Status Word : SW1 et SW2.
- Le second cas concerne la réception de donnée par le lecteur. Le lecteur attendant le nombre de d'octets Le, le rajoute au x quatre octets précédents. Le nombre d'octets envoyé par la carte peut être nul ou différent de celui spécifié, alors SW1 et SW2 renvoie un code d'erreur l'expliquant.

- Envoie de données par le lecteur sans réception de donnée. Dans ce cas l'octet Lc donne la taille des données qui suivent.
- Le dernier cas correspond à l'envoi de données avec réception de données. Dans ce cas le paramètre Lc est suivi des données puis du paramètre Le. En fait cette séquence est décomposée. Après réception de l'APDU, la carte envoie une réponse de statut sans donnée, à laquelle le lecteur répond par une commande GetResponse afin de recevoir les données de la carte.

A ce niveau de description, il devient clair que même si les protocoles utilisés sont restrictifs, la carte est un ordinateur indépendant relié par un réseau point à point avec un hôte pouvant être un PC, un téléphone ou un distributeur automatique bancaire. L'élément physique permettant de connecter une carte à puce à un PC est souvent appelé « lecteur » ou parfois encore « programmateur » de carte. Ces termes sont trompeurs et peuvent laisser à penser que la carte à puce se réduit à une simple mémoire qui peut simplement être lue ou programmée telle une EEPROM. On préfèrera le terme, plus juste même s'il est moins employé, de « coupleur » qui indique bien mieux sa fonction. La rigidité des protocoles de communication employés impose une architecture où la carte ne peut jouer qu'un rôle de serveur attendant passivement des requêtes. Cette limitation a été partiellement levée dans le domaine du GSM grâce à une utilisation détournée des APDU, mais la prochaine évolution consistera à migrer vers des piles de protocoles plus souples et plus usuels tels que par exemple HTTP/TLS/TCP/IP/USB. Des prototypes d'implémentation sur carte à puce ont déjà été démontrés.

2.5. Les cartes classiques ou « fermées »

Historiquement, des systèmes d'exploitations propriétaires ont été développés pour gérer les ressources des cartes. Normalement, une ou des applications de type serveur, s'appuyant sur le système d'exploitation, devraient interpréter les APDU de commande et calculer les réponses à envoyer. En pratique comme il n'était pas possible de charger des applications dans les cartes après leur fabrication (d'où le terme de carte « fermée »), le système d'exploitation et les applications n'étaient pas des entités distinctes mais un seul logiciel en mémoire morte.

La norme ISO 7816-4 a très tôt normalisé un jeu de commande APDU qui concerne en grande partie la manipulation de fichiers ce qui a eu pour effet de structurer les systèmes d'exploitation autour de cette notion et d'en faire la méthode de gestion de fait de la mémoire non volatile telle l'EEPROM.

Le système de fichier ISO 7816-4 se présente sous une forme arborescente classique. La racine de l'arborescence est dénommée MF (pour Master File), les répertoires DF (pour Directory File) et les fichiers EF (pour Elementary File). Il existe la notion de fichier (ou répertoire) courant sur lequel s'applique la commande envoyée. Les fichiers peuvent être structurés en enregistrements de longueur fixe ou variable. Ils sont protégés par des droits stockés avec le fichier sous forme de liste de contrôle d'accès (ou ACL pour Access Control List) permettant à la carte de décider si une commande peut être exécutée ou non sur le fichier courant en fonction du contexte de sécurité courant. Parmi les commandes normalisées on trouve les fonctionnalités suivantes :

- Sélection du fichier courant ;
- Lecture/écriture binaire dans le fichier courant;
- Lecture/écriture d'un enregistrement dans le fichier courant.

Une autre partie importante des commandes normalisées recouvre la gestion de la sécurité. On trouve en particulier ce qui est relatif à l'authentification du porteur de la carte et au chiffrement/authentification des échanges de données entre la carte et son hôte. D'un point de vue conceptuel tout ceci est classique et relève de l'authentification des utilisateurs sur un ordinateur d'une part et de la sécurité des protocoles de communication d'autre part. La partie authentification des utilisateurs est gérée par l'intermédiaire des codes secrets (ou PIN pour Personal Identification Number) généralement courts et présentables peu de fois avec échec pour éviter les attaques par force brute. La partie sécurité réseau est gérée par des protocoles permettant à la fois l'authentification mutuelle de la carte et de l'entité distante avec qui elle échange des données puis la dérivation de clés de session qui permettront de chiffrer et de contrôler l'intégrité des données échangées.

Les ACL des fichiers font référence aux PIN présentés avec succès au cours de la session courante et permettent d'exprimer des conditions d'accès telles que : le fichier est accessible en lecture sans condition et en lecture/écriture si le PIN N°1 a été présenté avec succès au cours de la session courante. L'utilisation de plusieurs PIN permet de concevoir une politique de sécurité multi-utilisateurs et d'introduire, par exemple, un administrateur en plus du porteur de la carte.

Des commandes plus spécifiques que la gestion des fichiers et de la sécurité sont nécessaires suivant le type de domaine qui utilise les cartes à puce. On trouvera par exemple un jeu de commandes spécifiques pour les cartes SIM des téléphones mobiles GSM, un autre pour les cartes de crédit à la norme EMV, encore un autre pour le porte-monnaie électronique français MONEO, etc.

Le résultat de cette standardisation au niveau des commandes APDU et de l'impossibilité de télécharger de nouvelles application dans une carte à puce est que les cartes à puces étaient en général développées spécifiquement pour un domaine bien particulier (GSM, bancaire, signature électronique, etc.). Elles interprétaient alors un général un sous-ensemble du jeu d'instructions ISO 7816-4 plus un jeu de commande spécifique au marché visé.

Outre les coûts de développement et de test d'un système complet en mémoire morte, il n'était pas possible pour des faibles volumes de créer des cartes particulières. Les intégrateurs de systèmes devaient alors s'adapter et utiliser, par exemple, les commandes disponibles sur une carte de signature électronique et les intégrer dans leur système de gestion des contrôles d'accès aux locaux d'une entreprise ou de la présence dans une cantine scolaire.

Pour reprendre le cas du contrôle des accès physiques aux locaux d'une entreprise, nous indiquons brièvement ci-après comment l'implémenter en utilisant simplement le jeu de commandes ISO 7816-4. Nous supposons qu'un employé ne peut accéder qu'à un nombre restreint de zones de l'entreprise et ceci pendant ses horaires de services. On peut coder ces deux informations dans sa carte à puce personnelle respectivement dans le fichier de zones autorisées Z et dans le fichier d'horaires autorisés H. De plus, pour éviter l'utilisation de badges volés, on peut attribuer à chaque utilisateur un PIN. On configure les droits d'accès au fichier pour qu'ils soient en lecture publique mais qu'ils ne soient accessibles en écriture que sur présentation d'un PIN administrateur.

Pour rentrer dans une zone protégée, l'employé devra insérer sa carte à puce et composer son code PIN. La séquence de commandes suivante sera alors envoyée par le sas d'accès à la carte :

- Présentation du code PIN à la carte (la carte répond par succès ou échec) ;
- Sélection du fichier Z ;
- Lecture du fichier Z ;
- Sélection du fichier H ;
- Lecture du fichier H.

Bien entendu cette première approche n'est pas sécurisée car rien ne permet d'authentifier la carte à puce. On peut donc rajouter une clef secrète dans la carte afin d'utiliser un protocole d'authentification entre la carte et le sas. Les commandes permettant d'échanger des nombres aléatoires et de dériver les clés de session nécessaires doivent donc être insérées en début de séquence. Par la suite toutes des APDU de commande et de réponse seront munies d'un cryptogramme qui peut être vérifié par le sas à l'aide de la clé de session. On authentifie alors les données du fichier Z comme provenant bien d'une carte officielle d'un employé de l'entreprise.

Pour compléter cet exemple, il faudrait définir la chaîne de sécurité complète comprenant par exemple la procédure pour attribuer une carte à un employé (et donc comment générer sa clef secrète et la transférer dans la carte de façon sécurisée), comment modifier les droits d'un employé, comment distribuer et stocker les clefs des sas, etc.

2.6. Rappel : les techniques de cryptographie

La cryptographie est l'étude des principes, méthodes et techniques mathématiques reliées aux aspects de sécurité de l'information telles la confidentialité, l'intégralité des données, authentification d'entités, et l'authentification de l'originalité des données. C'est un ensemble de techniques qui fournit la sécurité de l'information. Afin de protéger un message, on lui applique une transformation qui le rend incompréhensible ; c'est ce qu'on appelle le *chiffrement*, qui, à partir d'un *texte en clair*, donne un *texte chiffré* ou *cryptogramme*. Inversement, le *déchiffrement* est l'action qui permet de reconstruire le texte en clair à partir du texte chiffré. Dans la cryptographie moderne, les transformations en question sont des fonctions mathématiques, appelées *algorithmes cryptographiques*, qui dépendent d'un paramètre appelé *clef*.

La cryptanalyse étudie la sécurité des procédés de chiffrement utilisés en cryptographie. Elle consiste alors à casser des fonctions cryptographiques existantes, c'est-à-dire à démontrer leur sécurité. La cryptanalyse mêle une intéressante combinaison de raisonnement analytique, d'application d'outils mathématiques, de découverte de redondances, de patience, dans le but de trouver des faiblesses et, en particulier, de pouvoir décrypter des textes chiffrés. Le *décryptement* est l'action consistant à retrouver le texte en clair sans connaître la clef de déchiffrement.

La confidentialité est historiquement le premier problème posé à la cryptographie. Il se résout par la notion de chiffrement, mentionnée plus haut. Il existe deux grandes familles d'algorithmes cryptographiques à base de clefs : les algorithmes à *clef secrète* ou algorithmes symétriques, et les algorithmes à *clef publique* ou algorithmes asymétriques.

Algorithme à clé secrète

Dans la cryptographie conventionnelle, les clefs de chiffrement et de déchiffrement sont identiques : c'est la clef secrète, qui doit être connue des tiers communicants et d'eux seuls. Le procédé de chiffrement est dit symétrique. Les algorithmes symétriques sont de

deux types : les algorithmes de chiffrement en continu, qui agissent sur le texte en clair un bit à la fois et les algorithmes de chiffrement par blocs, qui opèrent sur le texte en clair par groupes de bits appelés blocs.

Un algorithme de chiffrement par blocs avec itération est un algorithme qui chiffre les blocs par un processus comportant plusieurs rondes. Dans chaque ronde, la même transformation est appliquée au bloc, en utilisant une sous-clef dérivée de la clef de chiffrement. En général, un nombre de rondes plus élevé garantit une meilleure sécurité, au détriment des performances.

Un cas particulier d'algorithme de chiffrement par blocs avec itération est la famille des chiffres de Feistel**.** Dans un chiffre de Feistel, un bloc de texte en clair est découpé en deux ; la transformation de ronde est appliquée à une des deux moitiés, et le résultat est combiné avec l'autre moitié par *ou exclusif*. Les deux moitiés sont alors inversées pour l'application de la ronde suivante. Un avantage de ce type d'algorithmes est que chiffrement et déchiffrement sont structurellement identiques. La méthode la plus employée pour concevoir un procédé de chiffrement est de chercher à réaliser une transformation suffisamment compliquée et irrégulière pour que son analyse soit difficile. Cette méthode empirique ne fournit aucune garantie quant à la résistance de l'algorithme résultant. Des exemples d'algorithmes symétriques d'utilisation courante aujourd'hui sont le DES (*Data Encryption Standard*), le DES triple à deux ou trois clefs, IDEA (*International Data Encryption Algorithm*), RC5 (Rivest's Code 5),...

L'algorithme DES

Les efforts conjoints d'IBM, qui propose Lucifer fin 1974, et de la NSA conduisent à l'élaboration du DES, l'algorithme de chiffrement le plus utilisé au monde. Le DES est un algorithme de chiffrement par blocs qui agit sur des blocs de 64 bits. C'est un chiffre de Feistel à 16 rondes. La longueur de la clef est de 56 bits. Généralement, celle-ci est représentée sous la forme d'un nombre de 64 bits, mais un bit par octet est utilisé pour le contrôle de parité. Les sous-clefs utilisées par chaque ronde ont une longueur de 48 bits.

Il y a donc pour le DES 2^{56} clés possibles, soit environ ... 72 millions de milliards possibilités. Les grandes lignes de l'algorithme sont :

Diversification de la clé. Le texte est découpé en blocs de 64 bits. On diversifie aussi la clé K, c'est-à-dire qu'on fabrique à partir de K 16 sous-clés $K_1,...,K_{16}$ à 48 bits. Les K_i sont composés de 48 bits de K, pris dans un certain ordre.

Permutation initiale**.** Pour chaque bloc de 64 bits x du texte, on calcule une permutation finie y=P(x). Une fois la permutation initiale réalisée, le bloc de 64 bits est scindé en deux blocs de 32 bits, notés respectivement **G** et **D** (pour gauche et droite, la notation anglo-saxonne étant *L* et *R* pour *Left and Right*). On note $\mathbf{G}_0$ et $\mathbf{D}_0$ l'état initial de ces deux blocs. Il est intéressant de remarquer que $\mathbf{G}_0$ contient tous les bits possédant une position paire dans le message initial, tandis que $\mathbf{D}_0$ contient les bits de position impaire..

Itération**.** Les blocs $\mathbf{G}_n$ et $\mathbf{D}_n$ sont soumis à un ensemble de transformations itératives appelées *rondes*. On applique 16 rondes d'une même fonction. A partir de $G_{i-1}D_{i-1}$ (pour i de 1 à 16), on calcule G_iD_i en posant :

$G_i=D_{i-1}$.

$D_{i-1}=G_{i-1}$ XOR $f(D_{i-1},K_i)$.

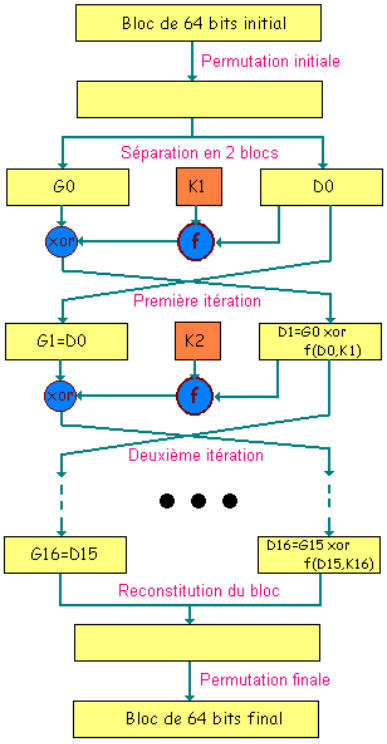
XOR est le ou exclusif bit à bit, et f est une fonction de confusion, suite de substitutions et de permutations.

Les 32 bits du bloc $\mathbf{D}_0$ sont étendus à 48 bits grâce à une table (matrice) appelé *table d'expansion* (notée **E**), dans laquelle les 48 bits sont mélangés et 16 d'entre eux sont dupliqués. La matrice résultante de 48 bits est appelée $\mathbf{D'}_0$ ou bien $\mathbf{E}[\mathbf{D}_0]$. L'algorithme DES procède ensuite à un *OU exclusif* entre la première clé $\mathbf{K}_1$ et $\mathbf{E}[\mathbf{D}_0]$. Le résultat de ce *OU exclusif* est une matrice de 48 bits que nous appellerons $\mathbf{D}_0$ par commodité (il ne s'agit pas du $\mathbf{D}_0$ de départ!).

$\mathbf{D}_0$ est ensuite scindé en 8 blocs de 6 bits, noté $\mathbf{D}_{0i}$. Chacun de ces blocs passe par des **fonctions de sélection** (appelées parfois *boîtes de substitution* ou *fonctions de compression*), notées généralement $\mathbf{S}_i$. Les premiers et derniers bits de chaque $\mathbf{D}_{0i}$ détermine (en binaire) la ligne de la fonction de sélection, les autres bits (respectivement 2, 3, 4 et 5) déterminent la colonne. La sélection de la ligne se faisant sur deux bits, il y a 4 possibilités (0,1,2,3). La sélection de la colonne se faisant sur 4 bits, il y a 16 possibilités (0 à 15). Grâce à cette information, la fonction de sélection "sélectionne" une valeur codée sur 4 bits.

Le bloc de 32 bits obtenu est enfin soumis à une permutation **P.** L'ensemble de ces résultats en sortie de **P** est soumis à un *OU Exclusif* avec le $\mathbf{G}_0$ de départ (comme indiqué sur le premier schéma) pour donner D1, tandis que le $\mathbf{D}_0$ initial donne $\mathbf{G}_1$.

Permutation finale : on applique à $G_{16}D_{16}$ l'inverse de la permutation initiale. $Z=P^{-1}(G_{16}D_{16})$ est le bloc de 64 bits chiffré à partir de x.



Régulièrement, le DES a fait l'objet de polémiques. Toute sa sécurité repose sur la fonction de confusion f, et en particulier à l'intérieur de celle-ci sur des boîtes S, tableau 4x16 d'entiers compris entre 0 et 15, aux valeurs mystérieuses. Certains ont affirmé que la NSA, qui a finalisé l'algorithme, a placé dans ces boîtes S des trappes qui lui permettaient de tout décrypter, tout en affirmant que l'algorithme est sûr. Toutefois, rien n'a

objectivement étayé cela. En particulier, le DES a toujours résisté aux travaux des cryptanalystes non basés sur la force brute. En revanche, ce qui a signé l'arrêt de mort du DES est l'extraordinaire progression de la puissance des ordinateurs. Le 17 juin 1997, le DES est cassé en 3 semaines par une fédération de petites machines sur Internet. Et on estime à cette date à quelques secondes le temps nécessaire à un Etat pour percer les secrets d'un message chiffré avec le DES.

Le DES triple est une variante du DES qui consiste à appliquer l'algorithme trois fois à chaque bloc : chiffrement, déchiffrement puis de nouveau chiffrement. Les clefs utilisées pour chaque application du DES peuvent être toutes les trois distinctes, ou bien on peut utiliser seulement deux clefs distinctes : une pour le chiffrement et une pour le déchiffrement. Dans tous les cas, la longueur efficace de la clef du triple DES ne dépasse pas 112 bits.

Cryptographie à clé publique

Le problème essentiel est alors la distribution des clés, ce secret que l'envoyeur et le destinataire doivent partager pour pouvoir respectivement chiffrer et déchiffrer. Les armées et les états ont recours aux valises diplomatiques pour ces échanges, mais ceci n'est pas accessible aux civils...

Le concept de cryptographie à clef publique fut inventé par Whitfield Diffie et Martin Hellman en 1976, dans le but de résoudre le problème de distribution des clefs posé par la cryptographie à clef secrète. De nombreux algorithmes permettant de réaliser un cryptosystème à clef publique ont été proposés depuis. Ils sont le plus souvent basés sur des problèmes mathématiques difficiles à résoudre, donc leur sécurité est conditionnée par ces problèmes, sur lesquels on a maintenant une vaste expertise. Mais, si quelqu'un trouve un jour le moyen de simplifier la résolution d'un de ces problèmes, l'algorithme correspondant s'écroulera.

Nombre des algorithmes proposés pour la cryptographie à clef publique se sont révélés rapidement non sûrs, ou non réalisables sur le plan pratique. Tous les algorithmes actuels présentent l'inconvénient d'être **bien plus lents que les algorithmes à clef secrète** ; de ce fait, **ils sont souvent utilisés non pour chiffrer directement des données, mais pour chiffrer une *clef de session* secrète**. Certains algorithmes asymétriques ne sont adaptés qu'au chiffrement, tandis que d'autres ne permettent que la signature. Seuls trois algorithmes sont utilisables à la fois pour le chiffrement et pour la signature : RSA, ElGamal et Rabin.

L'algorithme RSA

La méthode de cryptographie RSA a été inventée en 1977 par Ron Rivest, Adi Shamir et Len Adleman, à la suite de la découverte de la cryptographie à clé publique par Diffie et Hellman. Le RSA est encore le système cryptographique à clé publique le plus utilisé de nos jours. Il est intéressant de remarquer que son invention est fortuite: au départ, Rivest, Shamir et Adleman voulaient prouver que tout système à clé publique possède une faille. Cet algorithme repose sur la difficulté de factoriser des grands nombres.

Principe de fonctionnement

1. On commence par choisir deux grands nombres premiers, p et q, et on calcule $n = pq$. n est rendu public ; p et q doivent rester secrets et sont donc détruits une fois les clefs générées. p et q sont deux grands nombres premiers distincts. Leur génération se fait au hasard, en utilisant un algorithme de test de primalité probabiliste.

2. On choisit ensuite aléatoirement une clef publique e est un entier premier avec le produit (p-1)(q-1).

3. La clé privée d est obtenue grâce à l'algorithme d'Euclide, tel que ed=1 modulo (p-1)(q-1). Autrement dit, ed-1 est un multiple de (p-1)(q-1). On peut fabriquer d à partir de e, p et q.

Le couple (n,e) constitue la clé publique qui est rendue disponible par exemple en la mettant dans un annuaire. Le couple (n,d) constitue la clé privée.

Soit *m* le message en clair et *c* le cryptogramme. La fonction de chiffrement est, de façon simplifiée, $c = m^e \bmod n$ (si *m* est plus grand que *n*, il est séparé en morceaux de valeur inférieure à *n* et chaque morceau est chiffré séparément suivant cette formule). Du fait de la relation entre *e* et *d*, la fonction de déchiffrement correspondante est $m = c^d \bmod n$. La signature se fait de manière similaire, en inversant *e* et *d*, c'est-à-dire en chiffrant avec une clef privée et en déchiffrant avec la clef publique correspondante : $s = m^d \bmod n$ et $m = s^e \bmod n$.

Pour un cryptanalyste, retrouver la clef privée à partir de la clef publique nécessite de connaître $(p-1)(q-1) = pq-p-q+1 = n+1-p-q$, donc de connaître *p* et *q*. Pour cela, il doit factoriser le grand nombre *n*. Donc *n* doit être suffisamment grand pour que cela ne soit pas possible dans un temps raisonnable par rapport au niveau de sécurité requis. Actuellement, la longueur du module *n* varie généralement de 512 à 2048 bits suivant les utilisations. Compte tenu de l'augmentation des vitesses de calcul des ordinateurs et des avancées mathématiques en matière de factorisation des grands nombres, la longueur minimale des clefs doit augmenter au cours du temps.

Fonction de hachage à sens unique

Aussi appelée fonction de condensation, une **fonction de hachage** est une fonction qui convertit une chaîne de longueur quelconque en une chaîne de taille inférieure et généralement fixe ; la chaîne résultante est appelée *empreinte* (*digest* en anglais) ou condensé de la chaîne initiale.

Une fonction à sens unique est une fonction facile à calculer mais difficile à inverser. La cryptographie à clef publique repose sur l'utilisation de fonctions à sens unique à brèche secrète : pour qui connaît le secret (i.e. la clef privée), la fonction devient facile à inverser.

Une fonction de hachage à sens unique est une fonction de hachage qui est en plus une fonction à sens unique : il est aisé de calculer l'empreinte d'une chaîne donnée, mais il est difficile d'engendrer des chaînes qui ont une empreinte donnée, et donc de déduire la chaîne initiale à partir de l'empreinte. On demande généralement en plus à une telle fonction d'être sans collision, c'est-à-dire qu'il soit impossible de trouver deux messages ayant la même empreinte. On utilise souvent le terme fonction de hachage pour désigner, en fait, une fonction de hachage à sens unique sans collision.

La plupart des fonctions de hachage à sens unique sans collision sont construites par itération d'une fonction de compression : le message *M* est décomposé en *n* blocs $m_1,...,m_n$, puis une fonction de compression *f* est appliquée à chaque bloc et au résultat de la compression du bloc précédent ; l'empreinte *h(M)* est le résultat de la dernière compression.

Des exemples de fonctions ainsi conçues sont par exemple MD5, développé par Rivest en 1991, MD5 produit une empreinte de 128 bits à partir d'un texte de taille arbitraire. MD5 manipule le texte d'entrée par blocs de 512 bits ou bien SHA. SHA-1 est une amélioration de SHA publiée en 1994. SHA-1 produit une empreinte de 160 bits à partir d'un message de longueur maximale 2^{64} bits. Tout comme MD5, SHA-1 travaille sur des blocs de 512 bits.

Un code d'authentification de message (*Message Authentication Code,* MAC) est le résultat d'une fonction de hachage à sens unique dépendant d'une clef secrète : l'empreinte dépend à la fois de l'entrée et de la clef. On peut construire un MAC à partir d'une fonction de hachage ou d'un algorithme de chiffrement par blocs. Il existe aussi des fonctions spécialement conçues pour faire un MAC.

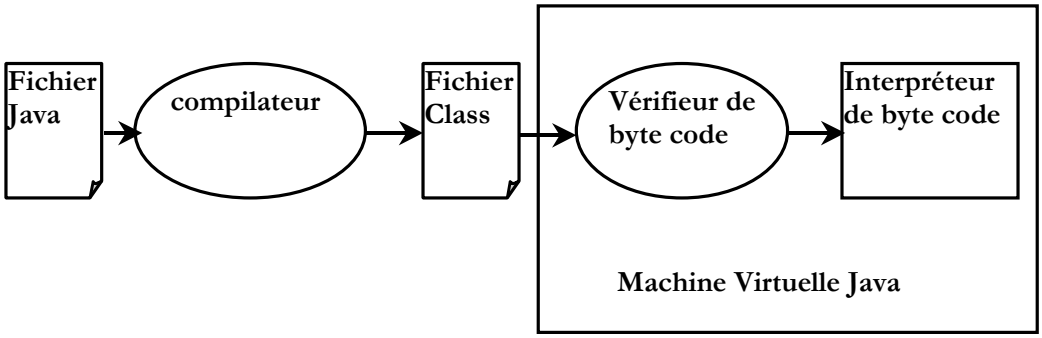
Une façon courante de générer un code d'authentification de message consiste à appliquer un algorithme de chiffrement symétrique en mode CBC au message ; le MAC est le dernier bloc du cryptogramme.

3. Java Card

3.1. Une introduction à Java Card

Afin de limiter le nombre de produits différents à développer pour satisfaire les besoins de tous les secteurs, séparer les applications du système d'exploitation et offrir aux programmeurs d'applications la portabilité de leur code entre les différentes cartes disponibles, la spécification Java Card a été introduite. Il s'agit d'une adaptation de la spécification Java qui était déjà disponible pour les ordinateurs et qui tient compte des spécificités de la carte. La spécification a subit une forte réduction permettant son implémentation dans l'espace mémoire réduit d'une carte à puce.

Les principes du langage Java orienté objet, d'API pour interagir avec l'environnement d'exécution et pour fournir des routines utilitaires au programmeur d'application ainsi que l'utilisation d'une machine virtuelle interprétant un langage intermédiaire (ou *byte code*) ont été conservés.



