

Cryptolib CPS v5

Manuel de Programmation

v1.5.0 du 16/10/2013

Sommaire

1	Introduction	4
2	La spécification du standard PKCS#11	5
2.1	Vue générale.....	5
2.2	Principes de sécurité.....	5
2.3	Compatibilité multi plate-forme	5
2.4	Types et structures.....	6
2.5	Les objets PKCS#11	6
2.6	Les services PKCS#11.....	7
2.7	Les mécanismes	7
3	Le Kit de Programmation.....	8
3.1	Description	8
3.2	Installation - Configuration.....	9
3.3	Options de compilation	9
4	Appel des services PKCS#11	10
5	Les services PKCS#11.....	10
6	Exemple de programmation de la Cryptolib CPS v5	11
6.1	Détection des lecteurs connectés au poste	11
6.2	Détection des cartes supportées	11
6.3	Détection de l'état carte.....	12
6.4	Gestion des sessions	15
6.5	Lecture des certificats	16
6.5.1	Lecture du certificat de signature.....	16
6.5.2	Lecture du certificat d'authentification	18
6.6	Operations cryptographiques	19
6.6.1	Authentification.....	20
6.6.2	Signature.....	21
7	Gestion des données métier.....	22
7.1	Recherche.....	22
7.2	Structuration des données.....	22
7.3	Gestion des informations carte.....	23
7.3.1	Recherche.....	23
7.3.2	Récupération des données.....	23
7.4	Gestion des informations porteur.....	26
7.4.1	Recherche.....	26
7.4.2	Récupération des données.....	26
7.5	Gestion des informations PS2	28
7.5.1	Recherche.....	28
7.5.2	Récupération des données.....	28
7.6	Gestion des situations d'exercice	29
7.6.1	Recherche d'une situation	29
7.6.2	Récupération des données.....	31
7.7	Gestion des situations de Facturation.....	33
7.7.1	Recherche de toutes les situations de facturation	33
7.7.2	Récupération des données.....	34
8	Spécificités du sans contact	36
9	Gestion du jeton d'établissement.....	37
9.1	Recherche du jeton	37

9.2	Récupération de la valeur du jeton	37
9.2.1	Obtention de la taille du jeton	37
9.2.2	Obtention de la valeur du jeton.....	38
9.3	Modification du jeton	38

1 Introduction

Ce document constitue le manuel de programmation de la Cryptolib v5, son but est de :

- présenter succinctement le standard PKCS#11,
- décrire les principes de développement,
- décrire les traitements à réaliser par les applicatifs pour s'interfacer avec cette librairie et gérer au mieux la nouvelle carte CPS3.

2 La spécification du standard PKCS#11

Le présent manuel se base sur la spécification du standard PKCS#11 version 2.20 publié par les 'RSA Laboratories'. Cette spécification décrit les interfaces PKCS#11 que doit implémenter un module cryptographique afin de respecter ce standard. Elle donne également toutes les informations nécessaires au développement d'une application utilisant l'interface PKCS#11.

Nous décrivons dans la suite du document les chapitres de la spécification qui intéresseront plus particulièrement le développeur.

2.1 Vue générale

Cette section correspond au chapitre 6 de la spécification. Celui-ci donne les concepts généraux de ce qu'est l'interface PKCS#11. Il évoque les contraintes et les règles d'accès sur un token PKCS#11 dans un contexte multi applicatifs. Il insiste notamment sur l'aspect multithread à prendre en compte lors de l'accès à un module PKCS#11. La carte à puce (i.e. token PKCS#11) est décrite en terme d'objets PKCS standard (certificats, clés publiques/privées, données annexes).

La gestion des sessions est abordée en détail afin de donner aux développeurs l'ensemble des règles importantes qui se rattachent à cette notion. La spécification décrit les différents types de sessions avec les droits qui leur sont associés. Cela indique notamment que la visibilité des objets (espace public, privé) est différente suivant le type de session. La session apparaît également comme un moyen de réaliser le cloisonnement entre les opérations cryptographiques et une zone permettant à une application de manipuler les objets PKCS de la carte.

Pour finir le chapitre 6 donne la liste des fonctions de l'interface PKCS#11 avec une description succincte de chacune d'elles.

2.2 Principes de sécurité

Le chapitre 7 de la spécification détaille les points importants de sécurité et expose comment restreindre l'accès aux objets privés de la carte,

- Par la saisie d'un code confidentiel appelé code PIN,
- En limitant la manipulation des objets eux-mêmes (ex. clés privées non extractibles).

Par ailleurs, il est indiqué que la notion de sécurité du dispositif cryptographique ne se limite pas uniquement au token et à l'implémentation de l'interface PKCS#11 qui dialogue avec le token, mais elle concerne également l'applicatif qui utilise cette interface.

2.3 Compatibilité multi plate-forme

Ce chapitre 8 de la spécification décrit les aspects multi plateformes du point de vue de l'alignement des structures C et sur la définition des constantes macros liés à la notion de pointeur.

Pour le développeur Cryptolib CPS v5, l'inclusion du fichier d'en-tête pkcs11.h suffit pour garantir la portabilité du code source.

2.4 Types et structures

Ce chapitre 9 de la spécification présente tous les types de données qui sont manipulables au travers de l'interface PKCS#11. Ces données sont pour l'essentiel des constantes, des types et des structures.

Pour le développeur, ce chapitre définit sous forme **typedefs** les types primitifs (CK_ULONG, CK_OBJECT_HANDLE, etc.) et les structures C PKCS (CK_SLOT_INFO, etc.) utilisables dans une application écrite en C/C++.

2.5 Les objets PKCS#11

Ce chapitre 10 de la spécification définit sous forme hiérarchique les différents objets qu'une application PKCS#11 peut manipuler par le biais des actions suivantes :

- Lecture
- Ecriture
- Copie
- Modification
- Suppression

Il existe des objets hardware, de mécanismes, de stockage. Ces derniers sont répartis selon les catégories suivantes :

- Objets certificats
- Objets clé publiques
- Objets clé privées
- Objets clés secrètes
- Objets données

Chaque objet comporte un ensemble d'attributs soit communs, soit spécifiques au type d'objet. La nature et la signification de chaque attribut sont détaillées dans ce chapitre.

2.6 Les services PKCS#11

Ce chapitre 11 de la spécification documente toutes fonctions de l'interface PKCS#11 en les regroupant par catégories de services :

- Fonctions générales
- Fonctions de gestion slot et token
- Fonctions de gestion de sessions
- Fonctions de gestion des objets PKCS
- Fonctions de chiffrement / déchiffrement
- Fonctions de hashing
- Fonctions de signature / vérification de signature

En préambule, la liste de tous les codes d'erreur est donnée avec la signification de chacun d'entre eux. Les règles concernant le passage des paramètres en entrée/sortie des fonctions sont détaillées.

Pour chaque fonction, la description des paramètres d'entrée et de sortie est indiquée. La liste des codes d'erreur potentiellement retournés par la fonction est précisée. Un exemple de code mettant en œuvre la fonction est également présent.

