
TUTORIAL

Installer un lecteur de cartes à puce USB sous Linux

Sebastien.Varrette@imag.fr

Version : 0.1 – Juin 2005

Résumé

Ce tutorial aborde l'installation d'un lecteur de cartes à puces sous Linux. Le lecteur utilisé pour réaliser ce tutorial est le lecteur USB Omnikey Cardman 3121 mais notre propos peut facilement s'étendre aux autres lecteurs, notamment ceux sur port série ou parallèle. L'objectif de cette installation est de gérer l'authentification des utilisateurs via une carte à puce. La réalisation de cet objectif (notamment la configuration des modules PAM etc... Voir aussi [4]) fera l'objet d'un document ultérieur. Ce tutorial se contente de détailler la phase d'installation et de configuration du lecteur, tout en proposant l'installation d'outils complémentaires permettant de manipuler le contenu de la carte à puce. Enfin, bien que la distribution utilisée soit une Debian, un effort tout particulier a été effectué pour réaliser au maximum les installations via les archives sources plutôt que par package, ceci dans le but de rester le plus générique possible.

Table des matières

1	Introduction : présentation des cartes à puce	3
2	Préliminaires : installations de logiciels sous Linux à partir des sources	3
2.1	Rappel : installation standard à partir de sources	3
2.2	Limites de la méthode standard	4
2.3	L'approche de l'outil <code>stow</code>	4
3	PC/SC lite	5
3.1	Présentation	5
3.2	Installation de PC/SC lite	6
3.2.1	Installation de la librairie <code>libusb</code>	6
3.2.2	Installation de <code>pcsc-lite</code>	6
3.3	Installation des drivers du lecteur	7
3.3.1	Préparation des répertoires d'accueil	7
3.3.2	Vérification préliminaires	7
3.3.3	Installation des drivers Omnikey	7
3.3.4	Installation de drivers génériques	8
3.3.5	Configuration du driver dans PC/SC Lite	8
3.3.6	Lancement du démon <code>pcscd</code>	9
3.4	Test de l'installation de PC/SC lite	10
3.5	Configuration démon pour un lancement au démarrage	10
3.6	Gestion des logs de <code>pcscd</code>	11
4	Mise en place d'un environnement de programmation Java	11
4.1	Installation du SDK (ex-JDK)	11
4.2	Creation des variables d'environnement <code>\$JAVA_HOME</code> et <code>\$CLASSPATH</code> - Mise à jour de la variable <code>\$PATH</code>	12
4.3	Installation de la font <code>symbol.ttf</code>	12
5	Installation du JavaCard Development Kit	13
5.1	Présentation	13
5.2	Installation du JCDK	13
5.3	Creation des variables d'environnement <code>\$JAVACARD_HOME</code> - Mise à jour de la variable <code>\$CLASSPATH</code> et <code>\$PATH</code>	14
6	Installation d'outils et d'applications compatibles PC/SC	14
6.1	Installation de <code>pcsc-perl</code>	14
6.2	Installation de <code>pcsc-tools</code>	15
6.3	Installation de OpenSC	15
6.4	Installation des frameworks MUSCLE	16
6.4.1	Prérequis : Installation de <code>libmusclecard</code>	16
6.4.2	Installation de <code>muscleTools</code>	16
6.4.3	Installation de <code>XCardII</code>	17
6.5	Installation de <code>MCardPlugin</code>	17

1 Introduction : présentation des cartes à puce

cf les documents suivants :

– http://www.jalix.org/ressources/miscellaneous/security/_IFT6271/Chap_14.pdf

– <http://people.cs.uchicago.edu/~dinoj/smartcard/javacardarch.html>

Dans la suite, les commandes shell nécessitant les droits utilisateurs (resp. root) seront préfixées par `user%` (resp. `root#`).

2 Préliminaires : installations de logiciels sous Linux à partir des sources

Afin de rester générique, les installations décrites dans ce tutorial sont faites à partir des sources, bien qu'il soit souvent plus pratique d'utiliser directement des paquets précompilés. Cependant, ce type d'installation garantit de bénéficier de la toute dernière version d'un logiciel.

2.1 Rappel : installation standard à partir de sources

Le fichier de sources se présente sous la forme d'une archive compressée `tar.gz` (au format général : `logiciel-version.tar.gz`). La méthode usuelle pour installer le logiciel est la suivante :

1. *Décompression de l'archive :*

```
user% tar xvzf logiciel-version.tar.gz
```

2. *Déplacement dans le répertoire des sources*

```
user% cd logiciel-version
```

3. *Configuration de la compilation.*

En général, un fichier `README` et/ou `INSTALL` existe et fournit la procédure à suivre. Il convient évidemment de les lire. Ensuite, le script `configure` permet de préparer automatiquement les fichiers nécessaires à la compilation. L'option `--prefix` permet de spécifier le répertoire cible de la compilation. Les fichiers générés seront alors installés dans le sous-répertoires de ce prefix (`$prefix/bin`, `$prefix/lib` ...). Par défaut, ce répertoire est en général `/usr/local`. Ainsi, pour lancer la configuration, on utilisera en général la commande :

```
user\% ./configure --prefix=/usr/local
```

(A noter que ce script admet de nombreuses autres option, détaillées dans le fichier `README` ou via la commande `./configure -help`).

