OpenSC

Le projet OpenSC fournit un ensemble de bibliothèques et d'outils pour travailler avec des cartes à puce (Token). Son principal objectif est de prendre en charge les cartes cryptographiques et de faciliter leur utilisation dans des applications de sécurité telles que l'authentification, le chiffrement des e-mails et les signatures numériques.

OpenSC met en œuvre les interfaces standard pour les cartes à puce, telles que l'API PKCS#11, le pilote de carte à puce de Windows et CryptoTokenKit sur macOS.

Dans le cadre d'esup-signature, OpenSC permet l'accès aux supports cryptographiques pour la signature avec des certificats eIDas (RGS**) pour les signatures cachet d'établissement (Utilisation d'un certificat cachet d'établissement à l'université de Rouen ou encore Instance Esup-Signature dédiée au consortium ESUP-Portail) ou pour les signature avec Esup-DSS-Client (Esup-DSS-Client).

En général les supports cartographiques (token) se présentent sous la forme d'un lecteur de carte a puce USB (une clé) accompagné d'une carte SIM. **Il faut donc qu'OpenSC supporte à la fois le lecteur et à la fois la carte SIM.**

La liste des matériels supportés nativement par les drivers libres intégrés est donnée ici : https://github.com/OpenSC/OpenSC/wiki/Supported-hardware-%28smart-cards-and-USB-tokens%29

Notez que si les drivers libres embarqués par OpenSC ne peuvent pas être utilisés, **il est aussi possible d'utiliser opensc avec les drivers propriétaires**. Le déploiement et configuration, notamment des postes clients, seront alors cependant un peu plus compliqués.

Sources

https://github.com/OpenSC/OpenSC

Documentation

La documentation est très bien faite, elle se trouve ici : https://github.com/OpenSC/OpenSC/wiki

Installation

L'installation est documentée ici :

- Pour Linux (dans le cadre de la signature cachet d'établissement) : https://github.com/OpenSC/OpenSC/wiki/Compiling-and-Installing-on-Unix-flavors
- Pour Windows : https://github.com/OpenSC/OpenSC/wiki/Windows-Quick-Start
- Pour MacOS : https://github.com/OpenSC/OpenSC/wiki/macOS-Quick-Start

Compatibilité matérielle

Si l'on ne parvient pas à signer directement avec le couple clé/carte SIM que vous possédez, vous pouvez contrôler votre matériel avec les commandes suivantes :

Fonctionnement du lecteur

Pour tester le fonctionnement de votre lecteur :

```
pkcs11-tool -L
```

Exemple de resultat correct :

```
Available slots:
Slot 0 (0x0): Feitian SCR301 (FFFFFFFFFFFF) 00 00
  token label        : Université de XXXX...
  token manufacturer : Gemalto
  token model        : PKCS#15 emulated
  token flags        : login required, rng, token initialized, PIN initialized
  hardware version   : 0.0
  firmware version   : 0.0
  serial num         : xxxxxxxxxxxxxx
  pin min/max        : X/X
```

A cette étape, il est important de vérifier que votre lecteur USB est bien compatible avec OpenSC, voir : https://github.com/OpenSC/OpenSC/wiki/Supported-hardware-%28smart-cards-and-USB-tokens%29

Même s'il n'est pas listé, il nous a été possible de signer en insérant la carte sim (avec la carte complète style carte de crédit) dans un vieux lecteur Identiv 4700F. La liste d'OpenSC n'est donc pas exhaustive.

N'hésitez pas à modifier le tableau ci-dessous pour y insérer les références des lecteurs fonctionnels chez vous

OpenSC	0.23.0	0.24.0
Feitian R301 (https://www.scardshop.com/lecteurs-cartes-sim/115-feitian-sim-r301-b6.html)	OK	OK
Identiv 4701F (https://www.shopnfc.com/fr/lecteurs-encodeurs-nfc/385-utrust-4701-f-double-interface-smart-card-reader.html)	OK	OK

Fonctionnement de la carte SIM

Ce point est plus problématique. OpenSC permet une abstraction matérielle mais la compatibilité est nativement limitée aux drivers libres embarqués par OpenSC.
Par exemple, les derrières carte SIM obtenues auprès de Certinomis (septembre 2023 ?) ne sont pas reconnues pas les drivers libres embarqués par OpenSC 0.23.0

Plusieurs choix s'offrent alors à l'exploitant :

- tenter de forcer l'usage d'un driver déjà intégré si on pense que la carte non reconnue correspond en fait à une variante d'une carte /driver reconnue dans opensc
- utiliser le driver propriétaire au travers toujours des utilitaires opensc, ce qui permet à esup-signature/esup-dss-client de préserver cette couche d'abstraction (en cours d'étude/développement)
- enfin pour le cachet serveur côté esup-signature, il est aussi possible de configurer directement le module/driver propriétaire (sans passer par opensc).