2.7 Les mécanismes

Ce dernier chapitre documente tous les mécanismes cryptographiques que peut potentiellement supporter un module PKCS#11.

3 Le Kit de Programmation

3.1 Description

Le Kit de Programmation est constitué de l'ensemble des répertoires et fichiers suivants :

- Répertoire **ProgrammeExemple** qui contient les sous-répertoires suivants:
 - **programmeExemple** : contient les exécutables et les sources du programme d'exemple Java.
 - **programmeExempleCPS3.jar** : programme d'exemple (ensemble de classes livrées sous forme d'une archive jar).
 - **programmeExempleJNI** : contient le binaire et les sources de la librairie JNI associée au programme d'exemple ainsi que le wrapper IAIK (cf. http://jce.iaik.tugraz.at/sic/Products/Core-Crypto-Toolkits/PKCS_11_Wrapper/license pour les précisions d'utilisation) pour les appels PKCS#11 depuis Java
 - **pgm_exemple_jni_w32.dll** : librairie du pont JNI pour Windows.
- Répertoire **DOC** qui contient les manuels suivants :
 - **CPS3_Manuel de programmation.pdf** : Manuel de programmation de la Cryptolib CPS v5 - (présent document) en format PDF.
 - **CPS3_Documentation Programme d'exemple.pdf** : documentation associée au programme d'exemple Java, document en format PDF
 - **CPMPRTAB.DOC** : Manuel de programmation du dictionnaire CPTAB des données de la CPS – Plates-formes Windows en format MS-Word 7
- Répertoire **DEV\Include** qui contient l'ensemble des fichiers de définitions et de prototypage nécessaires au développement d'une application :
 - PKCS11.H
 - CPTAB.H
- Répertoire **DEV\IncludeWin32** qui contient l'ensemble des fichiers de définitions et de prototypage pour Windows nécessaires au développement d'une application :
 - Win32.H
 - Sys_dep.H
- Répertoire **DEV\LIB** qui contient :
 - la librairie nécessaire à la phase d'édition de liens d'une application utilisant la Cryptolib CPS v5 : **CPS3_PKCS11_W32.LIB**
 - librairie de gestion du dictionnaire : **CPTABW32.LIB**

3.2 Installation - Configuration

Les installeurs de la Cryptolib CPS v5 fournis par l'ASIP Santé quelle que soit la plateforme ont été conçus :

- en respectant les spécifications d'installation de chaque OS.
- En prenant en compte les besoins des éditeurs pour l'intégration dans leur solution (installateurs silencieux, pas de redémarrage du poste nécessaire, ...).

Il est donc demandé d'utiliser ces installeurs en l'état.

Le programme d'exemple requiert la présence sur le poste de développement d'une JRE de version au moins égale à 1.5

A ce stade, il est déjà possible de lancer l'exécution des programmes de gestion de la carte CPS3 CPGESW32.EXE ou le programme d'exemple.

Les fichiers des répertoires \DEV\H et DEV\LIB doivent être correctement localisés de manière à être respectivement pris en compte dans les phases de compilation (INCLUDE....) et d'édition de liens (LIB..).

3.3 Options de compilation

Il n'y a pas d'options de compilation particulières à spécifier pour une application souhaitant s'interfacer avec la librairie Cryptolib CPS v5 autres que celles mentionnées au chapitre 8 des spécifications PKCS#11.

4 Appel des services PKCS#11

L'appel aux services d'un module cryptographique respectant le standard PKCS#11, nécessite l'acquisition d'une référence vers un objet CK_FUNCTION_LIST. On déclare alors dans le programme une variable pFunctionList de type CK_FUNCTION_LIST_PTR. La fonction générale C_GetFunctionList est ensuite appelée en passant comme argument un pointeur vers pFunctionList.

Si le code retour de C_GetFunctionList est CKR_OK, alors la référence pFunctionList est valide et pointe vers la liste des services PKCS#11.

On peut dès lors invoquer tous les services PKCS#11 via la variable pFunctionList.

L'extrait de code suivant montre comment initialiser un module PKCS#11 en général.

```
CK_FUNCTION_LIST_PTR pFunctionList = NULL_PTR;

CK_RV rc = C_GetFunctionList(&pFunctionList);

if (rc != CKR_OK)
{
    printf("Interface des services PKCS11 inaccessible\n");
    return rc;
}

/* initialisation avec flags */
pInitArgs.CreateMutex = NULL_PTR;
pInitArgs.DestroyMutex = NULL_PTR;
pInitArgs.flags = CKF_OS_LOCKING_OK; /* multithreading */
pInitArgs.LockMutex = NULL_PTR;
pInitArgs.UnlockMutex = NULL_PTR;
pInitArgs.pReserved = NULL_PTR;
rc = (*pFunctionList->C_Initialize)(&pInitArgs);

if (rc != CKR_OK && rc != CKR_CRYPTOKI_ALREADY_INITIALIZED)
{
    printf("L'initialisation du module PKCS11 a échouée\n");
    return rc;
}
```

5 Les services PKCS#11

Nous ne rappelons pas dans ce manuel l'ensemble des services PKCS#11. Le développeur se reportera à la spécification du standard PKCS#11 au chapitre 11 où sont exposés toutes les APIs avec leurs prototypes et les codes retours associés.

6 Exemple de programmation de la Cryptolib CPS v5

6.1 Détection des lecteurs connectés au poste

La détection du nombre de lecteurs connectés au poste de travail ne nécessite qu'un appel à C_GetSlotList. Il suffit de récupérer la taille de la liste de tous les slots.

L'extrait de code suivant montre comment détecter qu'aucun lecteur n'est connecté au poste.

```
CK_RV rc;
CK_ULONG ulCount;
/* Récupération de la taille de la liste des slots */
rc = (*pListeFonctions->C_GetSlotList)(CK_FALSE, NULL_PTR, &ulCount);

if (rc == CKR_OK)
{
    if (ulCount == 0)
    {
        /* ICI, AUCUN LECTEUR N'EST TROUVE */
    }
    else
    {
        /* SINON DETECTION CARTE */
    }
}
```

6.2 Détection des cartes supportées

La détection de présence de cartes CPS dans les lecteurs connectés au poste de travail se fait au moyen de l'appel à la fonction C_GetSlotList en passant la valeur CK_TRUE dans le paramètre tokenPresent. Une carte non supportée (ex : carte vitale) n'est pas visible par le module PKCS#11, donnant l'impression que le lecteur est vide.

L'extrait de code suivant permet de détecter qu'aucune carte n'est insérée/supportée.

```
CK_ULONG ulCount = 0;
CK_SLOT_ID* pSlotList = NULL_PTR;

/* Récupération du nombre de slots avec carte reconnue */
rc = (*pFunctionList->C_GetSlotList)(CK_TRUE, pSlotList, &ulCount);

if (rc == CKR_OK && ulCount > 0)
{
    pSlotList = (CK_SLOT_ID_PTR) malloc(pSlotList*sizeof(CK_SLOT_ID));
    /* Récupération de la liste des slots avec carte */
    rc = (*pFunctionList->C_GetSlotList)(CK_TRUE, pSlotList, &ulCount);
}

if (ulCount == 0)
{
    /* ICI, AUCUNE CARTE SUPPORTEE N'EST TROUVEE */
}
```

6.3 Détection de l'état carte

La notion d'état carte des API CPS est partiellement transposable dans le monde PKCS#11.

