Valérian REITHINGER

Créer et utiliser une partition chiffrée avec LUKS sous Linux

Page mise à jour le 18 avril 2018

Sommaire [masquer]

Quelques mots sur *LUKS*

LUKS, pour *Linux Unified Key Setup*, est le standard GNU/Linux pour le chiffrement des disques.

Une partition chiffrée est chiffrée via une clé, clé qui est générée lors de la création de la partition chiffrée et qui est protégée par un mot de passe (appelée *phrase secrète*).

LUKS a la particularité de supporter de multiples clés pour un même volume chiffré (ce qui permet de partager un accès sans divulger sa propre clé et/ou son propre mot de passe, de créer une clé/mot de passe de secours, ...).

Une des utilisations possibles (que je conseille et détaille par la suite) et de laisser *LUKS* stocker les clés et donc de

Installer les packets utiles

```
apt-get install cryptsetup
```

Chiffrements disponibles

Pour connaitre les paramètres de chiffrement compilés par défaut sur votre système (dont le type et la taille de la clé) :

```
cryptsetup --help
```

```
Default compiled-in device cipher parameters:

loop-AES: aes, Key 256 bits

plain: aes-cbc-essiv:sha256, Key: 256 bits, Password hashing: ripemd160

LUKS1: aes-xts-plain64, Key: 256 bits, LUKS header hashing: sha256, RNG: /dev/urandom
```

Créer une partition chiffrée avec LUKS

Repérer le disque (et ses actuelles partitions)

Repérer la partition à chiffrer, ou le disque sur laquelle elle est située si l'on veut modifier les partitions de ce disque.

```
fdisk -l
# ou
tree /dev/disk
# ou, si la partition était montée :
mount
```

Par la suite, le disque que je prend comme exemple est **/dev/sdX** sur lequel je chiffre la partition **/dev/sdX1**

Démonter la partition si elle était montée

Nous allons modifier la partition (création d'une partition chiffrée, formatage), il est donc nécessaire de la démonter. Si vous souhaitez modifier la table des partitions du disque, il est nécessaire de démonter toutes les partitions du disque.

```
umount /dev/sdX1
# ou
umount /mnt/mount_point_of_sdX1
```

Editer la table des partitions du disque (optionnel)

Si vous souhaitez modifier la table des partitions du disque, voila la marche à suivre. Il est bien entendu possible de ne chiffrer que certaines partitions d'un même disque.

```
# lancer fdisk pour le disque
#  l'argument est le périphérique disque (/dev/sdX dans mes exemples)
#  et non un lien (/dev/disk/by-label/...)
sudo fdisk /dev/sdX

# commandes fdisk :
```

```
# 'd' supprimer la partition
d # supprimer les partitions existantes

# 'g' créer une nouvelle table vide de partitions GPT
g # GPT est le standard à utiliser

# 'n' ajouter une nouvelle partition
n # entrer un numéro et une taille quand demandé

# 'w' écrire la table sur le disque et quitter
w
```

Créer une partition *LUKS* sur la partition

- ```
  cryptsetup --verbose luksFormat --verify-passphrase /dev/sdX1
  # WARNING!
  # ========
  # Cette action écrasera définitivement les données sur /dev/sdX1.
  #
  # Are you sure? (Type uppercase yes): YES
  # Saisissez la phrase secrète:
  # Vérifiez la phrase secrète:
  # Opération réussie.
  ```

  Choisissez une phrase secrète digne de sécurité !

  Pour choisir une clé de taille spécifique, utiliser l'argument
  – – key-size
  Attention aux ressources CPU nécessaires ! A mon sens, la valeur par défaut (256 bits) est suffisante.

Si la partition est un volume RAID

Attention, si vous chiffrez un volume RAID (type *md RAID array*), vous devriez ajouter l'argument – – align-payload=*valeur* : cela permet d'aligner les blocs chiffrés avec les bandes (stripes) du RAID

valeur = nombre de secteurs de 512 octets (bytes) dans une bande (stripe) RAID.
valeur = [RAID chunk size] x [Nbre disques utiles dans grappe] / 512

- **RAID chunk size** (en octets = bytes)
  $ mdadm –detail /dev/mdX | grep Chunk pour l'obtenir.
  Si donné en K, multiplier par 1024 ;
- **nombre de disques utiles dans la grappe RAID**
  N/2 pour RAID 1, N-1 for RAID 5 ;
- 512 octets (bytes) par secteur.