Concernant le langage, on retrouve la plupart des constructions syntaxiques, ainsi que les concepts orientés objet, y compris l'héritage et les interfaces. Le mécanisme d'exception a également été conservé. Cependant les types proposés sont plus retreints (les types *char*, *double*, *float*, *long*, et les tableaux multidimensionnels sont absents, le type *int* est optionnel), le clonage et le *multithreading* ne sont pas supportés. Une des spécificités de Java Card est la persistance par défaut des objets, en effet, ils sont alloués en EEPROM sauf indication contraire du programmeur. Ce mécanisme remplace la sérialisation qui n'est donc pas supportée. Une autre différence importante de Java Card est l'absence de ramasse-miettes automatique, il doit être activé volontairement depuis la version 2.2 de la spécification, avant, il n'était pas supporté. Dans tous les cas, ces spécificités nécessitent un style de programmation adapté et il n'est pas possible d'utiliser les mêmes motifs de conception qu'en Java pour ordinateur de bureau.

Concernant la machine virtuelle, le format du byte code a été modifié par rapport à Java à des fins d'optimisation. Bien sûr, toutes les instructions servant directement à supporter des caractéristiques du langage supprimées ont été retirées (comme par exemple les byte code relatifs au *multithreading*, et aux types *double*, *float*, et *long*). La taille des mots de la pile et des variables locales de la machine virtuelle est passée de 32 bits à 16 bits.

Une des particularités du langage Java est qu'il est fortement typé, et ne dispose pas explicitement de pointeur. Dès lors, un des éléments clef de la sécurité dans Java est le vérifieur de byte code. Celui-ci est une partie de la machine virtuelle Java effectuant une analyse de chaque applet chargée avant de l'exécuter. Cette analyse vise à vérifier que

l'applet est bien conforme aux règles de typage de Java, et qu'elle a bien été générée par un compilateur Java valide.

Dans l'architecture retenue pour Java Card nous remarquons deux éléments importants. Le premier concerne la machine virtuelle même qui est séparée en deux parties : une première partie hors de la carte vérifie le code et effectue des transformations sur ce code, et une deuxième partie sur la carte charge le code et l'exécute. Ce découpage en deux parties est le résultat de constatations faites sur les cartes à puce. Le peu de mémoire disponible, aussi bien de la mémoire volatile que non volatile, en font une denrée rare qu'il ne faut pas gaspiller. Cette répartition de la machine virtuelle permet de gagner de la place en mémoire, en se consacrant uniquement à l'exécution des applications.

Le deuxième point important est la modification même du fichier *Class*. En effet, ce n'est pas le fichier *Class* qui est chargé sur la carte, mais un fichier CAP pour *applet convertie* (Converted APplet). Etant donné que la carte n'est utilisée que comme un interpréteur de code, il s'agit de faciliter l'installation et prévoir un format plus simple à exécuter pour une plate-forme disposant de peu de ressources. De ce constat est né le format de fichier CAP. En effet, le fichier *Class* n'est pas directement exécutable et doit subir une phase importante d'éditions de liens. Au contraire, le format du fichier CAP est prévu pour une exécution immédiate et repose sur une phase d'édition de liens simplifiée. Pour obtenir le fichier CAP, un outil placé hors de la carte transforme le fichier *Class* en fichier CAP une fois la vérification effectuée.

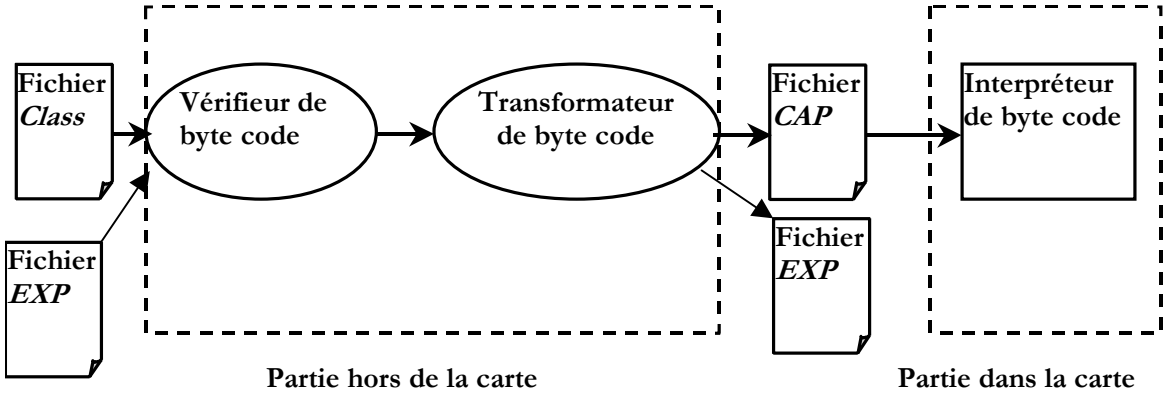


Figure Machine virtuelle Java Card utilisant un transformateur de byte code

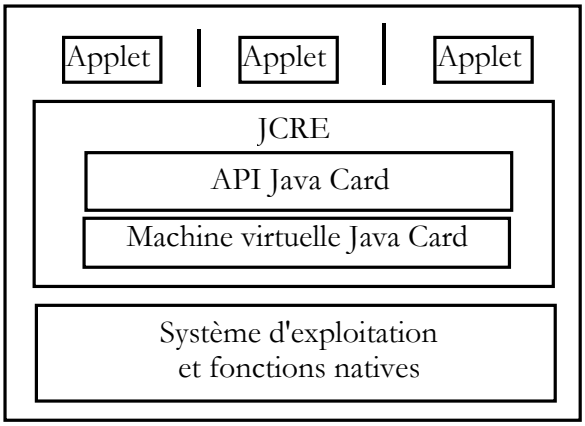
La spécification de la Java Card est constituée principalement de trois spécifications différentes :

- la spécification de l'API. Elle décrit les paquetages permettant d'utiliser un sous-ensemble du langage Java, d'utiliser les modules fournis par l'environnement Java Card et de faire appel aux fonctions de sécurité accessibles par les applets.
- la spécification de la Java Card Virtual Machine. Elle contient notamment une machine virtuelle spécifique, ainsi que des mécanismes de sécurité comme le pare-feu.
- la spécification du Java Card Runtime Environment. Il intègre les deux spécifications précédentes plus des modules complémentaires.

Quelle que soit l'implémentation utilisée, elle doit au minimum implémenter ces trois spécifications. Bien entendu, il est possible d'introduire des mécanismes

complémentaires, ainsi que des API complémentaires pour faciliter le développement des applications embarquées.

D'un point de vue global, une Java Card peut être représentée comme un système comportant deux zones différentes. La première est figée : elle est constituée du système d'exploitation et des extensions éventuellement fournies par les fournisseurs de la carte. Elle contient le JCRE et le système d'exploitation sous-jacent. La deuxième est modifiable. Elle est constituée des applets chargées dans la carte.



Structure d'une Java Card

L'application se situe exclusivement dans la zone modifiable, et ne communique qu'au travers du JCRE. En conséquence, la seule connaissance de l'interface du JCRE doit suffire pour permettre de développer des applets. Pour que le système fonctionne dans sa globalité, il est nécessaire que l'implémentation du JCRE soit conforme à sa spécification, et que les mécanismes décrits dans la spécification permettent effectivement d'assurer les propriétés spécifiées par ailleurs dans le standard.

Pour des contraintes liées à la sécurité des applications et pour des limites inhérentes aux capacités matérielles des cartes, certaines fonctionnalités présentes dans le standard Java ne sont pas présentes dans la Java Card. Le tableau suivant présente quelques-unes des différences parmi les plus significatives.

Ces contraintes ne remettent cependant pas en cause l'approche objet telle qu'elle est prévue par le standard Java. Ainsi, les applications Java Card intègrent certaines possibilités comme, par exemple, l'allocation dynamique de ressources mémoires. Ces possibilités sont en revanche à utiliser de façon contrôlée, puisque le standard Java Card ne prévoit pas par défaut de mécanismes de ramasse-miettes.

Fonctionnalités Java supportées	Fonctionnalités Java non supportées
• Certains types de base : booléen, byte, short • Tableaux à une dimension • Packages, classes, interfaces et exceptions du langage Java • Héritage, méthodes virtuelles, surcharge, création dynamique d'objets	• Certains types de base : long, double, float • Caractères et chaînes de caractères • Tableaux à plusieurs dimensions • Chargement dynamique de classes • Security manager • Ramasse-miettes et finalisation • Multi-tâches • Sérialisation d'objets • Clonage d'objets

Tableau Fonctionnalités Java supportées et non supportées

Concernant la gestion de la mémoire, le développeur doit également faire attention au fait que les applications sont embarquées dans des espaces mémoires restreints. Il est donc nécessaire que la gestion de l'espace mémoire soit explicitée. Dans la pratique, il est impératif de maîtriser les bornes supérieures liées à l'utilisation de ce mécanisme. De plus, le contenu des objets peut être alloué de façon persistante ou non, en fonction des choix du développeur. Certains objets seront dits "persistant", et d'autres "transient". Parmi les objets non-persistants, certains auront leur contenu effacé à chaque fin de session, tandis que d'autres auront leur contenu effacé lors de la remise à zéro de la carte. En conséquence, le développeur doit envisager les modes d'utilisation de la carte en général, et de son application en particulier. Cet effort de conception va se traduire concrètement par le fait que certains objets seront stockés dans la RAM, et d'autres dans l'EEPROM. En conséquence, ces choix influeront sur les temps d'exécution, la durée de vie de la carte et bien sûr la gestion de la mémoire. L'approche objet n'est donc pas remise en cause, mais les technologies utilisées aujourd'hui imposent encore quelques réflexions peut-être obsolètes demain.

Contrairement à la JVM d'un ordinateur classique, la machine virtuelle embarquée tourne toujours, le cycle d'horloge devient infini quand la carte à puce n'est pas alimentée. Elle crée les objets dans la zone EEPROM pour sauvegarder les informations de manière persistante. La sécurité est directement intégrée au niveau du système d'exploitation. Outre la traditionnelle application de débit/crédit de compte, la Java Card peut supporter tous types d'applications Java compatibles avec la norme Java Card de Sun.

Malgré ces restrictions, les avantages des Java Card sont nombreux: la portabilité (indépendance vis-à-vis du constructeur de la carte), la possibilité d'exécuter plusieurs applications sur une même carte, la possibilité de changer les applications sur une carte à puce selon le profil de l'utilisateur, la flexibilité du langage Java, la compatibilité avec d'autres standards existants (Europay/MasterCard/Visa).

3.2. Installation d'une Applet

Les applets sont donc les applications qui s'exécutent dans la carte en interagissant avec le JCRE. Elles sont identifiées par un identifiant unique (AID = Application IDentifier) défini par la norme ISO7816. Plusieurs applets peuvent résider dans la carte mais une seule est active à la fois. Lorsque l'application est chargée dans la carte elle est inaccessible tant qu'elle n'est pas installée.

La méthode statique *install* est invoquée une seule fois lors de l'installation de l'applet sur la carte. Elle contient principalement les allocations d'objets persistants de l'applet, dont l'objet représentant l'applet elle-même qui sera enregistré auprès de l'environnement d'exécution. L'enregistrement marque le début d'utilisation de l'application, il permet que l'applet soit sélectionnable. Il est possible de fournir des paramètres lors de l'installation afin de personnaliser l'applet. Si on souhaite plusieurs instances de l'applet il est nécessaire d'invoquer pour chacune la méthode *install*.

Pour enregistrer l'application il faut utiliser l'une des deux méthodes suivantes :

```
protected final void register ()
```

ou bien

```
protected final void register (byte[] bArray, short bOffset,
byte blength);
```

La première méthode enregistre en utilisant la valeur par défaut fournie dans le fichier CAP, la seconde utilise le premier paramètre pour spécifier l'AID. Lorsque plusieurs instances sont créées il est nécessaire d'utiliser la seconde méthode (l'AID est forcément unique). L'AID est composé de deux éléments. Le premier est le RID (Registered IDentifier) d'une longueur de cinq octets (donné par un organisme international ou national cf. le premier *nibble*) et le second élément optionnel, est le PIX (Proprietary application Identifier eXtension) pouvant aller jusqu'à onze octets. Le tampon bArray pouvant véhiculer plusieurs paramètres il est nécessaire d'indiquer le début et la longueur de l'AID.

3.3. Sélection d'une Applet

Il faut donc sélectionner l'application en utilisant l'AID en envoyer une APDU SELECT avec comme paramètre l'AID de l'application choisie. En effet plusieurs applets peuvent cohabiter sur une même carte Java et avant de leur envoyer une APDU de commande, l'entité dialoguant avec la carte doit d'abord envoyer une APDU de commande spécifique pour sélectionner l'applet à laquelle il souhaite envoyer les commandes suivantes.

La sélection se fait à l'aide d'un identifiant d'applet, ou AID (pour Applet IDentifier), attribué à l'installation. Lorsque qu'une applet est sélectionnée, l'environnement d'exécution invoque sa méthode *select* afin de lui permettre d'initialiser toutes les données de session.

Lorsqu'une autre applet doit être sélectionnée le JCRE doit désactiver l'applet courante en appelant la méthode *deselect*. Cette méthode peut être utilisée afin d'effacer certaines données. Il faut remarquer que cette méthode n'est pas forcément appelée (reset ou arrachement de la carte) et /ou qu'elle peut générer des exceptions et que dans ce cas le JCRE ignore ces exceptions.

3.4. Le traitement des APDU

Toutes les APDU envoyées à la carte sont transmises à l'applet sélectionnée en invoquant sa méthode *process*. L'applet peut alors analyser les octets de classe, d'instruction et éventuellement les paramètres P1 et P2 pour déterminer le traitement à effectuer avant d'envoyer une réponse.

La classe APDU donne une vison objet de la manipulation du tampon d'entrée/sortie qui peut être vu comme un objet de communication global à la carte. L'applet ne peut stocker les données de l'APDU que dans des variables locales afin d'éviter des fuites de données

vers une autre applet. Lorsque le JCRE reçoit un APDU du terminal, il le transfert dans cet objet et appel la méthode *process* avec l'APDU en paramètre. Les données de retour sont à leur tour écrites dans ce tampon.

```
process(APDU myApdu) throws ISOException{
// On obtient le tableau contenant l'APDU de commande
byte[] buffer = myApdu.getBuffer();
```

Comme nous l'avons vu précédemment, un APDU est composé d'un en-tête et potentiellement de données. Les octets de classe et d'instructions sont utilisés pour structurer son application.

```
if ( buffer[ISO7816.OFFSET_CLA] == (byte)0x32 ) {
   switch ( buffer[ISO7816.OFFSET_INS] )
    case..: …
         short lg=(short) myApdu [ISO7816.OFFSET_LC] &0xFF;
         short nb = myApdu.setIncommingAnd Receive();
if ( buffer[ISO7816.OFFSET_CLA] !== EXPECTED_VALUE )
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
```

Dans cet exemple on utilise le code de classe pour s'assurer qu'il s'agit d'une commande pour cette application et si ce n'est pas le cas on jette une exception. Ensuite on exécute différentes commandes en fonction du champ instruction.

Le nombre de données véhiculées par cet APDU est obtenu en inspectant le cinquième champs : `OFFSET_LC` qui est une valeur non signée qui doit être convertie. Si la valeur retournée est non nulle il est possible de demander au JCRE de rester en mode réception et d'accepter des données dans le tampon. Si le nombre de données attendues `(lg)` est supérieur à la taille du tampon il est nécessaire de demander la suite des données autant de fois que nécessaire (lg-nb modulo taille du buffer) en utilisant la méthode `receiveBytes(short )`.

Le renvoi de donnée vers le terminal nécessite le retournement de la ligne (ligne half-duplex) à l'aide de la méthode `setOutgoing ()` comme le montre l'exemple suivant.

Cette méthode ne fait QUE indiquer au JCRE qu'il doit passer en mode émetteur et retourne le nombre de données attendues par le terminal (paramètre Le de l'APDU). À partir de cet instant, si des données continuent d'arriver elles seront perdues.

```
short le = myApdu.setOutgoing();
// retournement de la ligne
if ( le < 2 ) ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
// cette méthode doit renvoyer deux données si
// le terminal en demande moins => erreur
myApdu.setOutgoingLength((byte)2);
// indication du nb de données retournées
buffer[0] = (byte)(balance >> 8);
buffer[1] = (byte)(balance & 0xFF);
```

```
// stockage dans le buffer des deux octets de donnée
myApdu.sendBytes((short)0, (short)2);
// émission de deux données à partir de l'index 0
```

Il est possible si il n'est pas nécessaire d'envoyer des données supplémentaires, de réaliser ces trois opérations avec la seule méthode `setOutgoinAndSend(short offset, short le)`.

Les octets de statut sont envoyés automatiquement par le JCRE si tout c'est bien passé (SW=9000). Si une exception a été levée, comme par exemple ci dessus, le statut renvoyé par le JCRE sera celui de l'exception si cette dernière n'est par rattrapée par l'application.

3.5. Le mécanisme de partage

Même si cela est possible, il n'est pas recommandé d'utiliser le tampon d'entrée/sortie comme mécanisme de partage entre différentes applications.

3.6. Utilisation de l'API crypto

L'API dispose d'un package appelé `javacardx.crypto` pour le support des fonctions cryptographiques. Ce package contient une classe appelée Cipher utilisable pour les fonctions de cryptage et de décryptage. Elle dispose de différents algorithmes. Pour l'utiliser quatre étapes sont nécessaires :

- Instanciation de la classe Cipher
- Instanciation d'un objet DESKey
- Positionner l'objet pour du cryptage ou du décryptage
- Exécuter l'algorithme

La première étape est l'instanciation de la classe Cipher en choisissant l'algorithme. La méthode getInstance () est statique, elle prend deux paramètres.

```
Cipher cipher =Cipher.getInstance(Cipher.ALG_DES_CBC_NOPAD,
true);
```

Le premier paramètre défini l'algorithme utilisé (voir la table ci-après), alors que le second paramètre spécifie si l'objet pourra être partagé ou non entre différentes applets.

```
1 ALG_DES_CBC_NOPAD for the DES CBC algorithm without padding
2 ALG_DES_CBC_ISO9797_M1 for DES CBC algorithm with ISO9797
padding of method 1 scheme
3 ALG_DES_CBC_ISO9797_M2 for DES CBC algorithm with ISO9797
padding of method 2 scheme
4 ALG_DES_CBC_PKCS5 for DES CBC algorithm with PKCS5 scheme
padding
5 ALG_AES_BLOCK_128_CBC_NOPAD for AES block 128 CBC algorithm
without padding
6 ALG_AES_BLOCK_128_ECB_NOPAD for AES block 128 ECB algorithm
without padding
7 ALG_DES_ECB_NOPAD for DES EBC algorithm without padding
8 ALG_DES_EBC_ISO9797_M1 for DES EBC algorithm with ISO9797
padding of method 1 scheme
```

```
9 ALG_DES_EBC_ISO9797_M2 for DES EBC algorithm with ISO9797
padding of method 2 scheme
10 ALG_RSA_ISO14888 for RSA algorithm with ISO14888 padding
11 ALG_RSA_NOPAD for RSA algorithm without padding
12 ALG_RSA_PKCS1 for RSA algorithm with PKCS1 padding
```

La seconde étape concerne l'instanciation de l'objet DESKey suivi de l'initialisation de l'objet par la fonction setKey().

```
DESKey desKey = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES,false);

byte [] keyBytes = {(byte)0x01,(byte)0x02,(byte)0x03,(byte)0x04};}

desKey.setKey(keyBytes, (short)0);
```

La troisième étape consiste à positionner l'objet Cipher pour la fonction requise (second paramètre) ainsi que d'initialiser avec la clé construite précédemment (premier paramètre).

```
cipher.init(desKey, Cipher.MODE_DECRYPT);
```

Finalement, il ne reste plus qu'à encrypter ou décrypter le bloc de donnée en utilisant la méthode doFinal(). Cette méthode possède cinq paramètres, les trois premiers spécifient les données d'entrée que l'on veut chiffrer ou déchiffrer, les deux derniers spécifient le tableau de sortie et l'offset dans ce tableau.

```
byte []inputData = {(byte)0x01,(byte)0x02,(byte)0x03,(byte)0x04};}
byte []outputData = new byte[8];
cipher.doFinal( inputData,(short)0, (short)inputData.length,
outputData, (short)0);
```

3.7. Exemple

Concernant les API, celles-ci ont été très fortement limitées. On trouve simplement un petit sous-ensemble du *package* `java.lang` ainsi que des nouveautés pour gérer les spécificités de la carte et de nombreux algorithmes cryptographiques. Une application pour Java Card doit obligatoirement contenir une classe étendant la classe `javacard.framework.Applet`. Le squelette d'une application est donc le suivant :

```
public class MonApplet extends javacard.framework.Applet{
      public static void install(byte[] tab, short debut, byte
long) throws

ISOException {
            MonApplet monInstance = new MonApplet();
            …
      }
      public boolean select(){
            // Initialisation des informations de session
            return true;
      }
      process(APDU apdu) throws ISOException{
            // On obtient le tableau contenant l'APDU de commande
```

```
            byte[] buffer = apdu.getBuffer();

            if ( buffer[ISO7816.OFFSET_CLA] == (byte)XX ) {
                  switch ( buffer[ISO7816.OFFSET_INS] ) {
                        case YY:
                        // Exécuter la commande et envoyer la
                  réponse
                        break;
                        case ...
                  }
            }
      }
}
```

3.8. Les RMI

Dans la dernière version (2.2) de la spécification Java Card a été introduit une version simplifiée des appels de méthode à distance de Java (ou RMI pour Remote Method Invocation) permettant de dispenser le programmeur de la gestion des APDU. Cependant, le protocole RMI est toujours transporté dans des APDU.

En résumé, Java Card permet

- de séparer les applications pour carte à puce de leur environnement d'exécution, autorisant la diffusion de cartes à puce génériques et d'applets portables,
- le chargement d'application dynamiquement durant la vie de la carte,
- la présence de plusieurs applications sur une même carte pouvant être sélectionnées successivement,
- de structurer des applications librement en gérant des objets persistants, sans forcément passer par un système de fichier.

4. Sécurité physique

La sécurité physique des cartes est un élément important de la chaîne sécuritaire des systèmes comportant des cartes à puce. En effet, contrairement à un élément comme un serveur que l'on peut considérer comme étant en lieu sûr, les cartes à puce se trouvent en permanence ou peuvent tomber facilement entre les mains d'un attaquant. Dans le cas d'un porte-monnaie électronique, son porteur a des intérêts contraires à l'émetteur de la carte et peut chercher à attaquer sa propre carte pour la créditer frauduleusement. Dans le cas d'une carte SIM, l'abonné à un réseau de téléphone mobile n'a pas intérêt, en général, à attaquer sa propre carte, mais un voleur aura intérêt à attaquer la carte pour en extraire le PIN ou contourner sa présentation afin d'utiliser frauduleusement le téléphone.

Les attaques physiques reposent sur l'observation ou la modification du support d'exécution des programmes, c'est-à-dire sur le microprocesseur et ses connexions externes.

4.1. Les attaques invasives

Les attaques invasives supposent l'accès aux composants élémentaires et aux bus internes du microprocesseur afin de lire/écrire/effacer des données ou même de modifier sa structure. Il en résulte que la carte à microprocesseur attaquée est généralement détruite et inutilisable.

Les attaques invasives supposent l'utilisation d'équipements rares et coûteux, tel qu'un FIB (Focus Ion Beam), et habituellement destinés à faire de la mise au point ou du test de composants chez les fournisseurs de composants. Un outil comme le FIB outil permet à la fois d'observer un circuit comme avec un microscope électronique à balayage, d'y creuser des cavités, et même de déposer de la matière (isolante ou conductrice) à sa surface. Précisons que l'utilisation de tels équipements requiert un niveau de connaissance et de compétence élevé. De ce fait, le risque lié aux attaques invasives est souvent considéré comme peu élevé.

Figure 4 : A gauche, piste conductrice coupée, à droite conection établie à l'aide d'un FIB

La Figure 4 illustre un exemple d'attaque physique. Certains circuits d'un microcontrôleur de carte à puce peuvent être désactivés à l'aide de fusibles. C'était en particulier le cas pour des circuits de test utilisés pour obtenir des états internes du microcontrôleur. Une fois les tests passés à la fin de la fabrication, ces circuits de tests

devenaient une menace pour la sécurité de la carte. Dans certains cas, ils pouvaient être réactivés à l'aide d'un FIB en reconstituant une piste coupée par un dépôt de matière isolante.

Au fils des ans les fabricants de circuits pour carte à puce ont perfectionné les défenses contre les attaques physiques invasives. La mémoire ROM est en général située au plus prés du substrat de silicium afin de rendre plus difficile sa lecture (il faut en effet retirer finement toutes les couches de métal au dessus, sans pour autant détruire celle de la ROM). Les circuits sont couramment surmontés d'une couche de métal formant un bouclier qui empêche d'intervenir sur le circuit sans le détruire. Les mémoires non volatiles sont chiffrées et des cellules d'adresse voisines n'ont pas une localisation physiquement proche. Les bus eux-mêmes entre le processeur et les mémoires sont chiffrés et multiplexés de telle sorte que si un attaquant arrive à poser une sonde sur une piste d'un bus interne d'un composant, l'information obtenue sera chiffrée et correspondra à des instants différents à des bits différents des mots circulant sur le bus. Finalement, en dehors des blocs mémoire, toute la conception du circuit est brouillée et le placement et le routage sont fait en dépit des règles habituelles pour rendre très difficile toute rétro-conception.

4.2. Les attaques non-invasives

En opposition avec les attaques invasives, les attaques non-invasives ne nécessitent pas de «détruire» la carte à microprocesseur cible. Ainsi, certaines attaques non-invasives peuvent être réalisées à l'insu et au détriment du porteur de la carte. De ce fait, les risques liés aux attaques non-invasives sont généralement considérés comme réels.

Parmi les attaques non-invasives, nous distinguons les attaques basées sur l'observation de l'environnement de la carte de celles basées sur sa perturbation.

L'observation consiste essentiellement à détecter et exploiter des «canaux cachés» pouvant laisser passer de l'information sur l'état interne de la carte, et en particulier, pour acquérir de l'information sur les secrets contenus dans la carte. Actuellement, ce sont essentiellement les canaux caché temporels ou l'analyse du signal de tension d'alimentation qui sont exploités (attaques de type SPA (Simple Power Attack) ou DPA (Differential Power Attack)). Le rayonnement électromagnétique provoqué par le fonctionnement du microprocesseur constitue également une source d'information.

La perturbation des entrées/sorties offre de nombreuses possibilités d'attaques. En appliquant des valeurs hors normes de façon continue ou transitoire, il est possible de perturber le microprocesseur et de modifier sélectivement une partie de la RAM, des registres ou bien de provoquer un décodage erroné d'une instruction. La plupart du temps, ce genre d'expérience n'aboutit à rien d'exploitable, mais, en systématisant les essais, il est possible de découvrir des comportements hautement intéressants. Il peut s'agir, par exemple, de la transmission de données après la limite du tampon d'entrées/sorties (donc potentiellement de secrets) ou de la modification en RAM de droits d'accès permettant ensuite d'accéder à des informations protégées.