Cette phase permet de vérifier les dépendances de librairie et de générer un fichier `Makefile` assurant une compilation spécialisée à votre machine. Toute erreur soulevée par cette commande doit être corrigées avant de passer à l'étape suivante.

4. *Compilation du logiciel.*

```
user\% make
```

5. *Installation du logiciel.* L'outil `make` permet également d'automatiser la phase d'installation. De même que précédemment, aucune option n'est normalement nécessaire.

```
root# make install
```

Remarque : les droits root sont en général requis afin de disposer des droits d'écriture dans le répertoire cible.

Cette suite de commandes installe le logiciel dans `/usr/local` (répertoire cible). Les exécutables sont stockés dans le sous-répertoire `bin` (éventuellement `sbin`) du répertoire cible, les bibliothèques dans le sous-répertoire `lib` du répertoire cible, etc...

2.2 Limites de la méthode standard

Cette méthode fonctionne parfaitement, mais pose des problèmes de mise à jour ou de suppression d'un logiciel. Il est en effet impossible de situer tous les fichiers installés lors du processus de compilation/installation. Pour désinstaller le logiciel, on peut utiliser la commande

```
root# make uninstall
```

depuis le répertoire des sources configurés. Cependant, de façon général, ce répertoire est supprimé après l'installation si bien qu'on se retrouve très vite avec des centaines de fichiers, d'origines diverses, dans le répertoire `/usr/local`.

2.3 L'approche de l'outil stow

Bob Glickstein a proposé de résoudre le problème par l'utilisation du logiciel `stow` [2].

`stow` est un petit script (500 lignes) écrit en langage PERL, sous licence GPL. Le principe est d'effectuer la phase d'installation avec un répertoire cible différent de `/usr/local`, et spécifique pour chaque logiciel.

On crée donc un répertoire `/usr/local/stow` dans lequel on placera un répertoire par logiciel installé. Ainsi, le logiciel `toto` version 1.0.14 sera configuré pour s'installer dans le répertoire `/usr/local/stow/toto-1.0.14/` via la commande

```
user% ./configure --prefix=/usr/local/stow/toto-1.0.14/
```

A l'issue de la phase d'installation (par `make install`), on se retrouve avec les exécutables générés stockés dans le sous-répertoire `bin` de `/usr/local/stow/toto-1.0.14/`, les bibliothèques dans le sous-répertoire `lib` etc...

C'est là que `stow` intervient. La commande suivante est à lancer dans le répertoire `/usr/local/stow` :

```
root# stow toto-1.0.14
```

Le script va alors mettre en place des liens symboliques pour que chaque fichier de `/usr/local/stow/toto-1.0.14/` soit également visible dans `/usr/local`, à la même place.

Ainsi, les fichiers et répertoires de `/usr/local/stow/toto-1.0.14/lib` sont également visibles dans `/usr/local/lib`. Et de même pour les répertoires `bin/`, `man/`, `share/`, `etc/`, `sbin/`, `doc/` ...

On obtient donc un logiciel configuré pour fonctionner dans `/usr/local` (phase 4), installé dans un répertoire spécifique (phase 6). Et le tout fonctionne de façon transparente, grâce aux liens symboliques générés par `stow`.

Pour désinstaller la version 1.0.14 du logiciel `toto` (au profit de l'installation d'une nouvelle version par exemple), il suffit d'exécuter la commande suivante dans le répertoire `/usr/local/stow` :

```
root# stow -D toto-1.0.14
```

On se retrouve alors avec un `/usr/local` "propre", dans lequel ne subsiste plus aucun lien pointant vers le répertoire `/usr/local/stow/toto-1.0.14/`.

On vient donc d'effectuer une désinstallation de `toto`, et ceci en douceur.

Cette technique est particulièrement intéressante pour tester une nouvelle version d'un logiciel tout en gardant en réserve une version stable. Rien ne vous empêche d'installer les versions 1.0.14, 1.1.0 et 1.1.1 de `toto` dans divers répertoires, et d'utiliser `stow` pour choisir la version visible depuis `/usr/local`.

`stow` se révèle vite indispensable lorsqu'on a à gérer un nombre important de logiciels dans `/usr/local`. C'est un outil petit, stable et qui a fait ses preuves. Une lecture de la documentation fournie peut se révéler un plus pour utiliser certaines options de cet utilitaire. Dans toutes la suite, les installations seront faites en utilisant l'approche de cet outil.

3 PC/SC lite

3.1 Présentation

La gestion standard de la communication entre un ordinateur et une carte à puce a été définie par le groupe de travail PC/SC ¹. PC/SC est donc une API qui fournit aux développeurs un ensemble de primitives standard permettant de gérer les lecteurs de cartes à puces et de communiquer avec le lecteur et les cartes à puces qu'il contient. Ainsi, les opérations standards effectuées sur les cartes à puces sont gérées par cette API de façon homogène.

Chaque fournisseur de lecteur doit simplement fournir un driver spécifique à son lecteur implémentant chaque fonctions de cette API.

¹<http://www.pcscworkgroup.com>

3.2 Installation de PC/SC lite

3.2.1 Installation de la librairie libusb

PC/SC-lite utilise la librairie libusb pour accéder à un lecteur USB. Pour l'installer (sous Debian) :

```
apt-get install libusb-dev libusb-0.1-4
```

La librairie est installée dans `/usr/lib/libusb.so`

Sinon, il suffit de récupérer les sources sur le site <http://libusb.sourceforge.net/> et procéder à l'installation classique.

