# Créer un conteneur/coffre-fort chiffré v2

## Introduction

Ce document est une version mise à jour en juillet 2026 avec beaucoup de changements améliorant la sécurité du coffre.

L'idée ici est de créer un fichier coffre-fort de quelques Go, remplit au départ de nombre aléatoires, et qui servira à stocker vos dossiers et fichiers confidentiels.

On utilisera ici un cryptage fort, avec un mot de passe très long + un sel pour brouiller le mot de passe, + une itération permettant de ralentir les attaques, et enfin : une clé décalée dans le conteneur, pour accroître la difficulté à casser le coffre-fort via une attaque par dictionnaire.

Bien entendu, rien n'empêche ensuite de mettre un coffre dans un autre coffre, puis dans un troisième, etc.

## Logique globale

Dans un premier temps, il faut créer le fichier conteneur qui contiendra les fichiers et dossiers à protéger.

Pour cela, on utilise la commande

```
dd if=/dev/urandom of=/root/coffre.dat bs=1G count=3
```

qui utilise le générateur de nombres aléatoires du noyau pour créer le fichier de sortie ***coffre.dat***, ici de 3Go (à adapter bien entendu suivant vos besoins et l'espace disque restant sur votre disque dur…).

L'étape suivante est d'utiliser la commande

```
losetup -f
```

qui permet de trouver le prochain périphérique de bouclage libre et de le lier au fichier conteneur via :

```
losetup /dev/loopX /root/coffre.dat
```

A partir de là, un

```
losetup -a
```

montre la liaison conteneur / périphérique de bouclage.

L'étape suivante est d'utiliser `openssl` pour générer la clé, à partir de sa taille, de mot de passe, du sel et du nombre d'itérations.

```
    openssl kdf \
        -keylen "$key_length" \
        -kdfopt "digest:$HASH_ALGO" \
        -kdfopt "iter:$iterations" \
        -kdfopt "salt:$salt" \
        -kdfopt "pass:$password" \
        -binary \
        PBKDF2
```

C'est cette étape cruciale qui permet de générée une clé « brouillée », qui va ralentir les attaques par dictionnaires via PBKDF2, la condition étant un long mot de passe (au moins 32 caractères) et un sel si possible également complexe.

Plus le nombre d'itérations est grand, plus la génération est longue, et la durée d'attaque allongée.

A cette étape, il ne faut pas descendre en dessous de 1 500 000.

`Openssl` génère donc une clé de 512 octets « brouillée » qui va être envoyée à `cryptsetup` via un pipe (le script n'écrit pas de fichiers de clé en utilisation normale hors tests).

C'est au final `cryptsetup` qui va s'occuper de chiffrer à la volée les données du conteneur, en utilisant le périphérique de bouclage, l'algorithme de chiffrement *$CYPHER*, le décalage de la clé dans le conteneur *$offset*, la taille de la clé *$KEYSIZE* et comme nous l'avons dit plus haut la clé « brouillée » par `openssl`.

```
cryptsetup create "$DEV_MAPPER" "$LOOP" \
            -c "$CYPHER" \
            -o "$offset" \
            -s "$KEYSIZE" \
            --key-file -
```

Cette commande va créer une entrée dans ***/dev/mapper/*** avec le nom *$DEV_MAPPER* choisit en paramètres au début du script.

Ce nouveau périphérique s'émule ensuite comme une partition classique de disque dur.

Il faut donc, la première fois, la formater via `mkfs.ext4` pour pouvoir ensuite la monter via `mount` dans le dossier *$MOUNT_POINT* et l'utiliser comme un dossier classique non chiffré.

La fermeture du conteneur se fait dans l'ordre inverse :

➔ démontage du *MOUNT_POINT* via `umount`

➔ destruction du ***/dev/mapper/DEV_MAPPER*** via `cryptsetup remove`

➔ destruction du périphérique de bouclage via `losetup -d`

et généralement vérification finale manuelle via `df` que tout est ok !

## Avantages

Chiffrement fort (AES-XTS 512 bits)

Protection contre la force brute (PBKDF2)

Absence de traces (100% PIPE)

Vérification par token (confirmation d'intégrité)

## Inconvénients

Le mode PLAIN ici utilisé n'offre PAS de protection contre :

➔ La corruption du conteneur

➔ La modification accidentelle des métadonnées

➔ La récupération après suppression de l'en-tête

Dit autrement, si ,pour une raison ou une autre, le fichier conteneur est corrompu, vous perdez tout !

Retenez donc qu'une sauvegarde régulière du conteneur de base n'est pas un luxe, et doit se faire évidemment quand le conteneur est démonté.

Dans la pratique, je n'ai jamais eu de pertes sur 20 ans d'utilisation de `cryptsetup` – mais il faut toujours mieux rester prudent…

## Pourquoi ne pas utiliser LUKS ?

Parce que je n'aime pas l'idée de slots de clés multiples, qui pour moi sont autant de portes d'entrées potentielles.

**LUKS** utilise une clé dérivée pour chiffrer les données et on peut donc changer le mot de passe du conteneur, voire en assigner plusieurs, sans rechiffrer les données.

On peut voir ça comme un avantage, mais pour ma part, si je travaillais pour la sécurité, je viserais la corruption d'un seul slot pour pourrir tout le système, là où le type **PLAIN** reste « tout ou rien », pour le meilleur ou pour le pire...

Bref, je ne nie pas les avantages de **LUKS** : métadonnées structurées dans l'en-tête, protection contre la corruption, dérivation de clé intégrée, ...

C'est juste que j'estime la méthode tout ou rien plus sécurisée, parce qu'offrant moins de portes d'entrées potentielles.

## Pourquoi un mode de test dans le script ?

Le script propose un mode *debug* et surtout une option permettant de tester que la corruption d'un seul paramètre fait échouer tout le montage.

Pourquoi une telle précaution ?

Parce que j'ai eu quelques surprises dans les syntaxes avant de trouver la bonne combinaison, entre la documentation sur le web et les fichiers man de base, et que parfois, mon mot de passe était simplement ignoré !

Après avoir soigneusement créé votre conteneur, je vous conseille vivement l'*option 4*, pour lancer les autotests, et vérifier que tout est ok.

