Une image contenant capture d’écran, texte, bleu vert, Rectangle

Description générée automatiquement

Une image contenant Police, Graphique, texte, logo

Description générée automatiquement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SPECIFICATIONS | | | | |
|  | | | | |
|  | | | | |
| Statut : En cours | | | Classification : Public | | | Version : V1.0.1 |

**Historique du document**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Rédigé par | | Vérifié par | | Validé par | |
| 0.1 | ASIP | Le 18/12/2018 | ASIP | Le 18/12/2018 | ASIP | Le 18/12/2018 |
| Motif et nature de la modification : Création du document | | | | | |
| Motif et nature de la modification : | | | | | |
| 1.0.1 | Support de la carte CPS4, mise à la charte graphique ANS | | | | | |

**SOMMAIRE**

[1. Préambule 4](#_Toc176858091)

[2. Introduction 5](#_Toc176858092)

[3. La carte CPS3 6](#_Toc176858093)

[4. La carte CPS4 7](#_Toc176858094)

[5. Architecture CryptoLib CPS3 8](#_Toc176858095)

[5.1. Projet Visual Studio de la CryptoLib CPS3 9](#_Toc176858096)

[6. OpenSC 10](#_Toc176858097)

[6.1. Présentation 10](#_Toc176858098)

[6.2. Modifications OpenSC 10](#_Toc176858099)

[6.2.1. Configuration par défaut 11](#_Toc176858100)

[6.3. Adaptations PKCS#11 11](#_Toc176858101)

[7. Support de la carte CPS4 15](#_Toc176858102)

[7.1. Driver carte 15](#_Toc176858103)

[7.1.1. Initialisation 15](#_Toc176858104)

[7.1.2. Transmission des APDUs 16](#_Toc176858105)

[7.1.3. Sélection de fichier sur la carte 17](#_Toc176858106)

[7.1.4. Gestion des codes PIN de la carte 18](#_Toc176858107)

[7.1.5. Opérations cryptographiques 19](#_Toc176858108)

[7.1.6. Gestion des mises à jour de carte 22](#_Toc176858109)

[7.2. Support PKCS#15 des nouveaux objets CPS2ter 22](#_Toc176858110)

[7.2.1. Initialisation 22](#_Toc176858111)

[7.2.2. Utilisation 22](#_Toc176858112)

[8. Support CPS3 23](#_Toc176858113)

[8.1. Gestion IAS 23](#_Toc176858114)

[8.2. Driver carte 23](#_Toc176858115)

[8.2.1. Initialisation 24](#_Toc176858116)

[8.2.2. Sélection de fichier sur la carte 25](#_Toc176858117)

[8.2.3. Gestion des codes PIN de la carte 26](#_Toc176858118)

[8.2.4. Opérations cryptographiques 27](#_Toc176858119)

[8.2.5. Gestion des mises à jour de carte 28](#_Toc176858120)

[9. Support du GALSS 30](#_Toc176858121)

[9.1. Gestion du driver 30](#_Toc176858122)

[9.2. Gestion des lecteurs GALSS 31](#_Toc176858123)

[9.3. Gestion de la carte 32](#_Toc176858124)

[10. Spécificités ASIP-Santé 34](#_Toc176858125)

[10.1. Gestion du cache 34](#_Toc176858126)

[10.1.1. Présentation 34](#_Toc176858127)

[10.1.2. Implémentation 34](#_Toc176858128)

[10.2. SmartCard Logon 37](#_Toc176858129)

[10.3. Initialisation du module PKCS#11 38](#_Toc176858130)

[10.3.1. Configuration des traces 38](#_Toc176858131)

[10.3.2. Initialisation des drivers lecteurs 39](#_Toc176858132)

[10.3.3. Initialisation des drivers carte 39](#_Toc176858133)

[10.3.4. Détection des lecteurs 40](#_Toc176858134)

[10.3.5. Détection de présence carte 40](#_Toc176858135)

[10.3.6. Détection du type de carte 41](#_Toc176858136)

[10.3.7. Enregistrement des objets carte 42](#_Toc176858137)

[10.4. Traces 44](#_Toc176858138)

[10.4.1. Traces PKCS#11 44](#_Toc176858139)

[10.4.2. Traces OpenSC 48](#_Toc176858140)

# Préambule

Documents de référence

|  |  |  |
| --- | --- | --- |
| Appellation | Type du document | Nom du document |
| [R1] | Spécifications | PKCS #11 v2.20: Cryptographic Token Interface Standard |
| [R2] | Spécifications | ASIP\_CPS3\_Données-métier\_v1.0.2 |
| [R3] | Spécifications | ANS1-001\_Profil-Electrique-SSCD-Chipdoc4\_v1-3.pdf |

Terminologies et abréviations

|  |  |
| --- | --- |
| Abréviation | Nom du document |
| CPS | Carte de Professionnel de Santé |
| GALSS | Gestionnaire d’Accès au Lecteur Santé Social |
| PSS | Protocole Santé Social |
| CryptoLib | Ensemble des composants librairie PKCS#11, CSP, CCM, TokenDriver |
| PC/SC | Personal Computer/Smart Card |
| CSP | Cryptographic Service Provider |
| PKCS#11 | Standard définissant une interface générique d’accès aux périphériques cryptographiques. (Public Key Cryptographic Standards) |
| CAPI | Microsoft Cryptography API |
| CCM | CAPI Certificate Manager |
| PKCS#15 | Cryptographic Token Information Format Standard |
| IAS ECC | Identification, Authentification, Signature, Carte Européenne du Citoyen |
| PIN | Personal Identification Number |
| AMO | Assurance Maladie Obligatoire |

# Introduction

La Direction de l’ASIP-Santé a acté la réalisation d’une nouvelle version de ses librairies cryptographiques (ou CryptoLib) afin d’intégrer l’accès à la CPS3. Cette CryptoLib offre une compatibilité ascendante pour la prise en compte de la CPS2ter par les applications actuellement sur le terrain.

Cette CryptoLib a été réalisée pour les environnements Windows, Mac et Linux. Des spécifications ont été rédigées afin de permettre aux éditeurs de réaliser eux-mêmes des portages vers d’autres OS cibles.

Un objectif secondaire est d’engager la fin de vie des API-CPS. Lors de la migration, toutes les applications souhaitant mettre en œuvre la CPS3 devront s’appuyer sur la CryptoLib.

Le présent document constitue les spécifications internes des adaptations effectuées sur la libraire PKCS#11 OpenSC en vue d’adresser :

* le support des cartes CPS3 au format IAS ECC,
* le support des cartes CPS4 au format NXP avec la nouvelle applet ChipDoc

# La carte CPS3

La carte CPS3 dans la nouvelle architecture CryptoLib est le successeur de la carte CPS2ter. A cette fin, il s’agit de ne pas provoquer une rupture de service avec les applicatifs du poste PS fonctionnant sur la CPS2ter.

Cette nouvelle carte va offrir une compatibilité ascendante en proposant un fonctionnement bi – mode :

* Un mode CPS3 conforme au standard IAS ECC comprenant
  + Les données du volet CPS2ter codifiées selon la norme IAS
  + Une partie sans contact
* Un mode CPS2ter comprenant
  + Les données du volet CPS2ter codifiées selon le standard PKCS#15

# La carte CPS4

La carte CPS4 dans l’architecture CryptoLib actuelle est le successeur de la carte CPS3. A cette fin, il s’agit de travailler indifféremment sur tous les objets des volets CPS et CPS2ter vus comme des objets PKCS#11 pour les applicatifs du poste PS fonctionnant actuellement sur la CPS3v3.

Cette nouvelle carte va offrir une compatibilité ascendante en proposant un fonctionnement bi – mode :

* Un mode CPS3 conforme au format NXP (et sa nouvelle applet ChipDoc) comprenant
  + Les données du volet CPS2ter codifiées selon le standard PKCS#15
  + Les données du volet CPS accessibles comme actuellement sur la carte CPS3v3

# Architecture CryptoLib CPS3

Le schéma ci-dessous présente l'architecture fonctionnelle la CryptoLib CPS3.



Figure : Architecture CryptoLib CPS3

Plusieurs choses sont à noter par rapport à ce schéma d’origine de l’architecture :

* Le driver Galss v3 est toujours présent dans le code la CryptoLib CPS3 mais n’est plus utilisé sur le terrain, ceci avec l’abandon des lecteurs PSS Sesam-Vitale.
* Le driver ca rte CPS2ter a été retiré de l’architecture
* Un nouveau driver carte CPS4 a été développé pour adresser le support de la carte CPS4
* Lorsqu’il s’agit d’adresser la carte CPS4, les objets CPS2ter ne sont plus émulés mais sont maintenant bien présents physiquement sur la carte CPS4 et lus tels quels

Dans les chapitres qui suivent, nous ne décrirons que l’architecture de la librairie représentée par le rectangle intitulé ‘Librairie PKCS#11 pour cartes CPS2ter… ’

## Projet Visual Studio de la CryptoLib CPS3

Dans la suite du document, toute référence à l’implémentation C d’une fonctionnalité de la CryptoLib CPS3 est indiquée par un tableau à 3 colonnes.

Quand il s’agit de la fonctionnalité elle-même :

* La première colonne indique le projet et le fichier source ou est implémentée la fonctionnalité,
* La deuxième colonne indique le nom de la fonction qui rend ce service,

Quand il s’agit de la rubrique ‘Appel’

* La première colonne indique le projet et le fichier source ou est implémentée la fonctionnalité,
* La deuxième colonne indique le nom de la fonction qui appelle ce service

|  |  |
| --- | --- |
| Projet / Fichier source | Fonction |
| Implémentation de la configuration des traces | |
| libopensc / ctx.c | load\_parameters(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | process\_config\_file() |

Figure : Exemple de référence au code source

# OpenSC

Le module PKCS#11 CryptoLib CPS3 repose sur le framework OpenSC de version 0.11.2.

## Présentation

OpenSC propose une architecture permettant de gérer les cartes à puces au format PKCS#15. OpenSC propose deux composants permettant respectivement d'initialiser et d'utiliser ce type de carte. Dans le cadre du projet de support de la carte CPS3, seul le composant permettant l’appel aux fonctions cryptographiques de la carte et de lire et d’écrire des objets PKCS#11 sera utilisé.