Dans les API CPS, l'état carte indiquait :

- Le nombre d'essais restant sur le code porteur et le code de déblocage.
- Que le code porteur est actif ou non.

Dans l'interface PKCS#11, la notion de code porteur actif n'existe que dans le contexte de l'application. Chaque application doit, par sécurité, demander la saisie du code porteur pour pouvoir utiliser les données (i.e. clés) privées de la carte. Une application ne sait donc pas qu'une autre application a déjà demandé le code porteur et que ce dernier est actif sur la carte CPS.

Pour savoir si le code porteur est actif pour l'application en cours d'exécution, il faut obtenir les informations de la session via la fonction `C_GetSessionInfo` et vérifier le champ `state` de la structure `CK_SESSION_INFO`. Si le champ `state` vaut `CKS_RO_USER_FUNCTIONS` ou `CKS_RW_USER_FUNCTIONS`, le code porteur est actif (cf. exemple Gestion des sessions).

D'autre part dans la structure `CK_TOKEN_INFO` obtenu par l'appel à la fonction `C_GetTokenInfo`, il figure :

- Le numéro de série de la carte est renseigné dans le champ 'serialNumber'
- l'état de saisie du code PIN / PUK qui par analyse du champ 'flags' peut renseigner sur le nombre d'essais restants.
- Le type de carte est renseigné dans le champ 'label'. Plus précisément,
 - Si la carte en cours est de type CPS2ter, le champ 'label' contient **CPS2ter-<identifiant logique carte>**
 - Exemple : CPS2ter-2100665689
 - Si la carte en cours est de type CPS3, le champ 'label' contient **CPS3v<version>-<identifiant logique carte>**
 - Exemple : CPS3v1-2300723698

Le code suivant affiche l'état de saisie du code PIN / PUK en supposant que le nombre d'essais possibles de saisie du code PIN est 3.

```
/* Obtention de l'état carte */
CK_TOKEN_INFO tokenInfo;
CK_SLOT_ID slotID;

rc = (*pFunctionList->C_GetTokenInfo)(slotID, &tokenInfo);
if (rc == CKR_OK)
{
    /* Transposition du flag */
    if (typeUtilisateur == CKU_USER)
    {
        if (infosCarte.flags & CKF_USER_PIN_FINAL_TRY)
            printf("Il reste un essai.\n");
        else if (infosCarte.flags & CKF_USER_PIN_COUNT_LOW)
            printf("Il reste deux essais.\n");
        else if (infosCarte.flags & CKF_USER_PIN_LOCKED)
            printf("La carte est bloquee.\n");
        else
            printf("Aucun code PIN erroné saisi.\n");
    }
    else if (typeUtilisateur == CKU_SO)
    {
        if (infosCarte.flags & CKF_SO_PIN_FINAL_TRY)
            printf("Il reste un essai pour le code PUK.\n");
        else if (infosCarte.flags & CKF_SO_PIN_COUNT_LOW)
            printf("Au moins un code PUK faux saisi.\n");
        else if (infosCarte.flags & CKF_SO_PIN_LOCKED)
            printf("La carte est definitivement bloquee.\n");
        else
            printf("Aucun code PUK erroné saisi.\n");
    }
}
```

Le code suivant teste le type de carte CPS en cours d'utilisation.

```
CK_TOKEN_INFO tokenInfo;
CK_SLOT_ID slotID;
CK_ULONG rc;
CK_BBOOL bCarteCPS3 = CK_FALSE;

/* Obtention d'un identifiant slot valide
...*/
/* Obtention de l'état carte */

rc = (*pFunctionList->C_GetTokenInfo)(slotID, &tokenInfo);

if (rc == CKR_OK)
{
    bCarteCPS3 = CK_FALSE;
    if (strstr((const char *)tokenInfo.label, "CPS3"))
        bCarteCPS3 = CK_TRUE;
}
```

6.4 Gestion des sessions

L'état d'une session est révélé dans la structure CK_SESSION_INFO par l'appel à la fonction C_GetSessionInfo

L'extrait de code suivant montre comment tester l'état de la session pour savoir si la session courante est loguée ou non et le type d'authentification (normale ou SO). Il faut prendre soin de considérer les sessions ouvertes en lecture seule (CKS_RO_PUBLIC_FUNCTIONS, CKS_RO_USER_FUNCTIONS) et celles ouvertes en lecture/écriture (CKS_RW_PUBLIC_FUNCTIONS, CKS_RW_USER_FUNCTIONS).

```
/******  
/* OBTENTION INFOS SESSION */  
/******  
CK_SESSION_INFO sessionInfo;  
CK_SESSION_HANDLE hSession;  
  
memset(&sessionInfo, 0, sizeof(sessionInfo));  
  
/* Ouverture d'une session */  
rc = (*pFunctionList->C_OpenSession)(slotID, CKF_RW_SESSION |  
CKF_SERIAL_SESSION, NULL_PTR, &hSession);  
  
/* Récupération des informations sur la session */  
rc = (*pFunctionList->C_GetSessionInfo)(hSession, &sessionInfo);  
  
if (rc == CKR_OK)  
{  
    printf("Session Information\n> slotID: %ld\n> state: %ld\n> flags:  
%ld\n> ulDeviceError:\n    %ld\n\n", sessionInfo.slotID, sessionInfo.state, sessionInfo.flags, sessi  
onInfo.ulDeviceError);  
  
    /******  
    /* CHARACTERISATION SESSION */  
    /******  
  
    /* Si la session est non loguée, l'utilisateur n'est pas authentifié  
*/  
    if ((sessionInfo.state == CKS_RO_PUBLIC_SESSION)  
        || (sessionInfo.state == CKS_RW_PUBLIC_SESSION))  
        printf("Utilisateur non logué\n");  
    /* Si la session est loguée, l'utilisateur est déjà authentifié */  
    else if ((sessionInfo.state == CKS_RO_USER_FUNCTIONS)  
             || (sessionInfo.state == CKS_RW_USER_FUNCTIONS))  
        printf("Utilisateur authentifié.\n");  
}  
else  
{  
    switch (rc)  
    {  
        /* Si l'erreur retournée est l'une des suivantes,  
afficher cr */  
        case CKR_DEVICE_ERROR:  
        case CKR_DEVICE_REMOVED:  
        case CKR_SESSION_CLOSED:  
        case CKR_SESSION_HANDLE_INVALID:  
            printf("Erreur : 0x%08X\n", rc);  
    }  
}
```

6.5 Lecture des certificats

La carte CPS contient un certificat de signature et un certificat d'authentification. La recherche des certificats est effectuée en spécifiant le label de l'objet à rechercher (attribut CKA_LABEL).