Le première test doit fonctionner (montage avec les paramètres), les autres doivent tous échouer.

## Licence

Le script est sous GPLv2 et fournit tel que, sans aucune garantie, à utiliser à vos risques périls, comme tout autre logiciel libre.

## Le script final

```
#!/bin/bash
# Script PBKDF2 - Version avec DEV_MAPPER configurable
# Version 7.13 - Nom du mapping personnalisable

# Licence GPLv2
# Jean Luc BIELLLMANN - contact@alsatux.com

set -euo pipefail

# Configuration
CONTAINER_PATH="/root/coffre.dat"
# Point de montage du container
MOUNT_POINT="/mnt"
# Nom du device dans /dev/mapper/
DEV_MAPPER="coffre"
# chiffrement
CYPHER="aes-xts-plain64"
# Longueur de la clé
KEYSIZE=512
# Algo de hashage
HASH_ALGO="sha512"

# Token FIXE (fichier caché à laisser dans le container)
TOKEN_FILE=".secret_token"
TOKEN_CONTENT="TOKEN_123456789_ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Mode débogage (1=activé, 0=désactivé)
DEBUG=1

# Couleurs
RED='\033[0;31m'
GREEN='\033[0;32m'
```

```
YELLOW='\033[1;33m'
BLUE='\033[0;34m'
CYAN='\033[0;36m'
BOLD='\033[1m'
NC='\033[0m'

print_ok() { echo -e "${GREEN}✅ $1${NC}"; }
print_error() { echo -e "${RED}❌ $1${NC}"; }
print_warning() { echo -e "${YELLOW}⚠ $1$
{NC}"; }
print_info() { echo -e "${BLUE}ℹ️ $1${NC}"; }
print_title() { echo -e "\n${BOLD}📌 $1$
{NC}"; }
print_debug() {
    if [ "$DEBUG" -eq 1 ]; then
        echo -e "${CYAN}🔍 $1${NC}" >&2
    fi
}
print_cmd() {
    if [ "$DEBUG" -eq 1 ]; then
        echo -e "${YELLOW}🔧 CMD: $1${NC}" >&2
    fi
}

show_hash_summary() {
    local hash="$1"
    local prefix="${2:-}"
    if [ ${#hash} -le 16 ]; then
        echo "${prefix}${hash}"
    else
          echo "${prefix}${hash:0:16}...${hash:
-8}"
    fi
}

generate_salt() {
    local salt_password="$1"
    local salt
    salt=$(echo -n "$salt_password" | sha512sum
| cut -d' ' -f1 | cut -c1-64)

    print_debug "Génération du sel"
    print_debug "Sel généré: $salt"

    echo "$salt"
    return 0
}

# Génération PBKDF2 avec debug visible (sortie
hex)
generate_pbkdf2_debug() {
    local password="$1"
    local iterations="$2"
    local salt="$3"
    local key_length="$4"

    local key_bits=$((key_length * 8))

    echo "" >&2
                                        print_debug
"========================================="
    print_debug "🔐 GÉNÉRATION PBKDF2 (DEBUG)"
    print_debug "   - Mot de passe: (masqué)"
    print_debug "   - Itérations: $iterations"
```

```
    print_debug "   - Sel: $salt"
    print_debug "   - Longueur clé: $key_length
octets ($key_bits bits)"
    print_debug "   - Hash: $HASH_ALGO"
                                    print_debug
"========================================="

    # Commande complète visible
     local cmd="openssl kdf -keylen $key_length
-kdfopt    digest:$HASH_ALGO    -kdfopt    iter:
$iterations  -kdfopt  salt:$salt  -kdfopt  pass:
$password -binary PBKDF2"
    print_cmd "$cmd"

    # Génération de la clé en hex pour debug
    local key_hex
    key_hex=$(openssl kdf \
        -keylen "$key_length" \
        -kdfopt "digest:$HASH_ALGO" \
        -kdfopt "iter:$iterations" \
        -kdfopt "salt:$salt" \
        -kdfopt "pass:$password" \
        -binary \
        PBKDF2 2>/dev/null | xxd -p -c 256 | tr
-d '\n')

    local ret=$?
    if [ $ret -ne 0 ]; then
         print_error "❌ Échec de la génération
PBKDF2 (code: $ret)"
        return $ret
    fi
```

```
    local key_len=${#key_hex}
    print_debug "✅ Génération PBKDF2 réussie"
      print_debug "    - Longueur hex: $key_len
caractères"
          print_debug  "       -  Condensat:  $
{key_hex:0:32}...${key_hex: -8}"

    return 0
}

#  Génération  PBKDF2  pour  la  vraie  execution
(sortie binaire)
generate_pbkdf2_binary() {
    local password="$1"
    local iterations="$2"
    local salt="$3"
    local key_length="$4"

    # Exécution silencieuse - sortie binaire
    openssl kdf \
        -keylen "$key_length" \
        -kdfopt "digest:$HASH_ALGO" \
        -kdfopt "iter:$iterations" \
        -kdfopt "salt:$salt" \
        -kdfopt "pass:$password" \
        -binary \
        PBKDF2 2>/dev/null

    return $?
}
```

```
# Vérification du superblock ext4
check_ext4_superblock() {
    local device="$1"
    if [ ! -b "$device" ]; then
        return 1
    fi

      print_debug "Vérification du superblock
ext4 sur $device"

    local magic
       magic=$(dd if="$device" bs=1 skip=1080
count=2 2>/dev/null | xxd -p 2>/dev/null | tr
-d '\n')

      print_debug "Signature détectée: $magic
(attendue: 53ef)"

    if [ "$magic" = "53ef" ]; then
        print_debug "✅ Superblock ext4 valide"
        return 0
    else
            print_debug "❌ Superblock ext4
invalide"
        return 1
    fi
}

# Fonction de montage avec vérification STRICTE
mount_with_key_and_check_strict() {
    local password="$1"
    local salt="$2"
    local iterations="$3"
    local offset="$4"
    local test_name="${5:-Test}"

     # Utiliser un nom de mapping unique pour
les tests
    local test_mapper="pbkdf2_test_$$"
                                        local
test_device="/dev/mapper/$test_mapper"

    print_title "TEST: $test_name"
    print_info "Paramètres:"
    print_info "   - Itérations: $iterations"
    print_info "   - Offset: $offset octets"
     print_info "    - Sel: $(show_hash_summary
"$salt")"
    print_info "   - Mot de passe: (masqué)"
       print_info "      - Mapping de test:
$test_mapper"

    # Debug : génération de la clé visible
    local key_length=$((KEYSIZE / 8))
      print_info "Debug de la génération de
clé..."
       if ! generate_pbkdf2_debug "$password"
"$iterations" "$salt" "$key_length"; then
             print_error "Debug de génération
échoué"
        return 1
    fi
    echo ""
```

```
    local LOOP
    LOOP=$(losetup -f)
    if [ -z "$LOOP" ]; then
            print_error "Pas de loop device
disponible"
        return 1
    fi
    print_debug "Loop device: $LOOP"