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

Figure : Architecture OpenSC

Comme nous l’avons indiqué précédemment, OpenSC utilise en interne le format PKCS#15 pour gérer les données de la carte. Pour supporter un nouveau type de carte, il faut donc implémenter un driver carte spécifique.

Remarque : une carte peut être PKCS#15 et nécessiter, malgré tout, l’écriture d’un driver carte spécifique pour réaliser des ordres carte qui sortiraient du standard (ISO7816-4).

Si le contenu de la carte n'est pas au format PKCS#15, OpenSC prévoit l’implémentation d’un émulateur qui permettra de transformer le contenu de la carte au format PKCS#15.

Pour envoyer des ordres vers la carte, OpenSC gère des drivers lecteurs tels que celui s’appuyant sur le standard PC/SC. Le support d’un nouveau type de lecteurs requiert l’ajout d’un driver lecteur correspondant.

## Modifications OpenSC

### Configuration par défaut

La CryptoLib CPS3 repose sur une configuration par défaut afin de rendre désormais optionnelle l’utilisation du fichier **opensc.conf** par OpenSC.

La configuration par défaut est une constante chaîne de caractères OPENSC\_CONFIG\_STRING définie dans le fichier opensc\_conf.h.

Les paramétrages définis sont les suivants.

* ‘reader\_drivers’ : liste des noms des drivers lecteurs à charger,
* section ‘reader\_driver’
  + ‘connect\_reset’ : autoriser le reset de la carte en cas de déconnexion du driver (seulement pour le PCSC),
* ‘card\_drivers’ : liste des noms des drivers carte à charger,
* section ‘card\_driver’
  + ‘no\_cache\_file\_list’ : ensemble de flags concernant les fichiers carte à ne pas mettre en cache dans le cas du driver carte ‘cps3’,
* section ‘framework pkcs15’
  + ‘use\_caching’ : utiliser le cache fichier ; dans le cas présent, la valeur est à ‘yes’
  + ‘enable\_pkcs15\_emulation’ : utiliser l’émulation PKCS#15 quand cela est nécessaire ; dans le cas présent, la valeur est à ‘yes’
  + ‘try\_emulation\_first’ : détection de la carte en utilisant l’émulation PKCS#15 d’abord ; dans le cas présent, la valeur est à ‘no’
  + ‘enable\_builtin\_emulation’ : utiliser l’émulation par défaut

## Adaptations PKCS#11

|  |  |  |
| --- | --- | --- |
| **Source** | **Fonction(s)** | **Action** |
| pkcs11\framework-pkcs15.c | pkcs15\_init\_slot | Adaptation ASIP : Le label du token ne doit pas contenir le label du code porteur. |
| pkcs11\slot.c | init\_slot\_info | Adaptation ASIP : Mise à blanc du ManufacturerID. |
| pkcs11\pkcs11-global.c | -Ajout d'un include - IC\_GetInfo | Adaptation ASIP : Personnalisation de la librairie avec notamment ajout de version, copyright, nom de produit… |
| pkcs11\pkcs11-global.c | IC\_GetTokenInfo | Mise à jour du compteur d'essais du code porteur et du code de déblocage au niveau de la récupération des informations carte (token). |
| pkcs11\framework-pkcs15.c | register\_mechanisms | Ajout de l'algorithme : RSA avec SHA-256.  Ajout de l'algorithme : RSA PSS avec SHA-1 et SHA-256. |
| pkcs11\framework-pkcs15.c | pkcs15\_set\_attrib pkcs15\_cert\_set\_attribute pkcs15\_prkey\_set\_attribute pkcs15\_pubkey\_set\_attribute pkcs15\_dobj\_set\_attribute | Ajout du chemin du fichier à modifier sur la carte pour la modification d'attribut. |
| pkcs11\framework-pkcs15.c | sc\_pkcs15\_update\_file | Ajout d'une fonction de modification de la valeur de l'objet qui ne nécessite pas de profil de carte. |
| pkcs11\misc.c | is\_session\_valid | Ajout d'une fonction permettant de tester la validité de la session. |
| pkcs11\framework-pkcs15.c | register\_mechanisms | Conditionner la possibilité de générer des clés à la capacité de la carte. |
| pkcs11\framework-pkcs15.c | register\_mechanisms | Conditionner l'ajout des algorithmes de signature avec hash en fonction de la configuration. |
| pkcs11\framework-pkcs15.c | register\_mechanisms | Remonter le flag CKF\_ENCRYPT dans le cas de l’utilisation de l’algorithme RSA\_PKCS |
| pkcs11\pkcs11-global.c | mutex\_create | Correction sur l'allocation mémoire du mutex. |
| pkcs11\framework-pkcs15.c | pkcs15\_create\_token | Gestion du cas du sans contact, dans lequel il n'y a pas de code porteur. |
| pkcs11\openssl.c | sc\_pkcs11\_register\_openssl\_mechanisms | Adaptations ASIP : Suppression des algorithmes non souhaités. |
| pkcs11\pkcs11-global.c | IC\_GetSlotInfo | Forçage de la détection des lecteurs. |
| pkcs11\pkcs11-global.c  pkcs11\slot.c | IC\_GetSlotList IC\_GetSlotList card\_detect | Gestion de la déconnexion/reconnexion des lecteurs. |
| pkcs11\pkcs11-global.c | déclaration d’un booléen unblock\_wait\_for\_slot\_event IC\_Finalize IC\_WaitForSlotEvent | Gestion du déblocage du WaitForSlotEvent pour Firefox lors de l'arrêt du programme. |
| pkcs11\pkcs11-global.c | IC\_WaitForSlotEvent | Gestion du waitForSlotEvent en mode bloquant. |
| pkcs11\pkcs11-global.c | IC\_GetSlotList | Non prise en compte des slots virtuels |
| pkcs11\slot.c | slot\_token\_removed | Conservation de l'id du slot |
| pkcs11\framework-pkcs15.c pkcs11\mechanism.c | pkcs15\_unbind sc\_pkcs11\_unregister\_all\_mechanisms | Libérer la liste des mécanismes associées à la carte pour éviter les doublons en cas de réintroduction. |
| pkcs11\framework-pkcs15.c | pkcs15\_set\_attrib | Modification de la valeur de l'objet uniquement dans le cas d'une initialisation PKCS#15. |
| pkcs11\pkcs11-session.c | IC\_Login | Interdiction de login SO si une session en lecture seule existe. |
| pkcs11\pkcs11-global.c | IC\_GetSlotList | Ne pas conditionner la détection des lecteurs au passage du paramètre pSlotList==NULL. |
| pkcs11\pkcs11-spy.c | init\_spy | Interdiction de charger un autre module PKCS#11 en version release. |
| pkcs11\pkcs11-object.c | IC\_DestroyObject | Ne pas supprimer l'objet en mémoire si la fonction n'est pas supportée. |
| pkcs11\slot.c | card\_detect | Pour éviter de boucler sur l'évènement slot en cas de changement de carte, tester si le retrait carte a déjà été effectué. |
| pkcs11\framework-pkcs15.c | pkcs15\_prkey\_sign | Si la revalidation du code porteur a échoué, il faut remettre le code d’erreur initial. |
| pkcs11\pkcs11-object.c | IC\_SetAttributeValue | Tester si la session permet de modifier un objet token. |
| pkcs11\pkcs11-object.c | IC\_SetAttributeValue | Tester si l'objet est modifiable. |
| pkcs11\pkcs11-object.c  pkcs11\pkcs11-session.c | IC\_CreateObject  IC\_DestroyObject  IC\_GetAttributeValue  IC\_SetAttributeValue  IC\_FindObjectsInit  IC\_FindObjects  IC\_FindObjectsFinal  IC\_DigestInit  IC\_Digest  IC\_DigestUpdate  IC\_DigestFinal  IC\_SignInit  IC\_Sign  IC\_SignUpdate  IC\_SignFinal  IC\_DecryptInit  IC\_Decrypt  IC\_GenerateKeyPair  IC\_UnwrapKey  IC\_SeedRandom  IC\_GenerateRandom  IC\_VerifyInit  IC\_Verify  IC\_VerifyUpdate  IC\_VerifyFinal  IC\_OpenSession  IC\_GetSessionInfo  IC\_Login  IC\_Logout  IC\_InitPin  IC\_SetPin | Tester la validité de la session. |
| pkcs11\mechanism.c | sc\_pkcs11\_encrypt\_init | Initialisation du chiffrement RSA : tester qu’il s’agit bien de la clé publique |
| pkcs11\mechanism.c | sc\_pkcs11\_encrypt | Chiffrement RSA effectif : utilisation des APIs OpenSSL 0.9.8n |
| pkcs11\openssl.c | sc\_pkcs11\_encrypt\_data | Chiffrement RSA effectif :avec les APIs OpenSSL 0.9.8n |

# Support de la carte CPS4

## Driver carte

Ce driver carte est une extension fonctionnelle du driver ISO 7816 inclus à l’origine dans OpenSC. Il est implémenté dans le fichier card-cps4.c.

Les fonctions ajoutées et exposées au framework OpenSC sont :

* cps4\_select\_file
* cps4\_match\_card;
* cps4\_init;
* cps4\_check\_sw;
* cps4\_set\_security\_env;
* cps4\_internal\_authenticate;
* cps4\_pin\_cmd;
* cps4\_compute\_signature;
* cps4\_get\_pin\_counter;
* cps4\_compute\_RSA\_priv;

Ces fonctions permettent à OpenSC d’adresser des appels à la carte CPS4 comme s’il s’agissait d’une carte au format ISO 7816.