3.2.2 Installation de pcsc-lite

Maintenant, on peut récupérer les sources PCSC lite sur le site <http://alioth.debian.org/projects/pcsc-lite/> Au moment de lecture de ce tutorial, il s'agit de l'archive `pcsc-lite-1.2.9-beta7.tar.gz`.

Vous pouvez éventuellement vérifier la signature de l'archive récupérée à partir du fichier `asc` associé via la commande :

```
user% gpg --verify pcsc-lite-1.2.9-beta7.tar.gz.asc \  
                pcsc-lite-1.2.9-beta7.tar.gz
```

Mais il faut disposer de la clé publique du site. (perso, je ne l'ai pas trouvée)

On décompresse l'archive et on procède à l'installation (éventuellement, modifier le répertoire en paramètre de l'option `--enable-libusb` pour correspondre à votre installation) :

```
user% tar xvzf pcsc-lite-1.2.9-beta7.tar.gz  
user% cd pcsc-lite-1.2.9-beta7  
user% ./configure --prefix=/usr/local/stow/pcsc-lite-1.2.9-beta7 \  
                --enable-libusb=/usr --enable-daemon --enable-debug \  
                --enable-threadsafes --enable-runpid=/var/run/pcscd.pid  
user% make  
root# make install  
root# cd /usr/local/stow  
root# stow pcsc-lite-1.2.9-beta7/
```

Il faut ensuite créer le device `pcsc` :

```
root# mkdir /dev/pcsc
```

Il faut enfin ajouter le répertoire `/usr/local/lib` dans le fichier de configuration `/etc/ld.so.conf` (qui gère l'édition de lien dynamique) s'il n'y figure pas déjà. Il faut ensuite mettre valider l'ajout des bibliothèques par la commande

```
root# ldconfig
```

3.3 Installation des drivers du lecteur

3.3.1 Préparation des répertoires d'accueil

On placera les drivers dans le répertoire `/usr/lib/pcsc/drivers`. Il faut donc créer ce répertoire :

```
root# mkdir -p /usr/lib/pcsc/drivers
```

PC/SC Lite détecte normalement automatiquement les lecteurs sur port USB en parcourant le répertoire `pcsc/drivers` du répertoire d'installation. Il convient donc de faire pointer `/usr/local/stow/pcsc-lite-1.2.9-beta7/pcsc/` sur `/usr/lib/pcsc/` :

```
root# ln -sf /usr/lib/pcsc /usr/local/stow/pcsc-lite-1.2.9-beta7/pcsc
```

3.3.2 Vérification préliminaires

Comme le lecteur fonctionne sur le port USB, les modules `usbcore` et `usb-uhci` doivent être chargés. On le vérifie par les commandes :

```
root# lsmod | grep usbcore
root# lsmod | grep usb-uhci
```

Si ce n'est pas le cas, les insérer avec la commande `modprobe <module>`. Le port USB doit être monté dans le système de fichier

3.3.3 Installation des drivers Omnikey

Les drivers linux du lecteur Omnikey Cardman 3121 peuvent être récupérés sur le site <http://www.omnikey.com/index.php?id=5> sous la référence "CardMan Desktop USB 3121 (CCID)". Au moment où ce tutorial est écrit, l'archive récupérée ainsi est `ifdokccid_lnx-2.1.0.tar.gz`

L'installation s'effectue à l'aide des commandes suivantes :

```
user% tar xvzf ifdokccid_lnx-2.1.0.tar.gz
user% cd ifdokccid_lnx-2.1.0
root# ./install -d /usr/lib/pcsc/drivers
```

La commande `lsusb` liste les périphériques attachés aux ports usb. Par exemple, pour le lecteur omnikey :

```
user% lsusb | grep -i omnikey
Bus 001 Device 002: ID 076b:3021 OmniKey AG CardMan 3121
```

Sur cet exemple,

- 076b est le "Vendor ID";
- 3021 est le "Product ID"

Il convient alors de mettre à jour le fichier

`/usr/lib/pcsc/drivers/ifdokccid_lnx-2.1.0.bundle/Contents/Info.plist` avec ces deux valeurs (entre les balises `<key>ifdVendorID</key>` et

<key>ifdProductID</key>) si celles-ci n'y figurent pas.

Remarque :

Dans l'état actuel des choses, le driver du lecteur correspond au fichier `ifdokccid_lnx-2.1.0.bundle/Contents/Linux/ifdokccid.so` dans le répertoire `/usr/lib/pcsc/drivers/`. Vis-à-vis de PC/SC et par le biais des liens symboliques installés, le driver se retrouve donc installé dans le répertoire `/usr/local/stow/pcsc-lite-1.2.9-beta7/pcsc/drivers`.

Dans mon cas, un petit bug dans la gestion du chemin au driver qui limite la taille de ce chemin à 100 caractères pose problème. Pour que tout fonctionne correctement dans la suite, il conviendra de renommer

```
/usr/lib/pcsc/driver/ifdokccid_lnx-2.1.0.bundle
en
/usr/lib/pcsc/driver/ifdokccid.bundle
```

3.3.4 Installation de drivers génériques

Pour ceux qui n'ont pas directement un driver pour leur lecteurs de cartes, le site <http://pcsc-lite.alieth.debian.org/ccid.html> fournit un driver générique réalisé par la communauté MUSCLE [1] et plus particulièrement Ludovic Rousseau.