    print_cmd "losetup $LOOP $CONTAINER_PATH"
       if ! losetup "$LOOP" "$CONTAINER_PATH"
2>/dev/null; then
        print_error "Échec attachement loop"
        losetup -d "$LOOP" 2>/dev/null
        return 1
    fi
    print_ok "Loop device attaché"

    local temp_mount="/tmp/mount_test_$$"
    mkdir -p "$temp_mount" 2>/dev/null

     print_debug "Point de montage temporaire:
$temp_mount"

      # Génération de la clé en binaire et
création du mapping
    print_info "Création du mapping de test..."

          local cmd_crypt="cryptsetup create
$test_mapper $LOOP -c $CYPHER -o $offset -s
$KEYSIZE --key-file -"
    print_cmd "$cmd_crypt"
```

```
    # Exécution réelle avec sortie binaire
       if ! generate_pbkdf2_binary "$password"
"$iterations" "$salt" "$key_length" | \
             cryptsetup create "$test_mapper"
"$LOOP" \
            -c "$CYPHER" \
            -o "$offset" \
            -s "$KEYSIZE" \
            --key-file - 2>/dev/null; then
        print_error "Échec création mapping"
        losetup -d "$LOOP" 2>/dev/null
        rmdir "$temp_mount" 2>/dev/null
        return 1
    fi
    print_ok "Mapping créé: $test_device"

    # Montage en lecture seule
    print_debug "Montage en lecture seule..."
            print_cmd "mount -t ext4 -o
noatime,nodev,nosuid,ro $test_device
$temp_mount"

             if ! mount -t ext4 -o
noatime,nodev,nosuid,ro "$test_device"
"$temp_mount" 2>/dev/null; then
        print_error "Échec montage"
             cryptsetup remove "$test_mapper"
2>/dev/null
        losetup -d "$LOOP" 2>/dev/null
        rmdir "$temp_mount" 2>/dev/null
        return 1
```

```
    fi
    print_ok "Montage réussi"

    # Vérification STRICTE du token
    print_debug "Vérification du token..."

    local result=1
    if [ -f "$temp_mount/$TOKEN_FILE" ]; then
        local content
        content=$(cat "$temp_mount/$TOKEN_FILE"
2>/dev/null)
        print_debug "Token trouvé: $content"
                    print_debug  "Token  attendu:
$TOKEN_CONTENT"

         if [ "$content" = "$TOKEN_CONTENT" ];
then
            print_ok "✅ Token correspond !"
            result=0
        else
            print_error "❌ Token incorrect !"
            result=1
        fi
    else
        print_warning "⚠  Token absent"
        result=1
    fi

    # Nettoyage
    print_debug "Nettoyage..."
    print_cmd "umount $temp_mount"
    umount "$temp_mount" 2>/dev/null
```

```
    print_cmd "cryptsetup remove $test_mapper"
            cryptsetup   remove   "$test_mapper"
2>/dev/null

    print_cmd "losetup -d $LOOP"
    losetup -d "$LOOP" 2>/dev/null

    rmdir "$temp_mount" 2>/dev/null

    print_debug "Nettoyage terminé"

    return $result
}

# Fonction de montage normale
mount_with_key_and_check() {
    local password="$1"
    local salt="$2"
    local iterations="$3"
    local offset="$4"
    local token_check="${5:-1}"
    local test_name="${6:-Test}"

    local test_mapper="pbkdf2_test_$$"
                                                local
test_device="/dev/mapper/$test_mapper"

    print_debug "Montage avec vérification"
          print_debug   "       -   Token   check:
$token_check"
```

```
        print_debug "     - Mapping de test:
$test_mapper"

    # Debug : génération de la clé visible
    local key_length=$((KEYSIZE / 8))
       print_info "Debug de la génération de
clé..."
        if ! generate_pbkdf2_debug "$password"
"$iterations" "$salt" "$key_length"; then
             print_error "Debug de génération
échoué"
        return 1
    fi
    echo ""

    local LOOP
    LOOP=$(losetup -f)
    if [ -z "$LOOP" ]; then
             print_error "Pas de loop device
disponible"
        return 1
    fi
    print_debug "Loop device: $LOOP"

    print_cmd "losetup $LOOP $CONTAINER_PATH"
       if ! losetup "$LOOP" "$CONTAINER_PATH"
2>/dev/null; then
        print_error "Échec attachement loop"
        losetup -d "$LOOP" 2>/dev/null
        return 1
    fi
    print_ok "Loop device attaché"
```

```
    local temp_mount="/tmp/mount_test_$$"
    mkdir -p "$temp_mount" 2>/dev/null

       # Génération de la clé en binaire et
création du mapping
    print_info "Création du mapping de test..."

          local cmd_crypt="cryptsetup create
$test_mapper $LOOP -c $CYPHER -o $offset -s
$KEYSIZE --key-file -"
    print_cmd "$cmd_crypt"

    # Exécution réelle avec sortie binaire
       if ! generate_pbkdf2_binary "$password"
"$iterations" "$salt" "$key_length" | \
             cryptsetup create "$test_mapper"
"$LOOP" \
            -c "$CYPHER" \
            -o "$offset" \
            -s "$KEYSIZE" \
            --key-file - 2>/dev/null; then
        print_error "Échec création mapping"
        losetup -d "$LOOP" 2>/dev/null
        rmdir "$temp_mount" 2>/dev/null
        return 1
    fi
    print_ok "Mapping créé: $test_device"

    # Montage en lecture seule
    print_debug "Montage en lecture seule..."
```

```
            print_cmd   "mount   -t   ext4   -o
noatime,nodev,nosuid,ro            $test_device
$temp_mount"

              if   !   mount   -t   ext4   -o
noatime,nodev,nosuid,ro          "$test_device"
"$temp_mount" 2>/dev/null; then
        print_error "Échec montage"
              cryptsetup  remove  "$test_mapper"
2>/dev/null
        losetup -d "$LOOP" 2>/dev/null
        rmdir "$temp_mount" 2>/dev/null
        return 1
    fi
    print_ok "Montage réussi"

    local result=0

    if [ "$token_check" -eq 1 ]; then
        print_debug "Vérification du token..."

          if [ -f "$temp_mount/$TOKEN_FILE" ];
then
            local content
                                  content=$(cat
"$temp_mount/$TOKEN_FILE" 2>/dev/null)
                    print_debug  "Token  trouvé:
$content"
                  print_debug  "Token  attendu:
$TOKEN_CONTENT"
```

```
                                if   [   "$content"   =
"$TOKEN_CONTENT" ]; then
                                 print_ok  "✅  Token
correspond !"
                result=0