Pour obtenir la liste des drivers supportés nativement :

```
opensc-tool -D
```

En version 0.23.0 on obtient :

```
Available card drivers:
  cardos           Siemens CardOS
  cyberflex        Schlumberger Cyberflex
  gemsafeV1        Gemalto GemSafe V1 applet
  starcos          STARCOS
  tcos             TCOS 3.0
  oberthur         Oberthur AuthentIC.v2/CosmopolIC.v4
  authentic        Oberthur AuthentIC v3.1
  iasecc           IAS-ECC
  belpic           Belpic cards
  entersafe        entersafe
  epass2003        epass2003
  rutoken          Rutoken driver
  rutoken_ecp      Rutoken ECP and Lite driver
  myeid            MyEID cards with PKCS#15 applet
  dnie             DNIe: Spanish eID card
  MaskTech         MaskTech Smart Card
  esteid2018       EstEID 2018
  idprime          Gemalto IDPrime
  coolkey          COOLKEY
  muscle           MuscleApplet
  sc-hsm           SmartCard-HSM
  mcrd             MICARDO 2.1 / EstEID 3.0 - 3.5
  setcos           Setec cards
  PIV-II           Personal Identity Verification Card
  cac              Common Access Card (CAC)
  itacns           Italian CNS
  isoApplet        Javacard with IsoApplet
  gids             GIDS Smart Card
  openpgp          OpenPGP card
  jpki             JPKI(Japanese Individual Number Cards)
  npa              German ID card (neuer Personalausweis, nPA)
  cac1             Common Access Card (CAC 1)
  nqapplet         NQ-Applet
  default          Default driver for unknown cards
```

Pour lister les périphériques présents sur votre machine et obtenir le code ATR de votre carte SIM:

```
opensc-tool -a
```

Pour afficher le nom de la carte telle qui reconnue par OpenSC :

```
opensc-tool -n
```

Driver correctement détecté

Exemple de résultat avec les "anciennes" carte certinomis "Gemalto IDPrime" :

```
# opensc-tool -a
Using reader with a card: Feitian SCR301 (FFFFFFFFFFFF) 00 00
3b:7f:96:00:00:80:31:80:65:b0:85:59:56:fb:12:0f:fe:82:90:00
```

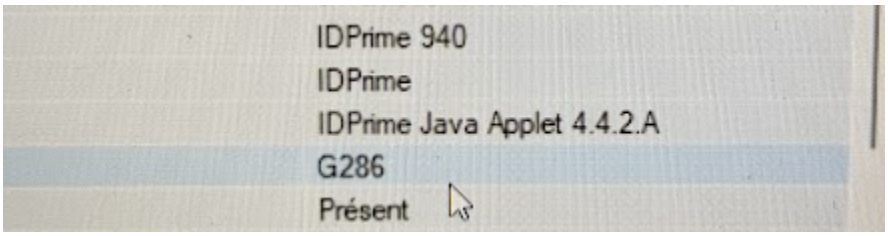
Ici le code ATR est **3b:7f:96:00:00:80:31:80:65:b0:85:59:56:fb:12:0f:fe:82:90:00** . Si on le cherche dans le dépot de OpenSC (ici le lien pour cherches directement sur github: https://github.com/search?q=repo%3AOpenSC%2FOpenSC+%223b%3A7f%3A96%3A00%3A00%3A80%3A31%3A80%3A65%3Ab0%3A85%3A59%3A56%3Afb%3A12%3A0f%3Afe%3A82%3A90%3A00%22&type=code)

On obtient :

src/libopensc/card-idprime.c

```
57            "ff:ff:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:00:00:ff:ff:ff",
58            "Gemalto IDPrime 930/3930",
59            SC_CARD_TYPE_IDPRIME_930, 0, NULL },
60          { "3b:7f:96:00:00:80:31:80:65:b0:85:59:56:fb:12:0f:fe:82:90:00",
61            "ff:ff:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:00:00:ff:ff:ff",
62            "Gemalto IDPrime 940",
63            SC_CARD_TYPE_IDPRIME_940, 0, NULL },
```