Exemple :

- /dev/md3 : 4 HDD de 2 TB en RAID 5

```
# mdadm --detail /dev/md3
[...]
 Raid Level : raid5
 Raid Devices : 4
 Total Devices : 4
[...]
 State : clean
 Active Devices : 4
```



```
[...]
```
```

```
Layout : left-symmetric
 Chunk Size : 512K
[...]
```

- valeur = 512 x 1024 x (4-1) / 512 = 3072
- Consultez l'aide de mdadm ou ce site pour plus d'informations.

Ouvrir la partition chiffrée

Avant de formater, il est nécessaire *d'ouvrir* la partition chiffrée (et il faudra le faire à chaque démarrage ou après la *fermeture* de la partition chiffrée).

```
1 cryptsetup -v luksOpen /dev/sdX1 monVolume
```

Le mot de passe vous est demandé.

Attention !
A présent, pour accéder au contenu de la partition chiffrée /dev/sdX1, il **ne faut pas** utiliser /dev/sdX1 mais le *mapper* créé lors de l'ouverture de la partition chiffrée : **/dev/mapper/monVolume**
(le nom du *mapper* a été spécifié lors de l'*ouverture* avec *cryptsetup luksOpen /dev/sdX1* ***monVolume***).

Formater la partition chiffrée

Maintenant que la partition est *ouverte* et que l'on y accède via le *mapper*, nous pouvons la formater.

```
1 mke2fs -t ext4 -L monVolume /dev/mapper/monVolume
```

Voila ! Il est à présent possible de monter le *mapper*, et de se servir de notre partition chiffrée. Voir ci-dessous.

Ouvrir puis montrer la partition chiffrée

Ouvrir la partition chiffrée

```
1 cryptsetup -v luksOpen /dev/sdX1 monVolume
```

- **/dev/sdX1** est la partition sur laquelle a été créée la partition chiffrée LUKS ;
- **monVolume** est le nom donné au *mapper* ;
- Bien-sûr, le mot de passe vous est demandé.

Attention !
Pour accéder au contenu de la partition chiffrée /dev/sdX1, il **ne faut pas** utiliser /dev/sdX1 mais le *mapper* :
/dev/mapper/monVolume

Monter le mapper de la partition chiffrée

```
1 # Création du point de montage (une fois pour toute)
2 mkdir /mnt/monVolume
3
4 # Montage du mapper
5 mount -v /dev/mapper/monVolume /mnt/monVolume
```

Démontrer puis fermer la partition chiffrée

```
umount -v /dev/mapper/monVolume
# ou
umount -v /mnt/monVolume
```

Fermer la partition chiffrée

```
cryptsetup -v luksClose monVolume
```

Scripts bash

Ouvrir une partition chiffrée

Afficher ce script au format texte

```
#!/bin/bash

#########################
# Info #
#########################
SCRIPT_NAME=$(echo $0 | sed "s/^.*\///g") #sed in order to only keep script's name, without dir path where script is
SCRIPT_AUTHOR="Valérian REITHINGER (@:valerian@reithinger.fr ; web:www.valerian.reithinger.fr)"
SCRIPT_VERSION="1.1 (04/jan/2018)"
SCRIPT_QUICK_DESCRIPTION="Mount a Luks encrypted filesystem"
SCRIPT_ARG_MIN_NB=3 # optional arg(s) not counted
SCRIPT_ARG_MAX_NB=4 # optional arg(s) not counted

#########################
# Versions #
#########################
# 1.0 (25/apr/2017) created
# 1.1 (04/jan/2018) light the code

#########################
# Dependencies #
#########################
# * 'cryptsetup' command, tested