            else
                 print_error "❌ Token incorrect
!"
                result=1
            fi
        else
                print_warning "⚠  Token absent,
fallback sur superblock"

                        if  check_ext4_superblock
"$test_device"; then
                   print_ok "✅ Superblock ext4
valide (fallback)"
                result=0
            else
                 print_error "❌ Superblock ext4
invalide"
                result=1
            fi
        fi
    else
            print_debug  "Vérification  du  token
désactivée"
    fi

    # Nettoyage
    print_debug "Nettoyage..."
```

```
    print_cmd "umount $temp_mount"
    umount "$temp_mount" 2>/dev/null

    print_cmd "cryptsetup remove $test_mapper"
            cryptsetup   remove   "$test_mapper"
2>/dev/null

    print_cmd "losetup -d $LOOP"
    losetup -d "$LOOP" 2>/dev/null

    rmdir "$temp_mount" 2>/dev/null

    print_debug "Nettoyage terminé"

    return $result
}

# Mode diagnostic
diagnostic_mode() {
    echo ""
                                                echo
"============================================="
    echo -e "  ${BOLD}🔬 MODE DIAGNOSTIC${NC}"
                                                echo
"============================================="
      echo  -e  "   ${YELLOW}🐛 DÉBOGAGE ACTIVÉ$
{NC}"
                                                echo
"============================================="
    echo ""
      echo  -e  "${BLUE}ℹ️   Le  mode  diagnostic
teste la robustesse du système${NC}"
```

```
     echo -e "${BLUE}   en vérifiant que seuls
les paramètres CORRECTS${NC}"
    echo -e "${BLUE}   permettent le montage du
conteneur.${NC}"
    echo ""
      echo -e "${YELLOW}⚠   Les tests avec des
paramètres ERRONÉS DOIVENT ÉCHOUER${NC}"
     echo -e "${YELLOW}   C'est le comportement
attendu et SOUHAITABLE !${NC}"
     echo -e "${YELLOW}   Un échec signifie que
le  système  a  rejeté  les  mauvais  paramètres.$
{NC}"
    echo ""
      echo  -e  "${GREEN}✅ Un  SUCCÈS  avec  des
paramètres ERRONÉS indiquerait${NC}"
        echo  -e  "${GREEN}      une  FAILLE  DE
SÉCURITÉ !${NC}"
    echo ""

    if [ ! -f "$CONTAINER_PATH" ]; then
                print_error  "Fichier  conteneur
introuvable: $CONTAINER_PATH"
        return 1
    fi

    local size
           size=$(stat   -c%s   "$CONTAINER_PATH"
2>/dev/null)
    echo ""
      print_info "Conteneur: $CONTAINER_PATH ($
((size/1048576))Mo)"
    echo ""
```

```
     echo "📝 Entrez le mot de passe PRINCIPAL
CORRECT :"
    read -s password_correct
    echo ""

    echo "📝 Entrez le mot de passe pour le SEL
CORRECT :"
    read -s salt_password_correct
    echo ""

      echo "📝 Entrez le nombre d'itérations
CORRECT :"
    read -r iterations_correct

         echo "📝 Entrez l'offset CORRECT
(octets) :"
    read -r offset_correct

     if ! [[ "$iterations_correct" =~ ^[0-9]+
$ ]] || [ "$iterations_correct" -lt 1000 ];
then
        print_error "Itérations invalides"
        return 1
    fi

    if ! [[ "$offset_correct" =~ ^[0-9]+$ ]] ||
[ "$offset_correct" -lt 0 ]; then
        print_error "Offset invalide"
        return 1
    fi
```

```
    local salt_correct
                    salt_correct=$(generate_salt
"$salt_password_correct")

    echo ""
                                            echo
"========================================="
    print_title "PARAMÈTRES CORRECTS"
    echo "   - Mot de passe: (masqué)"
    echo "   - Sel: $salt_correct"
         echo  "       -  Sel  (condensat):  $
(show_hash_summary "$salt_correct")"
    echo "   - Itérations: $iterations_correct"
    echo "   - Offset: $offset_correct octets"
    echo "   - Token: $TOKEN_CONTENT"
                                            echo
"========================================="
    echo ""

    echo -e "${YELLOW}⚠  Rappel : Seul le TEST
1 doit réussir.${NC}"
      echo -e "${YELLOW}    Les tests 2 à 6
DOIVENT ÉCHOUER.${NC}"
    echo ""
    echo -n "Appuyez sur ENTRÉE pour lancer les
tests... "
    read -r

    local test_results=()

    # Test 1 : Paramètres corrects
```

```
    print_title "TEST 1 : PARAMÈTRES CORRECTS (RÉFÉRENCE)"
    echo "   → Résultat attendu: ✅ SUCCÈS"
    echo "   → Ce test vérifie que le système fonctionne avec les bons paramètres"
    echo ""

    if mount_with_key_and_check "$password_correct" "$salt_correct" "$iterations_correct" "$offset_correct" 0 "Paramètres corrects"; then
        print_ok "Montage réussi ✓"
        test_results+=("✅ TEST 1: Paramètres corrects → SUCCÈS")
    else
        print_error "Échec du montage avec les paramètres corrects !"
        test_results+=("❌ TEST 1: Paramètres corrects → ÉCHEC")
    fi

    echo ""
    echo -n "Appuyez sur ENTRÉE pour continuer... "
    read -r

    # Test 2 : Mot de passe bidon
    print_title "TEST 2 : MOT DE PASSE BIDON"
    echo "   → Mot de passe: 'motdepassebidon123'"
    echo "   → Résultat attendu: ❌ ÉCHEC (token absent)"
    echo "   → Un succès indiquerait qu'un mot de passe incorrect peut ouvrir le conteneur"
    echo ""

    local fake_password="motdepassebidon123"

    if mount_with_key_and_check_strict "$fake_password" "$salt_correct" "$iterations_correct" "$offset_correct" "Mot de passe bidon"; then
        print_error "⚠  Le montage a réussi avec un mot de passe bidon !"
        test_results+=("❌ TEST 2: Mot de passe bidon → SUCCÈS (FAILLE DE SÉCURITÉ!)")
    else
        print_ok "Échec du montage ✓ (comme attendu)"
        test_results+=("✅ TEST 2: Mot de passe bidon → ÉCHEC (bon)")
    fi

    echo ""
    echo -n "Appuyez sur ENTRÉE pour continuer... "
    read -r

    # Test 3 : Sel bidon
    print_title "TEST 3 : SEL BIDON"
    echo "   → Mot de passe sel: 'fauxsel123'"
    echo "   → Résultat attendu: ❌ ÉCHEC"
    echo "   → Un sel différent génère une clé différente → montage impossible"
```

```
    echo ""

    local fake_salt_password="fauxsel123"
    local fake_salt
                      fake_salt=$(generate_salt