Une fois l'archive récupérée (`ccid-0.9.3.tar.gz` au moment où ce tutorial est rédigé), le fichier `README` détaille les lecteurs supportés. Bien que non testée, l'installation se déroule de la façon suivante :

```
user% tar xvzf ccid-0.9.3.tar.gz
user% cd ccid-0.9.3
user% ./configure --enable-libusb=/usr
user% make
root# make install
```

Normalement, le script de configuration détecte automatiquement le répertoire d'installation de `pcsc-lite` (dans ce tutorial, `/usr/local/stow/pcsc-lite-1.2.9-beta7`) et y place les drivers dans le répertoire `$prefix/pcsc/drivers/`.

3.3.5 Configuration du driver dans PC/SC Lite

Cas des lecteurs sur port USB Dans le cas des lecteurs USB, la détection des lecteurs se fait automatiquement dans la mesure où le répertoire `$prefix/pcsc/drivers` existe bien (c'est pourquoi un lien a été fait de `$prefix/pcsc` sur `/usr/lib/pcsc`) et possède un format standard (c'est normalement le cas à ce point du tutorial puisqu'on a utilisé un script constructeur).

Cas des autres lecteurs Pour les autres types de lecteurs (notamment ceux sur port série²), la configuration s'effectue via le fichier `/etc/reader.conf`.

²On rappelle qu'avec des lecteurs USB, il **NE FAUT PAS** passer par là (la détection est automatique)

il ne s'agit pas d'éditer directement ce fichier. On utilise le répertoire `/etc/reader.conf.d/` dans lequel on crée un fichier de conf par lecteur. Ainsi pour le lecteur `toto_Cardman` serait configuré via le fichier `/etc/reader.conf.d/toto_Cardman.conf` contenant les lignes suivantes :

```
#===== Configuration Toto Cardman =====
FRIENDLYNAME      "Toto CardMan"
DEVICENAME        /dev/ttys0
LIBPATH           /path/to/pcsc/drivers/toto_driver.so
CHANNELID         0x000001
```

Exécuter `man reader.conf` pour plus d'informations sur la signification de chaque directive.

Ensuite, la commande

```
root# update-reader.conf
```

parse le répertoire `/etc/reader.conf.d/` pour créer le fichier `/etc/reader.conf` (ainsi, éventuellement, plusieurs lecteurs peuvent être configurés).

Remarque : pour le détail de la configuration d'autres lecteurs, se référer au tutorial de Damien Sauveron [7].

3.3.6 Lancement du démon pcscd

A ce stade, il est possible de lancer le démon `pcscd` de `pcsc-lite`. On préférera dans un premier temps une exécution en mode debug au premier plan (options `--apdu --foreground`), afin de cerner les éventuels problèmes de configuration. On obtient normalement quelque chose de la forme :

```
root# pcscd --apdu --foreground
pcscdaemon.c:242:main() pcscd set to foreground with debug send to stderr
pcscdaemon.c:446:main() pcsc-lite 1.2.9-beta7 daemon ready.
hotplug_libusb.c:371:HPAddHotPluggable() Adding USB device: 001:002
readerfactory.c:1066:RFInitializeReader() Attempting startup of OMNIKEY \
  CardMan 3x21 00 00.
readerfactory.c:940:RFBindFunctions() Loading IFD Handler 3.0
OMNIKEY CardMan CCID IA32 v2.1.0 support.linux@omnikey.com
Card ATR: 3B 6E 00 FF 53 46 53 45 2D 43 31 36 34 2D 56 05 02 00
```

Sinon, les messages d'erreur devraient être suffisamment explicites pour comprendre et résoudre le problème. On pourra éventuellement utiliser la mailing list de muscle (<http://www.linuxnet.com/list.html>).

Remarque : Lors de mon problème de chemin trop long vers le driver, j'avais obtenu le retour suivant :

```
root # pcscd --apdu --foreground
pcscdaemon.c:242:main() pcscd set to foreground with debug send to stderr
pcscdaemon.c:446:main() pcsc-lite 1.2.9-beta7 daemon ready.
hotplug_libusb.c:371:HPAddHotPluggable() Adding USB device: 001:002
readerfactory.c:1066:RFInitializeReader() Attempting startup of OMNIKEY \
```

```

CardMan 3x21 00 00.
dyn_unix.c:32:DYN_LoadLibrary() /usr/local/stow/pcsc-lite-1.2.9-beta7/pcsc/
drivers/ifdokccid_lnx-2.1.0.bundle/Contents/Linux/ifdokc:
/usr/local/stow/pcsc-lite-1.2.9-beta7/pcsc/drivers/ifdokccid_lnx-2.1.0.bundle/
Contents/Linux/ifdokc: cannot open shared object file: No such file or directory
readerfactory.c:219:RFAddReader() OMNIKEY CardMan 3x21 init failed.

```

3.4 Test de l'installation de PC/SC lite

Dans l'archive des source de l'installation de pcsc-lite (pcsc-lite-1.2.9-beta7.tar.gz) se trouve un répertoire src/ dans lequel figure, après compilation, l'exécutable "testpcsc".