#########################
# Config #
#########################
DEBUG=false

#########################
# To Do #
#########################
#

#########################
# Tests #
#########################
# On Debian with V1.1 : OK

#########################################################################################################
# MAIN sub-Functions #
#########################################################################################################
#######################################
# check dependencies #
#######################################
CheckDependencies()
{
 # cryptsetup
 if hash cryptsetup 2&gt;/dev/null; then
 if $DEBUG; then EchoDebugMsg "CheckDependencies(): 'cryptsetup' founded"; fi
 else
 EchoErrorMsg "no 'cryptsetup' command was found on this system!"
# DisplayHelpMsg "
```

```
###############################
# Display help msg
DisplayHelpMsg()
{
 echo "$SCRIPT_NAME: $SCRIPT_QUICK_DESCRIPTION"

echo
 echo "Usage: $SCRIPT_NAME /encrypted_dev 'mapper_name' /mount_point [key]"
 echo
 echo " /encrypted_dev : path of the encrypted device (disk/partition) to open"
 echo " 'mapper_name' : name of the mapper which will manage the crypt disk (free label)"
 echo " /mount_point : path where the opened encrypted disk will be mount (must exist!) "
 echo " key (optionnal) : key to unlock the Luks device"
 echo
 echo "[optional args]"
 echo " -v or --verbose : verbose mode"
 echo " -q or --quiet : quiet mode (do not display anything, except warnings or errors)"
 echo
 echo "[other args]"
 echo " -h or --help : display this help message"
 echo " --version : display script version"
 echo
 echo "Example: $ $SCRIPT_NAME /dev/disk/by-uuid/10d0f53c-9545-47b1-8a1b-e309d36dada8 ext_HDD_4TB_LaCie /mnt/ext_HDD_4TB_LaCie"
 echo
 echo "Location: $0"
 echo "Version: $SCRIPT_VERSION"
 echo "Author: $SCRIPT_AUTHOR"
}

###################################
# Print script version #
###################################
DisplayDescriptionMsg()
{
 echo $SCRIPT_QUICK_DESCRIPTION
}

###############################
# DisplayVersionMsg #
###############################
DisplayVersionMsg()
{
 echo $SCRIPT_VERSION
}

###############################
# CheckArgs #
###############################
CheckArgs()
{
 #Check args (nb needed, help, version, ...)
 ARG_NB=$1 #nb of args given to this script
 TAB_ARGS=("${@}") #array of args
 TAB_ARGS=("${TAB_ARGS[@]:1}") #(need to remove 1st element = nb of args)
 if $DEBUG; then EchoDebugMsg "$ARG_NB arg(s) given : ${TAB_ARGS[@]}"; fi
 #-----------------------------
 # Init variables -
 #-----------------------------
 VERBOSE=false
 QUIET_MODE=false

NOToptARGS=0

CRYPTDEVICE=""
 MAPPERNAME=""
 MOUNTPOINT=""
 LUKSKEY=""