"$fake_salt_password")
    echo "   → Sel généré: $fake_salt"
         echo   "       →   Sel   (condensat):   $
(show_hash_summary "$fake_salt")"

           if   mount_with_key_and_check_strict
"$password_correct"                  "$fake_salt"
"$iterations_correct"   "$offset_correct"   "Sel
bidon"; then
          print_error "Le montage a réussi avec
un sel bidon !"
         test_results+=("❌ TEST 3: Sel bidon →
SUCCÈS (FAILLE!)")
    else
           print_ok "Échec du montage ✓ (comme
attendu)"
         test_results+=("✅ TEST 3: Sel bidon →
ÉCHEC (bon)")
    fi

    echo ""
         echo   -n   "Appuyez   sur   ENTRÉE   pour
continuer... "
    read -r

    # Test 4 : Itérations bidon
    print_title "TEST 4 : ITÉRATIONS BIDON"
```

```
     echo "    → Itérations: 1000 (au lieu de
$iterations_correct)"
    echo "   → Résultat attendu: ❌ ÉCHEC"
    echo "   → Un nombre d'itérations différent
génère une clé différente"
    echo ""

    local fake_iterations=1000

           if   mount_with_key_and_check_strict
"$password_correct"              "$salt_correct"
"$fake_iterations"            "$offset_correct"
"Itérations bidon"; then
         print_error "Le montage a réussi avec
des itérations bidon !"
          test_results+=("❌ TEST 4: Itérations
bidon → SUCCÈS (FAILLE!)")
    else
           print_ok "Échec du montage ✓ (comme
attendu)"
          test_results+=("✅ TEST 4: Itérations
bidon → ÉCHEC (bon)")
    fi

    echo ""
         echo   -n   "Appuyez   sur   ENTRÉE   pour
continuer... "
    read -r

    # Test 5 : Offset bidon
    print_title "TEST 5 : OFFSET BIDON"
```

```
        local  fake_offset=$((offset_correct  +
4096))
    if [ $((fake_offset % 512)) -ne 0 ]; then
                  fake_offset=$((fake_offset  -
(fake_offset % 512) + 512))
    fi
     echo "   → Offset: $fake_offset octets (au
lieu de $offset_correct)"
    echo "   → Résultat attendu: ❌ ÉCHEC"
    echo "   → Un offset incorrect déchiffre au
mauvais endroit"
    echo ""

           if   mount_with_key_and_check_strict
"$password_correct"             "$salt_correct"
"$iterations_correct"   "$fake_offset"   "Offset
bidon"; then
          print_error "Le montage a réussi avec
un offset bidon !"
        test_results+=("❌ TEST 5: Offset bidon
→ SUCCÈS (FAILLE!)")
    else
          print_ok "Échec du montage ✓ (comme
attendu)"
        test_results+=("✅ TEST 5: Offset bidon
→ ÉCHEC (bon)")
    fi

    echo ""
         echo  -n  "Appuyez  sur  ENTRÉE  pour
continuer... "
    read -r
```

```
    # Test 6 : Tout bidon
    print_title "TEST 6 : TOUT BIDON"
    echo "   → Tous les paramètres sont faux"
    echo "   → Résultat attendu: ❌ ÉCHEC"
      echo "    → Combiné, toutes les erreurs
doivent échouer"
    echo ""

           if   mount_with_key_and_check_strict
"$fake_password"                    "$fake_salt"
"$fake_iterations" "$fake_offset" "Tout bidon";
then
         print_error "Le montage a réussi avec
tous les paramètres bidon !"
        test_results+=("❌ TEST 6: Tout bidon →
SUCCÈS (FAILLE!)")
    else
          print_ok "Échec du montage ✓ (comme
attendu)"
        test_results+=("✅ TEST 6: Tout bidon →
ÉCHEC (bon)")
    fi

    echo ""
         echo  -n  "Appuyez  sur  ENTRÉE  pour
continuer... "
    read -r

    # Résumé
    echo ""
```

```
                                                    echo
"============================================="
    echo -e "  ${BOLD}📊 RÉSUMÉ DES TESTS${NC}"
                                                    echo
"============================================="
    echo -e "${BLUE}ℹ️  Lecture des résultats :
${NC}"
      echo  -e  "${GREEN}     ✅  →  Comportement
correct${NC}"
       echo  -e  "${RED}     ❌  →  Comportement
incorrect (faille de sécurité)${NC}"
    echo ""

    local nb_ok=0
    local nb_fail=0

    for result in "${test_results[@]}"; do
        if [[ "$result" == ✅* ]]; then
            echo -e "${GREEN}$result${NC}"
            ((nb_ok++))
        else
            echo -e "${RED}$result${NC}"
            ((nb_fail++))
        fi
    done

    echo ""
                                                    echo
"============================================="
    echo -e "  ${BOLD}📈 BILAN${NC}"
                                                    echo
"============================================="
```

```
          echo  "       Tests  passés:  $nb_ok/$
{#test_results[@]}"
            echo   "        Échecs:   $nb_fail/$
{#test_results[@]}"

    if [ $nb_fail -eq 0 ]; then
        print_ok "✅ Tous les tests sont passés
!"
          echo -e "${GREEN}    La sécurité est
bonne : seuls les paramètres corrects"
          echo -e "    permettent le montage du
conteneur.${NC}"
    else
        print_error "❌ Des tests ont échoué !"
           echo -e "${RED}    La sécurité est
compromise : des paramètres erronés"
          echo -e "    permettent le montage du
conteneur.${NC}"
    fi
                                                    echo
"============================================="
}

# Menu création
create_container() {
    echo ""
                                                    echo
"============================================="
    echo "  🔐 CRÉATION DU MAPPING"
                                                    echo
"============================================="
```

```
    if [ ! -f "$CONTAINER_PATH" ]; then
            print_error "Fichier introuvable:
$CONTAINER_PATH"
        return 1
    fi

    echo "📝 Mot de passe PRINCIPAL :"
    read -s password_main
    echo ""

    echo "📝 Mot de passe pour le SEL :"
    read -s password_salt
    echo ""

    echo "📝 Nombre d'itérations (100000+) :"
    read -r iterations

       echo "📝 Offset (octets, multiple de
512) :"
    read -r offset

     if ! [[ "$iterations" =~ ^[0-9]+$ ]] ||
[ "$iterations" -lt 100000 ]; then
        print_error "Itérations invalides"
        return 1
    fi

      if ! [[ "$offset" =~ ^[0-9]+$ ]] ||