Enfin, les attaques de type DFA consistent à induire des fautes lors de la manipulation de secrets (par exemple, lors d'une signature RSA) et à comparer les différents résultats produits. Elles peuvent permettre de remonter, très rapidement, aux secrets manipulés. Les attaques DFA, comme les précédentes, peuvent être effectuées dans un cadre invasif ou combinées avec des attaques invasives, ce qui accroît alors leurs capacités.

Les contre-mesures contre les attaques de type SPA ou DPA sont implémentées au niveau matériel et logiciel. Les fabricants de circuits ont introduit des mécanismes tels que des horloges désynchronisées perturbant l'analyse du signal par un attaquant. Les fournisseurs de carte à puce ont quant à eux développé des algorithmes cryptographiques ou toutes les données manipulées sont masquées par des nombres aléatoires afin de dé corréler les données secrètes des signaux observés.

En ce qui concerne l'induction de fautes, les parades sont relativement simples et bien connues : il s'agit essentiellement de faire des vérifications d'intégrité sur les données ou de répéter des calculs sensibles plusieurs fois avant de s'assurer que leurs résultats sont identiques. La partie délicate de la tâche consiste à trouver le juste compromis entre les performances, la place mémoire et la sécurité.

5. Sécurité logique

Nous venons de voir l'état de l'art dans les attaques et les contre-mesures sur le matériel. Il est évidemment que la complexité des cartes d'aujourd'hui ouvre la porte à de nouvelles attaques : les attaques logiques. Ces attaques bien connues de l'informatique généraliste consistent essentiellement à détecter des vulnérabilités dans le code et de construire ensuite une attaque afin de l'exploiter. Les contre mesures à mettre en œuvre sont issues du domaine de la sûreté de fonctionnement.

Pour réaliser des systèmes sûrs de fonctionnement, deux techniques complémentaires que sont la prévention des fautes et la tolérance aux fautes sont en général utilisées [Avi-01]. La prévention consiste à réduire la présence de fautes soit par des techniques d'élimination d'erreur soit par des techniques d'évitement et, par conséquent à identifier les actions les plus appropriées à mener pour l'amélioration du processus de développement du système. Parmi ces techniques nous pouvons citer l'utilisation de composants fiables, de méthodologies de développement du matériel et du logiciel, et des méthodologies de fabrication. Ces techniques d'évitement, bien que nécessaires, n'éliminent jamais toutes les fautes. De plus, parvenu à un certain stade, l'augmentation des mesures d'évitement a un effet marginal sur la probabilité de présence de fautes résiduelles. Ainsi, l'évitement de fautes n'éliminant pas les défaillances de composants même s'il retarde leur occurrence, il est nécessaire de développer les systèmes en les dotant de mécanismes de tolérance aux fautes. Si la tolérance aux fautes est un mécanisme acceptable dans un environnement (*e.g.* calculateurs aéronautique) il peut devenir totalement inacceptable si le coût de fabrication devient le premier critère commercial. Dans ce contexte, seules les techniques d'évitement ou d'élimination sont possibles.

L'évitement des fautes cherche à prévenir la présence de fautes ou l'occurrence de pannes touchant la structure du système. Ceci est obtenu par exemple en contraignant la démarche de conception par des règles régissant celle-ci ou l'utilisation de techniques garantissant par construction l'absence de fautes. Parmi les méthodes possibles, il est possible d'utiliser des spécifications formelles associées à des techniques de validation par la preuve et à la génération automatique de code à partir des spécifications.

L'élimination des fautes cherche à détecter la présence de fautes puis à les localiser dans le but de les extraire ensuite. Les diverses techniques de test fonctionnel ou structurel répondent par exemple à ces objectifs. De nombreux travaux de recherche ont porté sur l'utilisation de modèles pour générer de manière automatique des suites de test.

Ces deux techniques ont un point commun l'utilisation de modèles et de différentes techniques de vérification et de génération de code. Si l'utilisation de techniques formelles dans le milieu académique est bien répandue, il n'en est pas de même dans l'industrie (sauf peut-être dans le domaine de la vérification de circuits). Rares sont les industries pour lesquelles l'utilisation de techniques formelles est intégrée dans leur processus de développement. Le contexte de la carte à puce est différent de celui que l'on rencontre généralement. La défaillance d'une carte à puce n'a pas de conséquences catastrophiques pour l'individu, contrairement aux systèmes de transport ou d'énergie. Par contre les pertes financières peuvent être très importantes. Si ces méthodes sont techniquement applicables à la carte à puce, il n'est absolument pas évident qu'elles soient économiquement intéressantes. Ces techniques d'évitement en utilisant des modèles mathématique et de la synthèse de code peuvent résoudre les attaques logiques contre des fautes d'implémentation des cartes à puce.

5.1. Eviter les vulnérabilités

Eviter les fautes en concevant a priori le bon système est un moyen de garantir que ce dernier répond bien aux spécifications. Ceci est d'autant plus crucial pour les fonctions de sécurité quand ces dernières deviennent complexes dans leur compréhension, leur implémentation et leur vérification. Pour les cartes à puce, certains modules sont très coûteux à tester (développement de plusieurs milliers d'applications de test) et la conséquence d'une vulnérabilité peut être grave.

Dans ce contexte des cartes à puce ouvertes, la possibilité d'autoriser le chargement de code en dehors de l'espace de confiance du fabricant de la carte induit de nouveaux risques d'atteinte à la sécurité. En effet, il n'y a aucune raison d'estimer que le nouveau code chargé a été développé de manière à ne pas interférer avec les applications existantes. Dès lors, un des principaux problèmes pour le déploiement de nouvelles applications est de pouvoir fournir l'assurance que ces nouvelles applications ne risquent pas de compromettre l'intégrité de la carte ou la confidentialité des autres données qui y sont stockées. Dans le cas de Java, la politique de sécurité est répartie dans plusieurs composants : l'interpréteur, le vérifieur, les librairies de base. Cette politique définit des propriétés devant être respectées par tout programme. Par exemple, il n'est pas possible de construire un pointeur à partir d'une valeur entière, Java étant un langage fortement typé.

Un point clef de cette politique de sécurité est le vérifieur de byte code. Cette partie de la machine virtuelle analyse statiquement les programmes chargés afin de s'assurer de leur innocuité. De plus, afin d'effectuer cette analyse, il vérifie la syntaxe (binaire) des programmes chargés. La construction correcte d'un tel vérifieur est d'une importance capitale, elle permet d'assurer la sécurité du système dans sa globalité. Il est donc nécessaire d'apporter une preuve mathématique de la conformité de l'implémentation d'un vérifieur vis-à-vis de sa spécification en synthétisant le code à partir de la spécification et en démontrant chaque étape de raffinement.

5.2. Méthodes formelles

Les méthodes de développement rigoureuses se basent sur l'élaboration de modèles mathématiques pour vérifier ou assurer les propriétés du système considéré. Le modèle est une abstraction mathématique du monde réel obtenue après suppression de certains détails d'implémentation ou après le choix de certaines hypothèses et de caractéristiques essentielles du système.

Certaines méthodes donnent la possibilité de raffiner ce modèle abstrait dans un modèle concret. En informatique, ce modèle concret se trouve alors très proche d'une implémentation. Il s'agit alors de synthétiser ce modèle concret dans un langage de programmation et obtenir ainsi un code exécutable. Ce programme obtenu représente l'implémentation formelle du système spécifié, en préservant les propriétés inclues au niveau le plus abstrait.

La modélisation mathématique a de nombreux avantages dont les principaux sont :

- une précision dans la spécification formelle supérieure à celle fournie par une description informelle en langage naturel qui est souvent ambiguë. C'est la force de la modélisation : lever les ambiguïtés,
- des fondements permettant d'énoncer des propriétés et d'étudier le comportement du système dans son ensemble.

La modélisation peut prendre diverses formes, en fonction de la nature des mathématiques retenues pour exprimer et formaliser le système. Il existe différentes techniques de modélisation certaines reposent sur des machines d'état abstraites, d'autres sur de automates et enfin les modèles fonctionnels qui eux sont basés sur la notion de fonction dans le sens mathématique du terme.

A partir de l'élaboration d'un modèle d'une application, nous disposons d'une base pour raisonner sur ces modèles. S'offrent alors deux techniques différentes pour prouver ou assurer que les propriétés de l'application sont respectées par le modèle construit. Il est possible d'utiliser des vérifieurs de modèle qui en parcourant l'ensemble des états possibles de la spécification effectue la vérification par rapport aux propriétés qui sont attendues sur ce modèle. Cette vérification est entièrement automatisée, elle consiste à explorer tous les cas possibles. L'autre technique, à base de prouveurs de théorèmes ne reposent pas sur des systèmes finis et décidables. Ils permettent d'apporter la preuve de correction du programme ou du système à modéliser. Dans la suite de ce chapitre nous utiliserons la notation issue de la méthode B.

5.3. Description du vérifieur

Un élément important de la politique de sécurité d'une Java Card est le *vérifieur de byte-code*. La vérification de byte-code a pour objectif de s'assurer du respect de contraintes statiques sur le code chargé. Ces contraintes assurent que le byte-code peut être exécuté sans risque par la machine virtuelle et ne peut pas outrepasser les mécanismes de sécurité de haut niveau. La vérification consiste à effectuer une analyse statique du code mobile chargé, également appelé applet dans le cadre du langage Java. Cette analyse assure que le fichier contenant l'applet est un fichier valide, qu'il n'y a pas de débordement de pile, que le flot d'exécution reste confiné sur du byte-code valide, que chaque argument d'une instruction est d'un type correct et que les appels de méthodes sont effectués conformément à leurs attributs de visibilité (`public`, `protégé`, `privé`). Le premier point correspond à de la vérification structurelle alors que les suivants concernent la vérification du typage.

Il n'existe pas de description précise des fonctionnalités du vérifieur de byte code. Il existe des descriptions de la machine défensive Java comme décrit par Lindholm (Lindholm *et al)*, 1996. Cette machine défensive contient la sémantique statique et opérationnelle de chaque instruction, incluant implicitement les tests que doit faire un vérifieur de byte code ainsi qu'une explication succincte sur le principe de l'analyse de flot de données réalisée par le vérifieur. La construction correcte d'un tel vérifieur est d'une importance capitale, elle permet d'assurer la sécurité du système dans sa globalité. C'est donc dans ce contexte que nous montrons comment les méthodes formelles peuvent apporter une preuve mathématique de la conformité de l'implémentation d'un vérifieur vis-à-vis de sa spécification.

Pour ce faire nous formalisons le vérifieur de byte-code pour le langage Java Card complet (hormis les instructions `jsr` et `ret` qui sont traitées séparément), et nous montrons comment qu'une implémentation synthétisée à partir de cette formalisation remplit les contraintes de la carte à puce.

Vérification structurelle

La vérification structurelle consiste à s'assurer que le fichier chargé est un fichier valide. En fait, cela permet de garantir que le fichier contient bien la description des classes Java et du byte-code, et que les informations qu'il contient sont cohérentes entre elles. Par

exemple, le vérifieur s'assure que toutes les structures sont de la taille appropriée et que les parties référencées existent réellement. Ces tests ont pour objectif de s'assurer que le fichier chargé ne peut pas être mal interprété par le vérifieur de type ou la machine virtuelle.

En dehors des tests structuraux purement dédiés à la vérification du format binaire, d'autres tests, plus en relation avec le contenu du fichier, sont effectués. Ces tests assurent par exemple qu'il n'y a pas de cycle dans la hiérarchie d'héritage des classes, ou que les méthodes finales ne sont pas surchargées.

Dans le cas de Java Card, les tests structuraux sont plus importants que dans le cas de Java. Cela vient du fait que le format du fichier CAP[12] utilisé pour stocker les paquetages Java Card a été conçu pour une installation simple et une édition de liens simplifiée (Sun, 2000). Par exemple, la majorité des références vers d'autres composants est donnée sous la forme de décalage dans le composant, et ainsi, la vérification structurelle s'assure que ces décalages sont confinés à ce composant.

Un fichier au format CAP est constitué de plusieurs composants qui contiennent chacun des informations spécifiques du paquetage Java Card. Par exemple, le composant *Method* contient le byte-code de chaque méthode et le composant *Class* contient les informations sur les classes comme les références vers la superclasse et les méthodes déclarées. De plus, l'algorithme de vérification que nous utilisons nécessite la présence d'un composant supplémentaire qui est ajouté grâce au mécanisme d'extension de Java Card. Ainsi, avec ce composant additionnel, nommé *Proof component*, nous facilitons le processus de vérification tout en restant compatible avec la spécification Java Card.

Dans le cas de Java Card, nous distinguons les vérifications structurelles internes et externes. Les vérifications internes correspondent aux vérifications qui peuvent être effectuées sur un composant. Un exemple de ces vérifications est de vérifier que le composant *Class* est ordonné selon la hiérarchie des classes. Les vérifications structurelles externes correspondent aux tests assurant la cohérence entre les différents composants du paquetage ou avec les autres paquetages. Par exemple, un de ces tests consiste à vérifier que les méthodes déclarées dans le composant *Class* correspondent à des méthodes existantes dans le composant *Method*.

La vérification de type

Cette vérification est effectuée méthode par méthode et doit être faite pour chaque méthode du paquetage, c'est-à-dire pour chaque méthode contenue dans le composant *Method* du fichier CAP à vérifier.

La partie vérification du typage assure qu'aucune conversion de type interdite d'après les règles de typage du langage Java Card n'est effectuée par le programme. Par exemple, un entier ne peut pas être converti en référence sur un objet, le changement du typage ne peut être effectué qu'en ayant recours à l'instruction `checkcast` qui s'assure que le changement peut effectivement être effectué. De même, les arguments passés en paramètre à une méthode doivent être de types compatibles avec ceux attendus par la méthode.

[12] CAP pour Converted Applet, est le nom du format des fichiers que l'on charge dans les cartes à puce Java Card.

Comme les types des variables locales ne sont pas explicitement stockés dans le byte-code, il est nécessaire de retrouver le type de ces variables en analysant le byte-code. Cette partie de la vérification est la plus compliquée, et la plus coûteuse à la fois en temps et en mémoire. En effet, cela nécessite de calculer le type de chaque variable et de chaque élément dans la pile pour chaque instruction et chaque chemin d'exécution possible.

Dans le but de rendre une telle vérification possible, la spécification est assez restrictive sur les programmes qui sont acceptés. Des hypothèses sont prises par Sun Microsystem pour permettre la vérification. Seuls les programmes dont le type de chaque élément de la pile et de chaque variable locale est le même quelque soit le chemin pris pour atteindre cette instruction sont acceptés. Cela nécessite en particulier que la taille de la pile à une instruction donnée soit la même quelque soit le chemin menant à cette instruction.

5.4. Modélisation du vérifieur

Un vérifieur de byte code comprend deux parties relativement distinctes que sont la vérification de structure et la vérification de type. Ces deux parties sont distinctes par plusieurs aspects :

- d'un point de vue purement fonctionnel, ce sont les deux étapes successives de la vérification qui pourraient dans l'absolu être totalement séparées,
- d'un point de vue algorithmique, l'une est découpée en douze composants à traiter séparément, chaque composant nécessitant une analyse syntaxique d'un flot d'octets ; l'autre est constituée d'un traitement linéaire d'un ensemble de byte codes,
- du point de vue de la modélisation les deux vérifieurs diffèrent dans l'utilisation qui est fait de la méthode B.

Nous avons construit un modèle unique pour le vérifieur puisque dans notre architecture, le vérifieur de type s'appuie sur le vérifieur de structure aussi bien pour lui fournir des propriétés que des services dont il a besoin.

Le vérifieur de type est complètement modélisé en B sauf en ce qui concerne les allocations mémoire. Pour les accès aux composants, une interface contenant un modèle nécessaire au vérifieur de type a été modélisée. Cette interface est ensuite raffinée et complétée pour fournir non seulement les services au vérifieur de type mais aussi spécifier les tests du vérifieur de structure. Cette seconde partie n'est pas modélisée intégralement en B. Ceci est dû au fait que comme la vérification de structure comporte une vérification syntaxique d'un flot d'octets, il est difficile d'en fournir une représentation abstraite. Certains composants ont été complètement modélisés jusqu'à une interface qui permet de lire un octet dans un fichier. Nous montrons la faisabilité d'une modélisation s'appuyant sur des briques de base de très bas niveau. Certains composants n'ont pas été modélisés et ont été implémentés directement en C. Par contre, la partie concernant certains tests internes et la totalité des tests externes a été modélisée. Les tests internes font partie intégrante de la vérification de structure, les tests externes suivent le même schéma de raffinement que le vérifieur de type.

Le vérifieur de type : Le vérifieur de type est modélisé intégralement en B. Il est composé d'un modèle abstrait raffiné par un modèle concret. Le modèle abstrait comprend les boucles de haut niveau et la spécification complète de chaque test à effectuer sur chaque byte code. Le modèle concret l'implémente en s'appuyant sur les services fournis par le vérifieur de structure et une machine de base modélisant la mémoire volatile de travail (RAM).

Le vérifieur de structure : le vérifieur de structure ayant pour premier objectif, d'assurer l'analyse syntaxique des composants, nous avons mis en place une architecture où chaque composant est modélisé par une machine. Cette machine décrit les propriétés et les services d'accès à ce composant ainsi que les tests internes de ce composant.

La modélisation des douze composants du fichier CAP suit ainsi le même modèle. Il contient un ensemble de variables abstraites modélisant les informations contenues dans le composant, un invariant qui définit les propriétés des variables, une opération spécifique qui réalise l'analyse syntaxique et effectue les tests internes et des opérations de lecture permettant d'accéder aux informations dans le cas où l'analyse syntaxique aurait réussi. Pour ce faire, la machine contient une variable booléenne indiquant si les tests internes ont réussi, cette variable est initialisée à faux, est mise à jour par l'opération de test interne et est en pré condition des opérations de lecture.

L'interface du composant ne fournit que les propriétés et les variables nécessaires à la modélisation du vérifieur de type et des tests externes. Lors du raffinement de cette machine, la spécification est complétée pour permettre de décrire complètement le composant et les tests liés aux informations qu'il contient. Le raffinement se fait par collages successifs pour terminer sur une représentation élémentaire du fichier, c'est-à-dire une fonction qui associe à une adresse dans la mémoire, l'octet présent à cette adresse. Ici on ne peut pas parler de modèle concret, ni de modèle abstrait puisque la chaîne de raffinement avec les collages successifs vers une représentation concrète constitue la spécification du composant.

Modélisation : Nous n'entrerons pas dans le détail de la modélisation ni du vérifieur de structure ni du vérifieur de type qui est assez conséquente. Nous nous intéresserons juste à la partie formalisation. En effet, la plupart des erreurs rencontrées au cours de ce développement sont venue d'une mauvaise interprétation de la spécification informelle.

La méthodologie utilisée est traditionnelle pour les utilisateurs de la méthode B, elle consiste à réécrire la spécification informelle dans un format proche de B. Ainsi pour l'instruction *aaload*, le sommet de pile doit contenir une référence sur un tableau (*arrayref*) et un index dans cette même table (*index*). Après l'exécution de l'instruction *aaload*, les deux éléments précédents sont enlevés de la pile et remplacés par la valeur du tableau à l'index donné. L'inconvénient avec toute spécification informelle c'est que les informations essentielles sont souvent éclatées dans le document. Par exemple pour l'instruction *aaload*, le contenu de la pile n'explicite pas le typage requis ou d'une manière incomplète. La figure suivante donne *in extenso* la définition informelle de Sun.

Aaload

Load reference from array

Stack

…, *arrayref*, *index* ⇒

…, value

Description

The *arrayref* must be of type reference and must refer to an array whose components are of type reference. The index must be of type short. Both *arrayref* and index are popped from the operand stack. The reference value in the component of the array at index is retrieved and pushed onto the top of the operand stack.

Runtime If *arrayref* is null, *aaload* throws a *NullPointerException*. **Exceptions** Otherwise, if index is not within the bounds of the array referenced by *arrayref*, the *aaload* instruction throws an *ArrayIndexOutOfBoundsException*

Figure 1 Spécification informelle de l'instruction ***aaload***

La réécriture définit quels types sont attendus en entrée par l'instruction et comment cette même instruction modifie les types dans la pile. La figure suivante indique dans la première partie la sémantique opérationnelle de l'instruction. Il doit y avoir au moins deux éléments sur la pile pour que l'instruction puisse s'exécuter. Si la pile contient une référence sur un tableau d'objets et un *short*, alors l'instruction *aaload* enlève ces deux éléments de la pile et retourne le contenu du tableau à l'index indiqué par le short. Dans le second cas, la pile contient une référence *null* et un *short*. Dans ce dernier cas, l'instruction *aaload* retourne une référence *null*. Si la pile contient d'autres séquences de types que les deux précédentes, l'instruction ne peut pas s'exécuter et une erreur est détectée. La seconde partie de la spécification concerne la sémantique statique c'est-à-dire les tests qui doivent être vérifiés pour permettre l'exécution de l'instruction. Ces tests vérifient qu'il y a bien au moins deux éléments dans la pile et que ces deux éléments sont du type attendu, en accord avec la sémantique statique de l'instruction *aaload*.

La troisième partie décrit les modifications effectuées par l'instruction elle-même. Ces modifications concernent l'évolution de la pile et/ou des variables locales. Dans l'exemple suivant, les deux éléments se trouvant au sommet de la pile sont supprimés de la pile et un nouvel élément est ajouté au sommet de la pile. Le type de ce dernier élément est fonction du type du second élément dans la pile, avant l'exécution de l'instruction. Notons que dans ce cas précis, les variables locales ne sont pas modifiées. Cependant, d'autres instructions du langage les modifient. La quatrième partie de notre format de spécification contient les tests qui permettent d'assurer les propriétés en sortie de l'instruction. Dans notre exemple, il n'y a pas de tels tests de post-modification. Enfin, la cinquième et dernière partie indique les catégories d'exceptions qui peuvent être lancées par la dite instruction.

aaload
[…, refarray class, short] => […, ref class]
[…, null, short] => […, null]
Pre-modification tests:
1. The stack must contain at least two elements
   (1)
2. The two topmost elements of the stack have to be of types compatibles with refarray
   (2)

class and short.
Modifications: ***(3)***
The two topmost elements of the stack are removed. If the second element was a refarray type, then a reference of the same class is pushed onto the stack. Otherwise a type null is pushed.
Post-modification tests:
None
Throws ***(4)***
- NullPointerException
- ArrayOutOfBoundException

- SecurityException

*Figure 2 Réécriture de l'instruction **aaload***

Chaque instruction du langage Java Card est réécrite suivant ce format. Ce document devient le document de référence pour la modélisation. Ainsi, à partir de la description informelle, nous obtenons la spécification formelle du vérifieur de type.

Il est désormais envisageable de réaliser la formalisation de cette spécification en plusieurs étapes de raffinement jusqu'à l'obtention du code exécutable. Cette méthodologie de raffinement est celle utilisée pour le développement B. La traduction de l'informel vers le formel est une étape cruciale car l'obtention du code et la construction des preuves se fait à partir de cette spécification formelle. Ainsi, si une erreur se glisse dans la traduction, la phase de preuve peut être dans l'incapacité de la détecter. Nous pensons en particulier aux définitions des types attendus par les instructions. Ces définitions sont des postulats de base dans la modélisation. Une erreur dans leur écriture et au final, le vérifieur acceptera ou rejettera des programmes respectivement faux ou corrects, ce qui n'est pas le comportement attendu du vérifieur.

La figure suivante montre la modélisation en B de l'instruction *aaload*. Le cheminement du modèle abstrait suit notre réécriture de la spécification ainsi, ***(1)*** indique qu'il doit y avoir au moins deux éléments sur la pile. ***(2)*** montre que l'élément au sommet de la pile doit être un *short* et que le deuxième élément à partir du sommet de la pile doit être soit une référence sur un tableau, soit un *null*. Nous devons différencier ces deux derniers cas, car des traitements différents sont attendus pour chacun des cas. Parce que l'instruction *aaload* a deux traitements différents, la clause B **SELECT … THEN … WHEN … THEN … ELSE … END** est la plus adaptée à cette situation. Cette clause permet de spécifier une action gardée par une condition. Dans notre cas, tous les **SELECT** sont déterministes et la spécification globale est déterministe. L'opération retourne un résultat : une valeur est donnée à la variable contenant le résultat dans chaque cas. De plus, la pile n'est modifiée que dans les deux cas corrects acceptés par l'instruction : deux éléments sont supprimés de la pile et un nouveau est ajouté.

Dans le **SELECT** et dans le **WHEN**, le développeur indique les pré-conditions qui permettent l'exécution de l'instruction. Dans notre exemple, les pré-conditions indiquent qu'il doit y avoir au moins deux éléments sur la pile, que l'élément au sommet de la pile est un *short* et que le second élément à partir du sommet de la pile est soit une référence sur un tableau d'objets, soit *null*. Dans les clauses **SELECT** et **WHEN**, nous ajoutons les tests pour le traitement des exceptions ***(4)***. En fait, il s'agit d'une condition additionnelle qui vise à vérifier que si l'instruction est dans un handler d'exception, nous devons alors vérifier que pour tous les labels de preuve donnés pour chaque handler, les types des variables locales sont compatibles avec les types contenus dans les descripteurs des variables locales associés aux labels et qu'il n'y a pas de références non initialisées dans les variables locales. Cette condition additionnelle vient du fait que certaines instructions peuvent lancer des exceptions. Il faut donc tenir compte de ce comportement exceptionnel lors de leur modélisation puis de leur vérification. Ces conditions apparaissent en fait dans la clause THROWS de la spécification informelle. Dans cette clause, les exceptions que peut lancer l'instruction considérée sont énumérées.

Enfin, dans les clauses **THEN** et **ELSE**, nous donnons le comportement de l'instruction. Dans les deux cas corrects, la variable retournée par l'opération est mise à vrai (**TRUE**), indiquant que la vérification s'est bien passée. De plus, la pile est modifiée suivant la branche dans laquelle nous nous trouvons. Ainsi, dans les deux cas, deux éléments sont

supprimés de la pile. Ensuite, dans un cas, une référence est ajoutée sur la pile et dans l'autre cas un *null* est ajouté sur la pile. Le troisième et dernier cas correspond au cas d'erreur : il n'y a pas assez d'éléments sur la pile, le typage des éléments est incorrect ou alors la vérification du handler d'exception n'est pas correcte. L'instruction *aaload* retourne alors faux (**FALSE**) et ne modifie ni la pile, ni les variables locales. L'information qu'une erreur a été découverte est propagée et le processus de vérification est arrêté. Dans les cas où tout s'est bien passé, le processus de vérification continue par la vérification de l'instruction suivante.

```
bb ← verify_aaload =
BEGIN
   SELECT
        2 ≤ size(stack) ∧                                                    (1)
        last(stack) = c_short ∧                                              (2)
        last(front(stack)) = c_refarray ∧
        (pc ∈ dom(exception_handler)                                         (4)
         ⇒
         ∀label.(label ∈ exception_handler(pc)
              ⇒ COMPATIBLE(loc_var, loc_var_descriptor(label))) ∧
         c_uref ∉ ran(loc_var))
   THEN
        bb    := TRUE ||
        stack := front(front(stack)) ← c_ref                                 (3)
   WHEN
        2 ≤ size(stack) ∧                                                    (1)
        last(stack) = c_short ∧                                              (2)
        last(front(stack)) = c_null ∧
        (pc ∈ dom(exception_handler)                                         (4)
         ⇒
         ∀label.(label ∈ exception_handler(pc)
              ⇒ COMPATIBLE(loc_var, loc_var_descriptor(label))) ∧
         c_uref ∉ ran(loc_var))
   THEN
        bb    := TRUE ||
        stack := front(front(stack)) ←c_nul                                  (3)
   ELSE
        bb := FALSE
   END
END
```

*Figure3 Modélisation abstraite de l'instruction **aaload***

Le modèle formel du vérifieur de byte code embarqué est destiné à produire une implémentation formelle en C. Dans notre méthodologie de développement, il nous a fallu prendre en compte non seulement le développement formel, et en particulier le passage des besoins informels aux spécifications formelles mais également sa traduction et son intégration sur une plate-forme à fortes contraintes.