 if $DEBUG; then EchoDebugMsg "arg: $arg" ; fi
```

```
132  case "$arg" in
133  #-------------------------------
134  # OPT args -
135  #-------------------------------
136  #Help asked ?
137  "--help"|"-h")
138  DisplayHelpMsg
139  exit 0
140  ;;
141  #Version asked ?
142  "--version")
143  DisplayVersionMsg
144  exit 0
145  ;;
146  #Description asked ?
147  "--description")
148  DisplayDescriptionMsg
149  exit 0
150  ;;
151  #verbose ?
152  "--verbose"|"-v")
153  VERBOSE=true
154  ;;
155  #quiet mode
156  "--quiet"|"-q")
157  QUIET_MODE=true
158  ;;
159  #recursive mode
160  "--recursive"|"-r")
161  RECURSIVE_MODE=true
162  ;;
163  #-------------------------------
164  # not OPT args -
165  #-------------------------------
166  *)
167  ((NOToptARGS++))
168  if [ $NOToptARGS -eq 1 ]; then CRYPTDEVICE=$arg;
169  elif [ $NOToptARGS -eq 2 ]; then MAPPERNAME=$arg;
170  elif [ $NOToptARGS -eq 3 ]; then MOUNTPOINT=$arg;
171  elif [ $NOToptARGS -eq 4 ]; then LUKSKEY=$arg;
172  else : ;
173  fi
174  ;;
175  esac
176  done
177  #-------------------------------
178  # verbose mode is stronger than quiet mode
179  #-------------------------------
180  if $VERBOSE; then
181  if $QUIET_MODE; then
182  QUIET_MODE=false
183  EchoWarningMsg "you asked both verbose and quiet mode, verbose mode is
     stronger"
184  fi
185  fi
186  #-------------------------------
187  # check min/max of not opt args -
188  #-------------------------------
189  if [ $NOToptARGS -lt $SCRIPT_ARG_MIN_NB ] ; then
190  EchoErrorMsg "not enough arguments given!
     ($ARG_NB&lt;$SCRIPT_ARG_MIN_NB=min). Display help:"
191  DisplayHelpMsg;
192  exit 1
193  elif [ $NOToptARGS -gt $SCRIPT_ARG_MAX_NB ] ; then
194  EchoErrorMsg "to much arguments given! ($ARG_NB&gt;$SCRIPT_ARG_MAX_NB=max,
     see help with -h)"
195  exit 1
196  fi
197
198 if $DEBUG; then
199  echo " -&gt; VERBOSE = $VERBOSE"
200  echo " -&gt; QUIET_MODE = $QUIET_MODE"
201  echo " -&gt; CRYPTDEVICE = $CRYPTDEVICE"
```

```
208 ##############################
209 # CheckCRYPTDEVICE #
210 ##############################
211 CheckCryptedDevice()
212 {
213  if $VERBOSE; then EchoVerboseMsg "Checking encrypted device: $CRYPTDEVICE";
    fi
214
215 #Exists ?
216  if [[ -e $CRYPTDEVICE ]]; then
217  if $VERBOSE ; then EchoVerboseMsg " -&gt; OK, exists"; fi
218  else
219  EchoErrorMsg "Encrypted device '$CRYPTDEVICE' do not exists / is not
    readable"
220  exit -1
221  fi
222 }
223
224 ##############################
225 # CheckMapper #
226 ##############################
227 CheckMapper()
228 {
229  MAPPERPATH="/dev/mapper/$MAPPERNAME"
230
231 if [ "$1" == "mustNOTexist" ]; then
232  if $VERBOSE; then EchoVerboseMsg "Checking mapper: $MAPPERPATH (must NOT
    exist)"; fi
233  if [ -e $MAPPERPATH ]; then
234  EchoErrorMsg "Mapper $MAPPERPATH ever exist"
235  exit -1
236  else
237  if $VERBOSE ; then EchoVerboseMsg " -&gt; OK, do NOT exist"; fi
238  fi
239  elif [ "$1" == "mustexist" ]; then
240  if $VERBOSE; then EchoVerboseMsg "Checking mapper: $MAPPERPATH (must
    exist)"; fi
241  if ! [ -e $MAPPERPATH ]; then
242  EchoErrorMsg "Mapper $MAPPERPATH do NOT exist"
243  exit -1
244  else
245  if $VERBOSE ; then EchoVerboseMsg " -&gt; OK, exist"; fi
246  fi
247  else
248  EchoErrorMsg "CheckMapper(): unknown argument '$1'"
249  exit 1
250  fi
251 }
252
253 ##############################
254 # CheckMOUNTPOINT #
255 ##############################
256 CheckMountPoint()
257 {
258  if $VERBOSE; then EchoVerboseMsg "Checking asked mount point: $MOUNTPOINT";
    fi
259  if [[ -d $MOUNTPOINT ]]; then
260  if $VERBOSE ; then EchoVerboseMsg " -&gt; OK, exists"; fi
261  else
262  EchoErrorMsg "Asked mount point '$MOUNTPOINT' do not exists / is not
    writable / is not a directory"
263  exit -1
264  fi
265 }
266
267 ##############################
268 # CheckMOUNT #
269 ##############################
270 CheckMount()
271 {
272  if [ "$1" == "mustNOTbeMounted" ]; then
273  if $VERBOSE; then EchoVerboseMsg "Checking mount: $MOUNTPOINT (must NOT be
    mounted)"; fi
```