6.5.1 Lecture du certificat de signature

Une session doit avoir été préalablement ouverte.

```
CK_RV rc = CKR_OK;
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass = CKO_CERTIFICATE;
CK_OBJECT_HANDLE hCert;
/* l'objet à rechercher est public CKA_PRIVATE->CK_FALSE */
CK_BBOOL bFalse = CK_FALSE;
/* l'objet à rechercher est token CKA_TOKEN->CK_TRUE */
CK_BBOOL bTrue = CK_TRUE;
/* Nombre maximum d'objets à récupérer */
CK_ULONG ulMaxObjectCount = 1;
/* Label de l'objet */
char label[] = "Certificat de Signature CPS";

/* Template de recherche */
CK_ATTRIBUTE searchTemplate[] =
{
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bFalse, sizeof(bFalse)},
    {CKA_LABEL, label, strlen(label)}
};

/* Template de récupération de donnée */
CK_ATTRIBUTE templateAttr[] =
{
    {CKA_VALUE, NULL_PTR, 0}
};

CK_ULONG searchTemplateSize = sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize = sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);
```



```
rc = (*pFunctionList->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

hCert = CK_INVALID_HANDLE;
rc = (*pFunctionList->C_FindObjects)(hSession, &hCert, ulMaxObjectCount,
&ulMaxObjectCount);

if (hData != CK_INVALID_HANDLE)
{
    if (rc == CKR_OK)
    {
        rc = (*pFunctionList->C_GetAttributeValue)(hSession, hCert,
templateAttr, templateAttrSize);

        if (rc == CKR_OK)
        {
            templateAttr[0].pValue = new
CK_BYTE[templateAttr[0].ulValueLen];

            rc = (*pFunctionList->C_GetAttributeValue)(hSession, hData,
templateAttr, templateAttrSize);
            if (rc == CKR_OK)
            {
                /* Nécessite OpenSSL */

                X509* cert = d2i_X509(NULL, (const unsigned
char**) &templateAttr[0].pValue, templateAttr[0].ulValueLen);
                if (cert != NULL)
                {
                    char * subject =
X509_NAME_oneline(X509_get_subject_name(cert), NULL, NULL);
                    printf("\nSUBJECT NAME:  %s\n", subject);

                    char * issuer =
X509_NAME_oneline(X509_get_issuer_name(cert), NULL, NULL);
                    printf("\nISSUER NAME:   %s\n", issuer);

                    X509_free(cert);
                }
            }
        }
    }

    rc = (*pFunctionList->C_FindObjectsFinal)(hSession);
    return rc;
}
```

6.5.2 Lecture du certificat d'authentification

Une session doit avoir été préalablement ouverte (C_OpenSession).

```
CK_RV rc = CKR_OK;
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass = CKO_CERTIFICATE;
CK_OBJECT_HANDLE hCert;
/* l'objet à rechercher est public CKA_PRIVATE->CK_FALSE */
CK_BBOOL bFalse = CK_FALSE;
/* l'objet à rechercher est token CKA_TOKEN->CK_TRUE */
CK_BBOOL bTrue = CK_TRUE;
/* Nombre maximum d'objets à récupérer */
CK_ULONG ulMaxObjectCount = 1;
/* Valeur binaire de l'objet */
CK_BYTE *value = NULL;
/* Longueur de la valeur binaire de l'objet */
CK_ULONG lenValue = sizeof(value);
/* Label de l'objet */
char label[] = "Certificat d'Authentification CPS";

/* Template de recherche */
CK_ATTRIBUTE searchTemplate[] =
{
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bFalse, sizeof(bFalse)},
    {CKA_LABEL, label, strlen(label)}
};

/* Template de récupération de donnée */
CK_ATTRIBUTE templateAttr[] =
{
    {CKA_VALUE, NULL_PTR, 0},
    {CKA_ID, NULL, 0}
};

CK_ULONG searchTemplateSize = sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize = sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);
```

```
rc = (*pFunctionList->C_FindObjectsInit) (hSession, searchTemplate,
searchTemplateSize);

hData = CK_INVALID_HANDLE;
rc = (*pFunctionList->C_FindObjects) (hSession, &hCert, ulMaxObjectCount,
&ulMaxObjectCount);

if (hData != CK_INVALID_HANDLE)
{
    if (rc == CKR_OK)
    {
        rc = (*pFunctionList->C_GetAttributeValue) (hSession, hCert,
templateAttr, templateAttrSize);

        if (rc == CKR_OK)
        {
            templateAttr[0].pValue = new
CK_BYTE[templateAttr[0].ulValueLen];
            templateAttr[1].pValue = new
CK_BYTE[templateAttr[1].ulValueLen];

            rc = (*pFunctionList->C_GetAttributeValue) (hSession,
hData, templateAttr, templateAttrSize);
            if (rc == CKR_OK)
            {
                /* Nécessite OpenSSL */
                X509* cert = d2i_X509(NULL, (const unsigned
char**) &templateAttr[0].pValue, templateAttr[0].ulValueLen);
                if (cert != NULL)
                {
                    char * subject =
X509_NAME_oneline(X509_get_subject_name(cert), NULL, NULL);
                    printf("\nSUBJECT NAME: %s\n", subject);

                    char * issuer =
X509_NAME_oneline(X509_get_issuer_name(cert), NULL, NULL);
                    printf("\nISSUER NAME: %s\n", issuer);

                    X509_free(cert);
                }
                delete templateAttr[1];
                templateAttr[1] = NULL;
            }
        }
    }
}
rc = (*pFunctionList->C_FindObjectsFinal) (hSession);

return rc;
```

6.6 Operations cryptographiques

Chaque bi-clé est associée à un certificat. Ces trois objets (clé privé, clé publique et certificat) possèdent le même CKA_ID. Le CKA_ID des clés est obtenu à l'aide d'une recherche préliminaire sur le certificat associé.

6.6.1 Authentification

L'utilisateur doit avoir préalablement ouvert une session et s'être authentifié.

```
CK_RV rc = CKR_OK;
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_RSA;
CK_OBJECT_HANDLE hKey;
CK_ULONG ulMaxObjectCount = 1;
CK_BYTE_PTR pSignature = NULL;
CK_ULONG ulSignatureLen;

/* Template de recherche */
CK_ATTRIBUTE searchTemplate[]=
{
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_ID, &cka_id_keyAuth, sizeof(cka_id_keyAuth)},
};

CK_ULONG searchTemplateSize = sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);

rc = (*pFunctionList->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

hKey = CK_INVALID_HANDLE;
rc = (*pFunctionList->C_FindObjects)(hSession, &hKey, ulMaxObjectCount,
&ulMaxObjectCount);

if (hKey != CK_INVALID_HANDLE)
{
    if (rc == CKR_OK)
    {
        CK_MECHANISM mechanism = {CKM_RSA_PKCS, NULL_PTR, 0 };
        rc = (*pFunctionList->C_SignInit)(hSession, &mechanism, hKey);
        if (rc == CKR_OK)
        {
            rc = (*pFunctionList->C_Sign)(hSession, pRandomData,
            ulRandomDataLen, NULL, &ulSignatureLen);
            if (rc == CKR_OK)
            {
                pSignature = new CK_BYTE[ulSignatureLen];
                rc = (*pFunctionList->C_Sign)(hSession,
                ulRandomDataLen, pSignature,
pRandomData,
&ulSignatureLen);
            }
            delete pSignature;
            pSignature = NULL;
        }
    }
}

rc = (*pFunctionList->C_FindObjectsFinal)(hSession);
return rc;
```