L'ATR est présent dans le fichier card-idprime.c . On voit que cette carte est reconnue comme "Gemalto IDPrime 940". Cela correspond à l'information que l'on peut obtenir via le pilote Safenet sous windows :



OpenSC reconnait bien la carte comme tel également :

```
# opensc-tool -n
Using reader with a card: Feitian SCR301 (FFFFFFFFFFFF) 00 00
Gemalto IDPrime 940
```

Pour vérifier réellement que tout fonctionne, il faut vérifier l'authentification sur le matériel en lançant la commande suivante :

```
pkcs11-tool --login --test

#Sous MacOs : /Library/OpenSC/bin/pkcs11-tool --login --test
```

On doit alors entrer le code PIN. Le résultat doit ressembler à ça :

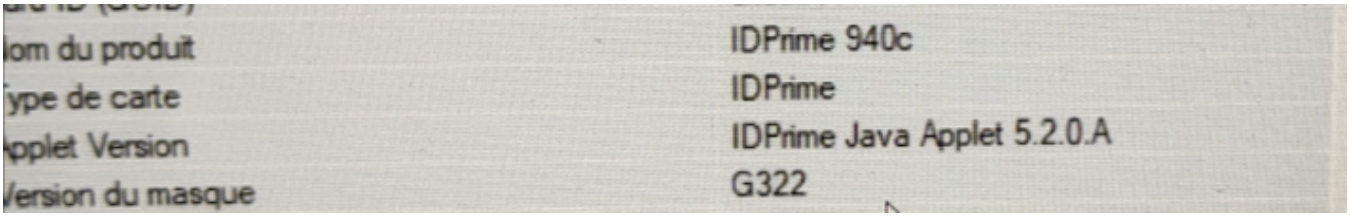
```
C_SeedRandom() and C_GenerateRandom():
  seeding (C_SeedRandom) not supported
  seems to be OK
Digests:
  all 4 digest functions seem to work
  MD5: OK
  RIPEMD160: OK
  SHA-1: OK
  SHA256: OK
Ciphers: not implemented
Signatures (currently only for RSA)
  testing key 0 (Private key 1)
  all 4 signature functions seem to work
  testing signature mechanisms:
    RSA-PKCS: OK
    SHA256-RSA-PKCS: OK
Verify (currently only for RSA)
  testing key 0 (Private key 1)
    RSA-PKCS: OK
Unwrap: not implemented
Decryption (currently only for RSA)
  testing key 0 (Private key 1) -- can't be used to decrypt, skipping
No errors
```

Driver non correctement détecté

Maintenant, lorsque l'on prend les dernières carte SIM livrées par Certinomis, l'ATR à changé :

```
# opensc-tool -a
Using reader with a card: Feitian SCR301 (FFFFFFFFFFFF) 00 00
3b:7f:96:00:00:80:31:80:65:b0:85:05:00:39:12:0f:fe:82:90:00
```

On ne trouve pas de référence précise à cet ATR dans OpenSC, mais si on ouvre le token dans Safenet on voit ceci :



La référence est IDPrime 940**c**

Cependant OpenSC la reconnait comme Gemalto IDPrime (generic) :

```
# opensc-tool -n
Using reader with a card: Feitian SCR301 (FFFFFFFFFFFF) 00 00
Gemalto IDPrime (generic)
```

Même si l'ATR de cette carte n'est ici pas directement référencé dans le code source d'OpenSC, avec le jeu des masques, la 940c est donc finalement vu comme une **Gemalto IDPrime (generic).**

Le souci est qu'avec ce driver+type, la 940c ne fonctionne pas correctement : l'authentification nous renvoie l'erreur suivante :

```
error: PKCS11 function C_SignFinal failed: rv = CKR_FUNCTION_NOT_SUPPORTED (0x54)
```

L'IDPrime 940**c** est en fait 'simplement' un nouveau modèle de la série des IDPrime 940, nous avons donc modifié le code d'OpenSC pour ajouter l'ATR de 940c pour faire en sorte qu'il soit reconnu comme "Gemalto IDPrime 940" (et non comme "Gemalto IDPrime (generic)").