```
 elif [ "$1" == "mustbeMounted" ]; then
 if $VERBOSE; then EchoVerboseMsg "Checking mount: $MOUNTPOINT (must be mounted)"; fi
 if [ $(mount | grep -c ${MOUNTPOINT::-1}) = 1 ]; then
 if $VERBOSE ; then EchoVerboseMsg " -&gt; OK, is mounted"; fi
 else
 EchoErrorMsg "$MOUNTPOINT is NOT mounted!"
 exit -1
 fi
 else
 EchoErrorMsg "CheckMount(): unknown argument '$1'"
 exit 1
 fi
}

######################
# EchoFormatedString #
######################
EchoFormatedString()
{
#EchoFormatedString "the string" thelength 'm'/'l'/'r' (middle/left/right)
 string=$1
 lengthAsked=$2
 alignment=$3

if [ $# -eq 0 ]; then
 if [ "$4" = "nonewline" ] ; then echo ""; else echo -n ""; fi
 return
 elif [ $# -eq 1 ]; then
 if [ "$4" = "nonewline" ] ; then echo "$string"; else echo -n "$string"; fi
 return
 elif [ $# -eq 2 ]; then
 FormatStringLength "$string" "$lengthAsked" 'l'
 if [ "$4" = "nonewline" ] ; then echo "$FORMATEDSTRING"; else echo -n "$FORMATEDSTRING"; fi
 return
 else
 FormatStringLength "$string" "$lengthAsked" "$alignment"
 if [ "$4" = "nonewline" ] ; then echo "$FORMATEDSTRING"; else echo -n "$FORMATEDSTRING"; fi
 return
 fi
}

######################
# FormatStringLength #
######################
FormatStringLength()
{
# FormatStringLength "the string" thelength 'm'/'l'/'r' (middle/left/right)
 string=$1
 lengthAsked=$2
 alignment=$3

if [ $# -eq 0 ]; then
 FORMATEDSTRING=""
 return
 elif [ $# -eq 1 ]; then
 FORMATEDSTRING="$string"
 return
 fi

sizeoforiginalstring=${#string}
 if [ $sizeoforiginalstring -ge $lengthAsked ]; then
 FORMATEDSTRING="$string"
 return
 fi

if [ "$alignment" = "m" ] ; then
 before=true
 while [ ${#string} -lt $lengthAsked ]; do
 if $before ; then
 before=false
```



```
 done
```

```
356
357 elif [ "$alignment" == "r" ]; then
358  while [ ${#string} -lt $lengthAsked ]; do
359  string=" $string"
360  done
361
362 else #[ "$alignment" == "l" ]; then
363  while [ ${#string} -lt $lengthAsked ]; do
364  string="$string "
365  done
366  fi
367  FORMATEDSTRING="$string"
368 }
369
370 ##############################
371 # EchoTXTColors #
372 ##############################
373 EchoInGreen()
374 {
375  echo -n -e "\033[39;32;49m" #$(BashTextStyles green)
376 }
377
378 EchoInRed()
379 {
380  echo -n -e "\033[39;31;49m" #$(BashTextStyles red)
381 }
382
383 EchoInYellow()
384 {
385  echo -n -e "\033[39;33;49m" #$(BashTextStyles yellow)
386 }
387
388 EchoInCyan()
389 {
390  echo -n -e "\033[39;36;49m" #$(BashTextStyles cyan)
391 }
392
393 EchoInLBlue()
394 {
395  echo -n -e "\033[39;94;49m" #$(BashTextStyles light-blue)
396 }
397
398 EchoInLBlack()
399 {
400  echo -n -e "\033[39;90;49m" #$(BashTextStyles light-black)
401 }
402
403 EchoInDefaultStyle()
404 {
405  echo -n -e "\033[39;0;49m" #$(BashTextStyles default)
406 }
407
408 ##############################
409 # EchoXXXXXMsg #
410 ##############################
411 EchoErrorMsg()
412 {
413  EchoInRed
414  echo -n "[ERROR] "
415  EchoInDefaultStyle
416  echo "$1"
417 }
418
419 EchoWarningMsg()
420 {
421  EchoInYellow
422  echo -n "[WARNING] "
423  EchoInDefaultStyle
424  echo "$1"
425 }
426
427 EchoDebugMsg()
428 {
```