6.6.2 Signature

L'utilisateur doit avoir préalablement ouvert une session et s'être authentifié.

```
CK_RV rc = CKR_OK;
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_RSA;
CK_OBJECT_HANDLE hKey;
CK_ULONG ulMaxObjectCount = 1;
CK_BYTE_PTR pSignature = NULL;
CK_ULONG ulSignatureLen;

/* Template de recherche */
CK_ATTRIBUTE searchTemplate[]=
{
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_ID, &cka_id_keySign, sizeof(cka_id_keySign)},
};

CK_ULONG searchTemplateSize = sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);

rc = (*pFunctionList->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

hKey = CK_INVALID_HANDLE;
rc = (*pFunctionList->C_FindObjects)(hSession, &hKey, ulMaxObjectCount,
&ulMaxObjectCount);

if (hKey != CK_INVALID_HANDLE)
{
    if (rc == CKR_OK)
    {
        CK_MECHANISM mechanism = {CKM_SHA256_RSA_PKCS, NULL_PTR, 0 };
        rc = (*pFunctionList->C_SignInit)(hSession, &mechanism, hKey);
        if (rc == CKR_OK)
        {
            rc = (*pFunctionList->C_Sign)(hSession, pData, pDataLen,
NULL,
&ulSignatureLen);
            if (rc == CKR_OK)
            {
                pSignature = new CK_BYTE[ulSignatureLen];
                rc = (*pFunctionList->C_Sign)(hSession, pData,
pDataLen,
pSignature, &ulSignatureLen);
            }
            delete pSignature;
            pSignature = NULL;
        }
    }
}

rc = (*pFunctionList->C_FindObjectsFinal)(hSession);
return rc;
```

7 Gestion des données métier

7.1 Recherche

Du point de vue du standard PKCS#11, les données métier de la carte CPS3 correspondent à des objets de classe CKO_DATA.

Il est fortement déconseillé de réaliser une recherche globale sur tous les objets de type CKO_DATA pour des raisons de performances. L'application devra privilégier la recherche des objets de données en spécifiant le label de l'objet à rechercher (attribut CKA_LABEL).

Pour la plupart des données métier recherchées, le label à utiliser est différent selon que l'on travaille avec une carte CPS2ter ou une carte CPS3.

7.2 Structuration des données

Les données métier de la carte sont retournées au format TLV (Tag, Length, Value). Leur interprétation est à la charge de l'application, en s'appuyant sur la description de ce format qui est donnée dans le document : ASIP_CPS3_Données-métier_vx.x.x.pdf.

7.3 Gestion des informations carte

7.3.1 Recherche

Pour la recherche des informations associées à la carte, la constitution du template de recherche est effectuée comme suit. Les attributs à définir dans le template sont CKA_CLASS, CKA_TOKEN, CKA_PRIVATE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA
- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte
- CKA_PRIVATE est positionné à faux (CK_FALSE) pour indiquer que l'on recherche un objet public
- CKA_LABEL : le renseignement de cet attribut vaut '**CPS2TER_ID_CARD**' pour une carte CPS2ter, '**CPS_ID_CARD**' sinon.

Par la suite, la recherche effective de l'objet est réalisée par les appels aux fonctions C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal dans cet ordre.

7.3.2 Récupération des données

La récupération des données est, elle aussi, basée sur un template. Ce template définit les attributs de l'objet PKCS qui seront retrouvés grâce à l'appel à la fonction C_GetAttributeValue. Pour notre propos, nous souhaitons obtenir la valeur de l'objet PKCS CKO_DATA. Nous définissons alors dans le template une seule structure CK_ATTRIBUTE avec les propriétés :

- type : CKA_VALUE correspond à un buffer de données qui contient au retour de l'appel C_GetAttributeValue la valeur binaire de l'objet CKO_DATA
- pValue : adresse du buffer de données
- ulValueLen : taille en octets du buffer de données

L'extrait de code suivant montre comment effectuer une recherche des informations carte suivant le type de carte. Il suppose qu'une session ait été déjà ouverte.

```
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass=CKO_DATA;
CK_OBJECT_HANDLE hData;
/* l'objet à rechercher est public CKA_PRIVATE -> CK_FALSE */
CK_BBOOL bFalse=CK_FALSE;
/* l'objet à rechercher est token CKA_TOKEN -> CK_TRUE */
CK_BBOOL bTrue=CK_TRUE;
/* Nombre maximum d'objets à récupérer */
CK_ULONG ulMaxObjectCount = 16;
/* Valeur binaire de l'objet */
CK_BYTE value[128];
/* Longueur de la valeur binaire de l'objet */
CK_ULONG lenValue = sizeof(value);
/* Label de l'objet */
char label[]="CPS2TER_ID_CARD";
CK_RV rc = CKR_OK;

/* Template de recherche */
CK_ATTRIBUTE searchTemplate[]={
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bFalse, sizeof(bFalse)},
    {CKA_LABEL, label, strlen(label)}
};

/* Template de récupération de donnée */
CK_ATTRIBUTE templateAttr [] =
{
    {CKA_VALUE, value, lenValue}
};

CK_ULONG searchTemplateSize=sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize=sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);

if(bCarteCPS3)
{
    strcpy(label, "CPS_ID_CARD");
    certTemplate[3].ulValueLen = strlen(label);
}
```



```
rc = (*pListeFonctions->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

while(rc == CKR_OK && ulMaxObjectCount)
{
    hData = CK_INVALID_HANDLE;
    rc = (*pListeFonctions->C_FindObjects)(hSession, &hData,
ulMaxObjectCount, &ulMaxObjectCount);

    if(hData != CK_INVALID_HANDLE)
    {
        if(rc == CKR_OK)
        {
            memset(value, 0, sizeof(value));
            rc = (*pListeFonctions->C_GetAttributeValue)(hSession,
hData, templateAttr, templateAttrSize);

            if(rc == CKR_OK)
            {
                /* Interprétation des donnees */
            }
        }
    }
}

rc = (*pListeFonctions->C_FindObjectsFinal)( hSession );
```

7.4 Gestion des informations porteur

7.4.1 Recherche

Pour la recherche des informations associées au porteur de la carte, la constitution du template de recherche est effectuée comme suit. Les attributs à définir dans le template sont CKA_CLASS, CKA_TOKEN, CKA_PRIVATE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA
- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte
- CKA_PRIVATE est positionné à faux (CK_FALSE) pour indiquer que l'on recherche un objet public
- CKA_LABEL : la valeur de cet attribut vaut 'CPS2TER_NAME_PS' pour une carte CPS2ter, 'CPS_NAME_PS' pour une carte CPS3.