Ainsi si le développement formel à l'aide de la méthode B permet d'obtenir une implémentation formelle en B0, celle-ci doit encore être traduite en C puis compilé pour le processeur cible de la carte à puce. Les processus de traduction et de compilation ne sont pas des processus qui ont été formalisés. Si la compilation a été qualifiée, ce n'est pas le cas de la traduction pour laquelle nous avons développé notre propre traducteur.

Pour ces raisons, nous ne pouvons entièrement nous reposer sur le développement formel pour assurer la correction du vérifieur. Il faut utiliser d'autres techniques. La figure suivante décrit les différentes phases du développement, de l'expression des besoins fonctionnels du vérifieur au code embarqué de ce même vérifieur.

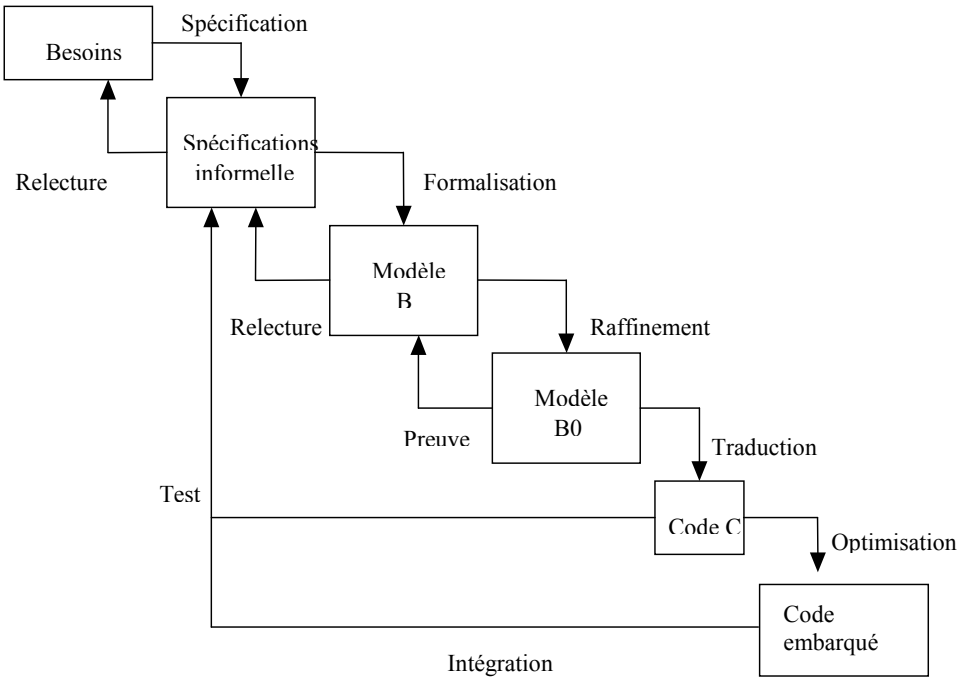


Figure 4 Cycle de développement complet

Le développement formel à lui tout seul n'assure pas une totale correspondance entre les besoins informels et le code produit. Cela est dû principalement aux activités en amont et en aval du processus de développement formel qui restent des activités manuelles. Il faut donc, en particulier dans une activité industrielle, tenir compte de cet aspect et compléter le développement formel par d'autres techniques comme le test et la relecture de code.

La relecture des spécifications formelles écrites en B permet de s'assurer manuellement que la traduction des besoins informels en spécifications formelles s'est faite correctement. C'est une étape importante car par la suite, le développement formel s'appuie sur cette traduction. Une erreur à ce niveau peut donc avoir des répercussions sur tout le développement.

Pour s'assurer que la traduction de l'implémentation formelle vers du code C puis vers du code compilé pour le processeur cible est correcte, nous ajoutons une phase de test. En fait, les tests peuvent être effectués à deux moments différents. Le premier se situe après la génération du code C issu de l'implémentation formelle. Le second sur le code installé dans la carte à puce. Cela permet de tester différentes parties du processus de développement comme la phase traduction et la phase d'intégration. Notons que les cas de tests ne sont pas générés à partir des spécifications formelles mais de l'expression des besoins fonctionnels. Cela nous permet en plus, de nous assurer de la conformité du code généré par rapport aux spécifications informelles.

L'intégration des parties de code développées formellement avec des parties de codes classiques est un point très important car il est illusoire d'imager développer formellement l'ensemble d'un système. Il faut d'abord modéliser les interactions entre ces deux développements. L'avantage d'utiliser B est la facilité avec laquelle nous pouvons utiliser les machines de base pour réaliser cette interface entre le développement formel et informel. Ces machines contiennent la spécification abstraite des blocs de base, c'est-à-dire la liste des variables, de leurs propriétés et de leurs services. Cette spécification abstraite est alors utilisée par les différents modèles du vérifieur de type et de structure. Cependant, ces machines de bases ne sont pas raffinées suivant la méthodologie B. En fait, une implémentation directe est fournie (en C ou en un autre langage). Il n'y a alors pas de preuve que les implémentations de ces machines de base correspondent effectivement à leurs spécifications. Cependant, ces machines de bases représentent des blocs logiciels très précis dont les implémentations ont déjà fait l'objet de la plus grande attention comme la gestion de la mémoire ou de la communication. Ces blocs ne sont de toutes les façons pas développables à l'aide de la méthode B. Nous assurons la correspondance par un processus classique la relecture de code.

L'un des principaux avantages de l'utilisation de la méthode B est la génération automatique de code qui conclut la phase d'implémentation. En fait, à la fin du processus de raffinement, la machine B obtenue est une implémentation écrite dans un sous-ensemble du langage B, le B0. Pour aborder cette génération de code, nous avons conçu un traducteur de code rudimentaire qui ne prend en compte que les implémentations en B0 du modèle formel. Ce qui n'est pas le cas de traducteurs comme celui développé par ClearSy. Pour une traduction plus simple, nous ajoutons les types des variables sous la forme d'assertions dans le code B0. Cela permet au traducteur de choisir le meilleur type C pour la variable à traduire. Cette simple considération nous permet en particulier de restreindre l'espace mémoire d'une variable à son strict nécessaire.

Plan d'évaluation

Les techniques d'évitement de faute par construction correcte ont un grand impact dans la communauté scientifique mais reçoivent assez peu d'attention de la part du monde industriel. Le seul cas où les industriels adoptent ces technologies, c'est sous la contrainte d'une certification, et les techniques formelles sont plus vues comme un coût supplémentaire plutôt qu'un moyen d'améliorer la qualité du logiciel. Il faut donc convaincre que ces technologies sont à même de passer l'échelle mais aussi sont compatibles avec les contraintes économiques rencontrées par les industriels. Nous avons donc évalué ce processus de développement en soumettant plusieurs hypothèses que nous cherchons à invalider. Pour ce faire, il nous faut disposer d'une base d'informations quantitatives sur le développement et pouvoir le comparer avec un développement traditionnel.

Ces hypothèses mesurent l'apport de la modélisation si elles ne sont pas invalidées. Nous définissons trois hypothèses pour évaluer l'impact et le résultat d'un développement formel sur un cas industriel :

- Hypothèse 1 : l'utilisation des méthodes formelles, et dans notre cas précis, l'utilisation de la méthode B, améliore la qualité du logiciel résultant de la modélisation.
- Hypothèse 2 : le surcoût d'un développement formel est acceptable.
- Hypothèse 3 : Le code généré par l'application de la méthode B n'entraîne pas une augmentation significative de l'occupation mémoire ou du temps d'exécution.

Les deux premières hypothèses nécessitent de pouvoir comparer un développement formel et un développement conventionnel. Nous devons mener ces deux développements par des équipes séparées et collecter les informations concernant les coûts et les erreurs trouvées dans les différentes étapes du développement.

Trois étapes ont été identifiées lors du développement :

- La translation de la spécification informelle vers un modèle formel abstrait, la vérification est faite par une revue de code,
- Le développement formel dont la validation est faite par raffinement et preuves,
- La translation de la spécification formelle détaillée vers un code exécutable, la vérification est faite par de la relecture et du test.

Durant la première phase les erreurs sont découvertes par la une revue effectuée par une personne externe au groupe de développeur. Le processus de revue est le premier moyen de s'assurer de correction de la formalisation, l'autre est le test. L'avantage de la revue sur le test est d'arriver plus tôt dans le cycle de développement et donc évite de perdre du temps dans une activité de preuve interactive. Durant cette phase 12 erreurs ont été relevées. La seconde phase est liée au processus de développement B. A chaque raffinement, le générateur d'obligation de preuve propose les lemmes liés au raffinement. Dans la méthodologie utilisée, nous ne réalisons que la preuve automatique suivie d'une revue des lemmes restants. Si ceux ci indiquent une erreur de raffinement alors cette erreur est corrigée. Mais nous ne réalisons pas de preuve interactive. Lorsque le dernier raffinement est atteint et que toutes les structures de données sont implémentables alors nous reprenons l'ensemble de lemmes avec le prouveur interactif. Certains nous avaient semblés justes alors qu'ils étaient erronés. Le processus de preuve peut alors être vu comme un outil de mise au point extrêmement puissant qui nous a permis de découvrir 29 lemmes faux. La dernière étape est la validation par le test. Cette étape nous assure dans une certaine limite que le passage du langage formel de bas niveau au langage exécutable est correct ainsi que le passage de la spécification informelle vers le modèle abstrait de notre système. Durant cette phase, 32 erreurs ont été découvertes liées à la formalisation (14) et au traducteur de code (9) et à des parties non modélisée (9).

Le premier tableau synthétise les métriques concernant le développement. En particulier, il apparaît que le vérifieur de structure est plus gros que le vérifieur de type. Ceci est dû au fait que le vérifieur de structure contient de nombreux tests, très différents. Chacun de ces tests nécessite une spécification et une implémentation, ce qui a pour conséquence d'augmenter la taille des modèles et du code C généré par la suite. Le vérifieur de type, quant à lui, peut être vu comme une seule entité incluant les règles de typage définies par le langage Java Card. Ainsi, il est possible de factoriser le code et d'éviter une expansion des modèles B et du code C généré. De plus, le vérifieur de structure contient des services d'accès aux données du fichier CAP qui sont utilisés par le vérifieur de type pour accéder notamment aux instructions des méthodes, aux définitions des classes et au type des champs. Cela explique également les différences apparaissant sur les nombres de composants B : le vérifieur de structure contient beaucoup plus de composants que le vérifieur de type.

	Structure	Typage	Utilitaires	Total
Nombre de lignes de B	35000	20000	3500	58500
Nombre de composants B	116	34	45	195

Nombre de POs générées	11700	18160	950	30810
Pourcentage de POs automatiquement prouvées	81 %	70 %	77 %	74 %
Nombre de machines de base utilisées	6	0	7	13
Nombre de lignes de C	7540	4250	860	12650

Figure 5 *Métriques sur le développement formel : complexité*

Ce tableau présente deux résultats remarquables : le premier concerne le nombre de lemmes ou obligations de preuve (POs). Les résultats montrent que le vérifieur de type génère bien plus de POs que le vérifieur de structure. La principale raison est le traitement par cas effectué par le vérifieur de type produisant ainsi un nombre considérable de lemmes. Le deuxième résultat concerne le nombre de lignes de C. Ce nombre est bien en dessous du nombre de lignes de B des modèles du vérifieur. Ce résultat vient du fait que la traduction du B vers du C ne prend en compte que les implémentations. Ainsi, les machines abstraites et les raffinements ne sont pas utilisés lors de la traduction. De plus, les clauses INVARIANT des boucles dans les implémentations ne sont pas traduites. Cela réduit drastiquement le nombre de lignes de B qui seront traduites en C.

La tableau suivant représente les résultats de l'exécution du vérifieur de byte code sur quelques applets. Ce sont des applets concrètes qui correspondent à des applications industrielles réelles déjà déployées sur les cartes à puce. La colonne *taille* contient la taille en octets des applets qui sont envoyées au vérifieur. Les colonnes *structure* et *type* contiennent le temps nécessaire pour exécuter chacune des phases de la vérification.

Le premier commentaire qui peut être extrait de ces résultats est que l'exécution du vérifieur de type est plus coûteuse en temps que celle du vérifieur de structure. En fait, le vérifieur de structure est une succession de tests. Certains sont simples, d'autres sont plus complexes, mais ce sont des vérifications de cohérence à effectuer en général une seule fois. Au contraire, le vérifieur de type effectue des tests pour chaque instruction. Afin de privilégier la faible consommation de mémoire volatile (RAM), il a été choisi de ne pas recourir au stockage temporaire dans cette mémoire de données dont le vérifieur de type peut souvent avoir besoin. En effet, pour réduire les besoins en mémoire, tant en RAM qu'en EEPROM, le code doit être parcouru à chaque fois qu'une information est demandée.

Applet	Taille (octets)	Vérifieur de Structure	Vérifieur de Type
Utils	4439	230 ms	1422 ms
pacapint	2375	50 ms	110 ms
Wallet	2762	100 ms	460 ms
TicTacToe	4988	130 ms	1372 ms
Applet 1	10394	611 ms	26 856 ms
Applet 2	22554	1161 ms	11 506 ms

Figure 6 *Métriques sur le développement formel : performance du logiciel*

La contrepartie évidente d'une telle approche est l'augmentation du temps d'exécution. Même si l'algorithme de vérification est linéaire, le temps nécessaire à la vérification des règles de typage est supérieur au temps de vérification de la cohérence des données contenues dans le fichier CAP. Si l'applet 1 nécessite près de 28 secondes pour être vérifiée, c'est parce que cette applet contient de nombreuses exceptions et que la vérification des exceptions est coûteuse en parcours de code. Une façon d'optimiser est d'essayer de faciliter l'accès aux données en stockant des données temporaires en mémoire volatile (RAM). Un compromis doit être trouvé entre l'utilisation de RAM et d'EEPROM et le temps d'exécution.

	Développement formel	Développement conventionnel
Coût du développement	12 semaines	12 semaines
Coût de la preuve	6 semaines	NA
Coût du test	1 semaines	3 semaines
Intégration	1 semaines	2 semaines
Total	**20** semaines	**17** semaines
Nombre de fautes détectées par la relecture	13	24
Nombre de fautes détectées par la preuve	29	-
Nombre de fautes détectées par le test	32	71
Total	**74**	**95**

Figure 7 *Métriques sur le développement formel : coût et qualité*

Ces métriques indiquent que la première hypothèse est vraie. Pour la seconde, il existe une surcharge liée à la méthodologie cependant nous ne prenons pas en compte le cycle de vie complet du produit. En effet les retours clients ne sont pas pris en compte ici. Probablement, il existe plus d'erreurs résiduelles dans le développement conventionnel que dans le développement formel. Les métriques (non-fournies ici) concernant la taille et les performances montrent que le code formel est sensiblement plus gros, mais les performances sont équivalentes.

Résultats : Le résultat majeur de ce travail d'évaluation est la démonstration de la non-invalidation des hypothèses et donc une grande confiance sur l'amélioration de la qualité du logiciel à un coût raisonnable. Nous avons aussi montré l'intérêt pour un industriel d'inclure dans son processus de développement de logiciels des méthodes formelles. En particulier, nous avons mis l'accent sur la conformité du code produit avec la spécification initiale.

Mais la comparaison relève aussi des manques. Le premier, et le plus marquant d'entre eux, concerne l'activité de preuve. Cette activité est clairement identifiée comme la partie coûteuse du développement formel. Elle révèle une tendance à risque liée à des outils et une méthodologie encore en cours d'évolution et d'amélioration. Cette méthodologie met en avant le comportement que doit avoir une équipe de développement face à la preuve, notamment quant à son organisation et à sa gestion :

- le découpage de la phase de preuve qui passe d'abord par une étape de correction du modèle et d'analyse des lemmes générés avant de se lancer dans la preuve des lemmes les uns après les autres,
- le développement de règles de preuve, un investissement et un coût en temps important sur les premiers développements, et donc plus encore sur le premier, mais sur lequel nous pouvons capitaliser.

Cette étude est encourageante quant à l'intérêt des méthodes formelles dans le domaine des cartes à puce. Pour parvenir aux résultats, il n'a pas été besoin d'optimiser totalement le prototype. Ainsi, pour améliorer le vérifieur, quelques optimisations ont déjà été identifiées : l'amélioration du modèle mémoire, l'élimination des tests redondant dans le vérifieur structurel, l'amélioration du traducteur de code pour prendre en compte des optimisations spécifiques. Enfin, la comparaison et l'analyse de la phase de preuve soulignent la nécessité d'améliorer les outils. Pour accompagner la méthodologie, les outils doivent subir des évolutions. Des améliorations sur la gestion des lemmes, de leurs hypothèses et de leurs preuves peuvent faciliter la phase de développement et de démonstration des lemmes, et avoir ainsi un impact significatif sur le développement en réduisant sa durée.

En utilisant une méthodologie de développement formel nous avons fourni une implémentation adaptée pour carte à puce, issue d'un modèle formel. La modélisation doit être utilisée avec précaution car les outils et les méthodes demandent encore des améliorations. Les parties du système où l'utilisation des méthodes formelles est nécessaire doivent être clairement identifiées. Ainsi, les interfaces entre les parties prouvées et non prouvées sont cruciales pour le système lui-même et doivent être définies avec le plus grand soin. Cette interaction entre les parties prouvées et non prouvées permet de réduire les coûts de développement, de réutiliser du code déjà développé et de réduire ainsi les risques tout en augmentant la confiance que nous pouvons mettre dans du code de plus en plus complexe.

5.5. Conclusions

Les techniques d'évitement de faute par construction correcte ont un grand impact dans la communauté scientifique mais reçoivent assez peu d'attention de la part du monde industriel. Le seul cas où les industriels adoptent ces technologies, c'est sous la contrainte d'une certification, et les techniques formelles sont plus vues comme un coût supplémentaire plutôt qu'un moyen d'améliorer la qualité du logiciel. Or ces technologies sont à même de passer l'échelle, mais elles sont aussi compatibles avec les contraintes économiques rencontrées par les industriels ce que nous démontrons en disposant d'une base d'informations quantitatives sur le développement et en le comparant avec un développement traditionnel.

Ces travaux sur les techniques d'évitement de faute par construction correcte de logiciel nous ont permis d'aborder un problème industriel en proposant une spécification complète d'un composant de sécurité d'une Java Card. Nous avons raffiné un tel composant pour obtenir un code exécutable sur une carte à puce en démontrant que ce code est conforme à la spécification initiale, relevant ainsi deux défis : la réalisation d'un vérifieur de byte code dans une carte à puce et la preuve de la correction de ce dernier.

Ajouter un vérifieur de byte code dans une Java Card permet à la carte d'assurer elle-même sa propre sécurité. L'indépendance de la carte vis-à-vis de sa propre sécurité permet entre autre de réduire les coûts d'architecture ainsi que le temps de déploiement. En effet, aujourd'hui, déployer de nouvelles applications sur une carte déjà dans les mains de l'utilisateur final requiert une importante infrastructure qui implique des centres de certification et des protocoles de cryptographie lourds et coûteux. Avec un vérifieur embarqué, l'infrastructure est plus réduite : il suffit d'envoyer l'application à la carte qui se charge elle-même de la vérifier et assure ainsi sa propre sécurité.

Le processus de preuve appliqué durant le développement formel du vérifieur de byte code assure sa correction. L'utilisation des méthodes formelles n'est pas sans impact. Un

compromis doit être trouvé pour obtenir la meilleure confiance dans le code pour le meilleur coût. Par exemple, la formalisation de chaque composant du fichier CAP n'est pas nécessaire. L'apport de la méthode B est marginal pour ce type de développement. Il faut se concentrer sur d'autres points comme le vérifieur de type où les apports des méthodes formelles sont plus importants.

Nous avons mis l'accent sur l'aspect méthodologique afin de fournir des données quantitatives sur le processus de développement formel afin d'aider des responsables de projets de choisir cette technologie. Nous avons aussi une méthodologie de développement adapté au domaine de la carte à puce afin de rendre un tel développement acceptable en terme de coût sans obérer la viabilité de la chaîne de confiance.

Il est à noter que le consortium GlobalPlatform[13] définissant les règles de sécurité gérant le chargement d'application et le cycle de vie de la carte, a modélisé complètement la spécification Card Specification v2.1.1 afin d'améliorer la qualité de la spécification. Ce travail était nécessaire au vu de la complexité de l'ensemble des règles de sécurité. La modélisation suivie de la phase de preuve a éliminé un certain nombre d'ambiguïtés dans la spécification ce qui montre l'intérêt des industriels à se prémunir contre les vulnérabilités des systèmes embarqués de confiance.

[13] www.globalplatform.org

6. Biométrie

(Cette partie est traitée par Claude BARRAL)

6.1. Définitions

La biométrie est un procédé pouvant être automatisé qui permet de reconnaître une personne sur la base d'une donnée physiologique ou comportementale.

Il existe trois moyens de prouver son identité, en utilisant :

- Quelque chose que l'on possède (pièce d'identité, badges etc…)
- Quelque chose que l'on sait (mot de passe, code PIN etc…)
- Quelque chose que l'on est (empreinte digitale, visage, œil, ADN etc…)

Ce troisième facteur d'identification est la biométrie.

La procédé biométrique consiste à comparer un échantillon candidat avec un échantillon de référence afin de déterminer s'il y a identité ou non entre ces deux échantillons.

Un système biométrique dit « multi-modal » utilise au minimum deux techniques biométriques différentes.

6.2. Petit historique

Dans l'antiquité, les Chinois et les Assyriens utilisaient déjà l'empreinte digitale pour authentifier des documents légaux.

L'utilisation d'éléments biométriques, à des fins d'authentification de personnes, date de la fin du 19ème siècle / début du 20ème siècle avec les travaux d'Alphonse Bertillon (anthropométrie criminelle) et Sir Francis Galton (identification criminelle, empreintes digitales).

6.3. Utilisation

L'utilisation de la biométrie peut avoir deux intérêts majeurs :

- Sécurité

Identification criminelle ou civile (lutte contre la multiplication d'identités), authentification du porteur d'un document légal, contrôle d'accès physique ou logique.

- Confort de l'utilisateur

Remplacement des badges, des mots de passe par quelque chose qui ne peut ni être perdue, ni être oubliée.

La biométrie peut être utilisée dans les contextes suivants : passeports, visas, cartes d'identité, permis de conduire, carte santé, distribution des aides sociales, contrôle au frontière, demandes d'asile, criminalité, recherche en paternité, enfants disparus, identification de terroristes, accès logiques (login : PC, réseau, mobiles GSM, PDA)…

La première application civile à grande échelle est le passeport électronique. Une norme internationale a été établie stipulant l'utilisation d'une puce électronique sans-contact qui devra stocker une photo numérique à des fins de reconnaissance faciale (obligatoire), empreinte digitale (optionnel, mais obligatoire en Europe) et iris (optionnel).

6.4. Les différentes modalités

Les « modalités » sont les différents types de techniques utilisées à des fins de reconnaissance biométrique. Ceci est séparé en deux grandes familles :

- Les données physiologiques
  - Empreinte digitale
  - Visage
  - Œil (iris, rétine)
  - Main (géométrie, réseau veineux, empreinte palmaire)
  - Forme de l'oreille
  - Voix
  - ADN (ou traces biologiques : salive, sang…)
  - …
- Les données comportementales
  - Signature
  - Frappe au clavier
  - Voix
  - Démarche
  - Mouvement des lèvres
  - …

La voix est effectivement une modalité pouvant tout aussi bien être traitée de manières physiologiques (analyse fréquentielle liée aux cordes vocales) que de manière comportementale (analyse du séquentielle de la parole)

Les données comportementales sont évidemment moins stables que les données physiologiques car facilement sujettes à modifications dues au stress ou à l'état de santé par exemple.

6.5. Identification et Vérification

L'**identification** est un procédé de reconnaissance dit «1 à n » : un échantillon n'ayant pas d'identité va être comparé à n échantillons de référence dans une base de données afin de lui attribuer une identité. Ceci est utilisé par exemple en sciences criminelles.

Identification = déterminer une identité, si connue dans la base de données

La **vérification** (ou **authentification**) est un procédé de reconnaissance dit « 1 à 1 » : un échantillon candidat dont l'identité est connue va être comparé à un échantillon de référence afin de prouver cette identité. Ceci est utilisé par exemple dans le passeport électronique où l'empreinte digitale capturée au poste frontière va être comparée à l'empreinte digitale de référence stockée dans la puce, ceci afin de vérifier que le porteur du passeport est bien son propriétaire légal.

Vérification = prouver une identité (prouver que l'on est bien la personne que l'on prétend être)

6.6. Les critères d'évaluation

Il existe sept critères d'évaluation d'un système biométrique : l'universalité, l'unicité, la permanence, la facilité de capture, la performance, l'acceptabilité et la résistance au contournement. En règle générale, ces critères se définissent sur trois niveaux : élevé, moyen, faible.

6.6.1. Universalité

La donnée biométrique utilisée est-elle disponible chez chaque personne ?

Certain handicaps rendent des techniques biométriques inutilisable : mutisme (voix), amputations (démarche, empreinte digitale, main), cécité (œil). On parle alors d'universalité moyenne ou faible.

A contrario, le visage a un niveau d'universalité élevé.

La proportion de personnes ne pouvant être enregistrés dans un système est le taux d'échecs à l'enregistrement (FTE : Failure To Enrol)

6.6.2. Unicité

La donnée biométrique utilisée est-elle assez unique pour obtenir une bonne fiabilité du système ?

Beaucoup de voix se ressemblent, les vrais jumeaux et les sosies ont des visages similaires, la géométrie d'une main varie peu (une pomme et cinq doigt en général !).

A contrario, l'empreinte digitale qui est utilisée depuis un siècle dans le monde policier a prouvé un bon niveau d'unicité. L'iris est reconnu pour avoir un niveau d'unicité très élevé.

6.6.3. Permanence

La donnée biométrique utilisée est-elle stable dans le temps ? Les phases de croissance et de vieillissement influent-elles fortement ?

On sait que la voix et le visage évoluent rapidement. La signature, la démarche, la frappe au clavier changent sensiblement avec le temps.

A contrario, l'iris et l'empreinte digitale sont connus pour être stables.