Une fois le démon lancé, exécuter cette commande. Vous devez obtenir :

```

user% ./testpcsc
MUSCLE PC/SC Lite unitary test Program

```

```

THIS PROGRAM IS NOT DESIGNED AS A TESTING TOOL FOR END USERS!
Do NOT use it unless you really know what you do.

```

```

Testing SCardEstablishContext      : Command successful.
Testing SCardGetStatusChange
Please insert a working reader     : Command successful.
Testing SCardListReaderGroups     : Command successful.
Group 01: SCard$DefaultReaders
Testing SCardListReaders          : Command successful.
Reader 01: OMNIKEY CardMan 3x21 00 00
Waiting for card insertion        : Command successful.
Testing SCardConnect              : Command successful.
Testing SCardControl              : Transaction failed. (don't panic)
Testing SCardGetAttrib            : Command successful.
ATR length: 18
Testing SCardGetAttrib            : Command successful.
3B 6E 00 FF 53 46 53 45 2D 43 31 36 34 2D 56 05 02 00
Testing SCardSetAttrib            : Transaction failed. (don't panic)
Testing SCardStatus               : Command successful.
Current Reader Name               : OMNIKEY CardMan 3x21 00 00
Current Reader State               : 0x0034
Current Reader Protocol            : T=0
Current Reader ATR Size            : 18 bytes
Current Reader ATR Value           : 3B 6E 00 FF 53 46 53 45 2D 43 31 36 34 2D 56 05 02 00
Press enter:
Testing SCardReconnect            : Command successful.
Testing SCardDisconnect           : Command successful.
Testing SCardReleaseContext       : Command successful.

```

PC/SC Test Completed Successfully !

3.5 Configuration démon pour un lancement au démarrage

En général sous Linux, le répertoire /etc/rcx.d/ contient une liste ordonnée des services à lancer au démarrage. Dans cet exemple, x correspond au résultat

de la commande `runlevel` (2 par défaut sous Debian, 5 sous Mandrake/Redhat etc...). Chaque fichier de ce répertoire est en fait un lien sur ceux du répertoire `/etc/init.d` qui contient classiquement l'ensemble des services pouvant être lancés au démarrage sous forme de scripts admettant les arguments `start`, `stop`, `restart` et éventuellement `status` pour respectivement lancer, stopper, relancer et voir l'état du service.

Vous pouvez évidemment faire un lien de `/etc/init.d/pcscd` sur `/usr/local/sbin/pcscd` mais pour une gestion plus standard et dans le respect la philosophie Debian, l'annexe A fournit un script de démarrage pour le démon `pcscd`.

Pour les autres distributions, vous pourrez vous inspirer des scripts suivants :

- <http://www.strongsec.com/smartcards/howto/html/SmartCard-Login-HOWTO-7.html#pcsc-example>
- l'archive des sources de `pcsc-lite` fournit également un script de démarrage dans le répertoire `etc/`. En effet, après compilation, ce répertoire contient le script `pcscd.init` qui devrait convenir.

Enfin, pour mettre en place le lancement du démon `pcscd` à chaque démarrage, il suffit d'utiliser la commande :

```
ln -sf /etc/init.d/pcscd /etc/rcx.d/S99pcscd
```

ou `x` est toujours le résultat de la commande `runlevel`.

3.6 Gestion des logs de `pcscd`

Par défaut, les logs du démon figurent dans le fichier `/var/log/messages`.

TODO : ajouter gestion des logs dans un fichier à part `/var/log/pcscd.log` (option `-d` de `pcscd`?) + `/etc/default/pcscd`

4 Mise en place d'un environnement de programmation Java

A ce stade, le lecteur est installé et configuré. Il reste à l'utiliser. Dans le cadre de ce tutorial, on s'intéresse principalement aux javacard. Il convient donc de disposer d'un environnement Java.

4.1 Installation du SDK (ex-JDK)

Pour cela, il suffit de se rendre sur le site de SUN³ pour récupérer le dernier SDK (Software Development Kit). Au moment où ce tutorial est rédigé, c'est le J2SE v 1.4.2_08 SDK qui a été récupéré sur le site <http://java.sun.com/j2se/1.4.2/download.html>.

³<http://java.sun.com/>

On obtient ainsi un script d'installation (`j2sdk-1_4_2_08-linux-i586.bin` dans notre cas). Les instructions d'installation sont disponibles en ligne sur le site <http://java.sun.com/j2se/1.4.2/install-linux.html>
Dans notre cas, elles se résument à la séquence de commandes suivantes :

```
chmod +x j2sdk-1_4_2_08-linux-i586.bin
cd /usr/local/stow
/path/to/SDK/binary/j2sdk-1_4_2_08-linux-i586.bin
cd ..
ln -sf stow/j2sdk1.4.2_08 java
```

4.2 Creation des variables d'environnement \$JAVA_HOME et \$CLASSPATH - Mise à jour de la variable \$PATH

Ces variables sont utilisées dans de nombreux programmes.
Pour ceux qui utilisent le shell `bash` (voir le résultat de la commande `echo $SHELL`), il s'agit d'ajouter les lignes suivantes dans le fichier de configuration `.bashrc` (présent dans les répertoires `/root/`, `/etc/skel/` et `~/`) :

```
# Gestion de Java
export JAVA_HOME=/usr/local/java
export CLASSPATH=$JAVA_HOME/lib
PATH=$PATH:$JAVA_HOME/bin
```