Après compilation en local :

- opensc (opensc-tool -n) reconnait bien la carte comme Gemalto IDPrime 940 (driver/type)
- la signature fonctionne correctement.

À la suite de cela, nous avons proposé cette modification à l'équipe d'OpenSC : https://github.com/OpenSC/OpenSC/pull/2941

Driver non détecté

Il arrive enfin que l'ATR ne corresponde à aucun ATR de référencé dans OpenSC : aucun driver n'est proposé / détecté par opensc et une authentification (via pkcs11-tool --login --test -p ******) nous renvoie alors l'erreur suivante :

```
Using slot 0 with a present token (0x0)
error: PKCS11 function C_GetTokenInfo failed: rv = CKR_TOKEN_NOT_RECOGNIZED (0xe1)
Aborting.
```

Si l'ATR n'est pas reconnu, il peut cependant malgré tout correspondre à un driver connu de OpenSC ... suivant les informations que vous aurez, si ça vous semble correspondre plus ou moins à un driver/modèle connu d'opensc, vous pouvez tenter de forcer l'usage d'un driver/type ou d'un autre ... comme dit en conclusion de ce paragrahe, il est évidemment (plus que) préférable de demande (dans la mesure du possible) un matériel compatible avec OpenSC.

OpenSC propose donc de forcer la reconnaissance d'ATR (non reconnu) via le fichier de configuration opensc.conf (/etc/opensc/opensc.conf sous linux ; /*Library*/OpenSC/etc/*opensc.conf sous MAC*)

Pour reprendre l'exemple ci-dessus, et si (et seulement si) l'ATR de notre "IDPrime 940c" n'avait pas du tout été reconnu, la configuration suivante aurait permis de la faire reconnaître (et fonctionner) en tant que Gemalto IDPrime
940

```
app default {
  card_atr 3b:7f:96:00:00:80:31:80:65:b0:85:05:00:39:12:0f:fe:82:90:00 {
    name = "Gemalto IDPrime 940C";
    driver = "idprime";
    type = "37004";
  }
}
```

le idprime est le nom du driver, on retrouve les noms des drivers dans les fichiers card-****.c dans la déclaration du sc_card_driver (ou en tapant la commande "opensc-tool -D")
Cf https://github.com/OpenSC/OpenSC/blob/master/src/libopensc/card-idprime.c

Pour 37004, cela correspond à l'enum SC_CARD_TYPE_IDPRIME_940 donné dans cards.h : https://github.com/OpenSC/OpenSC/blob/master/src/libopensc/cards.h

(SC_CARD_TYPE_IDPRIME_940 est défini implicitement à 37004 : il est à la 4ème place après SC_CARD_TYPE_IDPRIME_BASE qui est défini explicitement à 37000).

Conclusion

Pour résumer, OpenSC ne gère pas forcément tous les matériels mais il permet d'obtenir suffisamment d'informations pour permettre son identification.

Comme proposé plus haut pour les lecteurs USB, n'hésitez pas à recenser ci-dessous la liste des matériels et des ATR qui fonctionnent ou non avec OpenSC.

Dans la mesure du possible, il est préférable de demander un matériel pleinement opérationnel et supporté nativement par OpenSC.

Founisseur, Nom et ATR du certificat	OpenSC 0.23.0	OpenSC 0.24.0	OpenSC + Pilote propriétaire
Certinomis, Gemalto IDPrime 940 (3b:7f:96:00:00:80:31:80:65:b0:85:59:56:fb:12:0f:fe:82:90:00)	OK	OK	OK (libIDPrimePKCS11)
Certinomis, Gemalto IDPrime 940c (3b:7f:96:00:00:80:31:80:65:b0:85:05:00:39:12:0f:fe:82:90:00)	KO	OK	OK (libIDPrimePKCS11)
Certigna, Oberthur v7 (3b:db:96:00:80:b1:fe:45:1f:83:00:31:c1:64:08:40:22:30:0f:90:00:0a)	KO	KO	OK (libidop11)

Il est possible de vérifier le référencement de l'ATR via ce site :

Exemple pour Gemalto IDPrime 940

https://smartcard-atr.apdu.fr/parse?ATR=3b:7f:96:00:00:80:31:80:65:b0:85:59:56:fb:12:0f:fe:82:90:00

Exemple pour Gemalto IDPrime 940c

https://smartcard-atr.apdu.fr/parse?ATR=3b:7f:96:00:00:80:31:80:65:b0:85:05:00:39:12:0f:fe:82:90:00

L'ATR doit à minima y être présent pour pouvoir être pris en charge par OpenSC

Driver propriétaire

Si OpenSC embarque des drivers libres, il est aussi possible d'utiliser des drivers propriétaires avec les utilitaires proposés par OpenSC.