[ "$offset" -lt 0 ]; then
        print_error "Offset invalide"
        return 1
    fi
```

```
    local salt
    salt=$(generate_salt "$password_salt")

    echo ""
    echo "📌 RÉSUMÉ :"
    echo "   - Itérations: $iterations"
    echo "   - Offset: $offset octets"
    echo "   - Sel: $salt"
    echo "   - Token: $TOKEN_CONTENT"
        echo "       - Mapping: $DEV_MAPPER
(/dev/mapper/$DEV_MAPPER)"
    echo ""

      # Vérification si le fichier conteneur
semble déjà formaté
    local already_formatted=0
    if [ -f "$CONTAINER_PATH" ]; then
        local size
            size=$(stat -c%s "$CONTAINER_PATH"
2>/dev/null)
        if [ "$size" -gt 0 ]; then
            # Vérification rapide : chercher la
signature ext4
            local magic
                magic=$(dd if="$CONTAINER_PATH"
bs=1 skip=$((offset + 1080)) count=2
2>/dev/null | xxd -p 2>/dev/null | tr -d '\n')
            if [ "$magic" = "53ef" ]; then
                already_formatted=1
            fi
        fi
```

```
    fi

    if [ $already_formatted -eq 1 ]; then
        echo ""
            echo -e "${RED}⚠  ATTENTION : Un système de fichiers ext4 semble déjà présent !${NC}"
        echo -e "${RED}  Le formatage DÉTRUIRA toutes les données du conteneur !${NC}"
        echo ""
         echo -e "${YELLOW}  Si vous avez des données dans ce conteneur, arrêtez-vous maintenant.${NC}"
           echo -e "${YELLOW}  Le formatage efface TOUT de manière irréversible.${NC}"
        echo ""
         echo -n "Voulez-vous vraiment FORMATER ce conteneur ? (oui/NON) "
        read -r confirm_format

        if [[ ! "$confirm_format" =~ ^(oui|Oui|OUI)$ ]]; then
            print_warning "Formatage annulé"
            return 1
        fi

        echo ""
        echo -e "${RED}⚠  DERNIER AVERTISSEMENT :${NC}"
          echo -e "${RED}  Vous êtes sur le point de DÉTRUIRE toutes les données${NC}"
            echo -e "${RED}  du conteneur $CONTAINER_PATH${NC}"
        echo ""
            echo -n "Confirmer le formatage définitif ? (OUI/DESTRUCTION) "
        read -r confirm_final

           if [[ ! "$confirm_final" =~ ^(OUI|DESTRUCTION)$ ]]; then
            print_warning "Formatage annulé"
            return 1
        fi

        echo -e "${RED}⚠  FORMATAGE CONFIRMÉ - LES DONNÉES VONT ÊTRE DÉTRUITES${NC}"
        echo ""
    else
        echo ""
          echo -e "${BLUE}ℹ️  Aucun système de fichiers détecté sur ce conteneur.${NC}"
        echo -e "${BLUE}  Le formatage est sûr (création d'un nouveau conteneur).${NC}"
        echo ""
        echo -n "Confirmer le formatage ? (o/N) "
        read -r confirm_format

          if [[ ! "$confirm_format" =~ ^[OoYy]$ ]]; then
            print_warning "Opération annulée"
            return 1
        fi
```

```
    fi

    echo ""
         echo  -n  "Appuyez  sur  ENTRÉE  pour
continuer... "
    read -r

    # Debug de la génération de clé
    local key_length=$((KEYSIZE / 8))
       print_info "Debug de la génération de
clé..."
    if ! generate_pbkdf2_debug "$password_main"
"$iterations" "$salt" "$key_length"; then
             print_error "Debug de génération
échoué"
        return 1
    fi
    echo ""

    local LOOP
    LOOP=$(losetup -f)
    if [ -z "$LOOP" ]; then
        print_error "Pas de loop device"
        return 1
    fi

    print_cmd "losetup $LOOP $CONTAINER_PATH"
       if  !  losetup  "$LOOP"  "$CONTAINER_PATH"
2>/dev/null; then
        print_error "Échec attachement loop"
        return 1
    fi
```

```
          print_info   "Création   du   mapping
$DEV_MAPPER..."
          local   cmd_crypt="cryptsetup   create
$DEV_MAPPER  $LOOP  -c  $CYPHER  -o  $offset  -s
$KEYSIZE --key-file -"
    print_cmd "$cmd_crypt"

    # Exécution réelle avec sortie binaire
              if    !    generate_pbkdf2_binary
"$password_main"      "$iterations"      "$salt"
"$key_length" | \
        cryptsetup create "$DEV_MAPPER" "$LOOP"
\
            -c "$CYPHER" \
            -o "$offset" \
            -s "$KEYSIZE" \
            --key-file - 2>/dev/null; then
        print_error "Échec création mapping"
        losetup -d "$LOOP" 2>/dev/null
        return 1
    fi

    echo ""
     print_info "Formatage en ext4 et création
du token..."
                  print_cmd    "mkfs.ext4    -F
/dev/mapper/$DEV_MAPPER"

    if ! mkfs.ext4 -F "/dev/mapper/$DEV_MAPPER"
2>/dev/null; then
        print_error "Échec du formatage"
```

```
        return 1
    fi

    mkdir -p "$MOUNT_POINT" 2>/dev/null

                print_cmd    "mount    -t    ext4
/dev/mapper/$DEV_MAPPER $MOUNT_POINT"
                 if    !    mount    -t    ext4
"/dev/mapper/$DEV_MAPPER"         "$MOUNT_POINT"
2>/dev/null; then
            print_error "Échec du montage pour
création du token"
        return 1
    fi

                  echo    "$TOKEN_CONTENT"    >
"$MOUNT_POINT/$TOKEN_FILE"
    sync

                   print_ok    "Token    créé:
$MOUNT_POINT/$TOKEN_FILE"

    print_cmd "umount $MOUNT_POINT"
    umount "$MOUNT_POINT" 2>/dev/null

    print_cmd "cryptsetup remove $DEV_MAPPER"
    cryptsetup remove "$DEV_MAPPER" 2>/dev/null

    print_cmd "losetup -d $LOOP"
    losetup -d "$LOOP" 2>/dev/null

    echo ""
```

```
    print_ok "✅ Conteneur créé et formaté avec
succès !"
    echo ""
    echo "📌 Pour monter : Option 2 du menu"
    echo "   Device: /dev/mapper/$DEV_MAPPER"
}

# Menu montage
mount_container() {
    echo ""
                                          echo
"========================================="
    echo "  🔓 MONTAGE"
                                          echo
"========================================="

    if [ ! -f "$CONTAINER_PATH" ]; then
             print_error "Fichier introuvable:
$CONTAINER_PATH"
        return 1
    fi

    echo "📝 Mot de passe PRINCIPAL :"
    read -s password_main
    echo ""

    echo "📝 Mot de passe pour le SEL :"
    read -s password_salt
    echo ""

    echo "📝 Nombre d'itérations :"
    read -r iterations
```