Pour ressourcer le fichier de configuration, on utilise la commande `source` :

```
source ~/.bashrc
```

Pour ceux qui utilisent le shell `tcsh`, la syntaxe est légèrement différente. Le fichier de configuration est alors `.tcshrc` et les lignes à ajouter sont :

```
# Java variable
setenv JAVA_HOME /usr/local/java
set CLASSPATH=($JAVA_HOME/lib)
```

```
#par exemple
set path=(/bin /usr/bin /usr/local/bin /usr/local/bin/staroffice7 \
  /usr/X11R6/bin /usr/games $HOME/bin $JAVA_HOME/bin .)
```

4.3 Installation de la font `symbol.ttf`

Cette font d'affichage sera nécessaire par la suite. Le fichier correspondant peut être récupéré sur

<http://www.webpagepublicity.com/free-fonts/s/Symbol.ttf>

Ensuite, on copie ce fichier au bon endroit :

```
root# cp Symbol.ttf $JAVA_HOME/jre/lib/fonts/
root# chmod +w $JAVA_HOME/jre/lib/fonts/fonts.dir
```

Il faut ensuite éditer le fichier de configuration des fonts Java (le fichier `$JAVA_HOME/jre/lib/fonts/fonts.dir` en ajoutant la ligne :

`symbol.ttf -urw-symbol-medium-r-normal--0-0-0-0-p-0-adobe-fontspecific`

et en incrémentant de 1 le nombre de fonts du fichier (il s'agit du nombre présent au début du fichier - dans mon cas, ce nombre est devenu 73).

On dispose maintenant d'un environnement de programmation en Java. On va pouvoir procéder à l'installation le "JavaCard Development Kit"

5 Installation du JavaCard Development Kit

5.1 Présentation

Java Card Development Kit est un environnement de développement pour les applications pour cartes.

Sun Microsystems fournit sur son site⁴ les spécifications de la plateforme Java Card et le "Java Card Development Kit" - JCDK qui inclut une implémentation de référence pour cette spécification. Au moment où ce tutorial est écrit, la version 2.2.1 de la spécification inclut la spécification de trois éléments :

1. La "Java Card Virtual Machine" qui supporte un sous-ensemble du langage Java (adapté à l'environnement matériel limité des cartes à puces) ainsi que le format des fichiers utilisés pour installer des applets et les bibliothèques utilisées dans les cartes et les devices qui accueillent une technologie Java Card.
2. Le "Java Card Runtime Environment" traduit le comportement sur l'environnement d'exécution quelque soit l'implémentation de la JCVM, des classes de l'API Java Card etc...
3. une API spécifique aux plateformes Java Card qui complète le JCRE et reste compatible aux standards formels internationaux et industriels.

JCDK est un ensemble d'outils permettant de développer des applets basée sur l'API Java Card, et inclut notamment :

- C-JCRE, une implémentation de référence en C du JCRE qui fournit également un interpréteur pour la JCVM.
- des outils de développement, comme le "Java Card Converter" et le "Java Card Verifier" etc...

Complément : voir http://jfod.cnam.fr/tp_cdi/routaboul/javacard.pdf

5.2 Installation du JCDK

Il suffit de récupérer la dernière version du JCDK sur le site

<http://java.sun.com/products/javacard/downloads/>

Au moment où ce tutorial est écrit, il s'agit du "Java Card 2.2.1 Development Kit", récupérer sous forme d'une archive compressée `java_card_kit-2_2_1-linux-dom.zip`.

La procédure d'installation est la suivante :

⁴<http://java.sun.fr/products/javacard/>

```

root# cd /usr/local/stow
root# unzip /path/to/java_card_kit-2_2_1-linux-dom.zip
root# cd ..
root# ln -sf stow/java_card_kit-2_2_1-linux-dom.zip javacard

```

5.3 Creation des variables d'environnement \$JAVACARD_HOME - Mise à jour de la variable \$CLASSPATH et \$PATH

Comme au §4.2, il s'agit d'ajouter et de mettre à jour certaines variables d'environnement couramment utilisées dans votre shell.

– Pour ceux qui utilisent bash :

```

# Java
export JAVA_HOME=/usr/local/java
export JAVACARD_HOME=/usr/local/javacard
export CLASSPATH=$JAVA_HOME/lib:$JAVACARD_HOME/lib
PATH=$PATH:$JAVA_HOME/bin:$JAVACARD_HOME/bin

```

– Pour ceux qui utilisent tcsh :

```

# Java variable
setenv JAVA_HOME /usr/local/java
setenv JAVACARD_HOME /usr/local/javacard
set CLASSPATH=($JAVA_HOME/lib $JAVACARD_HOME/lib)

```

#par exemple

```

set path=(/bin /usr/bin /usr/local/bin /usr/local/bin/staroffice7 \
    /usr/X11R6/bin /usr/games $HOME/bin $JAVA_HOME/bin $JAVACARD_HOME/bin.)

```

Maintenant, le lecteur est installé et un environnement de développement des Java Cards est configuré. Les sections qui suivent détaillent l'installation d'outils complémentaires.