### Initialisation

L’initialisation du driver carte se fait en quatre étapes :

* Lors de l’initialisation d’OpenSC, le driver est appelé sur sa fonction sc\_get\_driver(). Cette fonction doit enrichir la structure qui lui est passée en paramètre en renseignant les fonctions qu’il implémente. Dans le cas du driver CPS4 il remplace les pointeurs de fonctions ISO 7816 par les fonctions spécifiques à la carte CPS4
* Ensuite lorsqu’OpenSC a détecté la présence d’une carte, il interroge le driver pour lui demander s’il sait utiliser celle-ci au travers de la fonction cps2ter\_match\_card.
* Si le driver lui fournit une réponse positive, OpenSC va alors initialiser la carte en appelant la fonction cps4\_init.
* OpenSC interroge le driver carte pour récupérer le modèle de celle-ci (fonction ajoutée au cours de l’implémentation de la CryptoLib CPS3 afin d’enrichir la structure PKCS#11 CKA\_TOKEN\_INFO)

|  |  |
| --- | --- |
| Initialisation du driver | |
| libopensc / card-cps4.c | cps4\_init(), retourne une structure sc\_card\_driver contenant stockant la liste des fonctions implémentées par ce driver |
| Appel | |
| libopensc / ctx.c | load\_card\_drivers () |

|  |  |
| --- | --- |
| Reconnaissance de la carte | |
| libopensc / card-cps4.c | cps4\_match\_card(), prend en paramètre une structure sc\_card\_t initialisée par OpenSC. Cette structure contient l’ATR de la carte. La carte est reconnue si son ATR est celui de la carte CPS4. |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |

|  |  |
| --- | --- |
| Initialisation de la carte | |
| libopensc / card-cps4.c | cps4\_init(), prend en paramètre une structure sc\_card\_t. Cette fonction enrichit la structure avec des informations concernant la carte : nom, algorithme et taille de clés supportés etc… |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |

|  |  |
| --- | --- |
| Lecture du modèle NXP de la carte | |
| libopensc / card-cps2ter.c | cps4\_get\_model (), recopie dans le paramètre model la valeur « NXP » |
| Appel | |
| pkcs11 / framework-pkcs15.c | pkcs15\_init\_token\_info() |

### Transmission des APDUs

La fonction réalisant la transmission des APDU n’a pas été surchargée par rapport au driver ISO 7816. Par contre la fonction analysant les champs SW1 et SW2 de l’APDU a été surchargée pour prendre en compte les valeurs spécifiques à la carte CPS4.

|  |  |
| --- | --- |
| Analyse des codes erreur | |
| libopensc / card-cps4.c | cps4\_check\_sw(), la fonction prend en paramètre la structure card et les valeurs SW1 et SW2 à analyser. Elle utilise le tableau cps2ter\_errors (défini dans le fichier card-cps2ter.c pour déterminer le type d’erreur renvoyée par la carte |
| Appel | |
| libopensc / card.c | sc\_check\_sw () |

### Sélection de fichier sur la carte

Deux fonctions sont utilisées lors de la sélection d’un fichier. La fonction construisant et envoyant l’APDU de sélection de fichier et la fonction analysant la réponse de la carte à cette sélection. Une troisième fonction permet de gérer le cas particulier de sélection d’applet sur la carte.

|  |  |
| --- | --- |
| Sélection de fichier | |
| libopensc / card-cps4.c | cps4\_select\_file(), la fonction prend en paramètre la structure sc\_card ainsi que le path du fichier à sélectionner. Cette fonction va appeler la fonction cps2ter\_internal\_select\_file qui va réaliser l’envoi d’APDU et la lecture de la réponse de la carte. Ensuite elle appelle la fonction cps2ter\_process\_fci pour interpréter la réponse de la carte. |
| Appel | |
| libopensc / card.c | sc\_select\_file() |

|  |  |
| --- | --- |
| Analyse de la réponse à une sélection de fichier | |
| libopensc / card-cps4.c | cps4\_process\_fci (), analyse la réponse de la carte pour déterminer le type de fichier. La fonction positionne sur le fichier des informations permettant de l’utiliser par la suite. |
| Appel | |
| libopensc / card-cps4.c | cps4\_select\_file() |

|  |  |
| --- | --- |
| Resélection de l’application ChipDoc suite au statut carte 69 85 | |
| libopensc / card-cps4.c | \_do\_reselect\_cps4\_aid (), effectue la resélection de l’applet ChipDoc grâce à son AID en passant par la sélection d’une application auxiliaire (IDS). Cette fonction prend en paramètre une structure carte sc\_card\_t |
| Appel | |
| libopensc / card-cps4.c | cps4\_select\_file() |

|  |  |
| --- | --- |
| Sélection d’applet | |
| libopensc / card-cps4.c | cps4\_select\_applet(), sur la carte CPS4 seule l’applet sélectionnée par défaut est utilisée. Cette fonction ne fait donc aucune action sur la carte. |
| Appel | |
| libopensc / card-cps4.c | cps4\_init() |

### Gestion des codes PIN de la carte

L’interface des drivers carte d’OpenSC ne prévoit initialement qu’une seule fonction de gestion des codes PIN qui, suivant les paramètres qui lui sont passés, devra réaliser :

* La présentation du code PIN ou du code PUK
* La modification du code PIN
* L’initialisation (ou déblocage) du code PIN

Dans le cadre de la CryptoLib CPS3, une fonction a été ajoutée afin de pouvoir lire le nombre d’essais restant pour la présentation du code PIN.

|  |  |
| --- | --- |
| Gestion du code PIN | |
| libopensc / card-cps4.c | cps4\_pin\_cmd(), la fonction prend en paramètre la structure card et une structure indiquant la fonction que l’on souhaite réaliser ainsi que les données nécessaires à cette fonction. |
| Appel | |
| libopensc / sec.c | sc\_pin\_cmd() |

***Cas particulier du déblocage du code PIN***

Pour la carte CPS4, le code PUK est utilisé seulement lors de l'opération de déblocage du code PIN. Cette opération est réalisée en un seul appel vers la carte au cours duquel sont transmis le code PUK et le nouveau code PIN.

Ce fonctionnement n'est pas compatible avec le standard PKCS#11 qui décompose le déblocage de la carte en plusieurs appels :

* Login SO en présentant le code PUK
* Changement du code PIN
* Logout

Les implémentations suivant la norme PKCS#11 suivront donc cet ordre et OpenSC redirigera ces appels vers les appels du driver carte correspondant.

Le driver carte CPS4 réalise donc les opérations suivantes lors de ces appels :

* Login en présentant le code PUK : appel au déblocage du code PIN en présentant le code PUK passé en paramètre et un code PIN généré aléatoirement. Le code PIN temporaire est mémorisé dans un contexte mémoire associé à la carte CPS2ter sous une forme chiffrée.
* Changement du code PIN : modification du code PIN par appel de l’APDU CPS4 correspondant en présentant le nouveau code PIN et le code PIN aléatoire généré précédemment. Le code PIN aléatoire est supprimé du contexte mémoire une fois la modification réalisée avec succès.
* Logout : aucune action particulière

|  |  |
| --- | --- |
| Construction de l’APDU de gestion des PINs | |
| libopensc / card-cps4.c | cps4\_build\_pin\_apdu(), cette fonction construit le buffer d’APDU en fonction de l’action à réaliser sur le code PIN dont la structure est passée en paramètre |
| Appel | |
| libopensc / card-cps4.c | cps4\_pin\_cmd() |

|  |  |
| --- | --- |
| Lecture du nombre d’essais restant | |
| libopensc / card-cps4.c | cps4\_get\_pin\_counter(), cette fonction sélectionne le fichier correspondant au code PIN passé en paramètre et analyse la réponse de la carte à cette sélection pour déterminer le nombre d’essai restant. Pour ce faire une fonction spécifique cps2ter\_select\_PIN\_file est appelée |
| Appel | |
| libopensc / pkcs15-pin.c | sc\_pkcs15\_get\_pin\_counter() |

### Opérations cryptographiques

Dans l’architecture OpenSC, un driver carte doit implémenter trois fonctions correspondant respectivement au positionnement d’un environnement de sécurité, à la signature de données et au déchiffrement de données. Dans le cas de la carte CPS2ter les deux dernières opérations sont implémentées par la même fonction cps2ter\_compute\_RSA\_priv.

|  |  |
| --- | --- |
| Positionnement de l’environnement de sécurité | |
| libopensc / card-cps4.c | cps4\_set\_security\_env(), cette fonction ne réalise aucune fonction particulière sur la carte, aucune APDU ne lui est donc envoyée. En revanche cette fonction enregistre dans les données privées du driver la clé privée à utiliser lors du prochain appel aux méthodes   * + cps4\_compute\_signature().   + cps4\_internal\_authenticate()   + cps4\_decipher() |
| Appel | |
| libopensc / sec.c | sc\_set\_security\_env() |

|  |  |
| --- | --- |
| Opération de signature | |
| libopensc / card-cps4.c | cps4\_compute\_signature(), cette fonction réalise les opérations de signature en mode PSO CDS ou Internal Authenticate. Cette fonction prend en paramètres la structure  carte sc\_card\_t, les données à signer, la longueur de ces données et la sortie de signature |
| Appel | |
| libopensc / sec.c | sc\_compute\_signature() |

|  |  |
| --- | --- |
| Opération de signature internal authentificate | |
| libopensc / card-cps4.c | cps4\_internal\_authenticate(), cette fonction réalise l’ opération de signature Internal Authenticate qui consiste en un déchiffrement des données. Cette fonction prend en paramètres la structure  carte sc\_card\_t, les données à déchiffrer, la longueur de ces données et la sortie du déchiffrement |
| Appel | |
| libopensc / sec.c | sc\_internal\_authenticate() |

|  |  |
| --- | --- |
| Opération de déchiffrement RSA | |
| libopensc / card-cps4.c | cps4\_decipher(), cette fonction réalise l’ opération de déchiffrement des données avec la clé d’authentification. Cette fonction prend en paramètres la structure  carte sc\_card\_t, les données à déchiffrer, la longueur de ces données et la sortie du déchiffrement |
| Appel | |
| libopensc / sec.c | sc\_decipher() |

#### Support RSA PSS

La signature RSA PSS a été réalisée dans le cadre du support du protocole TLS 1.3 pour l’authentification SSL sous le navigateur Firefox.

|  |  |
| --- | --- |
| Ajout de l’algorithme RSA PSS | |
| libopensc / framework-pkcs15.c | register\_mechanisms(), vérifie la présence du flag SC\_ALGORITHM\_RSA\_PAD\_PSS. Puis ajout du mécanisme correspondant avec l’appel sc\_pkc11\_register\_sign\_hash\_mechanism()  sc\_prkey\_sign() Activation de l’algorithme PSS sur détection de la demande d’utilisation de cet algorithme |
| libopensc / pkcs15-sec.c | sc\_pkcs15\_compute\_signature() Ajout de l’algorithme CPSV4\_ALG\_RSA\_SHA\_PKCS1\_PSS & CPSV4\_ALG\_RSA\_SHA256\_PKCS1\_PSS dans le contexte de sécurité |
| libopensc / padding.c | sc\_get\_encoding\_flags() Activation du padding PSS sur détection de cet algorithme |

|  |  |
| --- | --- |
| Vérification de la signature RSA PSS | |
| libopensc / openssl.c | sc\_pkcs11\_rsa\_pss\_verify() et fips\_rsa\_verify(), ces fonctions effectuent la vérification de signature RSA PSS avec les structures d’objets et APIs d’OpenSSL 0.9.8.n |
| libopensc / padding.c | sc\_get\_encoding\_flags() Activation du padding PSS sur détection de cet algorithme |