6.6.4. Facilité de capture

La donnée biométrique utilisée est-elle facile à acquérir ?

- Un premier paramètre est la disponibilité et le coût du matériel nécessaire :

La frappe au clavier ne nécessite aucun périphérique particulier (mis à part le clavier !)

Le visage, la démarche, la forme de la main se contentent d'une simple camera, voire même webcam.

La capture d'une image d'iris nécessite une caméra plus complexe; l'empreinte digitale nécessite un capteur dédié; la signature nécessite une tablette graphique.

- Un deuxième paramètre est la facilité d'utilisation :

Frapper au clavier ou signer sont simples, mais poser correctement son doigt sur un capteur ou laisser scanner sa rétine peut-être pénible.

6.6.5. Performance

Les modèles mathématiques et leur implémentation en algorithmes informatiques sont-ils assez fiables et mûrs ?

Ici encore l'empreinte digitale, l'iris, l'ADN donnent de bons résultats.

Le visage, la voix, la démarche ont des performances beaucoup plus mitigées.

Ceci sera plus amplement détaillé dans la section « fiabilité : taux d'erreurs ».

6.6.6. Acceptabilité

La grand public acceptera t-il le type de biométrie utilisé dans tel ou tel système ?

On aborde ici LE critère le plus important. Le problème est principalement culturel : certains associent les empreintes digitales au monde criminel, certaines maladies contagieuses par contact (SRAS en Asie) sont un frein à l'utilisation de matériel commun (capteur d'empreinte, scanner de main), le scan d'un iris manque de vulgarisation : beaucoup pensent qu'un laser balaye l'œil et peut causer de légers dommages, l'iris peut révéler certaines maladies et autres prises de drogues, ce qui est contraire au principe de vie privée…

A contrario, l'utilisation d'une photo pour un document d'identité étant le plus répandu, cela fait du visage le type de biométrie de loin le plus acceptable actuellement dans tous les pays.

6.6.7. Résistance au contournement

Peut-on facilement contourner le système ?

Une voix est imitable, enregistrable ; une signature est imitable ; une photo peut suffire à simuler la présence d'un visage…

A contrario, l'iris et l'empreinte digitale sont encore une fois des modalités plus intéressantes car plus difficilement contournables.

6.6.8. En résumé…

	Universality	Uniqueness	Permanence	Collectability	Performance	Acceptability	Circumvention
Face	High	Low	Medium	High	Low	High	Low
Fingerprints	Medium	High	High	Medium	High	Medium	High
Hand Geometry	Medium	Medium	Medium	High	Medium	Medium	Medium
Keystroke	Low	Low	Low	Medium	Low	Medium	Medium
Hand Vein	Medium	Medium	Medium	Medium	Medium	Medium	High
Iris	High	High	High	Medium	High	Low	High
Retinal Scan	High	High	Medium	Low	High	Low	High
Signature	Low	Low	Low	High	Low	High	Low
Voice Print	Medium	Low	Low	Medium	Low	High	Low
Facial Thermograms	High	High	Low	High	Medium	High	High
Odor	High	High	High	Low	Low	Medium	Low
DNA	High	High	High	Low	High	Low	Low
Gait	Medium	Low	Low	High	Low	High	Medium
Ear	Medium	Medium	High	Medium	Medium	High	Medium

Ce tableau est tiré d'une étude réalisée aux Etats-Unis vers la fin des années 90. On peut en conclure que l'empreinte digitale et l'iris sont les meilleurs compromis entre les différents critères.

6.7. Principe de fonctionnement d'un système biométrique

Le système se présente en deux phases : l'enregistrement (ou « enrolment ») et la vérification.

- ***Enregistrement***

C'est la première et unique étape qui a pour but de créer la donnée biométrique de référence. Cette étape est primordiale et doit être effectuée avec soin (moyens techniques, temps) : de la qualité de la donnée de référence générée dépendra la fiabilité du système !

Le procédé d'enregistrement se décompose ainsi :

- Capture de la donnée biométrique (en général plusieurs captures de la même donnée)
- Extraction des éléments caractéristique de cette (ces) donnée(s) (réduction conséquente en taille en vue de stockage ultérieur)
- Génération d'une donnée de référence (ou « template de référence »)
- Comparaison de la donnée de référence avec une nouvelle capture
- Stockage de la données de référence si étape précédente OK (sur un serveur, une carte à puce, etc…)

L'avantage de la génération d'un template est la réduction en taille de la donnée à stocker et à échanger lors d'une vérification, ceci économise la base de donnée et la bande passante d'un réseau.

- ***Vérification***

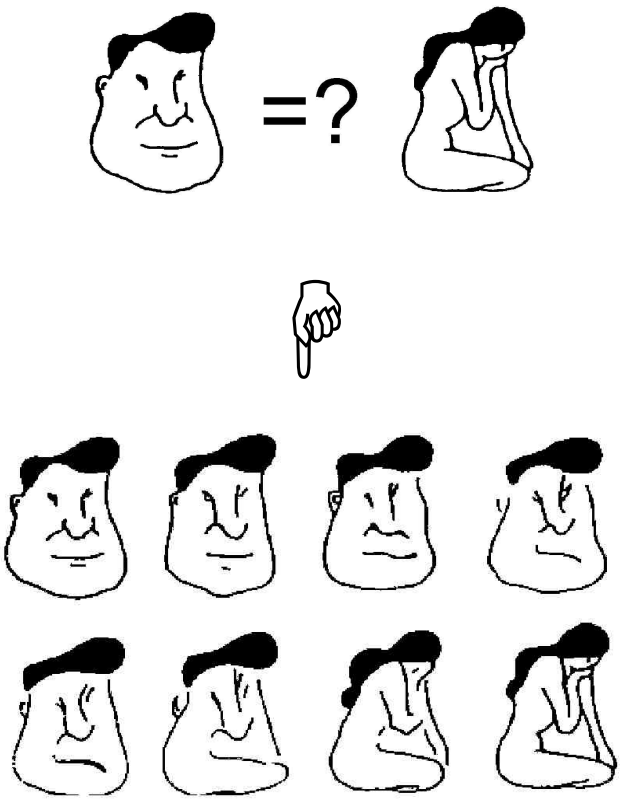
C'est l'étape récurrente, à chaque demande d'authentification, qui consiste à capturer la donnée biométrique candidate et à la comparer avec le template de référence pour déterminer s'il y a identité entre les deux échantillons.

Le procédé de vérification se décompose ainsi :

- Capture de la donnée biométrique de référence
- Extraction des éléments caractéristiques du candidat
- Lecture du template de référence
- Comparaison du template candidat avec le template de référence (« matching »)
- Décision (score 0 à 100% ou oui/non)

On distingue donc deux algorithmes indissociables de la biométrie : a/ l' « ***extraction*** » qui consiste à créer un template à partir d'une donnée brute b/ le « ***matching*** » qui consiste à comparer deux template.

Le matching est l'étape qui consiste à déterminer la « proximité » de deux données :



6.8. Fiabilité : Taux d'Erreurs

La comparaison de deux échantillons biométriques est un procédé statistique : à l'opposé d'une comparaison de mots de passe qui est déterministe (accepté si identique, rejeté sinon), chaque capture d'une même donnée biométrique est différente : on essai alors de déterminer un degré de similitude. Ceci amène à considérer deux taux d'erreurs :

- La fausse acceptation (faux positif) ou FAR (False Acceptance Rate)

C'est la probabilité d'accepter un intrus dans le système pensant qu'il s'agit d'une autre personne, elle, autorisée.

- Le faux rejet (faux négatif) ou FRR (False Rejection Rate)

C'est la probabilité de rejeter, au premier essai, une personne autorisée par le système. Un deuxième essai permet généralement de remédier au problème (la probabilité d'être également rejeté au deuxième essai, et aux suivants, est de plus en plus minime).

Pour une application donnée, la première étape consiste à déterminer le bon compromis entre FAR et FRR. Ces deux données évoluent de manière inversement proportionnelle : on fixe un seuil d'acceptation entre 0% et 100% de similitude, plus le seuil va vers 0% plus les fausses acceptations seront nombreuses, plus le seuil va vers 100% plus les faux rejets seront nombreux.

Il s'agit ici du compromis entre sécurité (FAR bas) et convivialité (FRR bas) du système.

Un système avec FAR 0% et FRR 100% refusera l'accès même à son utilisateur légitime mais reste donc hyper sécurisé !

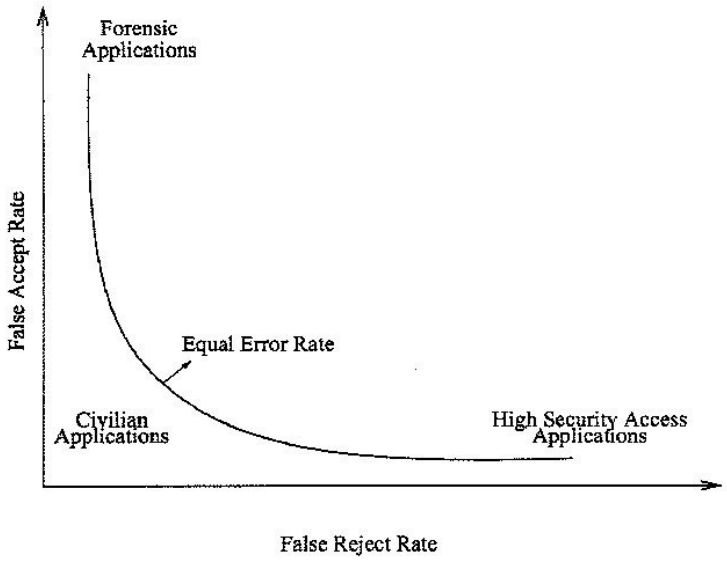
A des fins d'évaluation et test de performances des systèmes biométriques, on utilise souvent une valeur dite EER (Equal Error Rate), c'est la valeur pour laquelle FAR=FRR.

Cette valeur n'a pas lieu d'être prise comme référence dans un système réel car une application bien étudiée nécessitera de favoriser soit un FAR bas, soit un FRR bas.

Une application du type accès physique à une zone ultra sensible devra favoriser un FAR très bas : on ne veut surtout qu'un intrus puisse entrer, quitte à ce qu'une personne autorisée doivent procéder plusieurs fois à l'authentification.

Une application du type recherche criminelle devra favoriser un FRR très bas : on veut surtout ne pas « rater » le criminel, quitte à ce que plusieurs personnes puissent être suspectées et l'enquête affinée ensuite.

La fiabilité d'un système biométrique est déterminée à parts égales par la qualité du capteur, l'ergonomie de la capture et la performance des algorithmes.



6.9. Les capteurs

La plupart des données biométriques nécessitent une camera (plus ou moins sophistiquée) comme capteur : visage, iris, démarche, main, mouvement des lèvres, etc…

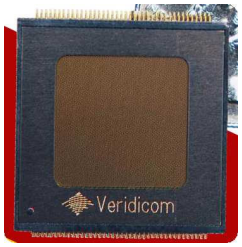
On considère également les capteurs suivant : microphones, claviers, tablettes graphiques et capteur d'empreintes digitales.

L'empreinte digitale nécessite un périphérique de capture dédié uniquement à cette fonction contrairement aux cameras, claviers et autres microphones. L'empreinte digitale représentant actuellement 50% du marché de la biométrie, on peut trouver beaucoup de sociétés proposant des capteurs basés sur différentes technologies.

Le premier « capteur » d'empreintes était le doigt encré déposé sur du papier. On distingue maintenant plusieurs type de technologie : optique (Digital Persona, Morpho Sagem…), pression (BMF…), ultrason (Stockburger…) et silicium.

Les capteurs optiques et silicium sont les plus répandus. On les trouve sous deux formats : le pavé de quelques cm² sur lequel on pose simplement le doigt, (PAD) , la barrette sur laquelle on déplace son doigt (Swipe)

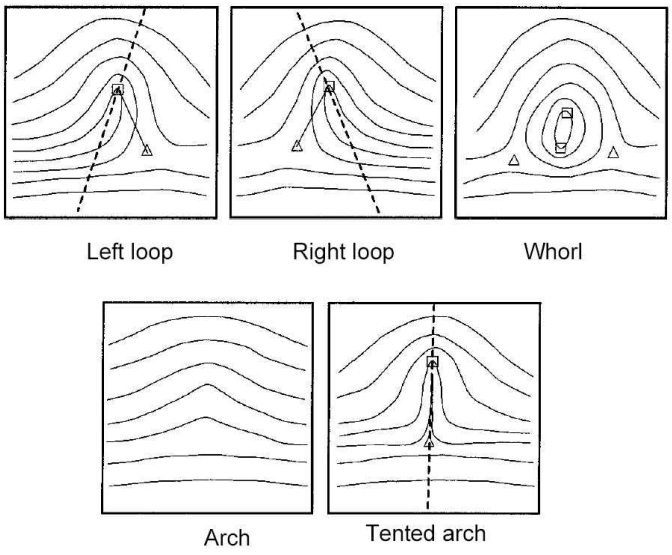
Les capteurs basés sur le silicium se présentent sous forme d'une puce électronique sur laquelle on vient poser son doigt. Ici encore différentes technologies sont utilisées : capacitif (Veridicom, Upek…) , thermique (Atmel), RF (Authentec).



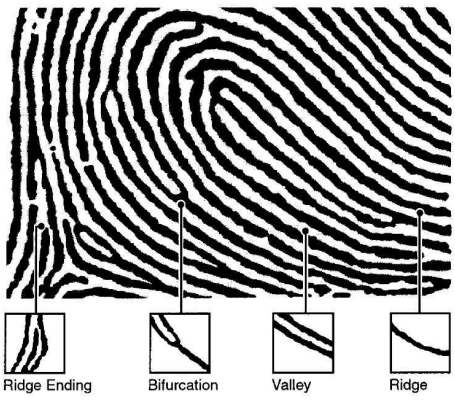
6.10. Empreinte, Visage & Iris

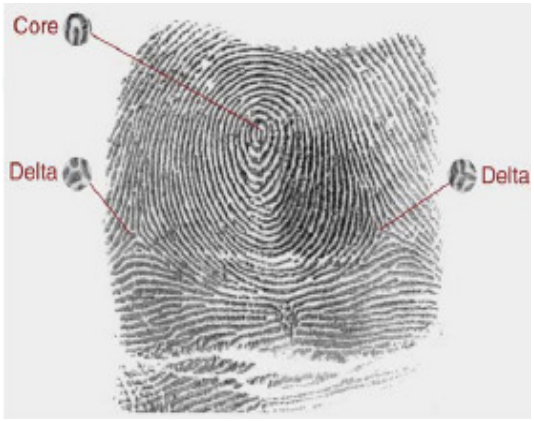
6.10.1. Empreinte

L'empreinte digitale se compose de lignes et de vallées formant un géométrique unique. On distingue quelques formes de base : boucle à droite, boucle à gauche, tourbillon, arche, arche aigue.



La vérification d'empreintes digitales se base sur l'identification de points caractéristiques : Minuties (fin de lignes, bifurcation de lignes), Deltas, Cores et Pores

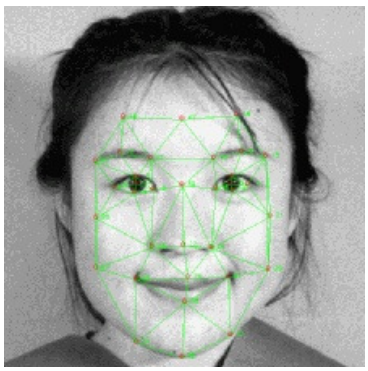
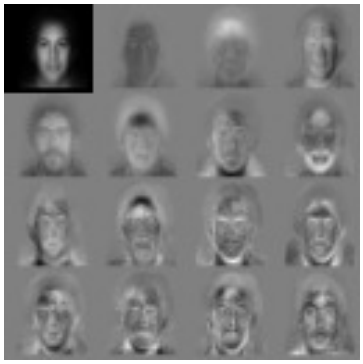
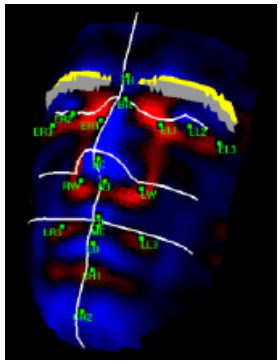




La donnée principale est la minutie ; core, delta et pores étant rarement pris en compte.

6.10.2. Visage

Il existe un grand nombre de techniques totalement différentes utilisées à des fins de reconnaissance faciale : Eigenfaces, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), neural networks, Gaussian Mixture Modeling (GMM), elastic graph…

Voir http://www.face-rec.org

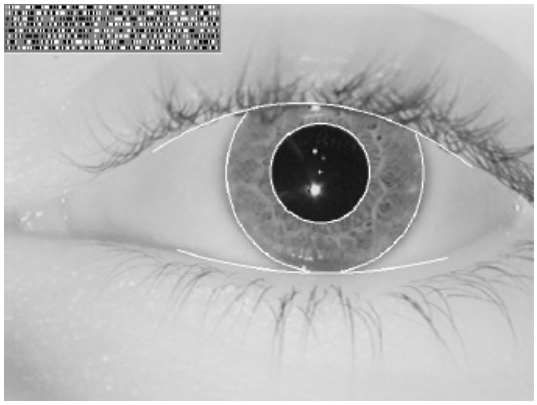
Les sources d'images sont variées : photographie, séquence vidéo, imagerie 3D, thermographie.

Les principaux facteurs altérant la reconnaissance faciale automatique sont l'illumination variable du visage, la position, l'expression (sourires, grimaces…) , le « bruit de fond » de l'image.

6.10.3. Iris

La méthode la plus répandue, connue sous le nom d'Iriscode a été inventé par John Daugman, professeur à Cambridge.

Voir www.iris-recognition.org



6.11. Biométrie et cartes à puce

6.11.1. Principes

Les avancées technologiques de ces dernières années permettent de multiples combinaisons dans la gestion de données biométriques. La carte à puce (ou, plus exactement, sa puce électronique de quelques mm²) est au carrefour des papiers d'identité et de la biométrie : elle permet d'authentifier de manière électronique le document d'identité au moyen d'outils de cryptographie (cette puce étant un petit microprocesseur pouvant effectuer des opérations complexes) et de stocker dans la puce (donc dans le papier d'identité) la donnée biométrique (empreinte digitale ou autre) de référence, ceci afin d'authentifier le porteur comme étant le possesseur *de droit* de ce papier d'identité.

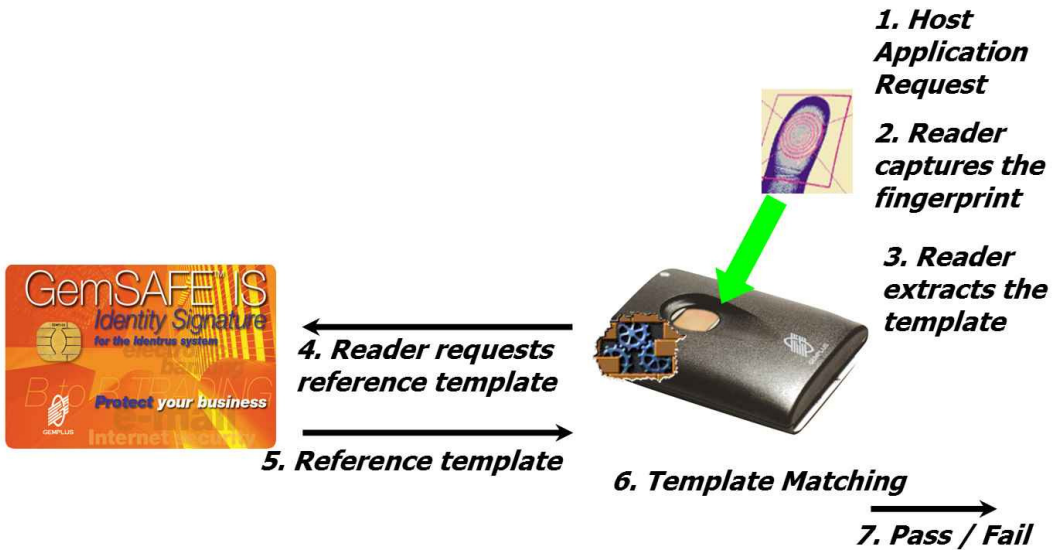
Cette petite puce électronique peut être totalement invisible à l'œil nu, noyée dans le plastique d'une carte d'identité ou dans la couverture d'un passeport, et peut communiquer *sans contact* (par ondes radio à très courte portée : quelques centimètres) avec un terminal prévu à cet effet. Toute communication entre la puce et le terminal est sécurisée au moyen d'outils de cryptographie.

Si l'application le permet, l'utilisation de cette puce électronique sécurisée comme support de stockage de la donnée biométrique de référence peut éviter la création d'un fichier centralisé contenant ces informations biométriques, à caractère très personnel. Trois solutions se présentent alors :

- Le stockage simple dans la puce (ou Storage-On-Card) :

  La donnée biométrique de référence peut être lue par le terminal, la comparaison avec la donnée biométrique candidate se fait dans le terminal, bénéficiant ici de la capacité de calcul élevée du terminal.

  Ce type de solution effectue une comparaison en quelques dizaines de millisecondes. En terme de sécurité, un terminal externe malveillant peut mémoriser une copie de la donnée biométrique de référence.

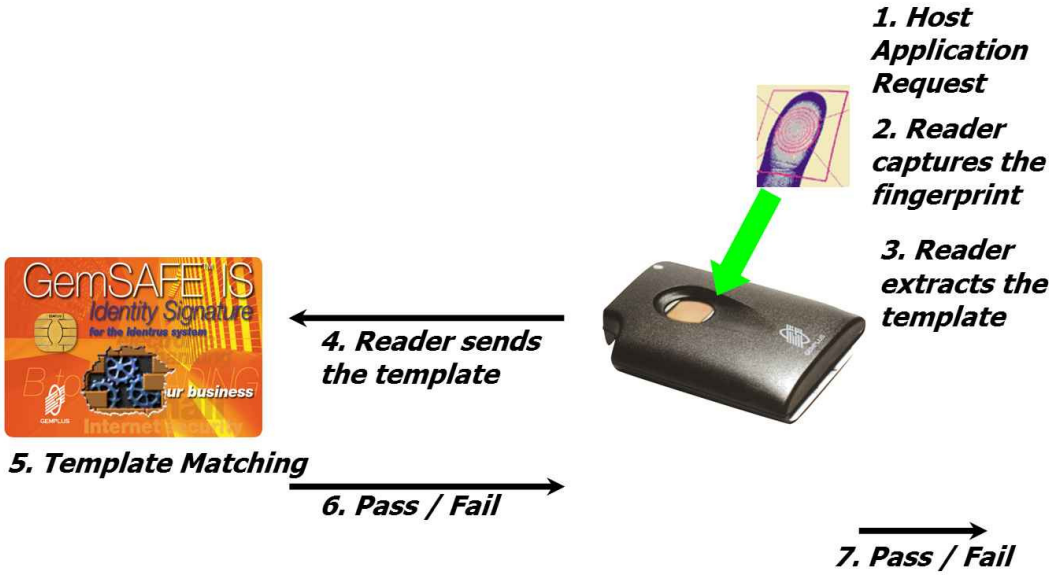


- La comparaison intégrale dans la puce (ou Match-On-Card) :

  Une fois l'enrôlement (enregistrement préalable de la personne dans le système) effectué, la donnée biométrique de référence, stockée dans la puce, n'est plus lisible

par le monde extérieur. Ceci accroît le niveau de sécurité : seule la donnée biométrique candidate est envoyée à la puce qui effectue la comparaison avec la donnée biométrique de référence. Ce type de solution effectue une comparaison en environ une seconde.

L'inconvénient majeur de ce type de solution est la taille de l'algorithme à stocker dans la puce et la nécessité d'une implémentation efficace (langage type assembleur, donc un manque de souplesse dans la portabilité et la mise en place de cette solution).

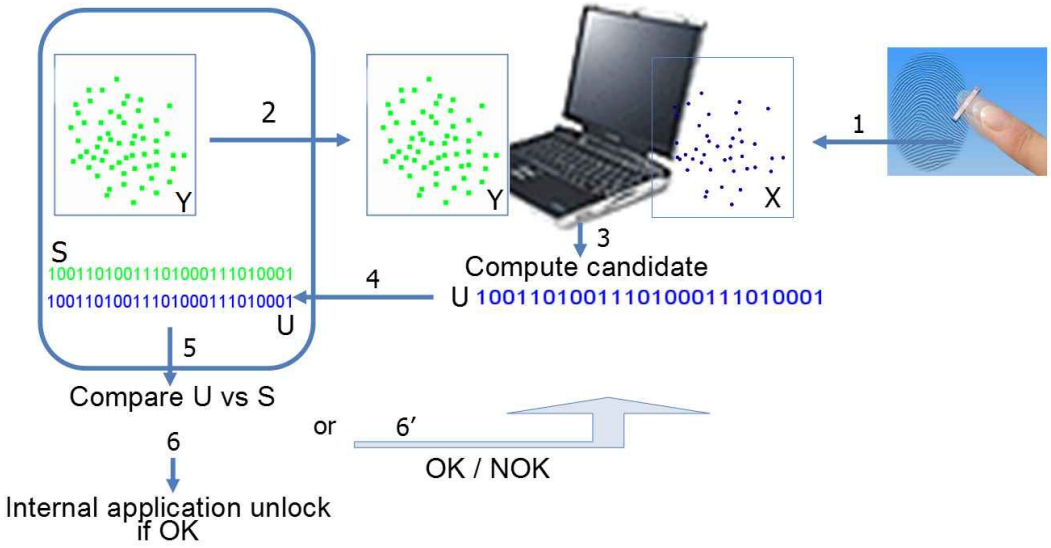


- La comparaison partielle dans la puce :

Cette solution a les avantages du Match-On-Card sans en avoir les inconvénients.

Les calculs complexes, et donc gourmands en temps, sont déportés de manière sécurisée dans le terminal externe (qui est considéré comme un plate-forme non sûre, et manipulera donc seulement une donnée non critique : un « bruit » généré à partir de la donnée biométrique de référence). La comparaison finale est faite dans la puce (à partir d'une donnée candidate générée par les précalculs du terminal externe) et se base sur un *secret* qui a été ajouté durant l'enrôlement à la donné biométrique de référence, ce secret restant interne à la puce.

Ce type de solution effectue une comparaison en quelques dizaines de millisecondes. La taille de l'algorithme à stocker dans la puce est infime (de l'ordre du kilo-octet), et peut être du code type Java, donc très portable et facile à mettre en place, et ce même après déploiement des puces sur le terrain.



La carte à puce (quelque chose que l'on possède) et la biométrie (quelque chose que l'on est) permettent de combiner deux des trois facteurs d'authentification. Il faut retenir que la biométrie n'est pas une donnée secrète et peut même laisser des traces, dans le cas des empreintes digitales par exemple. Seule la notion de secret permet d'obtenir un niveau de sécurité optimal, quitte à diminuer légèrement le niveau de confort d'utilisation. On ajoute alors le troisième facteur (quelque chose que l'on sait) par l'apport d'un mot de passe ou d'un code PIN.