Ainsi la commande pkcs11-tool admet en option --module

Ainsi on peut tester l'authentification via le code pin sur la carte, avec la commande suivante (libIDPrimePKCS11.so est le driver propriétaire de certinomis ici) :

```
# pkcs11-tool --login --test --module /usr/lib/pkcs11/libIDPrimePKCS11.so
```

Pour lister les objets :

```
#  pkcs11-tool -O --module /usr/lib/pkcs11/libIDPrimePKCS11.so

Using slot 0 with a present token (0x0)
Certificate Object; type = X.509 cert
  label:      PRM_CAC_QSCD
  subject:    DN: C=FR, O=Certinomis/organizationIdentifier=NTRFR-433998903, CN=Certinomis - Prime CA G2
  serial:     8CC62F2DAD75D1B7C2CC22354920843D6311D849
  ID:         da534a908725de6d7f86845a4bccb662afce71cd
Certificate Object; type = X.509 cert
  label:      PRM_CAC_QSCD
  subject:    DN: C=FR, O=Certinomis, OU=0002 433998903, CN=Certinomis - Root CA
  serial:     01
  ID:         51b4b5cb135859dcfb983733e0de571d25b3afc0
Certificate Object; type = X.509 cert
  label:      PRM_CAC_QSCD
  subject:    DN: C=FR, ST=75, L=Paris, O=CONSORTIUM ESUP PORTAIL/organizationIdentifier=NTRFR-508669561,
OU=0002 508669561/serialNumber=204277ILI131, CN=CONSORTIUM ESUP PORTAIL - SIGNATURES
  serial:     24F7DE1DDFD9B0A9E4C5B8F77A8B338C
  ID:         3de88247375e5fb19cab3c31f1776b98f738f02f
Public Key Object; RSA 4096 bits
  label:      PRM_CAC_QSCD
  ID:         3de88247375e5fb19cab3c31f1776b98f738f02f
  Usage:      encrypt, verify, wrap
  Access:     local
```

Par rapport au driver libre embarqué dans opensc, on constate au passage que les id sont différents, là où la clef publique avait pour id 0001 on a maintenant 3de88247375e5fb19cab3c31f1776b98f738f02f

Ainsi, pour lire le certificat, esup-signature/esup-dss-client devra utiliser comme commande non pas (comme donné par défaut, en dur actuellement) :

```
pkcs11-tool -r --id 0001 --type cert
```

mais :

```
pkcs11-tool -r --id 3de88247375e5fb19cab3c31f1776b98f738f02f --type cert --module /usr/lib/pkcs11
/libIDPrimePKCS11.so
```

Logs debug

Si vous souhaitez récupérer des informations de debuggage, vous avez 2 possibilités :

Variable d'environnement OPENSC_DEBUG

en positionnant une variable d'environnement OPENSC_DEBUG à 9

```
export OPENSC_DEBUG=9
```

Fichier de configuration opensc.conf

Vous pouvez renvoyer des logs dans un fichier en configurant rapidement opensc.conf

```
app default {
        debug = 3;
        debug_file = opensc-debug.txt;

}
```

Le fichier opensc.conf est chargé à chaque lancement d'opensc, pkcs11-tool, ...

Logs disponibles

Suite à cela, un simple

```
opensc-tool -a
```

vous renverra des logs de manière très verbeuse, dont l'APDU permettant d'identifier la carte sim *FFFFFFFFFFFF* (apdu qui parlera aux personnes s'étant frottés aux technologies PC/SC NFC au travers peut-être d'esup-nfc-tag 😉)

Attention, si vous envoyez des commandes nécessitant le code PIN de votre carte, celui-ci sera très certainement écrit dans ces logs.

En faisant un test d'authentification (login) avec ces messages en debug, vous pourrez par exemple voir le nom du driver utilisé, c'est à dire le driver sélectionné fonction de l'ATR présenté par votre carte SIM :

```
... sc_connect_card: card info name:'Gemalto IDPrime 940' ...
```