6 Installation d'outils et d'applications compatibles PC/SC

6.1 Installation de pcsc-perl

Ce wrapper Perl pour PC/SC [5] permet d'écrire rapidement et facilement des scripts Perl⁵ accédant les cartes insérées dans le lecteur.

La bibliothèque fournie permet donc de communiquer avec la carte à travers un script Perl en utilisant PC/SC.

La dernière version de ce wrapper (1.4.2 au moment où ce tutorial est rédigé) est disponible sur le site

<http://ludovic.rousseau.free.fr/software/pcsc-perl/>

La procédure d'installation est la suivante :

```

user% tar xvzf pcsc-perl-1.4.2.tar.gz
user% cd pcsc-perl-1.4.2
user% perl Makefile.PL

```

⁵<http://www.perl.org>

```
user% make
user% make test
root# make install
```

Des wrappers dans d'autres langages de programmation sont disponibles sur la page <http://www.linuxnet.com/middle.html>

6.2 Installation de pcsc-tools

[6] est un ensemble d'outils pour manipuler les cartes à puce à travers PC/SC. Ces outils permettent notamment de tester un driver PC/SC, une carte ou un lecteur, d'envoyer des commandes dans un environnement en ligne de commande ou graphique etc...

Les outils fournis par `pcsc-tools` sont détaillés dans la liste suivante :

- `pcsc_scan` qui scanne chaque lecteur PC/SC connecté et fournit des informations sur les cartes insérées. Cet exécutable utilise un script Perl - `ATR_analysis` - qui permet de récupérer et d'interpréter l'ATR⁶ de la carte insérée.
- `scriptor` (et sa version graphique par interface GTK `gscriptor`) est un script perl permettant s'envoyer des commandes à la carte insérée.

Une fois la dernière version du `pcsc-tools` (1.4.1 au moment où ce tutorial est rédigé) récupérée sur le site

<http://ludovic.rousseau.free.fr/software/pcsc-tools/>,
la procédure d'installation est la suivante :

```
user% tar xvzf pcsc-tools-1.4.1.tar.gz
user% cd pcsc-tools-1.4.1
user% make
root# make install
```

Attention, le `makefile` est un peu mal foutu - il aurait été appréciable de disposer d'un script `configure`.

Pour pouvoir utiliser la version graphique de `scriptor`, il est nécessaire d'installer des packages perl complémentaires :

```
apt-get install libglib-perl libgtk2-perl
```

6.3 Installation de OpenSC

Un 'card format' est une description détaillée de la façon dont sont représentés sur une carte des objets cryptographiques comme une clé ou un certificat.

PKCS#15 [3] est un standard pour le card format supportant tous les objets définis dans la norme PKCS#11. OpenSC⁷ est une bibliothèque (`libopensc`) permettant notamment de gérer les cartes compatibles avec le standard PKCS#15. L'utilisateur pourra ainsi initialiser, personnaliser et manipuler des cartes PKCS#15, signer et déchiffrer des données à partir de clés privées stockées sur la carte. Les

⁶Lorsqu'une carte est mise sous tension, elle émet une *réponse au démarrage* (Answer-To-Reset)

⁷<http://www.opensc.org/>

cartes supportées inclut Gles cartes Gemplus GPK, Schlumberger Cryptoflex, Finnish FINEID, Swedish eID, MioCOS et TCOS.

Elle pourra s'avérer utile par la suite.

La procédure d'installation de OpenSC (version 0.9.6 au moment où ce tutorial est rédigé) est la suivante :

```
user% tar xvzf opensc-0.9.6.tar.gz
user% cd opensc-0.9.6
user% ./configure --prefix=/usr/local/stow/opensc-0.9.6 \
                --with-pcsclite=/usr/local/stow/pcsclite-1.2.9-beta7
user% make
root# make install
root# cd /usr/local/stow/
root# stow opensc-0.9.6
```

6.4 Installation des frameworks MUSCLE

De nombreuses application développées dans le cadre de MUSCLE [1] sont disponibles sur le site <http://www.linuxnet.com/apps>.

Nous détaillons maintenant l'installation de certains outils qui seront utiles dans la suite, en particulier XcardII qui fournit une interface graphique à l'utilisateur et `muscletools` qui fournit un frontal en ligne de commande pour gérer votre carte à puce.

6.4.1 Prérequis : Installation de libmusclecard

Cette librairie est requise pour la suite. La dernière version est disponible sur le site <https://alioth.debian.org/projects/pcsclite/> (version 1.2.9-beta7 au moment où ce tutorial est rédigé). La procédure d'installation est la suivante :

```
user% tar xvzf libmusclecard-1.2.9-beta7.tar.gz
user% cd libmusclecard-1.2.9-beta7
user% bash
user% PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
user% ./configure --prefix=/usr/local/stow/musclecard-1.2.9-beta7
user% make
root# make install
root# cd /usr/local/stow
root# stow musclecard-1.2.9-beta7
root# ldconfig
```

6.4.2 Installation de muscleTools

`muscleTools` permet de gérer et de personnaliser des cartes supportées par MUSCLE ou tout objet cryptographique via cet outil en ligne de commande. Il permet notamment de lire et écrire des fichiers sur la carte, gérer les PINs, les objets, les clés, générer des clés sur la carte etc...