6.11.2. Sécurité

On peut identifier trois contextes d'utilisation :

- Face à face

Une personne surveille le processus d'authentification. C'est par exemple un contrôle d'identité face à un policier.

- Tierce partie de confiance

L'environnement est sous vidéo surveillance. C'est par exemple dans l'enceinte d'une banque, d'un magasin.

- Terminaux libres

Il n'y a aucune surveillance. C'est par exemple avoir un lecteur de biométrie et cartes chez soi pour du vote électronique, transactions sur Internet, etc…

Suivant le contexte d'utilisation, certaines attaques d'un système biométrie et cartes seront plus ou moins réalisables :

- Man-In-The-Middle

Un espion se place sur la ligne de communication et peut écouter, voire même modifier, les échanges.

- Substitution de la donné biométrique

C'est par exemple l'utilisation d'un faux doigt en silicone, ou gomme alimentaire, avec une empreinte copiée, ou la présentation d'une photo devant une camera.

- Substitution de la carte à puce

Présentation d'une vraie carte avec un template de référence modifié, d'une carte répondant « authentification OK » quelle que soit la donnée biométrique entrée (Match-On-Card).

- Manipulation de l'équipement de lecture (biométrie et/ou cartes)

On peut démonter facilement l'équipement afin de faciliter une attaque de type Man-in-the-middle.

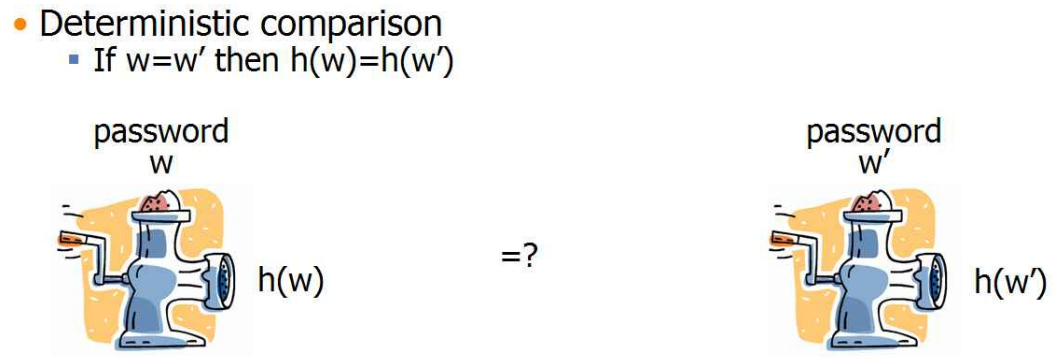
	Solutions
Attended terminals	**Only False card and Yes Card stay a problem.** **Secure graphical tags or the use of signed data are solutions.**
Trusted 3rd party	**Readers' integrity is guaranteed. So, we just have to protect against Cards attack and man-in-the-middle. The best solution seems to be the use of PKi, but a simplified scheme using only digital signatures may apply.**
Unattended terminals	**If it is possible, for the hacker, to open the reader, there is no simple way to protect the system.** **Two solutions can be studied:** • **The use of Strong PKI (One key pair for each reader, and one key pair for each card) combined with tamper resistant readers.**

6.11.3. Mots de passe et Biométrie

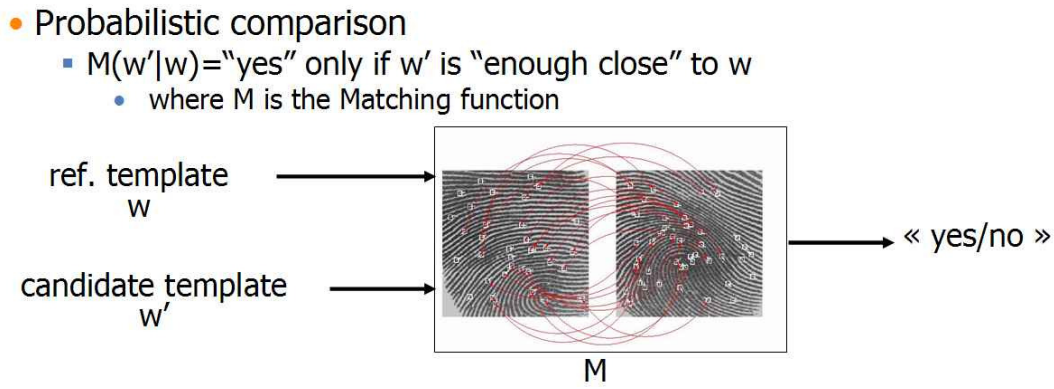
La principale différence réside dans le fait qu'un mot de passe (ou un code PIN) est un *secret* alors qu'une donnée biométrique est *publique*, on laisse beaucoup de traces derrière nous (empreintes digitales, enregistrements vidéos, etc…).

De plus un système d'authentification par mot de passe ne nécessite pas le stockage en clair de la donnée de référence, on stocke un « hashé » cryptographique du mot de passe à partir duquel on ne peut pas retrouver le mot de passe.

Ceci est lié au coté déterministe d'une authentification par mot de passe.



La donnée biométrique de référence doit être en clair pour être manipulé lors d'une authentification pour mesurer la *distance* entre les deux échantillons. Une fonction de hashage n'est pas utilisable dans ce contexte car le « hashé » de la donnée candidate est différent du « hashé » de la donnée de référence car les données de départ ne sont pas identiques. Au mieux, la donnée biométrique de référence est cryptée dans la base de donnée et doit être décryptée lors de son utilisation.



Une étude plus profonde des caractéristiques d'un mot de passe et d'une biométrie montre une opposition claire :

	Caractéristique	Code PIN	Biométrie
1	Secret	oui	non
2	Délégation	oui	non
3	Changeable	oui	non
4	Personnalisation	facile	difficile
5	Algorithme de comparaison	facile	moins facile
6	Confort de l'utilisateur	non	oui
7	Sensibilité aux regards indiscrets	oui	non

8	Sensibilité aux attaques *force brute*	oui	non
9	Contre-mesures	matures	immatures
10	Authentification « réelle »	non	oui
11	Processus de capture	facile	coûteux

- Secret

Un mot de passe est un secret, alors que la biométrie est une donnée publique. On peut faire une différence entre les biométries laissant des traces (empreintes digitales) et les biométries ne laissant pas de traces (géométrie de la main).

- Délégation

Suivant l'application considérée, la délégation est souhaitable (banque, GSM) ou doit être impossible (documents d'identité)

- Changeable

En cas de compromission, un mot de passe peut être annulé et un nouveau créé. Ce n'est pas aussi simple avec la biométrie !

- Personnalisation

Un code PIN est envoyé à l'utilisateur par courrier. En biométrie, la phase d'enregistrement de la personne nécessite le déplacement de celle-ci.

- Algorithme de comparaison

La comparaison entre deux codes PIN est triviale alors que la comparaison entre deux échantillons biométriques est beaucoup plus complexe.

- Confort de l'utilisateur

Un code PIN doit être mémorisé par l'utilisateur et l'on a souvent à gérer plusieurs codes PIN et mots de passe, alors que la biométrie ne nécessite aucun effort.

- Sensibilité aux regards indiscrets

La saisie d'un code PIN peut être espionnée alors qu'une donnée biométrique ne peut pas être rejouée sans l'utilisateur.

- Sensibilité aux attaques *force brute*

De manière automatisée, on peut essayer de jouer tous les mots de passe possible. Ce type d'attaque n'est pas possible avec la biométrie.

- Contre-mesures

Les contre-mesures contre les attaques de code PIN sont matures car étudiées depuis de nombreuses années (try counter, etc…). L'utilisation de la biométrie à grande échelle est beaucoup trop récente pour avoir expérimenté toutes les failles.

- Authentification utilisateur « réelle »

Le fait d'authentifier un porteur pour son code PIN est un artifice d'ordre juridique (« votre mot de passe est personnel, ne le communiqué pas »). La biométrie authentifie réellement, de manière physique, la personne.

- Processus de capture

Un clavier suffit pour saisir un code PIN alors que la plupart des biométries nécessitent un matériel de capture spécifique.

Cette opposition confirme donc la bonne complémentarité entre ces deux techniques. Le remplacement d'un code PIN par de la biométrie doit être étudié au cas par cas suivant l'application.

Ceci étant dit, dans certain cas où les mots de passe sont très mal gérés, la biométrie sera plus conviviale et même plus sûre : mots de passe notés sur des post-it, code PIN par défaut laissé actif (0000, 1234…). On utilisateur humain choisira trop souvent un mot de passe faible car il doit le mémoriser, ce qui le rend attaquable par force brute ou par dictionnaire.

Pour des utilisateurs n'ayant aucune conscience de sécurité, la biométrie est alors une solution de sécurité « faible mais facile ».

6.12. Les standards internationaux

Une norme est un ensemble de règles à suivre afin d'obtenir une interopérabilité entre différents systèmes traitant le même type de données.

L'ISO (International Standardization Organization) est l'organisme qui définie les normes au niveau international.

La CEN (Centre Européen de Normalisation) définie les normes au niveau Européen. On peut citer ici l'initiative TC224-WG15 (carte européenne du citoyen).

Les organismes nationaux régissent les règles au niveau de leur pays. En France, c'est l'AFNOR (Association Française de NORmalisation).

Chaque pays est représenté à l'ISO au travers de organisme national.

L'ISO est divisé en différents Sous-Comités (SC) traitant chacun d'un sujet commun. On peut citer, dans le cadre qui nous concerne, le SC17 (cartes d'identification), le SC27 (sécurité IT, e.g. cryptographie) et le SC37 (biométrie).

Au sein des normes régissant la carte à puce (ISO7816), on peut citer l'ISO 7816-11 qui traite de l'utilisation des données biométriques de l'utilisateur dans une carte.

Chaque SC est lui-même divisé en groupes de travail (WG). Au sein du SC17 on peut citer les WG3 (documents de voyages), WG4 (cartes à contact), WG8 (cartes sans contact). Au sein du SC37 il existe 6 WGs : WG1 (vocabulaire), WG2 (interfaces), WG3 (formats de stockage des données), WG4 (profils d'applications), WG5 (procédures de tests et d'évaluation), WG6 (aspects juridiques et société).

Il est à noter que les travaux de l'ISO en biométrie ont débuté en 2002 et sont très influencés par les travaux des organismes de normalisation aux Etats-Unis (ANSI et NIST). Parmi les travaux américains repris dans l'ISO SC37 WG2 on trouve la BioAPI (Biometrics Application Programming Interface) et le CBEFF (Common Biometric Exchange File Format).

Au sein de l'ISO SC37 WG3, les formats de stockages en cours de finalisation en normes internationales (sous la pression de l'utilisation dans les documents de voyages) sont :

- 19794-2 : format de stockage des minuties d'une empreinte

- 19794-4 : format de stockage d'une image d'empreinte
- 19794-5 : format de stockage d'une image de visage
- 19794-6 : format de stockage d'une image d'iris

D'autres initiatives de standardisation sont beaucoup moins avancées : signature, forme de la main, réseau veineux, autres formats de représentation d'une empreinte digitale.

La normalisation d'un sujet prend généralement plusieurs années. Les différentes étapes d'un norme ISO en cours de définition sont les WD (Working Draft), CD (Committee Draft), FCD (Final Committee Draft), FDIS (Final Draft for International Standard) et enfin IS (International Standard).

Chaque version de document est soumis aux votes (environ tous les 6 mois) et ils peuvent être reconduits plusieurs fois aux niveaux WD et CD (WD2, WD3… CD2…).

Les quatre normes biométrie citées ci-dessus ont atteinte le stade IS en 2005. Ce délai de trois ans (2002-2005) est extrêmement court et assez rare pour être noté.

6.13. Quelques références

D. Maltoni, D. Maio, A.K.Jain, S. Prabhakar: **Handbook of Fingerprint Recognition**, Springer, New York, 2003.

A. Jain, R Bolle, S Pankanti: **Biometrics - Personal Identification in Networked Society**, Kluwer Academic Publishers, 1999.

http://www.biometrics.org

http://biometrie.online.fr

7. Les applications

7.1. Le réseau GSM

L'histoire de la téléphonie mobile débute en 1982. A cette date, le Groupe Spécial Mobile, appelé GSM, est créé par la Conférence Européenne des administrations des Postes et Télécommunications (CEPT) afin d'élaborer les normes de communications mobiles pour l'Europe. Il y eut bien des systèmes de mobile analogique mais le succès de ce réseau ne fut pas au rendez-vous. Les années 80 voient le développement du numérique tant au niveau de la transmission qu'au niveau du traitement des signaux, avec pour dérivés des techniques de transmission fiables, et l'obtention de débits de transmission raisonnables pour les signaux. Ainsi, en 1987, le groupe GSM fixe les choix technologiques relatifs à l'usage des télécommunications mobiles: transmission numérique, multiplexage temporel des canaux radio, chiffrement des informations ainsi qu'un nouveau codage de la parole. Il faut attendre 1991 pour que la première communication expérimentale par GSM ait lieu. Au passage, le sigle GSM change de signification et devient Global System for Mobile communications.

Le concept cellulaire

Les réseaux de première génération possédaient des cellules de grande taille (50*km* de rayon) au centre desquelles se situait une station de base (antenne d'émission). Au tout début, ce système allouait une bande de fréquences de manière statique à chaque utilisateur qui se trouvait dans la cellule qu'il en ait besoin ou non. Ce système ne permettait donc de fournir un service qu'à un nombre d'utilisateurs égal au nombre de bandes de fréquences disponibles. La première amélioration consista à allouer un canal à un utilisateur uniquement à partir du moment où celui-ci en avait besoin permettant ainsi d'augmenter ``statistiquement'' le nombre d'abonnés, étant entendu que tout le monde ne téléphone pas en même temps. Mais ce système nécessitait toujours des stations mobiles de puissance d'émission importante (8 watts) et donc des appareils mobiles de taille et de poids conséquents. De plus, afin d'éviter les interférences, deux cellules adjacentes ne peuvent pas utiliser les mêmes fréquences. Cette organisation du réseau utilise donc le spectre fréquentiel d'une manière sous optimale.

C'est pour résoudre ces différents problèmes qu'est apparu le concept de **cellule**. Le principe de ce système est de diviser le territoire en de petites zones, appelées cellules, et de partager les fréquences radio entre celles-ci. Ainsi, chaque cellule est constituée d'une station de base (reliée au Réseau Téléphonique Commuté) à laquelle on associe un certain nombre de canaux de fréquences à bande étroite, sommairement nommés fréquences. Comme précédemment, ces fréquences ne peuvent pas être utilisées dans les cellules adjacentes afin d'éviter les interférences. Ainsi, on définit des motifs, aussi appelés clusters, constitués de plusieurs cellules, dans lesquels chaque fréquence est utilisée une seule fois.

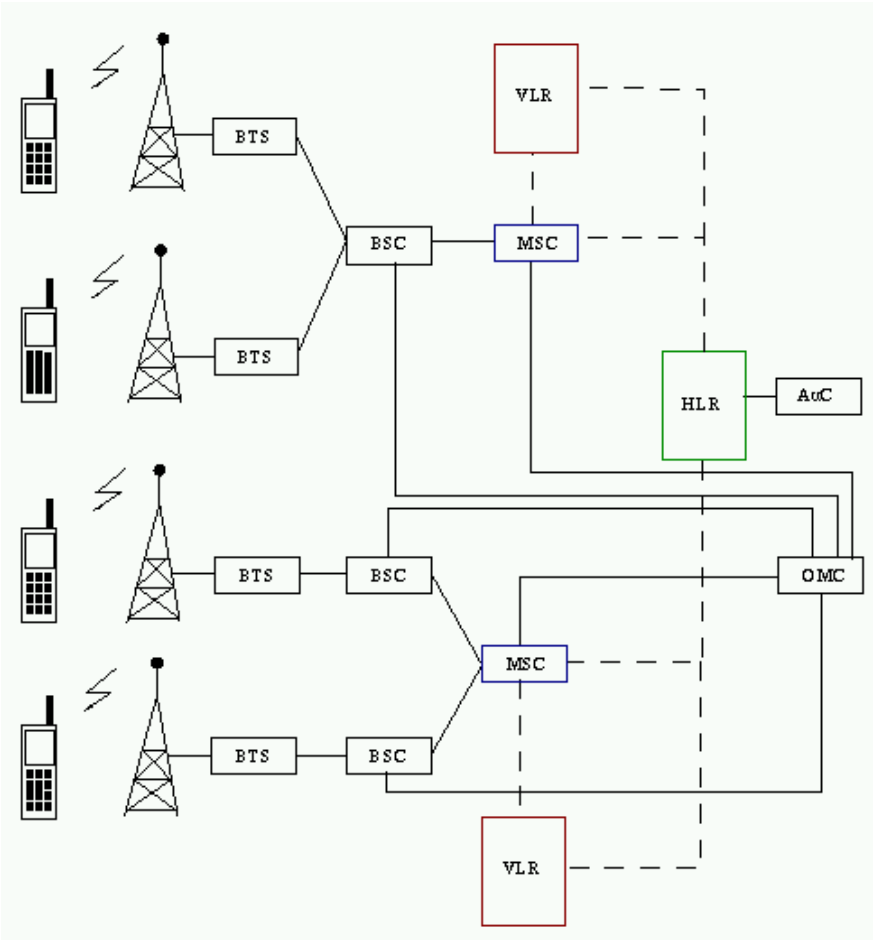
Pour éviter les interférences à plus grande distance entre cellules utilisant les mêmes fréquences, il est également possible d'asservir la puissance d'émission de la station de base en fonction de la distance qui la sépare de l'utilisateur. Le même processus du contrôle de la puissance d'émission est également appliqué en sens inverse. En effet, pour diminuer la consommation d'énergie des mobiles et ainsi augmenter leur autonomie, leur puissance d'émission est calculée en fonction de leur distance à la station de base. Grâce à des mesures permanentes entre un téléphone mobile et une station de base, les puissances

d'émission sont régulées en permanence pour garantir une qualité adéquate pour une puissance minimale.

Ainsi, dans une zone rurale où le nombre d'abonnés est faible et le terrain relativement plat, les cellules seront plus grandes qu'en ville où le nombre d'utilisateurs est très important sur une petite zone et où l'atténuation due aux bâtiments est forte. Un opérateur devra donc tenir compte des contraintes du relief topographique et des contraintes urbanistiques pour dimensionner les cellules de son réseau.

L'architecture du réseau

Le réseau GSM a pour premier rôle de permettre des communications entre abonnés mobiles (GSM) et abonnés du réseau téléphonique commuté. Le réseau GSM s'interface avec le réseau RTC et comprend des commutateurs. Le réseau GSM se distingue par un accès spécifique : la liaison radio que nous venons de décrire.



Le réseau GSM est composé de trois sous ensembles plus le mobile :

- Le sous système radio - BSS Base Station Sub-system assure et gère les transmissions radios,
- Le sous système d'acheminement - NSS Network Sub System (on parle aussi de SMSS *Switching and Management Sub-System* pour parler du sous système

d'acheminement). Le NSS comprend l'ensemble des fonctions nécessaires pour appels et gestion de la mobilité,

- Le sous-système d'exploitation et de maintenance - OSS (*Operation Sub-System*) qui permet à l'opérateur d'exploiter son réseau.

La mise en place d'un réseau GSM permet à un opérateur de proposer des services de type voix à ses clients en donnant l'accès à la mobilité tout en conservant un interfaçage avec le réseau fixe RTC existant.

Le sous système radio

Le sous-système radio gère la transmission radio. Il est constitué de plusieurs entités dont le mobile, la station de base BTS (*Base Transceiver Station*) et un contrôleur de station de base BSC (*Base Station Controller*).

Le mobile et la carte SIM (*Subscriber Identity Module*) sont les deux seuls éléments auxquels un utilisateur a directement accès. Ces deux éléments suffisent à réaliser l'ensemble des fonctionnalités nécessaires à la transmission et à la gestion des déplacements. Les terminaux (appareils) sont identifiés par un numéro d'identification unique de 15 chiffres appelé IMEI (*International Mobile Equipment Identity*).

La principale fonction de la carte SIM est de contenir et de gérer une série d'informations. Elle se comporte donc comme une mini base de données dont les principaux champs sont fournis dans le tableau suivant.

Paramètres	***Commentaires***
Données administratives	
Langage	Langue choisie par l'utilisateur
SN	Numéro de série de la carte
Données liées à la sécurité	
CHV1, CHV2	Mot de passe demandé à chaque connexion
PUK1, PUK2	Code pour débloquer une carte
Clé ***Ki***	Valeur unique, connue de la seule carte SIM et du HLR
RC1/2	Compteur d'erreur de CHV1/2
Données relatives à l'utilisateur	
IMSI	Numéro international de l'abonné
MSISDN	Numéro d'appel d'un téléphone GSM
Données de ``roaming''	
TMSI	Numéro attribué temporairement par le réseau à un abonné
Location updating status	Indique si une mise à jour de la localisation est nécessaire
Données relatives au réseau	
Mobile Country Code (***MCC***), Mobile Network Code (***MNC***),	Identifiants du réseau mobile de l'abonné

L'identification d'un mobile s'effectue exclusivement au moyen de la carte SIM. En effet, elle contient des données spécifiques comme le code PIN (*Personal Identification Number*) et d'autres caractéristiques de l'abonné, de l'environnement radio et de l'environnement de l'utilisateur.

L'identification d'un utilisateur est réalisée par un numéro unique IMSI (*International Mobile Subscriber Identity*) différent du numéro de téléphone connu de l'utilisateur MSIN (*Mobile Station ISDN Number*), tous deux étant stockés dans la carte SIM.

C'est la station de base qui fait le relais entre le mobile et le sous-système réseau. Elle réalise les fonctions de la couche physique et de la couche liaison de données. La station de base est l'élément central, que l'on pourrait définir comme un ensemble émetteur/récepteur pilotant une ou plusieurs cellules. Dans le réseau GSM, chaque cellule principale au centre de laquelle se situe une station base peut-être divisée, grâce à des antennes directionnelles, en plus petites cellules qui sont des portions de celle de départ et qui utilisent des fréquences porteuses différentes.

Le contrôleur de station de base Le contrôleur de station de base gère une ou plusieurs stations de base. Pour les fonctions des communications des signaux en provenance des stations de base, le BSC agit comme un concentrateur puisqu'il transfère les communications provenant des différentes stations de base vers une sortie unique. Dans l'autre sens, le contrôleur commute les données en les dirigeant vers la bonne station de base.

Dans le même temps, le BSC remplit le rôle de relais pour les différents signaux d'alarme destinés au centre d'exploitation et de maintenance. Il alimente aussi la base de données des stations de base. Enfin, une dernière fonctionnalité importante est la gestion des ressources radio pour la zone couverte par les différentes stations de base qui y sont connectées. En effet, le contrôleur gère les transferts intercellulaires des utilisateurs dans sa zone de couverture, c'est-à-dire quand une station mobile passe d'une cellule dans une autre. Il doit alors communiquer avec la station de base qui va prendre en charge l'abonné et lui communiquer les informations nécessaires tout en avertissant la base de données locale VLR (*Visitor Location Register*) de la nouvelle localisation de l'abonné.

Le sous système réseau

Le sous-système réseau, appelé Network Switching Center, joue un rôle essentiel dans un réseau mobile. Alors que le sous réseau radio gère l'accès radio, les éléments du NSS prennent en charge toutes les fonctions de contrôle et d'analyse d'informations contenues dans des bases de données nécessaires à l'établissement de connexions utilisant une ou plusieurs des fonctions suivantes: chiffrement, authentification et roaming. Il est constitué de des éléments suivants :

- Mobile Switching Center (MSC), le centre de commutation mobile est relié au sous-système radio via l'interface A. Son rôle principal est d'assurer la commutation entre les abonnés du réseau mobile et ceux du réseau commuté public. D'un point de vue fonctionnel, il est semblable à un commutateur de réseau, mis à part quelques modifications nécessaires pour un réseau mobile.
- Home Location Register (HLR), il existe au moins un enregistreur de localisation par réseau. Il s'agit d'une base de données avec des informations essentielles pour les services de téléphonie mobile et avec un accès rapide de manière à garantir un temps d'établissement de connexion aussi court que possible. Il contient toutes les informations relatives aux abonnés: le type d'abonnement, la clé

d'authentification K_i -cette clé est connue d'un seul HLR et d'une seule carte SIM-, les services souscrits, le numéro de l'abonné (IMSI), ainsi qu'un certain nombre de données dynamiques telles que la position de l'abonné dans le réseau.

- Authentication Center (AuC), l'opérateur doit pouvoir s'assurer qu'il ne s'agit pas d'un usurpateur. Le centre d'authentification remplit cette fonction de protection des communications en assurant le chiffrement des transmissions radio uniquement et l'authentification des utilisateurs du réseau au moyen d'une clé K_i, qui est à la fois présente dans la station mobile et dans le centre d'authentification.
- Visitor Location Register (VLR), cette base de données ne contient que des informations dynamiques, elle est liée à un MSC. Il y en a donc plusieurs dans un réseau GSM. Elle contient des données dynamiques qui lui sont transmises par le HLR avec lequel elle communique lorsqu'un abonné entre dans la zone de couverture du centre de commutation mobile auquel elle est rattachée. Lorsque l'abonné quitte cette zone de couverture, ses données sont transmises à un autre VLR.
- Equipment Identity Register (EIR), pour combattre le vol de terminal mobile, chaque terminal reçoit un identifiant unique IMEI (International Mobile station Equipment Identity) qui ne peut pas être modifié sans altérer le terminal. Un opérateur peut décider de refuser l'accès au réseau en fonction des listes qu'il possède.

Authentification et chiffrement

Il faut, pour éviter de fragiliser la confidentialité, la sécurité et la vie privée de l'abonné par le chemin radio, intégrer des fonctions de sécurité supplémentaire afin de protéger à la fois les abonnés mais aussi les opérateurs.

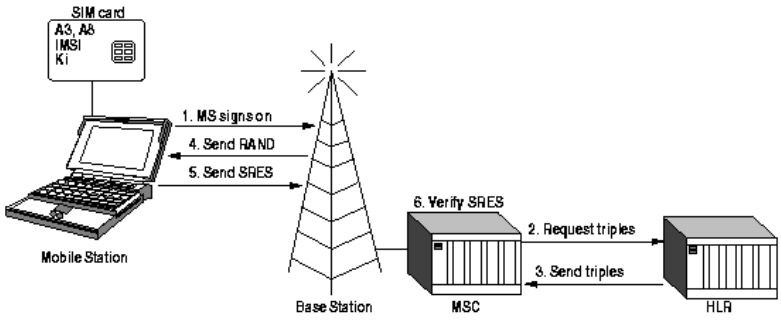
Afin d'éviter l'interception de l'IMSI lors du transfert sur la voie radio, il faut limiter cette transmission qui devient un moyen de tracer ou de suivre un abonné mobile en interceptant les messages de signalisation échangés sur le canal radio. Le réseau a recours à un IMSI temporaire le TIMSI. C'est le VLR qui établit la correspondance entre IMSI et TIMSI. L'IMSI est transmis à la mise sous tension du mobile puis le réseau lui affecte un TIMSI qui sera utilisé par la suite pour l'identifier. L'allocation d'un TIMSI est faite au moins à chaque changement de VLR et l'envoi du nouveau TMSI a lieu chiffré.