#### Authentification RSA PSS

|  |  |
| --- | --- |
| Détection de la clé d’authentification CPS4 | |
| libopensc / framework-pkcs15.c | sc\_is\_cps4\_auth\_operation(), effectue le test d’utilisation de la clé d’authentification de la carte CPS4. Retourne TRUE si c’est le cas. |

#### Chiffrement RSA

|  |  |
| --- | --- |
| Test des capacités de la clé pour le chiffrement | |
| libopensc / framework-pkcs15.c | pkcs15\_pubkey\_can\_do(), teste les capacités cryptographiques de la clé publique pour le chiffrement RSA. Cette fonction prend en paramètres la session sc\_pkcs11\_session, le handle de clé sc\_pkcs11\_object et le mécanisme utilisé |
| Appel | |
| libopensc / mechanism.c | sc\_pkcs11\_encrypt\_init(). |

|  |  |
| --- | --- |
| Réalisation du chiffrement | |
| libopensc / framework-pkcs15.c | pkcs15\_pubkey\_encrypt(), Appelle le traitement de chiffrement qui sera fait in fine par OpenSSL. Cette fonction prend en paramètres la session sc\_pkcs11\_session, le handle de clé sc\_pkcs11\_object ,le mécanisme utilisé, les données à chiffrer et la taille de ces données. |
| Appel | |
| libopensc / mechanism.c | sc\_pkcs11\_encrypt(). |

|  |  |
| --- | --- |
| Chiffrement OpenSSL | |
| libopensc / openssl.c | sc\_pkcs11\_encrypt\_data(), Appelle les APIs de chiffrement OpenSSL. Cette fonction prend en paramètres le mécanisme, l’opération en cours sc\_pkcs11\_operation\_t, les données à chiffrer, la taille de ces données et le buffer des données chiffrées. |
| Appel | |
| libopensc / mechanism.c | sc\_pkcs11\_encrypt(). |

#### Déchiffrement RSA

|  |  |
| --- | --- |
| Déchiffrement RSA en CPS4 | |
| libopensc / padding.c | sc\_pkcs1\_strip\_02\_padding(), effectue le retrait du padding PKCS#1 de type 2 |
| Appel | |
| libopensc / card-cps4.c | decipher() cette fonction prend en paramètres la structure carte et le cryptogramme. Elle effectue le déchiffrement RSA du cryptogramme. |

### Gestion des mises à jour de carte

Les cartes CPS4 pourront être mises à jour au cours de leur vie. Le driver carte CPS4 implémentera donc les fonctions liées à ces mises à jour.

## Support PKCS#15 des nouveaux objets CPS2ter

Le support PKCS#15 des objets CPS2ter est définie dans OpenSC pour la gestion des cartes CPS4. Il s’agit des fonctions PKCS11 standard permettant de rechercher et remonter tous les objets CPS2ter sous forme d’objets PKCS#15.

### Initialisation

L’initialisation du framework PKCS#15 représente l’essentiel des fonctionnalités de celui-ci. C’est en effet lors de cette phase que sera construite la structure PKCS#15 des objets CPS2ter lus sur la carte CPS4. Cette initialisation se fait en deux étapes.

* Reconnaissance et initialisation de la carte : le framework récupère et expose tous les libellés des objets CPS2ter lus sur la carte.
* Initialisation de la structure PKCS#15 : la structure PKCS#15 utilisée par OpenSC va être enrichie par le framework qui va y ajouter les objets qu’il s’attend à trouver sur une carte CPS4

### Utilisation

La recherche des objets CPS2ter remontés dans la structure PKCS#15 se fait de façon standard. Par conséquent, cette recherche s’effectue avec les appels suivants faits dans cet ordre : C\_FindObjectsInit(), C\_FindObjects() et C\_FindObjectsFinal() . Aucune modification n’a été faite dans ces fonctions de recherche pour le support de ces nouveaux objets de la CPS4.

|  |  |
| --- | --- |
| Ajout d’un objet CPS2ter au standard PKCS#15 | |
| libopensc / pkcs15.c | \_\_sc\_pkcs15\_add\_cps2ter\_object(), effectue l’ajout d’un objet CPS2ter à la structure PKCS#15 de la carte. Cette fonction prend en paramètre la structure PKCS#15 de la carte sc\_pkcs15\_card\_t et l’entrée CPS2ter à ajouter. |
| Appel | |
| libopensc / pkcs15.c | \_\_add\_cps2ter\_sit(), \_\_sc\_pkcs15\_search\_objects() |

|  |  |
| --- | --- |
| Ajout d’une situation d’exercice CPS2ter | |
| libopensc / pkcs15.c | \_\_add\_cps2ter\_sit(), effectue l’ajout d’une situation d’exercice CPS2ter à la structure PKCS#15 de la carte avec le libellé CPS2TER\_ACTIVITY\_XY\_PS. Cette fonction prend en paramètre la structure PKCS#15 de la carte sc\_pkcs15\_card\_t. |
| Appel | |
| libopensc / pkcs15.c | \_\_sc\_pkcs15\_search\_objects() |

# Support CPS3

## Gestion IAS

OpenSC comporte un driver 7816 qui sert de base à l’implémentation du driver carte CPS3 IAS. Cependant certaines fonctionnalités propres au standard IAS ne sont pas prévues dans l’architecture OpenSC, quelques modifications ont donc été apportées dans le code OpenSC pour les supporter. Par exemple lors de signature avec hashing, l'intégralité du hashing est réalisée par la librairie cryptographique OpenSSL alors que le le standard IAS impose que la dernière étape de hashing soit réalisée par la carte à puce.

|  |  |  |
| --- | --- | --- |
| **Source** | **Fonction(s)** | **Action** |
| libopensc/pkcs15.c | sc\_pkcs15\_bind\_internal | Le numéro de série devant figurer dans le token info est lu dans le fichier EF.SN.ICC dès l’initialisation du driver carte et n’a donc pas besoin d’être à nouveau lu lors de la lecture de la structure PKCS#15. |
| libopensc/pkcs15-sec.c | sc\_pkcs15\_compute\_hash | Ajout de l'opération de hashing faite par la carte pour la signature numérique |
| pkcs11/framework-pkcs15.c | pkcs15\_prkey\_sign | Appel à l'opération de hashing faite par la carte pour la signature numérique |
| pkcs11/mechanism.c | sc\_pkcs11\_signature\_update  sc\_pkcs11\_md\_update | Ajout des données pour l'opération de hashing faite par la carte lors de la signature numérique. |
| pkcs11/openssl.c | sc\_pkcs11\_openssl\_md\_final | Indiquer que le hash doit être partiel |

## Driver carte

Ce driver carte est une extension fonctionnelle du driver ISO 7816 inclus à l’origine dans OpenSC. Il est implémenté dans le fichier card-cps3r.c.

Les fonctions ajoutées et exposées à OpenSC sont :

* sc\_get\_driver
* cps3\_match\_card
* cps3\_init
* cps3\_get\_model
* cps3\_is\_visible
* cps3\_select\_file
* cps3\_select\_applet
* cps3\_get\_aid\_pkcs15
* cps3\_get\_pin\_counter
* cps3\_set\_security\_env
* cps3\_compute\_hash
* cps3\_compute\_signature
* cps3\_decipher
* cps3\_verify\_update
* cps3\_is\_valid

### Initialisation

L’initialisation du driver carte se fait en quatre étapes :

* Lors de l’initialisation d’OpenSC, le driver est appelé sur sa fonction sc\_get\_driver(). Cette fonction doit enrichir la structure qui lui est passée en paramètre en renseignant les fonctions qu’il implémente. Dans le cas du driver CPS3 il remplace les pointeurs de fonctions ISO 7816 par les fonctions spécifiques à la carte CPS3
* Ensuite lorsqu’OpenSC a détecté la présence d’une carte, il interroge le driver pour lui demander s’il sait utiliser celle-ci au travers de la fonction cps3\_match\_card.
* Si le driver lui fournit une réponse positive, OpenSC va alors initialiser la carte en appelant la fonction cps3\_init.
* OpenSC interroge le driver carte pour récupérer le modèle de celle-ci (fonction ajoutée au cours de l’implémentation de la CryptoLib CPS3 afin d’enrichir la structure PKCS#11 CKA\_TOKEN\_INFO)

Une fonction supplémentaire a été ajoutée dans le cas du driver CPS3. En effet l’ASIP souhaite que certains fichiers présents sur la carte ne soient pas remontés dans la structure PKCS#11. La fonction cps3\_is\_visible() a donc été ajoutée. Cette fonction est appelée lors du parsing de la structure PKCS#15 de la carte, pour chaque fichier elle demande au driver si celui-ci doit être visible au travers des objets PKCS#11.

|  |  |
| --- | --- |
| Initialisation du driver | |
| libopensc / card-cps3.c | sc\_get\_driver (), retourne une structure sc\_card\_driver contenant stockant la liste des fonctions implémentées par ce driver |
| Appel | |
| libopensc / ctx.c | load\_card\_drivers () |

|  |  |
| --- | --- |
| Reconnaissance de la carte | |
| libopensc / card-cps3.c | cps3\_match\_card(),permet au driver carte de vérifier si la carte présente dans l’un des slots doit lui être associée. Le traitement CPS3 consiste à lire le fichier EFSN qui contient un identifiant technique unique constitué d'un préfixe propre à l'organisme émetteur.  Pour différencier le mode sans contact, le driver carte essaie de lire un fichier qui n’est accessible qu’en mode contact. |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |

|  |  |
| --- | --- |
| Initialisation de la carte | |
| libopensc / card- cps3.c | cps3\_init(), prend en paramètre une structure sc\_card\_t. Cette fonction enrichit la structure avec des informations concernant la carte : nom, algorithme et tailles de clés supportés etc… |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |

|  |  |
| --- | --- |
| Lecture du modèle de la carte | |
| libopensc / card- cps3.c | cps3\_get\_model(), recopie dans le paramètre model la valeur « IAS ECC» |
| Appel | |
| pkcs11 / framework-pkcs15.c | pkcs15\_init\_token\_info() |

|  |  |
| --- | --- |
| Visibilité d’un fichier | |
| libopensc / card-cps3.c | cps3\_is\_visible(), vérifie si le path du fichier passé en paramètre est contenu dans la liste des paths des fichiers que l’ASIP ne souhaite pas voir au travers de l’interface PKCS#11 |
| Appel | |
| libopensc / pkcs15.c | sc\_pkcs15\_add\_df () |