```
EchoVerboseMsg()
{
 EchoInLBlue
 echo -n "[VERBOSE] "
 EchoInDefaultStyle
 echo "$1"
}

EchoOKMsg()
{
 EchoInGreen
 echo -n "[OK] "
 EchoInDefaultStyle
 echo "$1"
}

################################################################################################
# " MAIN " #
################################################################################################
#Check Dependencies
CheckDependencies

#Check args (nb needed, help, version, ...)
CheckArgs $# $*

CheckCryptedDevice

CheckMapper "mustNOTexist"

CheckMount "mustNOTbeMounted"

CheckMountPoint

verboseArg=""
if $VERBOSE; then verboseArg="-v"; fi

# luksOpen
if ! [ "$LUKSKEY" == "" ]; then
 if $VERBOSE; then EchoVerboseMsg "Trying to open $CRYPTDEVICE with 'crypsetup luksOpen' to mapper $MAPPERNAME (key given in arg)"; fi
 echo $LUKSKEY | cryptsetup -q $verboseArg luksOpen $CRYPTDEVICE $MAPPERNAME
else
 if $VERBOSE; then EchoVerboseMsg "Trying to open $CRYPTDEVICE with 'crypsetup luksOpen' to mapper $MAPPERNAME (key going to be asked)"; fi
 cryptsetup -q $verboseArg luksOpen $CRYPTDEVICE $MAPPERNAME
fi
CheckMapper "mustexist"

# mount
if $VERBOSE; then EchoVerboseMsg "Trying to mount $MAPPERPATH on $MOUNTPOINT ..."; fi
mount $verboseARg $MAPPERPATH $MOUNTPOINT

CheckMount "mustbeMounted"

if $QUIET_MODE; then
 :
else
 EchoOKMsg "$CRYPTDEVICE mounted on $MOUNTPOINT (via mapper $MAPPERPATH)"
fi

exit 0
```

Fermer une partition chiffrée

Afficher ce script au format texte

```
#!/bin/bash

#######################
# Info #
SCRIPT_VERSION="1.1 (04/Jan/2018)"
```

```
009 SCRIPT_QUICK_DESCRIPTION="Unmount a Luks encrypted filesystem"
010 SCRIPT_ARG_MIN_NB=1 # optional arg(s) not counted
011 SCRIPT_ARG_MAX_NB=1 # optional arg(s) not counted
012
013 ########################
014 # Versions #
015 ########################
016 # 1.0 (25/apr/2017) created
017 # 1.1 (04/jan/2018) light the code
018
019 ########################
020 # Dependencies #
021 ########################
022 # * 'cryptsetup' command, tested
023
024 ########################
025 # Config #
026 ########################
027 DEBUG=false
028
029 ########################
030 # To Do #
031 ########################
032 #
033
034 ########################
035 # Tests #
036 ########################
037 # On Debian with V1.1 : OK
038
039 ###############################################################################################
040 # MAIN sub-Functions #
041 ###############################################################################################
042 ######################################
043 # check dependencies #
044 ######################################
045 CheckDependencies()
046 {
047 # cryptsetup
048 if hash cryptsetup 2>/dev/null; then
049 if $DEBUG; then EchoDebugMsg "CheckDependencies(): 'cryptsetup' founded"; fi
050 else
051 EchoErrorMsg "no 'cryptsetup' command was found on this system!"
052 exit -1
053 fi
054 }
055
056 ################################
057 # DisplayHelpMsg #
058 ################################
059 # Display help msg
060 DisplayHelpMsg()
061 {
062 echo "$SCRIPT_NAME: $SCRIPT_QUICK_DESCRIPTION"
063
064 echo
065 echo "Usage: $SCRIPT_NAME /dev/mapper/name"
066 echo
067 echo " /dev/mapper/name : full path of the mapper which is opened for the
    crypt disk (and will be closed)"
068 echo
069 echo "[optional args]"
070 echo " -v or --verbose : verbose mode"
071 echo " -q or --quiet : quiet mode (do not display anything, except warnings
    or errors)"
072 echo
073 echo "[other args]"
074 echo " -h or --help : display this help message"
075 echo " --version : display script version"
076 echo
077 echo "Example: $ $SCRIPT_NAME /dev/mapper/bkpHDDcrypt"
078 echo
079 echo "Location: $0"
```