```
    echo "📝 Offset (octets) :"
    read -r offset

      if ! [[ "$iterations" =~ ^[0-9]+$ ]] ||
[ "$iterations" -lt 1000 ]; then
        print_error "Itérations invalides"
        return 1
    fi

       if ! [[ "$offset" =~ ^[0-9]+$ ]] ||
[ "$offset" -lt 0 ]; then
        print_error "Offset invalide"
        return 1
    fi

    local salt
    salt=$(generate_salt "$password_salt")

    echo ""
    echo "📌 RÉSUMÉ :"
    echo "   - Itérations: $iterations"
    echo "   - Offset: $offset octets"
    echo "   - Sel: $salt"
         echo "       - Mapping: $DEV_MAPPER
(/dev/mapper/$DEV_MAPPER)"
    echo ""
         echo -n "Appuyez sur ENTRÉE pour
continuer... "
    read -r

    # Debug de la génération de clé
```

```
    local key_length=$((KEYSIZE / 8))
       print_info "Debug de la génération de
clé..."
    if ! generate_pbkdf2_debug "$password_main"
"$iterations" "$salt" "$key_length"; then
             print_error "Debug de génération
échoué"
        return 1
    fi
    echo ""

    local LOOP
    LOOP=$(losetup -f)
    if [ -z "$LOOP" ]; then
        print_error "Pas de loop device"
        return 1
    fi

    print_cmd "losetup $LOOP $CONTAINER_PATH"
      if ! losetup "$LOOP" "$CONTAINER_PATH"
2>/dev/null; then
        print_error "Échec attachement loop"
        return 1
    fi

          print_info "Création du mapping
$DEV_MAPPER..."
          local cmd_crypt="cryptsetup create
$DEV_MAPPER $LOOP -c $CYPHER -o $offset -s
$KEYSIZE --key-file -"
    print_cmd "$cmd_crypt"
```

```
    # Exécution réelle avec sortie binaire
                if    !    generate_pbkdf2_binary
"$password_main"      "$iterations"      "$salt"
"$key_length" | \
         cryptsetup create "$DEV_MAPPER" "$LOOP"
\
            -c "$CYPHER" \
            -o "$offset" \
            -s "$KEYSIZE" \
            --key-file - 2>/dev/null; then
        print_error "Échec création mapping"
        losetup -d "$LOOP" 2>/dev/null
        return 1
    fi

    mkdir -p "$MOUNT_POINT" 2>/dev/null

             print_cmd   "mount   -t   ext4   -o
noatime,nodev,nosuid     /dev/mapper/$DEV_MAPPER
$MOUNT_POINT"
     if ! mount -t ext4 -o noatime,nodev,nosuid
"/dev/mapper/$DEV_MAPPER"         "$MOUNT_POINT"
2>/dev/null; then
        print_error "Échec montage"
               cryptsetup  remove  "$DEV_MAPPER"
2>/dev/null
        losetup -d "$LOOP" 2>/dev/null
        return 1
    fi

    # Vérification du token
    if [ -f "$MOUNT_POINT/$TOKEN_FILE" ]; then
```

```
        local content
                                      content=$(cat
"$MOUNT_POINT/$TOKEN_FILE" 2>/dev/null)
          if [ "$content" = "$TOKEN_CONTENT" ];
then
                print_ok "✅ Token vérifié avec
succès"
        else
                        print_warning  "⚠   Token
incorrect !"
                       print_info  "     Attendu:
$TOKEN_CONTENT"
             print_info "   Trouvé:  $content"
        fi
    else
               print_warning  "⚠   Token  absent
(fallback sur superblock)"
                        if   check_ext4_superblock
"/dev/mapper/$DEV_MAPPER"; then
                  print_ok  "✅ Superblock  ext4
valide"
        else
               print_warning "⚠  Superblock non
détecté"
        fi
    fi

    echo ""
          print_ok   "✅   Conteneur   monté   sur
$MOUNT_POINT"
    echo "   Device: /dev/mapper/$DEV_MAPPER"
    df -h "$MOUNT_POINT"
```

```
    cryptsetup status "$DEV_MAPPER" 2>/dev/null
|| true
}

# Menu démontage
unmount_container() {
                                                echo
"============================================="
    echo "  🔒 DÉMONTAGE"
                                                echo
"============================================="

    local mapper_name="$DEV_MAPPER"

                if    mount    |    grep    -q
"/dev/mapper/$mapper_name"; then
        print_cmd "umount $MOUNT_POINT"
        sync
           umount "$MOUNT_POINT" 2>/dev/null ||
umount -l "$MOUNT_POINT" 2>/dev/null
        print_ok "Démonté"
    fi

      if dmsetup ls | grep -q "$mapper_name";
then
                print_cmd  "cryptsetup  remove
$mapper_name"
              cryptsetup  remove  "$mapper_name"
2>/dev/null  ||  dmsetup  remove  "$mapper_name"
2>/dev/null
        print_ok "Mapping supprimé"
    fi
```

```
    local loops
       loops=$(losetup  -a  2>/dev/null  |  grep
"$CONTAINER_PATH" | cut -d: -f1)
    if [ -n "$loops" ]; then
        for loop in $loops; do
            print_cmd "losetup -d $loop"
            losetup -d "$loop" 2>/dev/null
        done
        print_ok "Loop devices nettoyés"
    fi

    echo "✅ Conteneur démonté"
}

# Menu principal
main() {
    if [ "$EUID" -ne 0 ]; then
        print_error "Root requis"
        exit 1
    fi

    echo ""
                                                echo
"============================================="
    echo -e "  ${BOLD}🔐 PBKDF2 + PLAIN${NC}"
                                                echo
"============================================="
    echo "  Conteneur: $CONTAINER_PATH"
    echo "  Point de montage: $MOUNT_POINT"
    echo "  Mapping: /dev/mapper/$DEV_MAPPER"
```

```
      echo "   Chiffrement: $CYPHER - $KEYSIZE
bits"
      echo "   🔒 100% PIPE - Aucun fichier
temporaire"
      echo -e "   ${YELLOW}🐛 Mode débogage:
ACTIVÉ${NC}"
                                              echo
"========================================="
    echo "  1: Créer (formate + token)"
    echo "  2: Monter"
    echo "  3: Démonter"
     echo -e "  ${YELLOW}4: 🔬 Mode diagnostic$
{NC}"
    echo "  5: Quitter"
                                              echo
"========================================="
    echo -n "Choix : "
    read -r choix

    case "$choix" in
        1) create_container ;;
        2) mount_container ;;
        3) unmount_container ;;
        4) diagnostic_mode ;;
        5) echo "Au revoir !" ; exit 0 ;;
         *) print_error "Choix invalide" ; exit
1 ;;
    esac
}

main "$@"
```