Pour mettre en œuvre les fonctions de d'authentification et de chiffrement des informations transmises sur la voie radio le réseau utilise des triplets comprenant un nombre aléatoire RAND, le résultat SRES de l'algorithme A3 avec comme paramètre d'entrée la clé K_i et le RAND, et la clé de chiffrement du chemin radio K_c obtenue par l'algorithme A8 avec comme paramètre d'entrée la clé K_i et l'aléa RAND. La clé K_c est envoyée par la carte SIM au mobile afin d'effectuer le chiffrement de la voix à l'aide de l'algorithme A5.

Chaque abonné à une clé Ki K_i qui lui est propre et les algorithmes A3, A5 et A8 sont identiques pour tous les abonnés. Les algorithmes A3 et A8 sont réalisés par un seul algorithme appelé COMP128.

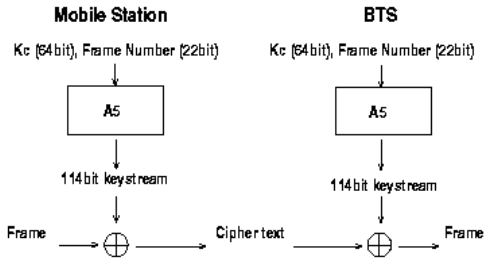
L'authentification permet de vérifier que l'identité transmise par le mobile IMSI sur la voie radio est correct afin d'éviter un détournement d'identité préjudiciable à l'abonné ou

l'utilisation frauduleuse de services auquel un abonné n'aurait pas souscrit. Le réseau ne calcul pas les triplets en temps réel.



L'AuC calcule des triplets (il possède donc la clé K_i de chaque abonné et les algorithmes A3 et A8) pour chaque abonné et les transmet au HLR qui les stocke en attendant une demande du VLR. Aucune donnée de sécurité clé ou algorithme ne circule donc sur le réseau ou la voie radio. Ce sont les BTS qui possèdent les algorithmes A5 car seule la voie radio en cryptée ensuite les messages circulent en clair sur le réseau interne ou le réseau filaire.

Lorsque le VLR reçoit une demande de connexion d'un IMSI il envoie un RAND au mobile. La carte SIM calcule la signature du RAND à l'aide de la clé K_i et l'algorithme A3 et envoie le résultat au VLR. Ce dernier compare le SRES rendu avec celui stocké et si ils correspondent, il envoie la clé K_c de chiffrement radio à la BTS permettant ainsi le cryptage du chemin radio.



Après l'authentification, le MSC prend la décision de passer en mode chiffré. Il transmet un message BSMAP CIPHER contenant la clé K_c de chiffrement au BSC, qui la retransmet à la BTS. La BTS envoie ensuite le message CIPHERING MODE au mobile pour lui indiquer le passage en mode chiffré en utilisant l'algorithme A5 comme décrit ci-dessus.

La carte SIM

La carte possède de nombreux éléments de sécurité comme les algorithmes A3 et A8, la clé K_i, mais aussi les code secrets CHV1 et CHV2. Le CHV1 (Card Holder Verification 1) est utilisé pour identifier l'abonné, il est demandé à chaque mise sous tension du mobile, ou plutôt de la carte SIM. Le code CHV2 est destiné à permettre un complément de personnalisation si l'opérateur du réseau le souhaite. L'utilisation de CHV1 n'autorise que les fonctions relatives à ce code et non à celles relatives à CHV2. Le blocage des CHV intervient lorsque l'utilisateur a entré un code erroné plus de trois fois. La carte SIM passe alors dans un état interdisant son utilisation sur le réseau GSM. Il devient nécessaire d'utiliser une clé de déblocage PUK (PIN Unblocking Key). Cette clé est elle-

même protégée par un compteur de ratification de dix tentatives. A ce moment là, la carte est irrécupérable.

La carte SIM est organisée suivant une arborescence. La racine MF (Master File) est constituée d'un fichier pouvant contenir des fichiers élémentaires EF (Elementary File) ainsi que des répertoires DF (Dedicated File) pouvant contenir eux aussi des EF.

Le répertoire racine contient le numéro de série dans un EF et deux répertoires GSM et TELECOM. Ce dernier comprend les informations liées au service de télécommunication (liste des numéros abrégés, dernier numéro composé, dernier message reçu,…). Le répertoire GSM contient l'IMSI, la liste des réseaux, la zone de localisation,…

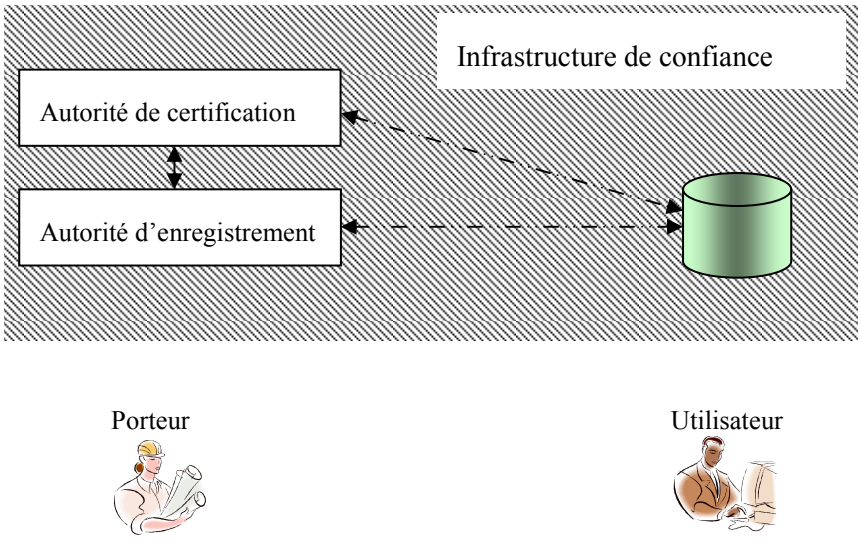
Les données sont accessibles au moyen de commandes APDU après avoir présenté les droits adéquats (ALW, always ; CHV1, présentation du code ; CHV2, idem ; ADM, authentification d'un administrateur ; NEV, never) en fonction de l'accès requis (lecture, modification, invalidation et revalidation.

7.2. Les infrastructures à clés publiques

La cryptographie a clés publiques rend la gestion des clefs plus simple mais induit de nouveaux problèmes. Chaque personne n'ayant qu'une clé publique Alice doit obtenir la clé publique de Bob soit en la demandant directement auprès de Bob soit en l'obtenant dans un répertoire central soit en l'ayant obtenu via sa propre base de donnée. Si elle demande à Bob sa clé est s'expose à une attaque de type *man in the middle* ou l'interception. Si un élément malveillant intercepte la réponse de Bob il peut échanger sa propre clé publique avec celle de Bob, il déchiffrera ainsi tous les messages d'Alice, tout en transmettant à Bob les messages car il possède la clé de Bob.

L'architecture

Un certificat de clé publique est constitué de la publique de quelqu'un signé par la clé privée d'une personne en qui on a confiance : une autorité de certification (CA), plus des informations complémentaire nom, adresse,.... En signant à la fois la clé et les informations supplémentaires on s'assure que les informations sont correctes et que la clé publique appartient bien à ladite personne. Evidement l'autorité ne doit apposer son sceau qu'en ayant confiance dans les informations qui lui sont fournies c'est le but de l'enregistrement par l'autorité d'enregistrement (RA). De même, un certificat peut être annulé il faut alors gérer des listes de révocation. Tous ces éléments forment une architecture à clé publique PKI.



Le porteur du certificat c'est celui dont le nom est référencé et qui possède la clé publique qui est présente dans le certificat et sa contrepartie la clé privée. En principe, il est le seul à connaître la clé privée correspondant au certificat. Il est équipé des matériels et logiciels nécessaire au stockage et à la gestion sûrs de son certificat et de sa clé privée.

L'utilisateur du certificat c'est l'entité qui sélectionne et valide le certificat du porteur. Il doit se fier au certificat afin de réaliser l'opération qu'il souhaite établir avec le porteur (validation d'une signature ou chiffrement). Cette confiance repose sur l'assurance que l'entité gérant les certificats dispose des procédures adéquates pour l'enregistrement et la publication des certificats. Il doit s'assurer que l'identité du destinataire est bien celle contenue dans le certificat et qu'aucun des certificats du chemin de certification n'est révoqué, il vérifie que les données du certificat n'ont pas été altérées en utilisant la clé publique de l'autorité de certification.

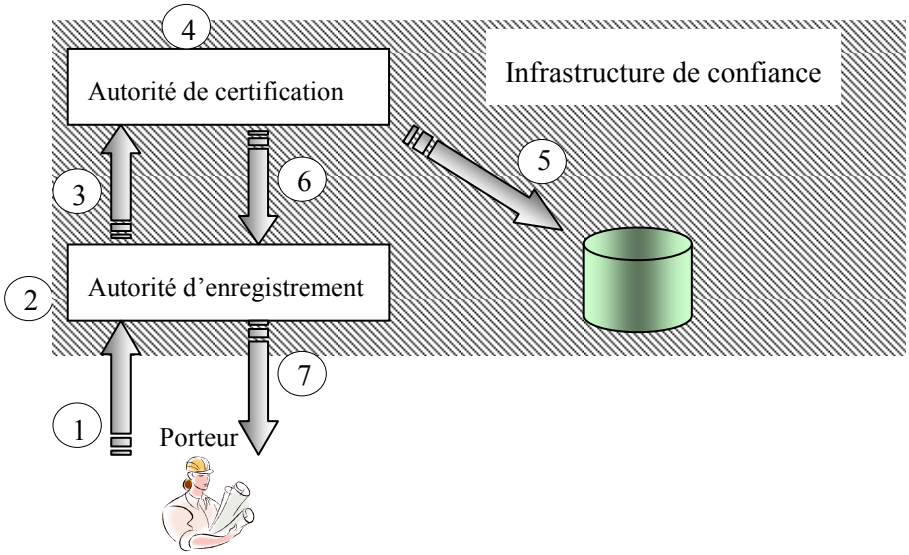
L'autorité de certification (CA) a la confiance d'un (hiérarchie de certificats) ou de plusieurs utilisateurs pour créer et attribuer les certificats.

L'autorité d'enregistrement (RA) a la responsabilité administrative relative à la gestion des demandeurs/porteurs de certificats. Il doit s'assurer de l'identité du demandeur en obtenant les pièces justificatives (carte d'identité,…) obtenir la clé publique (parfois génère la paire de clés) ou s'assurer que le demandeur est en possession de la clé privée associée à la clé publique ainsi que traiter les révocations des certificats. Une fois que le certificat est généré et donc signé avec la clé privée du CA il devient infalsifiable. C'est donc lors de l'enregistrement que peuvent se passer les fraudes.

Le service de publication est une composante qui rend disponible les certificats de clés publiques. Ce service peut être rendu par un annuaire, un serveur web, une messagerie… Le service de publication est en charge de la mise à jour des listes de révocation.

Les différentes phases

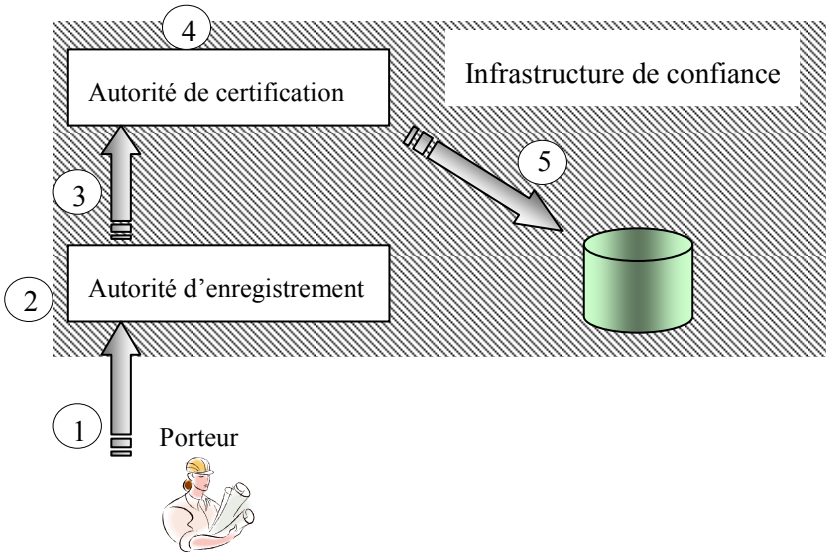
La création des certificats se décompose en plusieurs étapes. En 1, le porteur ou son représentant fait une demande, en 2 le RA procède à l'authentification et à la vérification des attributs du demandeur. Il est à noter qu'il est possible de vérifier ces attributs lors de la remise du certificat. La qualité de l'authentification détermine le niveau de confiance du certificat.



En 3 le RA transmet à l'AC la demande, en 4 l'AC génère le certificat en vérifiant uniquement l'unicité du numéro de certificat. En 5 place le certificat dans un service de publication accessible à tous. Le certificat est remis à l'AE en 6, qui le donne au porteur en 7 ou à son représentant si nécessaire. Si l'infrastructure prend en charge la génération de la paire de clé alors la clé privée est distribuée d'une manière sécurisée.

Afin de stocker la clé privée de manière sûre, il est possible d'utiliser une carte à puce. Les clés peuvent être générées par la carte, puis le certificat chargé dans la carte.

La révocation est un mécanisme destiné à rendre un certificat non valide définitivement (révocation) ou définitivement (suspension) avant sa date de fin de validité. Plusieurs causes peuvent être à l'origine d'une révocation comme la compromission d'une clé privée, modification d'un élément du certificat, …



En 1 le porteur demande à l'autorité d'enregistrement la révocation du certificat dont il fournit l'identifiant. L'AE procède en 2 à l'authentification du porteur et vérifie les privilèges de révocation. La demande est transmise à l'autorité de certification en 3, laquelle enregistre (4) la révocation et demande la publication dans les listes de révocation dudit certificat. Les listes peuvent devenir des très gros fichiers, et souvent la liste n'est pas publiée à chaque révocation de certificat ce qui entraîne une latence supplémentaire entre la décision et la révocation.

Pour pouvoir utiliser une clé publique avec sécurité il faut donc que l'utilisateur puisse déterminer à qui appartient une clé publique et à quelle fonction elle est destinée. Pour cela il faut que la clé publique correspondant à la clé privée asymétrique soit accompagnée d'informations décrivant son propriétaire, son usage. Il doit identifier quelle autorité est responsable de l'émission du certificat, il doit indiquer la période de validité du certificat et un numéro d'identification. Enfin il doit être infalsifiable et donc doit comporter la signature de l'autorité l'ayant certifié. Un certificat X509 contient les données suivantes :

Version du certificat
Numéro de série du certificat
Description de l'algorithme de signature du CA
Nom du CA qui a généré le certificat
Période de validité
Nom de l'utilisateur auquel appartient le certificat
Clé publique
Description de l'algorithme à utiliser avec la clé publique
Identification alternative du CA (optionnel)
Identification alternative de l'utilisateur (optionnel)
Extensions (optionnel)
Signature du CA

Les modèles de confiance

Le modèle qualifié de fermé concerne des communautés d'utilisateurs comme une entreprise. Le domaine de confiance est alors limité à ce périmètre. Ce modèle est de type hiérarchique avec une hiérarchie limitée à un seul niveau. Dans ce cas l'AC émet directement les certificats des utilisateurs finaux. Tous les utilisateurs appartiennent par définition au domaine de confiance fermé et chacun possède le certificat racine de l'AC dans son système.

La limite de ce modèle intervient lorsque les besoins de communications inter communautés se font sentir. Dès lors le certificat racine n'est pas reconnu par les acteurs externes au périmètre de confiance initial. Ils doivent donc s'accorder sur des règles d'interopérabilité afin que les certificats puissent être reconnus par les autres on parle alors de modèle en réseau. Un tel modèle peut utiliser un système hiérarchique, une certification croisée.

Le dernier modèle dit ouvert, est en fait le modèle le plus large où tout utilisateur connecté peut éventuellement échanger avec tout utilisateur. Dans ce cas, les utilisateurs obtiennent des certificats à partir d'opérateurs privés délivrant des certificats en ligne. Les contrôles sur l'identité du demandeur sont très légers, se limitant à l'unicité de l'adresse de messagerie électronique. Le niveau de confiance est donc plus faible mais peut suffire si le risque encouru est faible aussi.

Le dernier modèle de confiance est apparu avec PGP. OpenPGP est l'abréviation en anglais de "Open Pretty Good Privacy". Il s'agit d'une spécification pour sécuriser les données avec une infrastructure de gestion de clés (IGC) ; elle a été normalisée par le groupe OpenPGP de l'IETF. Le but principal de OpenPGP est le chiffrement et la signature des fichiers dans les réseaux ouverts grâce à l'utilisation des services de cryptographie à clé publique. OpenPGP est donc un moyen d'indiquer des privilèges aux utilisateurs en utilisant un certificat électronique. Dans ce cas, la clé publique d'un demandeur est signée par un ensemble de parrains qui se portent garant de lui. OpenPGP permet d'avoir plusieurs AC indépendantes et non spécialisées ; c'est-à-dire que chaque utilisateur est sa propre entité de confiance. Avec cette approche OpenPGP a une infrastructure décentralisée. OpenPGP utilise un mécanisme tolérant aux fautes appelé "Web of Trust" qui a été conçu de façon à ce que l'émetteur puisse ne pas être une AC professionnelle. N'importe quel utilisateur peut signer une clé publique et agir en tant qu'autorité de certification. L'ensemble des certificats devient peu à peu un ensemble de clés publiques interconnectées par ces signatures. Plus il y aura de signatures, plus la confiance en ce certificat sera effective. La validation du certificat est donc faite par la vérification des signatures dans le certificat et de la date de validité. Le niveau de confiance est donc très relatif et dépend de la confiance que l'on a dans ces parrains.

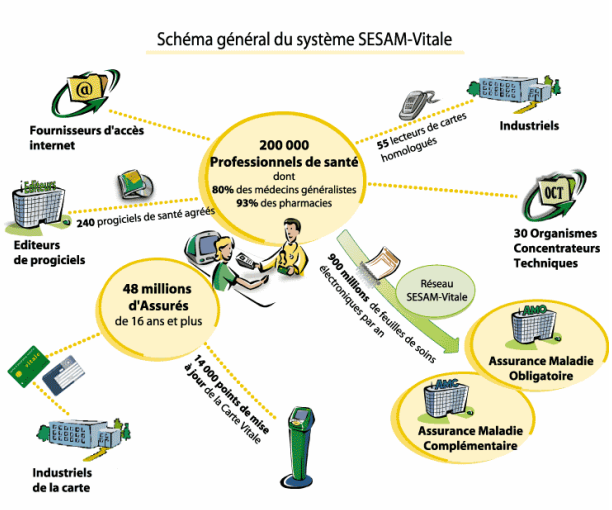
La carte professionnel de Santé

Le groupement d'intérêt public (GIP) « Carte professionnel de santé », gère la carte Sesam des professionnels (dite CPS) et la carte Vitale des assurés sociaux. C'est une carte à puce réservée aux personnels de santé. Les porteurs ont accès au réseau santé social (RSS). En relation avec la carte Vitale, elle permet d'émettre des feuilles de soin électronique et de les transmettre aux caisses d'assurance maladie. Pouvoir accéder au RSS offre une gamme de service comme l'échange confidentiel sur messagerie électronique, la télémédecine (échange de données entre praticiens, résultat d'examen), veille sanitaire,…

En tant qu'autorité de certification, le GIP gère des certificats de clés de signature pour signer des messages, des certificats de clé d'authentification pour un usage applicatif, des certificats de clé de chiffrement. Le GIP joue les différents rôles d'autorité dont la génération et la gestion des certificats racines et intermédiaires, génération des certificats délivrés aux utilisateurs, publication dans un annuaire des certificats valides, la fabrication et la gestion des cartes à puce support des certificats de signature et d'authentification.

Les certificats d'authentification et de signature sont générés lors de la phase de personnalisation de la carte. Les paires de clés asymétriques sont renouvelées en même temps que la carte.

Le professionnel dispose d'un lecteur double (Sesam+Vitale). Les feuilles de soin électroniques sont stockées sur le poste de travail et signée par la carte Sesame dans le lecteur. En fin de journée, les lots sont signés et télétransmis vers les caisses d'assurance maladie.



Il existe différent type de cartes suivant les professionnels de santé concernés : CPS pour les professionnels de santé libéraux, CPE pour les professionnels d'établissements, CDE pour les directeurs d'établissements. Dans les cartes CPS, il y a 2 certificats (2 clés privées) pour augmenter le niveau de sécurité : une clé d'authentification (pour le cryptage) et une clé de signature. Il propose également la possibilité d'utiliser ses certificats sur d'autres supports que la carte CPS certificat dans clé USB, certificat dans autres cartes, certificat sur poste... Le GIP peut délivrer des certificats pour les serveurs ce qui permet l'authentification réciproque. Le GIP a mis en place une PKI qui se porte garant des secrets qu'il va gérer, composée de.

- d'une autorité d'enregistrement (**AE**) : il certifie la qualité des individus porteurs de carte (en lien avec ordres, DDASS…). Il certifie que la personne décrite est bien celle qu'elle prétend être.
- d'une autorité de certification (**AC**) : il crée les clés et les certificats. Il crée et gère les secrets et en particulier les clés privées qui sont dans la puce.
- d'un service de publication (**SP**) : c'est un annuaire contenant les clés publiques des porteurs de cartes. Il y a 2 clés par personnes + une troisième pour l'utilisation de la messagerie sécurisée.

La partie publique de l'annuaire ne présente que les porteurs de cartes CPS (pas CPE par exemple) selon la volonté de la CNIL. Les porteurs de cartes CPE deviennent visibles dans l'annuaire uniquement s'ils utilisent la messagerie sécurisée. Pour chaque classe de clés et pour chacune d'entre elles, il existe une liste d'oppositions et révocations particulière.

Conclusion

La technologie des PKI est une innovation capitale pour gérer la sécurité dans les grands réseaux Intranet et surtout pour permettre le développement des échanges électroniques sur Internet. Elle permet à un utilisateur, grâce à son architecture originale, de réaliser des échanges sécurisés avec un grand nombre de serveurs sans que ceux-ci aient besoin de connaître l'utilisateur au préalable. Seule l'Autorité d'Enregistrement aura vérifié une fois pour toute l'identité de l'utilisateur pour le compte de toute la communauté. Elle offre, sous la forme d'une solution de sécurité unique, une panoplie de fonctions répondant à tous les besoins de sécurité des échanges électroniques (authentification, confidentialité, intégrité, non répudiation). Grâce à l'utilisation de standards et aux mécanismes de

certification croisée ou hiérarchiques, il est possible de créer des domaines de sécurité de très grande taille sur Internet, domaines de sécurité ingérables en pratique avec des solutions traditionnelles. Cette infrastructure permet d'envisager la dématérialisation des échanges et des processus de l'entreprise dans un contexte légal approprié.

7.3. La sécurisation des ordinateurs personnels

Introduction

Aujourd'hui, la sécurité demeure la première préoccupation des entreprises, en particulier pour le déploiement d'ordinateurs. Des risques tels que les attaques malveillantes, le vol, les virus, l'accès non autorisé aux systèmes, l'interception de données confidentielles, le piratage et un cryptage insuffisant peuvent transformer des données privées en informations publiques. Des pertes de revenu, des surcoûts informatiques, l'indisponibilité des systèmes, la perte de données et le vol d'informations sont quelques-unes des conséquences possibles de tels problèmes.

L'exécution fiable est constituée de l'environnement dans lequel s'exécutent les applications sécurisées. Cela exige une combinaison de matériel et de logiciel, notamment un système d'exploitation au noyau de sécurité inviolable pour segmenter la mémoire, afin que les applications sécurisées puissent s'exécuter simultanément avec les autres applications. L'exécution fiable vérifie également que tous les accès aux périphériques sont autorisés, en particulier l'accès au clavier.

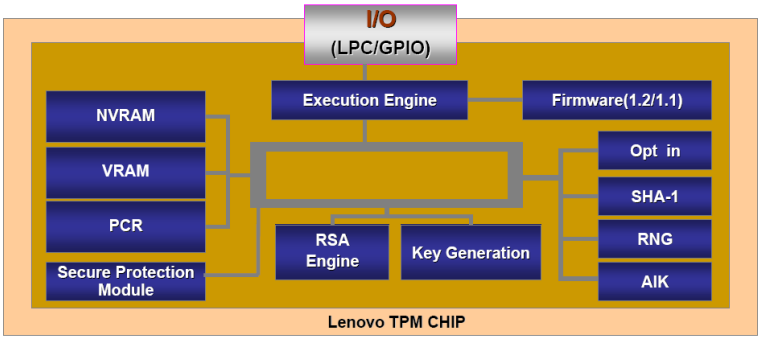
Les plates-formes de confiance sont apparues à partir de 1999 avec la mise en place du TCPA *(Trusted Computing Platform Alliance)* organisme en charge d'élaborer des spécifications pour cette architecture. Cette association s'est dissoute pour former le TCG (*Trusted Computing Group*). L'objectif est de définir une architecture matérielle qui serait commune aux prochaines versions des ordinateurs de façon à renforcer la sécurité. Microsoft qui est un membre actif de ce consortium travaille dans la même direction avec la spécification de NGSCB (*Next Generation Secure Computing Base*), reprenant en partie les travaux de TCG afin d'élaborer la prochaine version de son système d'exploitation. Les premiers travaux de TCG ont conduit à l'élaboration de version 1.2 de la spécification du matériel dédié à la sécurité le TPM (*Trusted Platform Module*).

Le TPM apporte les fondements en matière de sécurité, ce que l'on appelle aujourd'hui la "base de confiance essentielle". Les informations stockées sur la puce peuvent être utilisées par un processus d'attestation, c'est-à-dire des processus de mesure, d'enregistrement et de rapports qui interdisent les accès non autorisés ou la falsification de données. Le TPM constitue une base solide de protection qui peut être complétée par d'autres mesures de sécurité. Malgré ses caractéristiques élevées de sécurité, le TPM en lui-même n'est pas suffisant pour assurer un haut niveau de protection du système. Un modèle de sécurité plus complet est nécessaire prenant en compte l'intégration du système d'exploitation.

Architecture du TPM

L'architecture d'un TPM comprend un microcontrôleur avec des fonctionnalités cryptographiques. La figure suivante montre les différents blocs de base le composant. Afin de simplifier l'intégration dans une plateforme de type PC, le TPM utilise un bus spécifique le LPC (*Low Pin Count*) afin de se raccorder au chipset de la carte mère. Il est recommandé qu'un tel composant puisse être évalué au moins au niveau 3+ des critères

communs. Il doit garantir un minimum de résistance face aux attaques matérielles, bien que ce type d'attaque ne soit pas sa cible principale.