Par la suite, la recherche effective de l'objet est réalisée par les appels C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal dans cet ordre

7.4.2 Récupération des données

La récupération des données s'effectue comme au § 7.3.2,

L'extrait de code suivant montre comment effectuer une recherche des informations du porteur. Il suppose qu'une session ait été déjà ouverte.

```
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass=CKO_DATA;
CK_OBJECT_HANDLE hObject;
/* l'objet à rechercher est public CKA_PRIVATE -> CK_FALSE */
CK_BBOOL bFalse=CK_FALSE;
/* l'objet à rechercher est token CKA_TOKEN -> CK_TRUE */
CK_BBOOL bTrue=CK_TRUE;
/* Nombre maximum d'objets à récupérer */
CK_ULONG ulMaxObjectCount = 16;
/* Valeur binaire de l'objet */
CK_BYTE value[128];
/* Longueur de la valeur binaire de l'objet */
CK_ULONG lenValue = sizeof(value);

char label[]="CPS2_NAME_PS";
CK_RV rc = CKR_OK;
/* Template de recherche */
CK_ATTRIBUTE searchTemplate[]={
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bFalse, sizeof(bFalse)},
    {CKA_LABEL, label, strlen(label)}
};
/* Template de récupération de donnée */
CK_ATTRIBUTE templateAttr [] =
{
    {CKA_VALUE, value, lenValue}
};
CK_ULONG searchTemplateSize=sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize=sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);
```

```
if(bCarteCPS3)
{
    strcpy(label, "CPS_NAME_PS");
    certTemplate[3].ulValueLen = strlen(label);
}

rc = (*pListeFonctions->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

while(rc == CKR_OK && ulMaxObjectCount) {

    hData = CK_INVALID_HANDLE;
    rc = (*pListeFonctions->C_FindObjects)(hSession, &hData,
ulMaxObjectCount, &ulMaxObjectCount);

    if(hData != CK_INVALID_HANDLE) {
        if(rc == CKR_OK) {
            /* Ici, on récupère le CKA_VALUE de l'objet */
            memset(value, 0, sizeof(value));
            rc = (*pListeFonctions->C_GetAttributeValue)(hSession,
hData, templateAttr, templateAttrSize);

            if(rc == CKR_OK) {
                /* Interprétation des donnees */
            }
        }
    }
}

rc = (*pListeFonctions->C_FindObjectsFinal)( hSession );
```

7.5 Gestion des informations PS2

7.5.1 Recherche

La recherche des informations PS2 s'effectue en deux itérations car les données sont localisées sur deux fichiers carte différents. La constitution du template de recherche est effectuée comme suit. Les attributs à définir dans le template sont CKA_CLASS, CKA_TOKEN, CKA_PRIVATE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA
- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte
- CKA_PRIVATE est positionné à faux (CK_FALSE) pour indiquer que l'on recherche un objet public
- CKA_LABEL : la valeur de cet attribut vaut '**CPS2TER_INFO_PS**' pour la 1ere itération puis '**CPS2TER_QUAL_PS**' pour la 2eme dans le cas d'une CPS2ter. Pour une carte CPS3, seule la valeur '**CPS_INFO_PS**' n'a de sens. En effet, les informations de qualification du PS ne sont pas stockées dans les cartes CPS3.

Par la suite, la recherche effective de l'objet est réalisée par les appels C_FindObjectsInit, C_FindObjects, C_FindObjetsFinal dans cet ordre

7.5.2 Récupération des données

La récupération des données s'effectue comme au § 7.3.2,

L'extrait de code suivant montre comment effectuer une recherche des informations PS2. Il suppose qu'une session ait été déjà ouverte.

```
CK_OBJECT_CLASS objClass=CKO_DATA;
CK_OBJECT_HANDLE hData;
CK_BBOOL bFalse=CK_FALSE;
CK_BBOOL bTrue=CK_TRUE;
CK_ULONG ulMaxObjectCount = 16;
CK_BYTE value[1024];
CK_ULONG lenValue = sizeof(value);

/* Spécifier deux labels car les données PS2 sont
sur deux fichiers différents */
char *label[2]={ "CPS2TER_INFO_PS", "CPS2TER_QUAL_PS" };
int nbLabels = sizeof(label)/sizeof(char *);
int j;

CK_RV rc = CKR_OK;

CK_ATTRIBUTE searchTemplate[]={
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bFalse, sizeof(bFalse)},
    {CKA_LABEL, NULL, 0}
};

CK_ATTRIBUTE templateAttr [] =
{
    {CKA_VALUE, value, lenValue}
};
```

```
CK_ULONG searchTemplateSize=sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);

if(bCarteCPS3)
{
    label[0] = "CPS_INFO_PS";
    /* En CPS3, l'objet métier CPS_QUALIF_PS n'existe pas */
    nbLabels-- ;
}

for(j=0 ; j<nbLabels ;j++ )
{
    lenValue = sizeof(value);
    memset(value, '\0', lenValue);
    ulMaxObjectCount = 16;
    searchTemplate[3].pValue = label[j];
    searchTemplate[3].ulValueLen = strlen(label[j]);
    templateAttr[0].ulValueLen = lenValue;

    rc = (*pFunctionList->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

    while(rc == CKR_OK && ulMaxObjectCount) {
        hData = CK_INVALID_HANDLE;
        rc = (*pFunctionList->C_FindObjects)(hSession, &hData,
ulMaxObjectCount, &ulMaxObjectCount);

        if(hData != CK_INVALID_HANDLE) {

            if(rc == CKR_OK) {

                rc = (*pFunctionList->C_GetAttributeValue)(hSession,
hData, templateAttr, 1);

                if(rc == CKR_OK) {
                    /* Interprétation des donnees */
                }
            }
        }

        rc = (*pFunctionList->C_FindObjectsFinal)( hSession );
    } /* fin du for */
}
```

7.6 Gestion des situations d'exercice

7.6.1 Recherche d'une situation

On recherche une situation d'exercice de numéro donné, ce numéro variant entre 0 et 15.