Les sources peuvent être récupérées sur le site
<https://alioth.debian.org/projects/muscleapps/> Pour décompresser les sources :

```
user% tar xvzf muscleTools-0.9.2.tar.gz
user% cd muscleTools-0.9.2
```

Ensuite, le Makefile doit être modifier pour assurer une compilation correcte. Modifier les lignes définissant les variables CFLAGS et LIBS de la façon suivante :

```
CFLAGS = -g -Wall -I/usr/local/include/PCSC
LIBS = -I/usr/local/lib -lmusclicard -lpthread
```

Ensuite, l'installation est effectuée par la commande :

```
root# make
```

6.4.3 Installation de XCardII

xcardii : Graphical program to manage a MuscleCard smartcard
XCardII, a graphical program (using QT) to manage a MuscleCard (either a Java Card containing the MuscleCard applet or a Cryptoflex personalised as a MuscleCard).

Drag and drop files to the smartcard, manage PINs, objects, keys, etc. Import, export, and generate onboard keys. . Homepage : <https://alioth.debian.org/projects/muscleapps/>

TODO @ compléter. PAS TESTE! necessite moc, @ suivre

6.5 Installation de MCardPlugin

The plugin are used by the muscle framework to manage the different cards. For example the MCardPlugin is used to handle a Java Card using the MCardApplet and the CFlexPlugin is used for the Cryptoflex card.

Download the last version of MCardPlugin.

```
user% export CPPFLAGS=-I/usr/local/include/PCSC
user% export LDFLAGS=-I/usr/local/include/PCSC
user% ./configure
user% make
root# ./installBundle
```

TODO @ compléter. PAS TESTE!

A Script de démarrage du démon pcsd sous Debian

A adapté éventuellement (notamment pour la variable \$DAEMON)

TODO : prévoir ajout et gestion d'un fichier /etc/default/pcsd

```
#!/bin/sh
#
# /etc/init.d/pcsd
# Start/Stop/Restart PCSC Lite resource manager daemon
#
# Carlos Prados Bocos <cprados@debian.org>
# modifications by Ludovic Rousseau <ludovic.rousseau@free.fr> and
# Sebastien Varrette <Sebastien.Varrette@imag.fr>

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/sbin/pcsd
NAME=pcsd
DESC="PCSC Lite resource manager"

test -f $DAEMON || exit 0

if [ 'id -u' != "0" ] && [ "$1" = "start" -o "$1" = "stop" ] ; then
    echo "You must be root to start, stop or restart pcsd."
    exit 1
fi

set -e

case "$1" in
    start)
        echo -n "Starting $DESC: "
        start-stop-daemon --start --quiet --pidfile /var/run/$NAME.pid \
            --exec $DAEMON
        echo "$NAME."
        ;;

    stop)
        echo -n "Stopping $DESC: "
        start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/$NAME.pid \
            --exec $DAEMON
        sleep 2
        echo "$NAME."
        ;;

    restart|force-reload)
        echo -n "Restarting $DESC: "
        start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/$NAME.pid \
            --exec $DAEMON
        sleep 2
        start-stop-daemon --start --quiet --pidfile /var/run/$NAME.pid \
            --exec $DAEMON
        echo "$NAME."
        ;;
)
```

```

restart-if-running)
    echo -n "Stopping $DESC if it is running: "
    if start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.pid \
        --exec ${DAEMON} ;
    then
        echo "${NAME}."
        echo -n "Restarting $DESC: "
        sleep 2
        start-stop-daemon --start --quiet --pidfile /var/run/$NAME.pid \
            --exec ${DAEMON}
        echo "${NAME}."
    else
        echo "(not running)."
```

```

    fi
    ;;

status)
    if [ -e /var/run/$NAME.pid ] ; then
        if ps -p $(cat /var/run/$NAME.pid) &> /dev/null ; then
            echo "running"
            exit 0
        else
            echo "not running and /var/run/$NAME.pid exists"
            exit 1
        fi
    else
        echo "not running"
        exit 3
    fi
    ;;

*)
    N=/etc/init.d/$NAME
    echo "Usage: $N {start|stop|restart|force-reload|restart-if-running|status}" >&2
    exit 1
    ;;

esac

exit 0

```

Références

- [1] David Corcoran & al. Muscle - Movement for the Use of Smart Cards in a Linux Environment. <http://www.linuxnet.com/>.
- [2] Bob Glickstein and Guillaume Morin. Gnu stow, 1996. <http://www.gnu.org/software/stow/stow.html>.
- [3] RSA Laboratories. The Public-Key Cryptography Standards - PKCS #15 v1.1 : Cryptographic token information syntax. Technical report, RSA Laboratories, June 2000. <http://www.rsasecurity.com/rsalabs/node.asp?id=2141>.
- [4] Christophe Rippert. Analyse du rôle des cartes à puce dans un environnement réparti. Master's thesis, Université Joseph Fourier - Grenoble 1, June 2000. <http://www-verimag.imag.fr/~rippert/publis/rapport-dea.ps.gz>.
- [5] Ludovic Rousseau. pcsc-perl, 2004. <http://ludovic.rousseau.free.fr/software/pcsc-perl/>.
- [6] Ludovic Rousseau. pcsc-tools, 2004. <http://ludovic.rousseau.free.fr/software/pcsc-tools/>.
- [7] Damien Sauveron. Installation of Smart Card Software. Technical report, LaBRI, jan 2004. <http://damien.sauveron.free.fr/SmartCard-Howto.html>.