### Sélection de fichier sur la carte

Deux fonctions sont utilisées lors de la sélection d’un fichier. La fonction construisant et envoyant l’APDU de sélection de fichier et la fonction analysant la réponse de la carte à cette sélection. Une troisième fonction permet de gérer le cas particulier de sélection d’applet sur la carte.

|  |  |
| --- | --- |
| Sélection de fichier | |
| libopensc / card-cps3.c | cps3\_select\_file(), la commande de sélection de fichier est différente de celle du driver ISO 7816 lorsqu’il faut transmettre le chemin d’accès complet à la carte. Dans ce cas l’instruction P1 vaut 0x09 et le chemin d’accès commence par l’identifiant du MF (i.e. 0x3F) |
| Appel | |
| libopensc / card.c | sc\_select\_file() |

|  |  |
| --- | --- |
| Sélection d’applet | |
| libopensc / card-cps3.c | cps3\_select\_applet(), sur la carte CPS3 la sélection d’applet ne diffère pas de la sélection de fichier. Cette fonction appelle donc la fonction cps3\_select\_file(). |
| Appel | |
| libopensc / card-cps3.c | cps3\_init () |

|  |  |
| --- | --- |
| Sélection de l’applet ID à utiliser | |
| libopensc / card-cps3.c | cps3\_get\_aid\_pkcs15(), l’ID d’applet à utiliser diffère en mode contact et sans contact. Cette fonction utilise les informations positionnées par la fonction cps3\_match\_card() pour renvoyer l’ID correspondant au mode en cours. |
| Appel | |
| libopensc / dir.c | sc\_find\_pkcs15\_app () |

### Gestion des codes PIN de la carte

La carte CPS3 support les commandes ISO 7816 de gestion des codes PIN. Seule la commande de lecture du nombre d’essais restant a donc été implémentée dans ce driver.

|  |  |
| --- | --- |
| Lecture du nombre d’essais restant | |
| libopensc / card-cps3.c | cps3\_get\_pin\_counter(), cette fonction sélectionne le fichier correspondant au code PIN passé en paramètre, envoie la commande GET\_DATA à la carte puis analyse la réponse de la carte pour déterminer le nombre d’essais restant. |
| Appel | |
| libopensc / pkcs15-pin.c | sc\_pkcs15\_get\_pin\_counter() |

### Opérations cryptographiques

Dans l’architecture OpenSC, un driver carte doit implémenter trois fonctions correspondant respectivement au positionnement d’un environnement de sécurité, à la signature de données et au déchiffrement de données. Le format IAS imposant de faire réaliser le dernier hash par la carte lors d’une signature, une fonction de hashing a été ajoutée au driver CPS3.

|  |  |
| --- | --- |
| Positionnement de l’environnement de sécurité | |
| libopensc / card-cps3.c | cps3\_set\_security\_env(), permet de positionner l’environnement de sécurité.  La signature IAS nécessite de procéder à un hashage partiel des données et de faire calculer par la carte le hash du dernier bloc de données à signer. La commande de hashage du dernier bloc doit être précédée par une commande MSE SET afin de positionner le modèle de hashage. |
| Appel | |
| libopensc / sec.c | sc\_set\_security\_env() |

|  |  |
| --- | --- |
| Calcul d’empreinte numérique | |
| libopensc / card-cps3.c | cps3\_compute\_hash(), à l’origine cette fonction n’existe pas dans le driver carte OpenSC, elle a été ajoutée pour permettre de réaliser le hashage par la carte du dernier bloc de données lors d’une signature IAS.  Cette fonction reçoit en paramètre le hash partiel réalisé en amont par OpenSSL ainsi que le dernier bloc de données à signer d’une taille inférieure ou égale à 64 octets. |
| Appel | |
| libopensc / sec.c | sc\_compute\_hash() |

|  |  |
| --- | --- |
| Signature RSA | |
| libopensc / card-cps3.c | cps3\_compute\_signature(), est utilisée à la fois pour les opérations de signature numérique (PSO – Compute Digital Signature) et pour l’authentification (Internal authenticate for client/server authentication).  L’aiguillage sur la commande de signature ou d’authentification se fait sur la longueur des données à signer. Dans le code générique OpenSC, une modification a été réalisée sur l’appel de cette fonction pour ne passer une donnée à signer de longueur non nulle que dans le cas de l’authentification.  La commande de signature numérique ISO 7816 a été modifiée pour respecter les contraintes IAS. Dans l’implémentation du driver CPS3, cette fonction se contente de faire signer le hash précédemment réalisé par la carte.  Dans le cas de l’authentification, la commande diffère de l’implémentation 7816 par la valeur de l’instruction 0x88 et les paramètres P1 et P2 tous les deux à 0x00. A noter que la donnée à signer doit être d’une longueur inférieure ou égale à 40% de la taille de la clé (33 octets pour une clé de 1024 bits). |
| Appel | |
| libopensc / sec.c | sc\_compute\_signature() |

|  |  |
| --- | --- |
| Déchiffrement RSA | |
| libopensc / card-cps3.c | cps3\_decipher(), est invoquée pour réaliser le déchiffrement d’une donnée chiffré à l’aide de la clé d’authentification (cas du WinLogon). L’implémentation IAS diffère de celle du driver 7816 par l’octet de padding spécifié dans la commande et dont la valeur est 0x81 au lieu de 0x00. |
| Appel | |
| libopensc / sec.c | sc\_decipher() |

### Gestion des mises à jour de carte

Les cartes CPS3 ont vocation à être mises à jour au cours de leur vie. Cette mise à jour se traduit par la modification du fichier EF\_ACTUA spécifique aux cartes CPS. Cette mise à jour a nécessité l’implémentation de deux fonctions dans le driver carte CPS3 :

* cps3\_verify\_update : vérifie si la carte a été mise à jour depuis sa dernière utilisation
* cps3\_is\_valid : vérifie si la dernière mise à jour de la carte s’est bien déroulée

|  |  |
| --- | --- |
| Test de la présence de mise à jour | |
| libopensc / card-cps3.c | cps3\_verify\_update(), cette fonction vérifie si la carte a été mise à jour. (cf[. Gestion du cache](#_Gestion_du_cache)) |
| Appel | |
| libopensc / pkcs15.c | sc\_pkcs15\_bind\_internal() |

|  |  |
| --- | --- |
| Validité de la mise à jour | |
| libopensc / card-cps3.c | cps3\_is\_valid(), cette fonction renvoie un flag préalablement positionné par la fonction cps3\_verify\_update(). La mise à jour est considérée comme valide si le fichier EF\_ACTUA est absent ou bien s’il contient les dates de début et de fin de la dernière mise à jour. |
| Appel | |
| pkcs11 / misc.c | is\_session\_valid () |

# Support du GALSS

OpenSC ne permettant pas nativement d'adresser des lecteurs au travers de la couche GALSS, il a donc été nécessaire de développer un driver lecteur spécifique utilisé pour lire les cartes CPS3 et CPS2ter au travers des lecteurs PSS.

Les drivers lecteur étant cumulatifs au sein de l’architecture OpenSC, deux drivers différents ne doivent pas reconnaître un même lecteur au risque de remonter à l'applicatif deux lecteurs différents alors qu'un seul lecteur physique existe. Pour éviter que les lecteurs PC/SC soient vus au travers de deux drivers lecteur différents (driver PC/SC fourni par OpenSC et driver GALSS), le driver GALSS élimine de la liste des lecteurs détectés ceux utilisant la couche PC/SC et ne permet d'utiliser que les lecteurs de type PSS.

Si le driver lecteur GALSS ne trouve pas sur le poste le composant GALSS avec la version attendue, il ne retournera aucun lecteur.

Le driver lecteur est en charge de détecter la présence d’une carte dans un lecteur, de la mettre sous tension, et ainsi de pouvoir lui transmettre des APDU.

La liste des fonctions implémentées dans le driver GALSS est précisée ci-dessous.

## Gestion du driver

La gestion du driver consiste à initialiser puis finaliser celui-ci lors des phases d’initialisation et de finalisation de la CryptoLib CPS3.

|  |  |
| --- | --- |
| Initialisation du driver GALSS | |
| libopensc / reader-galss.c | galss\_init (), réalise le chargement des librairies Galss Client et Galss Info et initialise les pointeurs des fonctions de ces deux librairies. |
| Appel | |
| libopensc / ctx.c | load\_reader\_drivers () |

|  |  |
| --- | --- |
| Arrêt du driver | |
| libopensc / reader-galss.c | galss\_release (), permet de libérer les zones mémoires allouées en interne par le driver lecteur. |
| Appel | |
| libopensc / ctx.c | sc\_release\_context () |

## Gestion des lecteurs GALSS

La gestion des lecteurs GALSS dans le driver consiste à réaliser les actions suivantes

* Détecter les lecteurs GALSS lors de l’initialisation de la CryptoLib CPS3
* Verrouiller un lecteur avant de l’utiliser
* Déverrouiller un lecteur après l’avoir utilisé

|  |  |
| --- | --- |
| Détection des lecteurs GALSS | |
| libopensc / reader-galss.c | galss\_detect\_readers (), permet, au travers des fonctions du GalssInf, d’obtenir la liste des lecteurs configurés sur le poste et dont le nom de ressource commence par « CPS ». Afin de s’assurer de la présence physique du lecteur, une session Galss sera ouverte sur chaque lecteur trouvé. |
| Appel | |
| libopensc / ctx.c | sc\_ctx\_detect\_readers () |

|  |  |
| --- | --- |
| Verrouillage du lecteur | |
| libopensc / reader-galss.c | galss\_detect\_card\_presence (), permet de verrouiller l’accès au lecteur. Cette opération est réalisée au travers de la fonction DebutExclusivite du GalssCli. |
| Appel | |
| libopensc / card.c | sc\_lock () |

|  |  |
| --- | --- |
| Déverrouillage du lecteur | |
| libopensc / reader-galss.c | galss\_detect\_card\_presence (), permet de déverrouiller l’accès au lecteur. Cette opération est réalisée au travers de la fonction FinExclusivite du GalssCli. |
| Appel | |
| libopensc / card.c | sc\_unlock () |