Pour la recherche d'une situation d'exercice de numéro 'N', la constitution du template de recherche est effectuée comme suit. On définit une structure CK_ATTRIBUTE dans le template. Les attributs de cette structure sont CKA_CLASS, CKA_TOKEN, CKA_PRIVATE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA

- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte
- **CKA_PRIVATE est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet privé**
- CKA_LABEL : cet attribut est formaté selon le modèle '**CPS2TER_ACTIVITY_XY_PS**' où XY est le numéro de situation à rechercher variant entre 01 et 16. Pour une carte CPS3, le modèle à utiliser est '**CPS_ACTIVITY_XY_PS**'

Par la suite, la recherche effective de l'objet est réalisée par les appels C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal dans cet ordre

7.6.2 Récupération des données

La récupération des données s'effectue comme au § 7.3.2

L'extrait de code suivant montre comment rechercher la première situation d'exercice (de numéro 0). Il suppose qu'une session ait été déjà ouverte et que l'utilisateur s'est authentifié sur cette session.

```
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass=CKO_DATA;
CK_OBJECT_HANDLE hData;
/* l'objet à rechercher est privé CKA_PRIVATE -> CK_TRUE */
/* l'objet à rechercher est token CKA_TOKEN -> CK_TRUE */
CK_BBOOL bTrue=CK_TRUE;
/* Nombre maximum d'objets à recuperer */
CK_ULONG ulMaxObjectCount = 16;
/* Valeur binaire de l'objet */
CK_BYTE value[128];
/* Longueur de la valeur binaire de l'objet */
CK_ULONG lenValue = sizeof(value);
char label[24]="CPS2TER_ACTIVITY_";
/* On recherche la première situation */
int numSituation = 0;
CK_ATTRIBUTE searchTemplate[]={
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bTrue, sizeof(bTrue)},
    {CKA_LABEL, label, strlen(label)}
};

CK_ATTRIBUTE templateAttr [] =
{
    {CKA_VALUE, value, lenValue}
};

CK_ULONG searchTemplateSize=sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize=sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);
int compteur = 0;
```

```
/* Mettre le label sous la forme 'CPS(2TER)_ACTIVITY_XY' */
if( bCarteCPS3 ) {
    strcpy(label, "CPS_ACTIVITY_");
}
sprintf(label + strlen(label), "%02d_PS", numSituation);
searchTemplate[3].ulValueLen = strlen(label);

rc = (*pListeFonctions->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

while(rc == CKR_OK && ulMaxObjectCount)
{
    hData = CK_INVALID_HANDLE;
    rc = (*pListeFonctions->C_FindObjects)(hSession, &hData,
ulMaxObjectCount, &ulMaxObjectCount);

    if(hData != CK_INVALID_HANDLE)
    {
        compteur++;

        if(rc == CKR_OK)
        {
            memset(value, 0, sizeof(value));
            rc = (*pListeFonctions->C_GetAttributeValue)(hSession,
hData, templateAttr, templateAttrSize);

            if(rc == CKR_OK)
            {
                /* Interprétation des données */
            }
        }
    }
}
rc = (*pListeFonctions->C_FindObjectsFinal)( hSession );
```


7.7 Gestion des situations de Facturation

7.7.1 Recherche de toutes les situations de facturation

On recherche toutes les situations de facturations à partir de la situation d'indice 0.

Pour la recherche des situations de facturation, la constitution du template de recherche est effectuée comme suit. On définit une structure CK_ATTRIBUTE dans le template. Les attributs de cette structure sont CKA_CLASS, CKA_TOKEN, CKA_PRIVATE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA
- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte
- **CKA_PRIVATE est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet privé**
- CKA_LABEL : cet attribut vaut 'CPS_SIT_FACT'.

Par la suite, la recherche effective de l'objet est réalisée par les appels C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal dans cet ordre

Remarque : Dans le volet IAS de la carte CPS3, il n'y a pas de stockage des situations de facturation.

7.7.2 Récupération des données

La récupération des données s'effectue comme au § 7.3.2

L'extrait de code suivant montre comment retrouver toutes les situations de facturation. Il suppose qu'une session ait été déjà ouverte et que l'utilisateur s'est authentifié sur cette session.

```
/* classe d'objet à rechercher */
CK_OBJECT_CLASS objClass=CKO_DATA;
CK_OBJECT_HANDLE hData;
/* l'objet à rechercher est privé CKA_PRIVATE -> CK_TRUE */
/* l'objet à rechercher est token CKA_TOKEN -> CK_TRUE */
CK_BBOOL bTrue=CK_TRUE;
/* Nombre maximum d'objets à recuperer */
CK_ULONG ulMaxObjectCount = 16;

char label[15]="CPS_SIT_FACT";

CK_ATTRIBUTE searchTemplate[]={
    {CKA_CLASS, &objClass, sizeof(objClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_PRIVATE, &bTrue, sizeof(bTrue)},
    {CKA_LABEL, label, strlen(label)}
};

CK_ATTRIBUTE templateAttr [] =
{
    {CKA_VALUE, NULL_PTR, 0}
};

CK_ULONG searchTemplateSize=sizeof(searchTemplate)/sizeof(CK_ATTRIBUTE);
CK_ULONG templateAttrSize=sizeof(templateAttr)/sizeof(CK_ATTRIBUTE);
```

```
rc = (*pFunctionList->C_FindObjectsInit)(hSession, searchTemplate,
searchTemplateSize);

while(rc == CKR_OK && ulMaxObjectCount)
{
    hData = CK_INVALID_HANDLE;
    rc = (*pFunctionList->C_FindObjects)(hSession, &hData,
ulMaxObjectCount, &ulMaxObjectCount);

    if(hData != CK_INVALID_HANDLE) {

        if(rc == CKR_OK)
        {
            templateAttr[0].pValue = NULL_PTR;
            templateAttr[0].ulValueLen = 0;
            rc = (*pFunctionList->C_GetAttributeValue)(hSession,
hData, templateAttr, templateAttrSize);

            if(rc == CKR_OK)
            {
                templateAttr[0].pValue =
malloc(templateAttr[1].ulValueLen * sizeof(CK_BYTE));
                rc = (*pFunctionList->C_GetAttributeValue)(hSession,hData, templateAttr, templateAttrSize);
            }

            if(rc == CKR_OK) {
                /* Interprétation des données */
            }

        }
    }

rc = (*pFunctionList->C_FindObjectsFinal)( hSession );

if(templateAttr[0].pValue != NULL_PTR)
    free(templateAttr[0].pValue);
```

8 Spécificités du sans contact

Ce paragraphe ne s'applique qu'à la carte CPS3.

La partie sans contact de la carte CPS3 met à disposition des applications un certificat technique d'authentification ainsi que la bi-clé associée.

Du point de vue PKCS#11, le certificat technique est un objet de classe CKO_CERTIFICATE ayant pour label '**CPS – Certificat Technique**'.

La recherche de ce certificat est basée sur le template de recherche dont la structure CK_ATTRIBUTE est définie comme suit :

- CKA_CLASS : CKO_CERTIFICATE
- CKA_TOKEN : CK_TRUE
- CKA_PRIVATE : CK_FALSE
- CKA_LABEL : le nom indiqué ci-dessus en respectant la casse

En vue d'effectuer une signature, on recherche d'abord la clé privée correspondante.

Du point de vue PKCS#11, la clé privée est un objet de classe CKO_PRIVATE_KEY ayant pour label '**CPS_PRIV_TECH_AUT**'.

La recherche de cette clé est basée sur le template de recherche dont la structure CK_ATTRIBUTE est définie comme suit :

- CKA_CLASS : CKO_PRIVATE_KEY
- CKA_TOKEN : CK_TRUE
- CKA_PRIVATE : CK_FALSE
- CKA_LABEL : le nom indiqué ci-dessus en respectant la casse

9 Gestion du jeton d'établissement

Le jeton applicatif est un espace de stockage sur la partie sans contact de la carte CPS3 permettant à une application CPS3 d'écrire des données.