## Conclusion

`Cryptsetup` est un outil GNU/Linux particulièrement puissant pour chiffrer des conteneurs (fichiers), des partitions, des disques, y compris en RAID.

À noter que le chiffrement est « transparent » côté performances de la machine, et ne pose donc pas de problèmes particuliers, même sur des matériels anciens.

Cela étant, si le script ici proposé est bien optimisé pour les conteneurs, il pose une nouvelle question avec la montée en puissance des machines et de l'informatique quantique.

Une utilisation classique de `cryptsetup` est le chiffrement de sa partition ***/home*** pour protéger ses données en cas de vol.

Pour chiffrer la partition racine en revanche, la méthode ici proposée n'est pas complètement adaptée pour deux raisons :

➔ GRUB a besoin d'accéder à un ***/boot*** visible pour charger le noyau et le disque virtuel de démarrage (*RAMFS*), ce qui nécessite donc de prévoir une partition non chiffrée dédiée au ***/boot***.

➔ `cryptsetup` utilise les fichiers ***/etc/crypttab*** et ***/etc/fstab*** pour ouvrir les conteneurs chiffrés au démarrage de la machine. Là encore, si ***/etc*** n'est pas accessible au démarrage, le montage ne fonctionnera pas...

Pour aller plus loin - et là on touche aux limitations réelles de nos distributions - si GNU/Linux demande bien le mot de passe au démarrage de la machine quand il détecte des partitions chiffrées dans **/etc/crypttab**, il n'y a pas aujourd'hui de procédure standardisée pour intégrer le sel et les itérations PBKDF2 via `openssl`...

Cette étape qui paraît pourtant essentielle n'est tout simplement pas prévue !

A part Debian, qui propose l'option *check*=un script externe à créer, dans ***/etc/crypttab***, les autres distributions semblent ne pas avoir compris l'intérêt de pouvoir personnaliser complètement son espace chiffré sans stocker la clé.

On voudrait imposer **LUKS** partout, « flinguer » l'intérêt du mode **PLAIN**, et simplifier les attaques par dictionnaire, on ne s'y prendrait pas autrement...

Maintenant si vous cherchez un outil permettant de gérer des disques durs de sauvegarde chiffrés accessibles depuis plusieurs systèmes d'exploitation, `veracrypt` est un outil intermédiaire en mode graphique plutôt intéressant, fonctionnant sous les 3 OS majeurs.