## Gestion de la carte

La gestion des cartes dans le driver consiste à réaliser les actions suivantes

* Détecter la présence d’une carte dans le lecteur GALSS
* Se connecter à la carte
* Se déconnecter de la carte
* Transmettre une APDU à la carte

|  |  |
| --- | --- |
| Détection de la carte | |
| libopensc / reader-galss.c | galss\_detect\_card\_presence (), permet de détecter la présence d’une carte dans le lecteur en utilisant la commande lecteur associée. |
| Appel | |
| libopensc / sc.c | detect\_card\_presence () |

|  |  |
| --- | --- |
| Connexion à la carte | |
| libopensc / reader-galss.c | galss\_connect (), permet d’effectuer une connexion à la carte présente dans le lecteur en réalisant une mise sous tension. Afin de mettre la carte sous tension, il est nécessaire de vérifier qu’une session Galss est bien ouverte sur le lecteur et que la carte est bien présente. |
| Appel | |
| libopensc / card.c | sc\_connect\_card () |

|  |  |
| --- | --- |
| Déconnexion à la carte | |
| libopensc / reader-galss.c | galss\_disconnect(), permet d’effectuer une fermeture de session Galss |
| Appel | |
| libopensc / card.c | sc\_disconnect\_card () |

|  |  |
| --- | --- |
| Envoi d’APDU à la carte | |
| libopensc / reader-galss.c | galss\_transmit (), permet de transmettre une APDU à la carte présente dans le lecteur. Cette opération est réalisée au travers de la fonction Echange du GalssCli. |
| Appel | |
| libopensc / apdu.c | do\_single\_transmit () |

# Spécificités ASIP-Santé

## Gestion du cache

### Présentation

Un cache fichier a été mis en place dans le fonctionnement de la librairie PKCS#11 afin de sauvegarder sur le disque les données lues sur les cartes. Lorsque ces cartes sont par la suite réutilisées, leur contenu est lu depuis le cache fichier.

Le cache est constitué d’un ensemble de fichiers associés à une carte et aux différents objets lus sur cette carte.

Le nom d’un fichier d’objet cache est constitué comme suit :

<numéro\_de\_série\_carte>\_<CKA\_ID> où CKA\_ID est l’identifiant de l’objet PKCS#11 correspondant.

Les objets mis en cache sont les certificats de signature et d’authentification, les données métier (informations carte, informations porteur, situations d’exercice).

### Implémentation

#### Les fonctions

Les fonctions de gestion du cache sont regroupées au sein du fichier source *pkcs15-cache.c* du projet *libopensc*.

|  |  |
| --- | --- |
| Récupération du nom de fichier de cache « select » à utiliser | |
| libopensc / pkcs15-cache.c | generate\_select\_cache\_filename(), permet de construire le nom du fichier de cache à utiliser dans le cadre d’une instruction select. |
| Appel | |
| pkcs11 / pkcs15-cache.c | sc\_pkcs15\_read\_cached\_select\_file() |

|  |  |
| --- | --- |
| Récupération du nom de fichier de cache à utiliser | |
| libopensc / pkcs15-cache.c | generate\_cache\_filename(), permet de construire le nom du fichier de cache à utiliser. |
| Appel | |
| pkcs11 / pkcs15-cache.c | sc\_pkcs15\_read\_cached\_file() |

|  |  |
| --- | --- |
| Génération du fichier de cache « select » | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_select\_cache\_file(), permet de mettre en cache l’objet associé à une instruction select. |
| Appel | |
| libopensc / card.c | sc\_select\_cached\_file() |
| Génération du fichier de cache | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_cache\_file(), permet de mettre en cache un objet de la carte. |
| Appel | |
| libopensc / pkcs15.c | sc\_pkcs15\_bind\_internal() |

|  |  |
| --- | --- |
| Suppression d’un objet mis en cache | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_delete\_cached\_file(), permet de supprimer un objet mis en cache. |
| Appel | |
| Fonction inutilisée |  |

|  |  |
| --- | --- |
| Lecture d’un objet « select » mis en cache | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_read\_cached\_select\_file(), permet de lire un objet associé à une instruction select déjà mis en cache. |
| Appel | |
| libopensc / card.c | sc\_select\_cached\_file() |

|  |  |
| --- | --- |
| Lecture d’un objet de la carte mis en cache | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_read\_cached\_file(), permet de lire un objet de la carte déjà mis en cache. L’objet à lire est déterminé en fonction des paramètres passés en arguments. |
| Appel | |
| libopensc / pkcs15.c | sc\_pkcs15\_bind\_internal() |

#### Cas particuliers des données de situations

Les données de situations d’une carte CPS3 peuvent-être mises à jour. Si jamais les données de cette carte avaient été mises en cache, il faut procéder à la mise à jour de ces données dans le cache.

Le fichier EF\_ACTUA a pour fonction de refléter les mises à jour éventuelles de ces données. Initialement ce fichier est absent de la carte. Il est ensuite créé à la première mise à jour. Enfin il est lui-même mis à jour à chaque mise à jour ultérieure des données de situations.

Sur la base de ce fichier EF\_ACTUA, et à l’aide de la fonction *verify\_update()*, la CryptoLib CPS3 détermine les actions à mener quant aux données de situations mises en cache.  
Cette fonction, pour plus de souplesse dans les appels, est à implémenter pour chaque type de carte. A ce jour, seule la carte CPS3 comprend une implémentation, car elle est la seule carte à être concernée par ces mises à jour :

|  |  |
| --- | --- |
| Vérification de la mise à jour du fichier EF\_ACTUA pour la carte CPS3 | |
| libopensc / card-cps3.c | cps3\_verify\_update(), est l’implémentation de la fonction *verify\_update()* définie dans la structure *sc\_card\_operations* (définie dans libopensc / opensc.h).  Elle permet de vérifier si la mise à jour du cache est nécessaire, et le supprime dans ce cas (à l’exception du fichier EF\_DIR).  Algorithme :  Si le fichier EF\_ACTUA n’existe ni en cache, ni sur la carte, ne rien faire (a).  Si le fichier EF\_ACTUA n’existe pas en cache, mais existe sur la carte, ne rien faire (b).  Si le fichier EF\_ACTUA existe en cache et sur la carte, comparer les deux fichiers. Si les fichiers sont différents, supprimer les données de situations du cache (c), sinon ne rien faire (d).  A l’issue de l’appel à verify\_update() :  Dans les cas (a), (b) et (c) les fichiers de cache concernant les données de situations sont (re)créées, après lecture de ces dernières sur la carte.  Dans le cas (d), les fichiers de cache concernant les données de situations sont lus, et aucune lecture de données sur la carte n’est réalisée.  Attention : Dans le cas où le fichier EF\_ACTUA est vide ou incomplet, la carte est reconnue mais inutilisable. Un code d’erreur « Vendor Defined » est retourné par la CryptoLib CPS3 lors de l’ouverture d’une session. Ce code est : CKR\_ASIP\_BAD\_EF\_ACTUA (= CKR\_VENDOR\_DEFINED + 1). |
| Appel | |
| libopensc / pkcs15.c | sc\_pkcs15\_bind\_internal() |

|  |  |
| --- | --- |
| Fonction de parsing de fichier EF\_ACTUA pour la carte CPS3 | |
| libopensc / card-cps3.c | sc\_pkcs15\_parse\_ef\_actua() déchiffre le contenu du fichier EF\_ACTUA afin d’en extraire les informations nécessaires à déterminer si une mise à jour du cache est nécessaire, c’est-à-dire les dates de début et de fin de mise à jour.  Ces informations sont stockées dans une structure sc\_ef\_actua\_file. |
| Appel | |
| libopensc / card-cps3.c | cps3\_verify\_update() |

|  |  |
| --- | --- |
| Fonction de formatage de date pour le fichier EF\_ACTUA pour la carte CPS3 | |
| libopensc / card-cps3.c | get\_ef\_actua\_time() transforme une date initialement en chaîne de caractère vers une date en secondes.  C’est à l’aide de ce format de date que s’effectue la comparaison entre fichiers EF\_ACTUA afin de déterminer si une mise à jour du cache est nécessaire. |
| Appel | |
| libopensc / card-cps3.c | cps3\_verify\_update() |

|  |  |
| --- | --- |
| Fonction « outil » permettant de récupérer les chemins des fichiers concernés par la mise à jour du cache d’une carte CPS3 | |
| opensc / <plateforme cible> / src/system.c | sc\_get\_card\_cached\_files() récupère les chemins des fichiers concernés par la mise à jour du cache d’une carte CPS3.  Attention : Le fichier EF\_DIR mis en cache n’est jamais retourné par cette fonction.  Cette fonction est spécifique système. Pour les plateformes Windows elle utilise la librairie windows.h et pour les plateformes UNIX elle utilise la librairie dirent.h |
| Appel | |
| libopensc / pkcs15-cache.c | sc\_pkcs15\_delete\_cached\_file() |

## SmartCard Logon

Le mécanisme du SmartCard Winlogon est positionné lors de l’initialisation du module PKCS#11. Un paramètre réservé dans la structure initArgs PKCS#11 est passé à l’appel C\_Initialize(). Ce paramètre est une chaîne de caractères valant ‘winlogonDecrypt’.Le fait de positionner ce paramètre déclenche le déverrouillage de la fonctionnalité de déchiffrement de données chiffrées avec la clé d’authentification.

|  |  |
| --- | --- |
| Récupération du paramétrage de SmartCard Winlogon | |
| pkcs11 / pkcs11-global.c | containsReservedParameter(), prend en paramètre le champ pReserved de la structure initArgs |
| Appel | |
| pkcs11 / pkcs11-global.c | IC\_Initialize() |

## Initialisation du module PKCS#11

L’initialisation du module PKCS#11 enchaine les traitements suivants:

* création d’un contexte OpenSC
* la lecture de la configuration des traces
* l’initialisation des drivers lecteurs PCSC, GALSSv3, GALSSv4
* l’initialisation des drivers carte CPS3 et CPS2ter,
* la détection des différents lecteurs connectés au poste de travail,
* la détection du type des cartes insérées dans chaque lecteur précédemment reconnu,
* pour chaque carte détectée, lecture des différents objets (certificats, clés)

### Configuration des traces

La configuration des traces dépend de la plate-forme cible.