9.1 Recherche du jeton

Du point de vue PKCS#11, le jeton applicatif est un objet de classe CKO_DATA ayant pour label 'CPS_DATA'

Pour la recherche du jeton applicatif, le template de recherche est constitué d'une seule structure CK_ATTRIBUTE. Les attributs à définir dans la structure sont CKA_CLASS, CKA_TOKEN, CKA_MODIFIABLE et CKA_LABEL.

- CKA_CLASS vaut CKO_DATA.
- CKA_TOKEN est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet de la carte.
- CKA_MODIFIABLE est positionné à vrai (CK_TRUE) pour indiquer que l'on recherche un objet modifiable.
- CKA_LABEL : cet attribut vaut '**CPS_DATA**'.

Par la suite, la recherche effective de l'objet est réalisée par les appels C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal dans cet ordre.

9.2 Récupération de la valeur du jeton

Celle-ci s'effectue en deux temps. Il faut d'abord obtenir la taille du jeton, allouer le buffer de données puis récupérer au final la valeur du jeton.

9.2.1 Obtention de la taille du jeton

Pour l'obtention de la taille du jeton, un template de récupération de données est créé contenant une seule structure CK_ATTRIBUTE. La structure est définie comme suit.

- type vaut CKA_VALUE
- pValue vaut NULL_PTR pour signifier que seule la taille des données sera récupérée
- ulValueLen vaut 0

Par la suite, l'obtention de la taille du jeton est réalisée par l'appel C_GetAttributeValue. C'est la propriété **ulValueLen** qui est renseignée avec la taille désirée.

9.2.2 Obtention de la valeur du jeton

Par la suite, on alloue un buffer de données de la taille obtenue au § 9.2.1 dans la propriété ulValueLen.

Puis on redéfinit la structure CK_ATTRIBUTE du template créé au § 9.2.1 comme suit.

- type reste inchangé
- pValue vaut maintenant l'adresse du buffer de données alloué
- ulValueLen reste inchangé

Enfin, l'obtention de la valeur du jeton est réalisée par un second appel à C_GetAttributeValue

9.3 Modification du jeton

On alloue un nouveau buffer de modification de la taille obtenue au § 9.2.1 puis on copie les données modifiées dans ce buffer.

On utilise un nouveau template de modification contenant une seule structure CK_ATTRIBUTE définie comme suit.

- type : CKA_VALUE
- pValue : adresse du buffer de modification alloué
- ulValueLen : taille en octets du buffer de modification

On appelle par la suite la fonction C_SetAttributeValue en passant ce template de modification pour mettre à jour le jeton applicatif.

L'extrait de code suivant montre comment rechercher et modifier le jeton d'établissement.

```
CK_RV rv=CKR_OK;
CK_BBOOL bFalse=FALSE;
CK_BBOOL bTrue=TRUE;
CK_OBJECT_CLASS dataClass=CKO_DATA;
CK_CHAR dataLabel[]="CPS_DATA";

CK_ATTRIBUTE dataTemplate[]=
{
    {CKA_CLASS, &dataClass, sizeof(dataClass)},
    {CKA_TOKEN, &bTrue, sizeof(bTrue)},
    {CKA_MODIFIABLE, &bTrue, sizeof(bTrue)},
    {CKA_LABEL, dataLabel, strlen((char *)dataLabel)}
};

CK_ULONG dataTemplateSize=sizeof(dataTemplate)/sizeof(CK_ATTRIBUTE);
CK_ATTRIBUTE dataValueTemplate[]={CKA_VALUE, NULL, 0};
CK_ATTRIBUTE dataValueModTemplate[]={CKA_VALUE, NULL, 0};
CK_ULONG
dataValueTemplateSize=sizeof(dataValueTemplate)/sizeof(CK_ATTRIBUTE);
CK_OBJECT_HANDLE hObject=0;
CK_BYTE pin[]="1234";
CK_ULONG i;

rv=(*pFunctionList->C_Login)(hSession,CKU_USER,pin,strlen((char *)pin));

if (rv!=CKR_OK && rv!=CKR_USER_ALREADY_LOGGED_IN &&
rv!=CKR_USER_PIN_NOT_INITIALIZED)
    return rv;

if (rv==CKR_USER_PIN_NOT_INITIALIZED)
{
    rv = CKR_OK; // pas la peine d'aller plus loin, ce n'est pas
modifiable en sans contact
    return rv;
}

rv=(*pFunctionList->C_FindObjectsInit)(hSession, dataTemplate,
dataTemplateSize);
if(rv!=CKR_OK)
    return rv;

while (TRUE) {
    CK_OBJECT_HANDLE objectHandle[]={0,0};
    CK_ULONG i,nbObject=0;
    rv=(*pFunctionList->C_FindObjects)(hSession, objectHandle,
sizeof(objectHandle)/sizeof(CK_OBJECT_HANDLE), &nbObject);
    if(rv!=CKR_OK)
        return rv;

    for (i=0;i<nbObject;i++)
    {
        hObject=objectHandle[i];
    }

    if (nbObject!=sizeof(objectHandle)/sizeof(CK_OBJECT_HANDLE))
        break;
}

/* fin du while */
```

```
rv=(*pFunctionList->C_FindObjectsFinal)(hSession);
if(rv!=CKR_OK)
    return rv;

if (hObject==0)
{
    rv = CKR_OK; // c'est probablement une carte CPS2ter, rien à modifier
    donc
    return rv;
}

/* recuperation de la valeur du jeton */
rv=(*pFunctionList->C_GetAttributeValue)(hSession, hObject,
dataValueTemplate, dataValueTemplateSize);
if(rv!=CKR_OK)
    return rv;

dataValueTemplate[0].pValue=malloc(dataValueTemplate[0].ulValueLen*sizeof(CK_BYTE));
rv=(*pFunctionList->C_GetAttributeValue)(hSession, hObject,
dataValueTemplate, dataValueTemplateSize);
if(rv!=CKR_OK)
    return rv;

/* modification de la valeur du jeton */
dataValueModTemplate[0].pValue=malloc(dataValueTemplate[0].ulValueLen*sizeof(CK_BYTE));
dataValueModTemplate[0].ulValueLen=dataValueTemplate[0].ulValueLen;
for (i=0;i<dataValueTemplate[0].ulValueLen;i++)
{
    ((CK_BYTE_PTR)dataValueModTemplate[0].pValue)[i]=((CK_BYTE_PTR)dataValueTemplate[0].pValue)[i]+(CK_BYTE)i;
}

rv=(*pFunctionList->C_SetAttributeValue)(hSession, hObject,
dataValueModTemplate, dataValueTemplateSize);
if(rv!=CKR_OK)
    return rv;

/* recuperation de la nouvelle valeur du jeton pour comparaison */
rv=(*pFunctionList->C_GetAttributeValue)(hSession, hObject,
dataValueTemplate, dataValueTemplateSize);
if(rv!=CKR_OK)
    return rv;

if (memcmp(dataValueTemplate[0].pValue, dataValueModTemplate[0].pValue,
dataValueTemplate[0].ulValueLen)!=0) {
    /* Les données sont différentes */
}
}
```