```
083 # Print script version #
```

```
086 ###################################
087 DisplayDescriptionMsg()
088 {
089 echo $SCRIPT_QUICK_DESCRIPTION
090 }
091
092 ###############################
093 # DisplayVersionMsg #
094 ###############################
095 DisplayVersionMsg()
096 {
097 echo $SCRIPT_VERSION
098 }
099
100 ###############################
101 # CheckArgs #
102 ###############################
103 CheckArgs()
104 {
105 #Check args (nb needed, help, version, ...)
106 ARG_NB=$1 #nb of args given to this script
107 TAB_ARGS=("${@}") #array of args
108 TAB_ARGS=("${TAB_ARGS[@]:1}") #(need to remove 1st element = nb of args)
109 if $DEBUG; then EchoDebugMsg "$ARG_NB arg(s) given : ${TAB_ARGS[@]}"; fi
110 #-----------------------------
111 # Init variables -
112 #-----------------------------
113 VERBOSE=false
114 QUIET_MODE=false
115
116 NOToptARGS=0
117
118 MAPPERPATH=""
119
120 #-----------------------------
121 # Loop on args -
122 #-----------------------------
123 for arg in "${TAB_ARGS[@]}"
124 do
125 if $DEBUG; then EchoDebugMsg "arg: $arg" ; fi
126 case "$arg" in
127 #-----------------------------
128 # OPT args -
129 #-----------------------------
130 #Help asked ?
131 "--help"|"-h")
132 DisplayHelpMsg
133 exit 0
134 ;;
135 #Version asked ?
136 "--version")
137 DisplayVersionMsg
138 exit 0
139 ;;
140 #Description asked ?
141 "--description")
142 DisplayDescriptionMsg
143 exit 0
144 ;;
145 #verbose ?
146 "--verbose"|"-v")
147 VERBOSE=true
148 ;;
149 #quiet mode
150 "--quiet"|"-q")
151 QUIET_MODE=true
152 ;;
153 #recursive mode
154 "--recursive"|"-r")
155 RECURSIVE_MODE=true
156 ;;
157 #-----------------------------
158 # not OPT args -
```



```
164 fi
```

```
;;
esac
done
#--------------------------------
# verbose mode is stronger than quiet mode
#--------------------------------
if $VERBOSE; then
if $QUIET_MODE; then
QUIET_MODE=false
EchoWarningMsg "you asked both verbose and quiet mode, verbose mode is stronger"
fi
fi
#--------------------------------
# check min/max of not opt args -
#--------------------------------
if [ $NOToptARGS -lt $SCRIPT_ARG_MIN_NB ] ; then
EchoErrorMsg "not enough arguments given! ($ARG_NB<$SCRIPT_ARG_MIN_NB=min). Display help:"
DisplayHelpMsg;
exit 1
elif [ $NOToptARGS -gt $SCRIPT_ARG_MAX_NB ] ; then
EchoErrorMsg "to much arguments given! ($ARG_NB>$SCRIPT_ARG_MAX_NB=max, see help with -h)"
exit 1
fi