Sous Windows, elle consiste en :

* La lecture en base de registre **HKCU\Software\ASIP Sante\PKCS** des paramètres :
  + ‘Traces’ de type REG\_DWORD (1 pour l’activation des traces, 0 sinon)
  + ‘Debug’ de type REG\_SZ (chaîne de caractères spécifiant le niveau de trace, ‘10’ par exemple)
* Ou la lecture des deux variables d’environnement :
  + CPS3\_PKCS11\_TRACES (1 pour l’activation des traces, 0 sinon)
  + CPS3\_DEBUG (chaîne de caractères spécifiant le niveau de trace, ‘10’ par exemple)

Sous Linux, elle consiste en la lecture des deux variables d’environnement :

* CPS3\_PKCS11\_TRACES (1 pour l’activation des traces, 0 sinon)
* CPS3\_DEBUG (chaîne de caractères spécifiant le niveau de trace, ‘10’ par exemple)

|  |  |
| --- | --- |
| Implémentation de la récupération de la configuration des traces | |
| libopensc / ctx.c | load\_parameters(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | process\_config\_file() |

### Initialisation des drivers lecteurs

L’initialisation des drivers lecteurs consiste en la récupération par OpenSC de l’interface des fonctions de gestion lecteur exposées par chaque implémentation de driver lecteur.

On a les deux cas suivants :

* Initialisation des fonctions du driver GALSS, appel de la fonction sc\_get\_galss\_driver
* Initialisation des fonctions du driver PCSC, appel de la fonction sc\_get\_pcsc\_driver

|  |  |
| --- | --- |
| Implémentation de l’initialisation de tous les drivers lecteurs | |
| libopensc / ctx.c | load\_reader\_drivers(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | sc\_context\_create() |

### Initialisation des drivers carte

De façon analogue, L’initialisation des drivers carte consiste en la récupération par OpenSC de l’interface des fonctions de gestion carte exposées par chaque implémentation de driver carte. On a les deux cas suivants :

* Initialisation des fonctions du driver CPS2ter, appel de la fonction sc\_get\_cps2ter\_driver
* Initialisation des fonctions du driver CPS3, appel de la fonction sc\_get\_cps3\_driver

|  |  |
| --- | --- |
| Implémentation de l’initialisation de tous les drivers carte | |
| libopensc / ctx.c | load\_card\_drivers(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | sc\_context\_create() |

### Détection des lecteurs

La détection des lecteurs connectés s’effectue dans l’ordre PCSC puis GALSS.

|  |  |
| --- | --- |
| Détection de lecteur PCSC | |
| libopensc / reader-pcsc.c | pcsc\_detect\_readers(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | sc\_ctx\_detect\_readers() |

Un seul driver GALSS permet le support des composants logiciels GALSS v3.

|  |  |
| --- | --- |
| Détection de lecteur GALSS | |
| libopensc / reader-galss.c | galss\_detect\_readers(), prend en paramètre le contexte OpenSC |
| Appel | |
| libopensc / ctx.c | sc\_ctx\_detect\_readers() |

### Détection de présence carte

La détection de la présence d’une carte dans chaque lecteur se fait à l’aide du driver lecteur affecté.

|  |  |
| --- | --- |
| Test de présence carte avec driver PCSC | |
| libopensc / reader-pcsc.c | pcsc\_detect\_card\_presence(), prend en paramètre la référence du driver lecteur |
| Appel | |
| libopensc / sc.c | sc\_detect\_card\_presence() |
| pkcs11 / slot.c | card\_detect() |

|  |  |
| --- | --- |
| Test de présence carte avec driver GALSS | |
| libopensc / reader-galss.c | galss\_detect\_card\_presence(), prend en paramètre la référence du driver lecteur |
| Appel | |
| libopensc / sc.c | sc\_detect\_card\_presence() |
| pkcs11 / slot.c | card\_detect() |

### Détection du type de carte

La détection du type de carte s’effectue en sollicitant d’abord le driver CPS3 puis seulement le driver CPS2ter si le premier n’a pas reconnu la carte. Chaque driver envoie une instruction SELECT FILE pour vérifier la présence d’un fichier sur la carte.

|  |  |
| --- | --- |
| Détection d’une carte avec le driver CPS3 | |
| libopensc / card-cps3.c | cps3\_match\_card(), prend en paramètre la référence du driver lecteur |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |
| pkcs11 / slot.c | card\_detect() |

|  |  |
| --- | --- |
| Détection d’une nouvelle carte CPS4 avec le driver CPS4 | |
| libopensc / card-cps4.c | cps4\_match\_card(), prend en paramètre la référence du driver lecteur |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |
| pkcs11 / slot.c | card\_detect() |

Par la suite, chaque carte reconnue est initialisée avec en particulier la lecture de son numéro de série.

|  |  |
| --- | --- |
| Initialisation d’une carte reconnue CPS3 | |
| libopensc / card-cps3.c | cps3\_init(), prend en paramètre la référence de la structure carte |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |
| pkcs11 / slot.c | card\_detect() |

|  |  |
| --- | --- |
| Initialisation d’une carte reconnue CPS4 | |
| libopensc / card-cps4.c | cps4\_init(), prend en paramètre la référence de la structure carte |
| Appel | |
| libopensc / card.c | sc\_connect\_card() |
| pkcs11 / slot.c | card\_detect() |

### Enregistrement des objets carte

Pour les cartes de type CPS3 et CPS4, le traitement fait des lectures directes en PKCS#15. Pour les cartes de type CPS2ter, l’initialisation s’effectue via l’émulation PKCS#15. Au final, les objets de la carte ajoutés dans la structure PKCS#15 sont :

* Certificats
* code PIN et code de déblocage
* clés privées
* clés publiques
* données métier

|  |  |
| --- | --- |
| Enregistrement d’un certificat d’une carte reconnue CPS2ter via l’émulation | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_add\_cert(), prend en paramètres  - la référence de la structure PKCS#15 de la carte  - type d’objet permis,  - ID éventuelle de l'autorité émettrice du certificat  - Pointeur vers le chemin de l'objet sur la carte  - Pointeur vers l'ID de l'objet sur la carte  - Chaine de caractère CKA\_LABEL représentant l'objet  - pointeur vers l'objet d'authentification déverrouillant l'accès à cet objet  - pointeur vers la structure de flags caractérisant l'objet |
| Appels | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_cps2ter\_init() |
| libopensc / pkcs15-syn.c | sc\_pkcs15\_bind\_synthetic() |

|  |  |
| --- | --- |
| Enregistrement d’un objet d’authentification d’une carte reconnue CPS2ter via l’émulation | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_add\_pin(), prend en paramètres  - la référence de la structure PKCS#15 de la carte  - ID de l'objet d'authentification verrouillant l'accès à cet objet  - Chaine de caractère CKA\_LABEL représentant l'objet  - Pointeur vers le chemin de l'objet sur la carte  - Référence du code PIN qui sera utilisé dans le code pour différencier le code PIN du code PUK,  - Type de ce caractère utilisé dans le code PIN (SC\_PKCS15\_PIN\_TYPE\_ASCII\_NUMERIC pour la carte CPS2ter)  - Longueur minimale du code PIN  - Longueur maximale du code PIN  - Flags caractérisant le code PIN (type de padding, PIN SO, PIN local etc...)  - Nombre restant d'essai de code PIN avant blocage  - Nombre initial d'essai de code PIN avant blocage  - Caractère utilisé pour padder le code PIN s'il est plus court que la longueur maximale  - Flags caractérisant l'objet PKCS#15 (objet privé ou modifiable) |
| Appels | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_cps2ter\_init() |
| libopensc / pkcs15-syn.c | sc\_pkcs15\_bind\_synthetic() |

|  |  |
| --- | --- |
| Enregistrement d’une donnée métier d’une carte reconnue CPS2ter via l’émulation | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_add\_data(), prend en paramètres  - la référence de la structure PKCS#15 de la carte  - Pointeur vers l'ID de l'objet sur la carte  - Chaine de caractère CKA\_LABEL représentant l'objet  - Pointeur vers le chemin de l'objet sur la carte,  - Pointeur vers une structure de flags caractérisant l'objet  - pointeur vers l'objet d'authentification déverrouillant l'accès à cet objet |
| Appels | |
| libopensc / pkcs15-cps2ter.c | sc\_pkcs15emu\_cps2ter\_init() |
| libopensc / pkcs15-syn.c | sc\_pkcs15\_bind\_synthetic() |

|  |  |
| --- | --- |
| Création des objets de la carte dans la structure PKCS#15 | |
| libopensc / framework-pkcs15.c | pkcs15\_create\_tokens(), prend en paramètre la référence de la structure PKCS#15 de la carte |
| Appel | |
| pkcs11 / slot.c | card\_detect() |

## Traces

Le CryptoLib CPS3 peut gérer 2 types de traces :

* Les traces purement PKCS#11.
* Les traces internes à l’implémentation (plus fines), qui correspondent aux traces applicatives et natives d’OpenSC. Elles ont été adaptées afin de d’être plus proche de celle de la CryptoLib existante.

Remarque : Les données sensibles, telles que le code porteur, ou le code de déblocage, sont masqués dans les deux types de traces.

### Traces PKCS#11

#### Fichier

Le nom du fichier de traces PKCS#11 a le format suivant : cps3p11\_%p\_%t.log où %p correspond à l’identifiant de l’application (ou processus) et %t correspond à l’identifiant du thread.

Ainsi une application exécutant 3 threads aura autant de fichiers de traces PKCS#11.