Le TPM possède un accélérateur mathématique pour réaliser efficacement les opérations de cryptographie. Ces fonctions cryptographiques (SHA-1 et RSA) ne sont pas accessibles directement de l'extérieur. Il faut passer par le bus d'entrée sortie du TPM et sont API. Il est capable de réaliser du chiffrement/déchiffrement avec des clés d'une longueur maximale de 2048 bits. Cet algorithme est aussi utilisé pour la signature, donc s'assurer de l'intégrité pour réaliser des demandes d'extension de PCR. Le TPM peut calculer des valeurs de hachage pour des petits documents et dispose d'un générateur de nombre aléatoire et un générateur de clés.

Il possède la faculté d'être désactivé (module opt-in) sur demande de l'administrateur. Il dispose enfin de capacité de stockage de mémoire volatile et non volatile. Le matériel est inviolable car la mémoire scellée et infalsifiable est utilisée pour générer et stocker de façon sécurisée les clés et les fonctions de cryptage qui les utilisent pour déverrouiller le reste du système. On s'aperçoit de la grande similitude avec une carte une carte à puce inamovible.

Fonctionnement du système

La confiance dans le système correspond à l'attente que ce dernier se comportera de la manière attendue. Il est à noter que cette notion est assez proche de celle de la sûreté de fonctionnement. Les fonctionnalités attendues d'une plateforme de confiance sont une capacité de protection et d'isolation des processus, des mesures d'intégrité et la capacité à transférer ces mesures à un serveur distant afin que ce dernier puisse évaluer l'intégrité de la plateforme et prendre les mesures adéquates. On remarquera d'emblée que le TPM ne prend aucune décision et que c'est la plate forme qui effectue les mesures et les stocke dans le TPM.

Les capacités de protection représentent un ensemble de commandes avec des autorisations permettant l'accès à des zones protégées (mémoires, registre) dans lesquels il est possible de faire des manipulations sur des données sensibles. Le TPM possède de telles zones comme les registres PCR (*Platform Configuration Register*) destinés au stockage des mesures d'intégrité. Le TPM stocke aussi des clés cryptographiques pour authentifier les mesures réalisées.

Le fonctionnement d'une plate-forme informatique commence par le processus d'amorçage, le système d'exploitation est ensuite chargé, puis les applications sont exécutées, notamment les clients de courrier électronique ou les navigateurs. A chacune de ces étapes une mesure de l'état du système appelée attestation, est réalisée et stockée dans le TPM.

Une mesure d'intégrité est donc la mesure à un instant donné d'une partie pertinente de l'état d'un système ou d'élément pouvant affecter l'intégrité du système. Des entités externes effectuent ces mesures et les stockent dans les PCR de manière fiable. Comme les mesures sont effectuées par logiciel ces derniers peuvent être corrompus et indiquer des mesures ne correspondant pas à la réalité. Cependant la première mesure est effectuée par un BIOS modifié dont le logiciel a été audité comme fiable et stocké dans une partie non altérable du BIOS (flashage ultérieur impossible). Dès lors cette amorce, appelé *Root Of Trust* est capable de mesurer le premier élément logiciel suivant. L'ordinateur distant demandant l'état du système sera à même de détecter tout changement par rapport à la mesure lors du précédent démarrage.

Une attestation correspond à l'assurance de l'intégrité de cette mesure. Pour éviter toute modification lors du transfert vers un tiers, il est nécessaire de garantir sa non altération et aussi l'origine ou l'identité de la plateforme émettrice. Si la mesure correspond à un hachage du code, l'attestation correspond à une signature à l'aide d'une clé AIK (*Attestation Identity Key*). Il s'agit donc d'une opération qui prouve qu'une plateforme peut être fiable afin d'envoyer ces mesures à un tiers et fournit aussi la preuve de son identité.

Le fait que le TPM contienne une clé EK (*Endorsement Key*) soit un couple de clefs publiques/privée spécifique à chaque TPM: cela permet d'avoir une clef unique qui permet d'identifier le TPM de manière non univoque. Cette clef peut être ou ne pas être utilisée, tout comme le TPM peut être, normalement, totalement désactivé. Cette fonctionnalité, outre son utilité directe permettant d'identifier la machine, à son revers: celui de la vie privée. Toutefois, des fonctionnalités sont prévues dans les spécifications pour contrôler l'accès à cette clef.

Il est possible de définir de multiples identités chacune pouvant correspondre à des utilisations différentes de la machine. Il est possible de créer des *Attestation Identity Keys* (AIK), en nombre illimité, qui sont des *alias* de la clé EK. Globalement unique également, ils permettent de ne pas divulguer la clé (nécessairement unique et immuable à la machine) et donc de se créer des sortes d'identités virtuelles associées à la machine. Identités que l'on peut faire disparaître à loisir.

Il est possible de désactiver/réactiver, d'administrer un TPM avec la possibilité de "prendre possession" du TPM (*take ownership*), c'est-à-dire de placer un secret partagé au sein du TPM. La personne qui connaîtra ce secret sera reconnu par le TPM comme étant son propriétaire et aura donc des droits étendus sur celui-ci. Notamment celui de modifier les paramètres liés à la divulgation (ou non) de la clé EK.

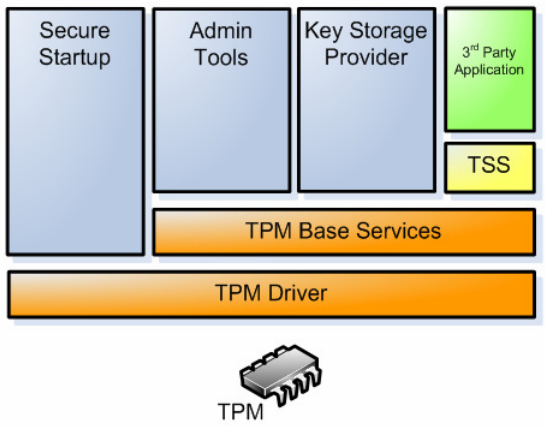
Il est possible d'auditer (et donc conserver une trace) des commandes exécutées sur le TPM. Cette vérification a posteriori ne peut être réalisée que par le propriétaire du TPM. Une certaine forme de délégation de pouvoir du propriétaire sur les commandes qu'il peut utiliser est possible. L'objectif est de pouvoir déléguer certains droits du propriétaire à des processus de confiance afin qu'ils puissent exploiter les services offerts par le TPM.

Windows et NGSCB

Les solutions logicielles aux problèmes de sécurité ont toujours des vulnérabilités. En effet, ces solutions utilisent un espace mémoire partagé et repose sur le système d'exploitation pour gérer la mémoire physique. Comme elles reposent sur le système elles doivent faire face aux vulnérabilités propre du système. Une solution basée sur du matériel permet de pallier les vulnérabilités logicielles. L'accès au matériel est plus facilement contrôlable et de plus moins sujet aux modifications non souhaitées.

L'utilisation d'un TPM permet de gérer de manière plus fiable les clés cryptographiques en les stockant dans des zones mémoire séparées des zones sous contrôle du système. de plus le TPM n'exécute pas de logiciel du système d'exploitation et n'est pas sujet aux vulnérabilités de ce dernier.

La figure suivante montre l'architecture générale d'utilisation du TPM par NGSCB. La couche d'accès au TPM est un driver conforme à la spécification TCG 1.2 permettant une meilleure stabilité de la plateforme. Le composant TBS (TPM Base Services) est en charge du partage des ressources du TPM. Les composants *Secure Startup*, *Admin Tools*, et *Key Storage Provider*, sont fournis par Microsoft et utilisent les services du TPM. Les applications tierces sont exécutées au dessus de la pile TSS (*TCG Software Stack*) qui est la spécification de l'API d'accès au TPM.



Le *Secure Startup* est conçu afin d'utiliser le TPM pour garantir l'intégrité de la partition Windows (les applications, les DLL et les fichiers stockés dans cette partition) et d'autoriser l'accès à la zone protégée de données uniquement après s'être assuré de l'intégrité de la séquence de démarrage du système. Ce composant communique avec le TPM dans les premiers instants du démarrage afin de contrôler les modifications intervenues depuis le précédent démarrage. Les fonctions de démarrage "en confiance" offrent la possibilité de stocker, dans des PCR, des hachages d'informations sur la configuration durant la séquence de démarrage. Une fois que l'ordinateur est démarré, des données (telles que des clés symétriques pour des fichiers cryptés) peuvent être "scellées" sous un PCR. Les données scellées ne peuvent être descellées que si le PCR possède la même valeur qu'au moment du scellage. Si on essaie de faire démarrer un autre système, ou si un virus a introduit un *back-door* dans le système d'opération, la valeur PCR ne correspondra pas et le descellement échouera, protégeant ainsi les données.

Microsoft fournit une interface WMI (*Windows Management Instrumentation*) qui encapsule la majeure partie des tâches administratives liées au TPM. Les administrateurs peuvent localement ou à distance, effectuer toute modification sur le TPM voir l'activer ou le désactiver (*opt-in*) en fonction des stratégies de sécurité de l'entreprise. Les fonctions d'initialisation et de gestion permettent au propriétaire d'utiliser ou non la puce, de la remettre à zéro, et d'en prendre possession. Ce groupe de fonctions est un peu complexe, afin d'assurer une séparation forte entre ce qui peut être fait au niveau du BIOS (démarrage), et ce qui peut l'être lors du fonctionnement normal de l'ordinateur, pour qu'aucun logiciel malicieux puisse effectuer des opérations sensibles.

Le dernier composant est le KSP (*Key Storage Provider*), il permet aux applications utilisant le TPM de générer de manière sûre des clés privées, d'encrypter des données et de signer. Par défaut, les clés générées sont sauvegardées dans le profil utilisateur. Néanmoins, il est possible de générer des extensions des registres PCR par le TPM pour le chiffrement de données.

Il existe de nombreuses déformation de cette approche et il est important de se rappeler TCG n'est pas NGSCB, NGSCB utilise un composant le TPM définit par TCG. Les spécifications de TCG ne définissent pas de fonctionnalités de DRM. Ce n'est juste qu'une application possible d'un composant "de confiance". Une puce TPM ne contrôle pas l'exécution, ni ne bloque une exécution. Ce que fait un TPM c'est de fournir une protection des clés privées et des données chiffrées d'un utilisateur, il permet de protéger des données sensibles de beaucoup d'attaques logicielles, y compris des virus, des vers et des chevaux de Troie

Conclusion

L'informatique fiable ou de confiance a pour but de sécuriser le système, afin qu'il soit inviolable et qu'il fonctionne d'une façon reconnue. Ses objectifs comprennent la confidentialité, la fiabilité, la sécurisation des communications, la prédictibilité et l'autorisation de la connectivité et des accès. L'informatique fiable consiste en une approche globale de la sécurité, qui inclut : l'amorçage, l'accès au système, l'accès aux périphériques, l'exécution d'applications et l'échange d'informations au moyen du courrier électronique. La sécurité dépend des avancées dans chacun de ces domaines, mais aucun d'entre eux, pris séparément, n'est suffisant.

8. Conclusion

Jusqu'à récemment la programmation des cartes à puce était réalisée en assembleur et longuement vérifiée avant la phase de masquage. La phase de masquage est une phase durant laquelle le système d'exploitation de la carte est figé dans la ROM de la carte sur le silicium. Le programme contenait à la fois le système opératoire et l'application. Le processus de développement était donc long ce qui était dû à la fois au langage de programmation et à la campagne de test associée au développement. Dans un tel schéma, toute évolution du code est difficile voire impossible. Ce processus de développement long et coûteux semble inadapté à certains marchés nécessitant un temps de déploiement (*time-to-market*) réduit. Dans ce cadre l'essentiel des problèmes de sécurité était lié aux attaques matérielles.

Depuis quelques années, sont apparues de nouvelles cartes à puce pouvant répondre à ces marchés. Elles sont basées sur des systèmes ouverts et autorisent le chargement dynamique de code. Généralement ces cartes permettent le chargement de plusieurs applications sur la même carte. Les cartes basées sur la norme *Java Card*, *MultOS* ou bien *Windows for Smart Card* font partie de cette nouvelle génération. Ces systèmes opératoires utilisent des machines virtuelles afin d'améliorer la portabilité, la compacité du code et la sécurité. En particulier le standard Java Card apporte à la carte à puce l'interopérabilité du langage Java. Ainsi, les applications développées pour Java Card peuvent s'exécuter sur toute carte implémentant la machine virtuelle Java Card. Le code Java Card généré est un code mobile qui peut transiter sur un réseau avant d'être chargé sur une carte à puce pour y être exécuté. La Java Card prévoit en effet la possibilité de charger du code après son déploiement. Cependant, le schéma de déploiement n'est pas exempt de problème

La possibilité de charger du code dans la carte en dehors d'un site sécurisé accroît les risques d'attaque logiques. Le risque est d'autant plus grand si les différentes applications chargées dans la carte peuvent échanger des informations entre elles. En effet avec l'apparition des cartes multi-applicatives, il devient possible de mener une attaque non plus contre la carte dans son ensemble mais contre une partie de celle-ci et ce depuis une autre partie. Il est donc nécessaire de se prémunir contre des applications hostiles. Concevoir des mécanismes de vérification embarqués est un défi au vu des ressources limitées de la carte, s'assurer de l'absence de vulnérabilité d'un tel composant en est un autre. C'est pourquoi l'utilisation modèles mathématiques et d'outil de vérification formel commencent à être utilisé dans le domaine de la carte à puce.

La carte est un outil de sécurité et prône en tant que tel une fermeture afin de minimiser les risques d'attaque. Cependant c'est dans l'ouverture vers l'extérieur et la connexion aux autres systèmes vers lesquels les acteurs de la carte se dirigent afin de répondre aux exigences du marché. Il faut, en plus de proposer de nouvelles fonctionnalités et de nouveaux services, continuer d'assurer le même niveau de sécurité qu'auparavant, c'est-à-dire garantir l'intégrité, la confidentialité et la disponibilité des données et des applications de la carte. De nouveaux mécanismes de sécurité doivent être mis en place afin de garantir la sécurité de la carte et lui permettre ainsi de préserver ses qualités de sécurité et de fiabilité dans les futures applications.

Bibliographie

Générale

1. Hendry, Michael, *Smart card security and applications*, Artech House, Boston, Mass, 1997
2. W. Rankl and W. Effing, *Smart Card Handbook*, John Wiley & Sons Ltd., Chichester, 1997
3. Smart Card Trends, les news de l'industrie de la carte : http://www.smartcardstrends.com/
4. Card Werk, le standard ISO 7816 : http://www.cardwerk.com/smartcards/ smartcard_standard_ ISO7816-1.aspx
5. Bruce Schneier *Applied Cryptography* Wiley, ISBN 0-471-12845-7
6. Pfleeger, *Security in Computing* Prentice Hall ISBN 0-13-799016-2
7. Hendry, *Smart card security and applications* Hartec HouseISBN 0-890006-953-0
8. Denning, *Cryptography and data security* Wesley, ISBN 0-201-10150-5

Certification

1. *The Common Criteria for Information Technology Security Evaluation (CC)*. August 1999. Version 2.1. Technically identical to International Standard ISO/IEC 15408:1999. http://csrc.nist.gov/cc/ccv20/ccv2list.htm
2. International Organization for Standardization (ISO). ISO/IEC TR 13335. *Guidelines for the Management of IT Security (GMITS)*. Note that this is a five-part technical report (not a standard);
3. R. Kruger and J. Eloff. Common Criteria Framework for the Evaluation of Information Technology Security Evaluation. In *IFIP TC11 13th International Conference on Information Security, (SEC'97)*, pages 197-209, 1997.
4. http://csrc.nist.gov/publications/nistir/nistir-7056.pdf

Attaques

1. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C. : *The Sorcerer's Apprentice Guide to Fault Attacks*. In : Proceedings of Workshop on Fault Detection and Tolerance in Cryptography, Italy (2004)
2. Coron, J.S., Kocher, P., Naccache, D. *: Statistics and Secret Leakage*. In : Proceedings of Financial Cryptography (FC2000). Volume 1962 of Lecture Notes in Computer Science., Springer-Verlag (2001) 157–173
3. Gandol, K., Mourtel, C., Olivier, F. : *ElectroMagnetic Analysis : Concrete Results*. In : Proceedings of CHES'2001. Volume 2162 of Lecture Notes in Computer Science., Springer-Verlag (2001) 251–261

4. Giraud, C., Thiebeauld, H. : *A survey on fault attacks*. In : Proceedings of CARDIS'04, Smart Card Research and Advanced Applications VI, Toulouse, France, Kluwer academic publisher (2004) 159–176
5. Hubbers, E., Poll, E. : *Reasoning about Card Tears and Transactions in Java Card*. In : Fundamental Approaches to Software Engineering (FASE'2004). Volume 2984 of Lecture Notes in Computer Science., Barcelona, Spain, Springer-Verlag (2004) 114–128
6. Kocher, P. : *Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems*. In : Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1996) 104–113
7. Kömmerling, O., Kuhn, M.G. : *Design Principles for Tamper-Resistant Smartcard Processors*. In : Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 9–20
8. Kocher, P., Jaffe, J., Jun, B. : *Differential Power Analysis*. In : Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1999) 388–397
9. Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G. : *Improving Smart Card Security using Self-timed Circuits*. In : Proceedings of ASYNC'02. (2002) 211–218
10. Quisquater, J.J., Samyde, D. : *ElectroMagnetic Analysis (EMA) : Measures and Countermeasures for Smart Cards*. In : Proceedings of E-smart 2001. Volume 2140 of Lecture Notes in Computer Science., Springer-Verlag (2001) 200–210
11. Skorobogatov, S., Anderson, R. : *Optical Fault Induction Attacks*. In : Proceedings of Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2002), San Francisco Bay (Redwood City), USA (2002)

Java Card

1. Bieber, P., Cazin, J., Girard, P., Lanet, J.L., Wiels, V., Zanon, G. : *Checking Secure Interactions of Smart Card : Applets* Extended Version. Computer Security 10 (2002) 369–398 Special issue on ESORICS 2000.
2. Chen, Z. and Di Giorgio, R. *Understanding Java Card 2.0*, March 1998, http://www.javaworld.com/javaworld/jw-02-1998/jw-03-javadev.html
3. Chen, Z. : Java Card *Technology for Smart Cards : Architecture and Programmer's Guide*. Addison-Wesley (2000)
4. Montgomery, M., Krishna, K. : *Secure Object Sharing in Java Card*. In : Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999)
5. Thomas Schaeck and Rinaldo Di Giorgio, *How to write OpenCard card services for Java Card applets*, JavaWorld, Oct 1998, http://www.javaworld.com/javaworld/jw-10-1998/jw-10-javadev.html
6. Oestreicher, M. : *Transactions in Java Card*. In : Proceedings of 15th Annual Computer Security Applications Conference (ACSAC), Phoenix, Arizona, USA (1999)

Les TPM

1) R. Anderson, "*Trusted Computing Frequently Asked Questions*", version 1.1, http://www.cl.cam.ac.uk/%7Erja14/tcpa-faq.html

2) P. England, B. Lampson et al, "*A Trusted Open Platform*", IEEE Computer Magazine, July 2003, http://download.microsoft.com/download/c/8/0/c80ea683-9900-46ff-9c67-d7f14b0d3787/trusted_open_platform_ieee.pdf

3) Microsoft Product Information, "*The Next-Generation Secure Computing Base: An Overview*", April 2003, http://www.microsoft.com/resources/ngscb/NGSCB_Overview. mspx

4) S. Pearson, B. Balacheff, et al, "*Trusted Computing Platforms"* HP professional Books, Prentice Hall PTR, 2003

5) N. Sumrall, B. Meinschein, S. Cannady , "*Safer Computing Usage Models in the business Environment*", Intel Developer Conference Spring 2003, http://cnscenter.future.co.kr/resource/rsc-center/presentation/intel/spring2003/S03USSFCS68_OS.pdf

6) Trusted Computing Group, https://www.trustedcomputinggroup.org/

Les sites des fabricants de carte

1. Axalto http://www.axalto.com/
2. Gemplus Ltd., http://www.gemplus.com
3. Giesecke & Devrient, http://www.gdm.de
4. Oberthur card system http://www.oberthurcs.com/pages/home/home.asp
5. Aspects Software (www.aspects-sw.com)
6. Card Base technology http://www.cardbase.com/
7. GlobalPlatform : GlobalPlatform. (http://www.globalplatform.org/)
8. Hive-Minded : Smartcard.NET. (http://www.hiveminded.com/)
9. IBM smart card solutions, http://www-306.ibm.com/software/wireless/wecos/tools.html
10. Mondex Solutions, http://www.mondex.com/ ou Multos (www.multos.com)
11. Mobile Mind http://www.mobile-mind.com/
12. Orga Card Systems Ltd., http://www.orga.com
13. TCG http://www.trustedcomputing.org
14. Trusted Logic http://www.trusted-logic.fr
15. ZeitControl Basic Card, http://www.basiccard.com/

GLOSSAIRE

AES - Advanced Encryption Standard. Algorithme de chiffrement basé sur la cryptographie symétrique, AES est le remplaçant du DES. Face aux vulnérabilités du DES vis-à-vis des attaques par force brute, l'institut NIST a lancé en janvier 1997 un appel d'offre pour le remplacement du DES par un nouvel algorithme de chiffrement par blocs de 128 bits, supportant des clés de chiffrement de 128, 192 et 256 bits minimum.

Authentification - Service de sécurité dont l'objectif est de valider l'identité d'une entité (utilisateur ou équipement). Il existe classiquement trois méthodes d'authentification permettant de prouver l'identité d'une entité :
- Authentification basée sur la connaissance d'un secret,
- Authentification basée sur la possession d'un objet,
- Authentification basée sur la biométrie.

Biométrie - Technique consistant à étudier les caractéristiques physiques d'une personne, comme une empreinte digitale, la géométrie de la main, la structure de l'œil ou le modèle de la voix.

CA - Certificate Authority - Une autorité de certification est un organisme tiers vérifiant l'identité d'un utilisateur au moyen d'un ensemble d'exigences en vue de délivrer un certificat numérique. Les autorités de certification proposent différents niveaux d'assurance, en fonction de la diligence mise en œuvre pour vérifier une identité individuelle.

Certification - Processus selon lequel une tierce partie assermentée atteste l'authenticité de l'identité d'un utilisateur.

Certificat de clé publique - Document signé de manière numérique par une autorité de certification, basé sur une vérification d'identité réalisée par une autorité d'enregistrement et contenant la clé publique de l'individu, une forme d'identité de l'individu ainsi qu'une période de validité limitée. Le format standard de certificat est X.509 v.3 (norme PKI-X). Les informations certifiées sont notamment :
- L'identité du porteur.
- La clé publique du porteur.
- La durée de vie du certificat.
- L'identité de l'autorité de certification émettrice.
- La signature de l'autorité de certification émettrice.

Challenge/Response - Autrement appelé stimulation/réponse en terminologie française, challenge/response est un procédé d'authentification basé sur un défi émis par un serveur. Le client ne peut relever ce défi que s'il possède un secret particulier. Le succès du défi prouve l'identité du client. Remarque : le protocole NTLM utilisé dans les systèmes Windows NT de Microsoft repose sur ce principe.

Clé privée - Clé exclusivement connue par un utilisateur individuel.

Clé publique - Clé largement publiée.

Cryptanalyse - Étude de la sécurité des procédés cryptographiques. La cryptanalyse consiste à déchiffrer un message dont on connaît généralement le procédé dechiffrement, mais pas les secrets.

Cryptographie - Science de la conservation sécurisée d'informations par des moyens et méthodes destinés à la transformation de données afin de cacher leur contenu, d'empêcher leur modification et leur utilisation frauduleuse.

DES - Data Encryption Standard - L'algorithme de cryptage de données (DES) est un système cryptographique symétrique, avec une taille de bloc de 64 bits et utilise une clé de 56 bits durant l'exécution (8 bits de parité sont enlevés de la clé complète de 64 bits). Dans le cadre de la communication, l'émetteur et le récepteur doivent connaître la même clé secrète qui peut être utilisée pour crypter ou décrypter le message ou encore pour générer et vérifier un code d'authentification de message (MAC).

ECC - Elliptic Curve Cryptosystem - Système cryptographique à clé publique basé sur les propriétés de courbes elliptiques.

EEPROM - Electrically erasable programmable read only memory.

Fonction de hachage - Fonction non réversible qui associe un ensemble de chaînes de caractères arbitraires à un ensemble de chaînes d'octets de longueur fixe. Une fonction de hachage résistant à la collision possède la propriété selon laquelle il est impossible de construire, par un calcul sur ordinateur, des données d'entrées distinctes associées aux mêmes données de sortie

GSM - Global System of Mobile Communications - Norme paneuropéenne pour téléphones mobiles.

Intégrité - Capacité à déterminer que les données reçues sont identiques aux données émises.

ISO (International Standards Organization) - La norme ISO 7816 définit les caractéristiques physiques, électriques ainsi que le protocole des cartes à puce.

Non-répudiation - Condition dans laquelle l'émetteur d'un message ne peut pas renier la validité du résultat du processus utilisé pour authentifier les données.

Personnalisation - Modification d'une carte à puce afin qu'elle représente des informations concernant une seule personne. Il existe deux types de personnalisation : graphique et logique. La personnalisation graphique modifie l'aspect visuel de la carte (nom du détenteur, photographie), la personnalisation logique les informations qu'elle contient sous forme électronique.

Signature numérique - Technique utilisant le système cryptographique à clé publique pour prouver qu'un message n'a pas été altéré.

Triple DES (3 DES) - Identique à DES, si ce n'est que les données d'entrée sont en fait cryptées trois fois avec deux ou trois clés.

PIN - Personal Identification Number - Numéro ou code que le détenteur d'une carte doit saisir pour confirmer qu'il est l'unique propriétaire de la carte.

RAM - Mémoire vive, mémoire à accès aléatoire – une mémoire réinscriptible dans l'ordinateur où les données actuellement utilisées sont conservées pour un accès rapide par la carte à puce.

ROM - Read only memory (mémoire ROM masquée).

RSA - L'un des premiers algorithmes à clé publique développé par Rivest, Shamir et Adelman. Il s'agit d'une norme ISO, basé sur la cryptographie asymétrique, RSA est l'algorithme le plus utilisé dans le monde et offre les services de sécurité essentiels tels que l'authentification, la confidentialité, l'intégrité et la signature. RSA connaît un essor important avec les développements des infrastructures de gestion de clés (PKI)

ICP - Infrastructure à clé publique - Cryptage d'informations au moyen de deux clés différentes possédant une relation mathématique pour le cryptage et le décryptage. Je crypte un message avec ma clé privée et vous le décryptez avec ma clé publique que vous vous êtes procuré sur un serveur de répertoires ou que je vous ai donnée. Si vous souhaitez m'envoyer un message, vous le cryptez avec ma clé publique et je le décrypte avec ma clé privée.

SIM - Subscriber Identification Module. Une carte à puce SIM contient les informations relatives à l'abonné dont son numéro de téléphone cellulaire.

SIM Toolkit - Une fonction du GSM permettant de télécharger des données via des codes insérés dans des SMS. C'est un moyen de mettre à jour les applications fonctionnant sur le téléphone et de permettre des transactions sécurisées nécessaires notamment pour le développement des applications bancaires mobiles.