if $DEBUG; then
echo " -> MAPPERPATH = $MAPPERPATH"
fi
}

##############################
# CheckMapper #
##############################
CheckMapper()
{
if [ "$1" == "mustNOTexist" ]; then
if $VERBOSE; then EchoVerboseMsg "Checking mapper: $MAPPERPATH (must NOT exist)"; fi
if [ -e $MAPPERPATH ]; then
EchoErrorMsg "Mapper $MAPPERPATH ever exist"
exit -1
else
if $VERBOSE ; then EchoVerboseMsg " -> OK, do NOT exist"; fi
fi
elif [ "$1" == "mustexist" ]; then
if $VERBOSE; then EchoVerboseMsg "Checking mapper: $MAPPERPATH (must exist)";
fi
if ! [ -e $MAPPERPATH ]; then
EchoErrorMsg "Mapper $MAPPERPATH do NOT exist"
exit -1
else
if $VERBOSE ; then EchoVerboseMsg " -> OK, exist"; fi
fi
else
EchoErrorMsg "CheckMapper(): unknown argument '$1'"
exit 1
fi
}

##############################
# CheckMOUNT #
##############################
CheckMount()
{
MOUNTPOINT=$MAPPERPATH

if [ "$1" == "mustNOTbeMounted" ]; then
if $VERBOSE; then EchoVerboseMsg "Checking mount: $MOUNTPOINT (must NOT be mounted)"; fi
if [ $(mount | grep -c ${MOUNTPOINT::-1}) = 1 ]; then
EchoErrorMsg "$MOUNTPOINT ever mounted!"
```

```
mounted)"; fi
if [ $(mount | grep -c ${MOUNTPOINT::-1}) = 1 ]; then
if $VERBOSE ; then EchoVerboseMsg " -> OK, is mounted"; fi
else
EchoErrorMsg "$MOUNTPOINT is NOT mounted!"
exit -1
fi
else
EchoErrorMsg "CheckMount(): unknown argument '$1'"
exit 1
fi
}

##############################
# EchoTXTColors #
##############################
EchoInGreen()
{
echo -n -e "\033[39;32;49m" #$(BashTextStyles green)
}

EchoInRed()
{
echo -n -e "\033[39;31;49m" #$(BashTextStyles red)
}

EchoInYellow()
{
echo -n -e "\033[39;33;49m" #$(BashTextStyles yellow)
}

EchoInCyan()
{
echo -n -e "\033[39;36;49m" #$(BashTextStyles cyan)
}

EchoInLBlue()
{
echo -n -e "\033[39;94;49m" #$(BashTextStyles light-blue)
}

EchoInLBlack()
{
echo -n -e "\033[39;90;49m" #$(BashTextStyles light-black)
}

EchoInDefaultStyle()
{
echo -n -e "\033[39;0;49m" #$(BashTextStyles default)
}

##############################
# EchoXXXXXMsg #
##############################
EchoErrorMsg()
{
EchoInRed
echo -n "[ERROR] "
EchoInDefaultStyle
echo "$1"
}

EchoWarningMsg()
{
EchoInYellow
echo -n "[WARNING] "
EchoInDefaultStyle
echo "$1"
}

EchoDebugMsg()
{
EchoInLBlack
```

```
{
EchoInLBlue
echo -n "[VERBOSE] "
EchoInDefaultStyle
echo "$1"
}

EchoOKMsg()
{
EchoInGreen
echo -n "[OK] "
EchoInDefaultStyle
echo "$1"
}

################################################################################################
# " MAIN " #
################################################################################################
#Check Dependencies
CheckDependencies

#Check args (nb needed, help, version, ...)
CheckArgs $# $*

#CheckMount "mustbeMounted"

CheckMapper "mustexist"

verboseArg=""
if $VERBOSE; then verboseArg="-v"; fi

# unmount
if $VERBOSE; then EchoVerboseMsg "Trying to UNmount $MAPPERPATH ..."; fi
umount $verboseARg $MAPPERPATH

CheckMount "mustNOTbeMounted"

# luksClose
if $VERBOSE; then EchoVerboseMsg "Trying to close mapper $MAPPERPATH ..."; fi
cryptsetup -q $verboseArg luksClose $MAPPERPATH

CheckMapper "mustNOTexist"

if $QUIET_MODE; then
:
else
EchoOKMsg "Mapper $MAPPERPATH and it's mount point successfully Close /
UNmounted"
fi

exit 0
```