Ce fichier est situé :

* Sous Microsoft Windows :

%ALLUSERSPROFILE%\santesocial\cps\log\

* Sous Linux :

/var/opt/santesocial/CPS/log/

#### Implémentation

|  |  |
| --- | --- |
| Fonction d’impression de traces PKCS#11 | |
| pkcs11 / pkcs11-spy.c | logP11(string   format , mixed   args), trace une chaîne de caractères formatée, avec le format : *format* et en utilisant les arguments : *args*.  Chaque chaîne tracée est précédée d’un « timestamp » du type : « Thu Sep 16 16:55:50.760 : ». Le formatage est identique à celui utilisé par la fonction fprintf() de la librairie standard de C stdio. |
| Exemple d’appel à la fonction de logP11() | |
| pkcs11 / pkcs11-spy.c | logP11( "%d: %s", count++, function); imprime la chaine : « Thu Sep 16 16:59:46.460 : 111: C\_FindObjects »  %d a été remplacé par « 111 » (la valeur de *count++*), et %s a été remplacé par « C\_FindObjects » (la valeur de *function*). Le tout a été préfixé par le « timestamp » : « Thu Sep 16 16:59:46.460 : » |

Certaines fonctions ont besoin de tracer un sous-ensemble d’information. Elles ne nécessitent donc pas, de ce fait, du pré fixage par un « timestamp ». L’utilisation de la fonction logP11() est dans ce cas délaissée au profit d’un simple appel à la fonction de la librairie standard de C : fprintf().

Contrairement à la fonction logP11(), la fonction fprintf() n’est pas chargée d’ouvrir et fermer le fichier. C’est donc à la fonction faisant appel à fprintf() de remplir ce rôle.

Ci-après les étapes clés du traçage à l’aide de la fonction fprintf() :

1. Déclaration du pointeur de fichier :

FILE \*f = NULL;

1. Récupération du pointeur de fichier :

f = get\_spy\_out\_by\_thread\_force\_open();

1. Impression d’une chaîne dans le fichier :

fprintf(f, "%s", *chaine*);

1. Vidage du tampon et fermeture du fichier :

fflush(f);

fclose(f);

|  |  |
| --- | --- |
| Fonctions de traces de sous-ensemble d’information | |
| pkcs11 / pkcs11-display.c | print\_enum(), chargée d’imprimer une énumération, comme par exemple une liste d’attributs dans le cadre d’une recherche d’objets.  Exemple (surligné en gris les chaînes imprimées par la fonction) :  Tue Sep 21 17:33:51.870 : 24: C\_FindObjectsInit Tue Sep 21 17:33:51.870 : [in] hSession = 0x1 Tue Sep 21 17:33:51.870 : [in] pTemplate[2]:  Tue Sep 21 17:33:51.870 : CKA\_CLASS CKO\_PRIVATE\_KEY Tue Sep 21 17:36:13.322 : CKA\_LABEL [size : 0xC (12)] Tue Sep 21 17:36:13.322 : 4350535F 50524956 5F534947 CPS\_PRIV\_SIG |
| pkcs11 / pkcs11-display.c | print\_boolean(), chargée d’imprimer un booléen.  Exemple (surligné en gris les chaînes imprimées par la fonction) :  Tue Sep 21 17:43:50.069 : 89: C\_FindObjectsInit Tue Sep 21 17:43:50.069 : [in] hSession = 0x1 Tue Sep 21 17:43:50.069 : [in] pTemplate[4]:  Tue Sep 21 17:43:50.069 : CKA\_CLASS CKO\_DATA  Tue Sep 21 17:43:50.069 : CKA\_TOKEN True Tue Sep 21 17:47:24.334 : CKA\_MODIFIABLE True Tue Sep 21 17:48:24.630 : CKA\_LABEL [size : 0x8 (8)] Tue Sep 21 17:48:24.630 : 4350535F 44415441 CPS\_DATA Tue Sep 21 17:48:24.646 : Returned: 0 CKR\_OK |
| pkcs11 / pkcs11-display.c | print\_generic(), chargée d’imprimer un tableau d’octets.  Exemple (surligné en gris les chaînes imprimées par la fonction) :  Tue Sep 21 17:30:27.434 : 23: C\_Login Tue Sep 21 17:30:27.434 : [in] hSession = 0x1 Tue Sep 21 17:30:27.434 : [in] userType = CKU\_USER Tue Sep 21 17:30:27.434 : [in] pPin[ulPinLen][size : 0x4 (4)] Tue Sep 21 17:30:27.434 : 2A2A2A2A Tue Sep 21 17:33:51.870 : Returned: 0 CKR\_OK |
| pkcs11 / pkcs11-display.c | print\_print(), chargée d’imprimer un objet et son label.  Exemple (surligné en gris les chaînes imprimées par la fonction) :  Tue Sep 21 17:41:53.351 : 59: C\_FindObjectsInit Tue Sep 21 17:41:53.351 : [in] hSession = 0x1 Tue Sep 21 17:41:53.351 : [in] pTemplate[2]:  Tue Sep 21 17:41:53.351 : CKA\_CLASS CKO\_PRIVATE\_KEY  Tue Sep 21 17:42:00.820 : CKA\_LABEL [size : 0xC (12)] Tue Sep 21 17:42:00.820 : 4350535F 50524956 5F415554 CPS\_ PRIV\_AUT Tue Sep 21 17:43:46.257 : Returned: 0 CKR\_OK |

Ces fonctions sont le plus souvent invoquées au travers du pointeur de fonction (display) de la structure *type\_specs* définie dans le source *pkcs11-display.h* du projet *pkcs11* comme suit:

typedef struct {

CK\_ULONG type;

const char \* name;

display\_func\* display;

void \* arg;

} type\_spec;

Exemple d’initialisation et d’utilisation d’une structure *type\_specs* :

// Initialisation

type\_spec ck\_attribute\_specs[] = {

{CKA\_CLASS , "CKA\_CLASS ", print\_enum, ck\_cls\_t },

{CKA\_TOKEN , "CKA\_TOKEN ", print\_boolean, NULL },

{CKA\_PRIVATE , "CKA\_PRIVATE ", print\_boolean, NULL },

{CKA\_LABEL , "CKA\_LABEL ", print\_print, NULL },

{CKA\_APPLICATION , "CKA\_APPLICATION ", print\_print, NULL },

{CKA\_VALUE , "CKA\_VALUE ", print\_generic, NULL },

{CKA\_OBJECT\_ID , "CKA\_OBJECT\_ID ", print\_generic, NULL },

{CKA\_CERTIFICATE\_TYPE , "CKA\_CERTIFICATE\_TYPE ", print\_enum,ck\_crt\_t}

};

…

// Utilisation

CK\_ATTRIBUTE\_PTR pTemplate  
…  
if(ck\_attribute\_specs[k].type == pTemplate[j].type) {  
// Afin d’inserer le « timestamp »  
logP11( " %s ", ck\_attribute\_specs[k].name);  
…  
// trace sans « timestamp » et en utilisant la fonction associée à *display* dans l’initialisation de ck\_attribute\_specs pour le type correspondant (ex : CKA\_CLASS => print\_enum, CKA\_LABEL => print\_print …)  
ck\_attribute\_specs[k].display(pTemplate[j].type,pTemplate[j].pValue, pTemplate[j].ulValueLen, ck\_attribute\_specs[k].arg);  
…  
}  
…

Se reporter à la fonction print\_attribute\_list() dans le source pksc11-display.c du projet pkcs11 pour plus de détails.

### Traces OpenSC

#### Fichier

Le nom du fichier de traces OpenSC a le format suivant : cps3opsc\_%p\_%t.log où %p correspond à l’identifiant de l’application (ou processus) et %t correspond à l’identifiant du thread.

Ainsi une application exécutant 3 threads aura autant de fichiers de traces OpenSC.

Ce fichier est situé :

* Sous Microsoft Windows :

%ALLUSERSPROFILE%\santesocial\cps\log\

* Sous Linux :

/var/opt/santesocial/CPS/log/

#### Implémentation

OpenSC dispose d’un fichier source dédié aux traces. Il s’agit du fichier *log.c* du projet *libopensc*. Il comprend notamment les 2 fonctions à utiliser pour les traces et décrites ci-après.

|  |  |
| --- | --- |
| Fonction de traces d’erreur OpenSC | |
| libopensc / log.c | \_sc\_error(sc\_context\_t \*ctx, const char \*format, mixed args), imprime une chaîne de caractères formatée, avec le format : *format* et en utilisant les arguments : *args* et en tenant compte du contexte : *ctx*.  Chaque chaîne tracée est précédée d’un « timestamp » du type : « Wed Sep 22 11:27:40.785 : ». Le formatage est identique à celui utilisé par la fonction fprintf() de la librairie standard de C stdio. |
| Exemple d’appel à la fonction de \_sc\_error() | |
| libopensc / reader-pcsc.c | \_sc\_error est redéfinie en sc\_error dans le fichier d’entête log.h. Ainsi les appels à la fonction \_sc\_error se font en sc\_error, comme suit :  sc\_error(ctx, desc ": 0x%08lx\n", rv); imprime la chaine : « Wed Sep 22 11:27:40.785 : SCardEndTransaction failed: 0x80100069 »  *desc* a été remplacé par *"SCardEndTransaction failed"*(sa valeur). %08lx a été remplacé par « 80100069 » (la valeur de *rv*)Le tout a été préfixé par le « timestamp » : « Wed Sep 22 11:27:40.785 : » |

|  |  |
| --- | --- |
| Fonction de log des traces de debug OpenSC | |
| libopensc / log.c | \_sc\_debug(sc\_context\_t \*ctx, const char \*format, mixed args), imprime une chaîne de caractères formatée, avec le format : *format* et en utilisant les arguments : *args* et en tenant compte du contexte : *ctx*.  Chaque chaîne loguée est précédée d’un « timestamp » du type : « Wed Sep 22 11:27:40.785 : ». Le formatage est identique à celui utilisé par la fonction fprintf() de la librairie standard de C stdio. |
| Exemple d’appel à la fonction de \_sc\_debug() | |
| pkcs11 / framework-pkcs15.c | \_sc\_debug est redéfinie en sc\_debug dans le fichier d’entête log.h. Ainsi les appels à la fonction \_sc\_debug se font en sc\_debug, comme suit :  sc\_debug(context, "Now computing hash of signature for %d bytes. %d bytes already hashed.\n", sign\_data->remaining\_msg\_len, sign\_data->msg\_len); imprime la chaine : « Wed Sep 22 11:27:39.129 : Now computing hash of signature for 40 bytes. 64 bytes already hashed. »  le premier %d a été remplacé par « 40 » (la valeur de sign\_data->remaining\_msg\_len).  le second %d a été remplacé par « 64 » (la valeur de sign\_data->msg\_len).Le tout a été préfixé par le « timestamp » : « Wed Sep 22 11:27:39.129 : » |