Chiffrement de la Master Key de Cryptsetup à l'aide d'une PKI : pourquoi et comment ?

Levent Demir, Jean-Michel Tenkes, Vincent Roca

Abstract— Cryptsetup est un outil Linux permettant de chiffrer un conteneur qui travaille en liaison avec LUKS qui gère les clés de chiffrement. Le but de ce travail est de modifier Cryptsetup/LUKS afin de protéger la clé principale (appelée Master Key (MK) dans ce document) via un chiffrement clé publique/privée. Nous décrivons le fonctionnement de Cryptsetup, sa modification, présentons une preuve de concept et les bénéfices apportés.

1. Introduction

1.1 Fonctionnement de Cryptsetup et gestion de la clé de chiffrement MK

Sous Linux, l'outil Cryptsetup [1] offre un moyen simple et commode pour chiffrer un « disque complet » (ou « full disk encryption ») et donc protéger tous les fichiers inclus. Ce « disque complet » peut toutefois être une partition, une clé USB, ou un conteneur. Dans cet article nous utiliserons le terme conteneur (ou container en anglais) pour désigner ce « disque complet ».

L'outil Cryptsetup s'appuie sur le module noyau dm-crypt [2] [3], activé dans toutes les distributions Linux récentes. De plus Cryptsetup fonctionne étroitement avec la surcouche LUKS [4] (Linux Unified Key Setup) qui permet de gérer les clés de chiffrement de façon très souple (ainsi un conteneur peut être partagé par 8 utilisateurs). Cette souplesse est possible car deux clés principales [5] sont utilisées :

- Key Encryption Key (KEK), ou Derived Key sur le schéma : cette clé permet de chiffrer la MK ;
- Master Key (MK) : cette clé permet de chiffrer les données stockées dans le conteneur. Elle est parfois appelée Data Encryption Key (DEK).

La Figure 1 montre le processus complet de gestion des clés dans LUKS. Durant la phase de chiffrement de la MK :

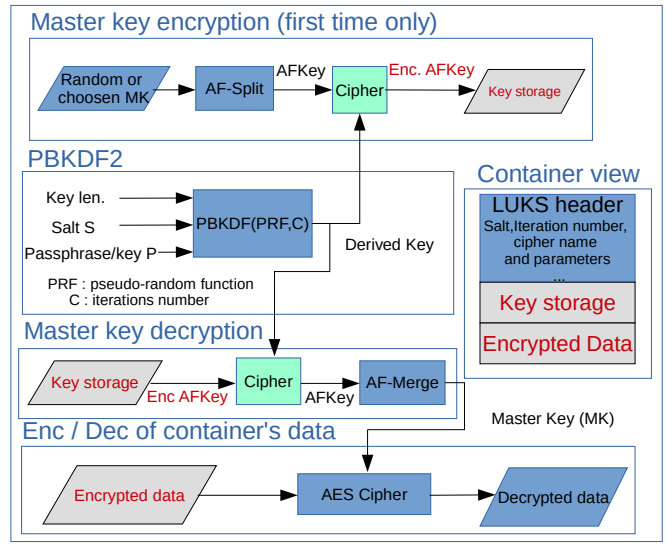


Figure 1: gestion de la Master Key dans LUKS.

- Lors de l'initialisation du conteneur l'utilisateur entre une passhprase qui est dérivée à l'aide de PBKDF2 pour obtenir une clé dérivée ;
- Une MK est ensuite soit générée aléatoirement soit spécifiée par l'utilisateur ;
- La MK est « diffusée » grâce à la fonction AF-Split [5] (AFKey sur le schéma) puis chiffrée avec la clé dérivée. Elle est alors stockée dans le conteneur, après l'entête LUKS.

(NB : la fonction AF-Split/Merge offre une protection anti-forensic. Pour cela la clé de 128 bits est étendue sur plusieurs kiloctets, tous les blocs de 16 octets

- *L.Demir is with Inria and Incas ITsec, France. Email: levent.demir@inria.fr*
- *J-M Tenkes is with Incas ITsec, France. Email: jm.tenkes@incas-itsec.com*
- *V.R is with Inria, France. Email: vincent.roca@inria.fr*

étant dépendants les uns des autres. Ceci augmente la probabilité de supprimer réellement la clé puisqu'il suffit de retirer quelques octets pour empêcher définitivement la récupération de la clé.)

Durant la phase de déchiffrement de la MK :

- L'utilisateur entre sa passphrase ce qui permet de re-générer la clé dérivée ;
- La clé étendue (AFKey) est déchiffrée ;
- La fonction AF-Merge récupère la MK.

1.2 Motivations

Le but de ce travail est de modifier le processus de chiffrement de la MK afin d'utiliser une paire clé publique/privée. Ainsi, un utilisateur souhaitant créer un conteneur chiffrera la MK avec sa clé publique, ou alors une entité de confiance créera le conteneur pour l'utilisateur et chiffrera la MK avec sa clé publique. Lorsque l'utilisateur souhaitera accéder au conteneur, il devra charger sa clé privée (à travers un canal sécurisé) pour déchiffrer la MK puis le contenu du conteneur.

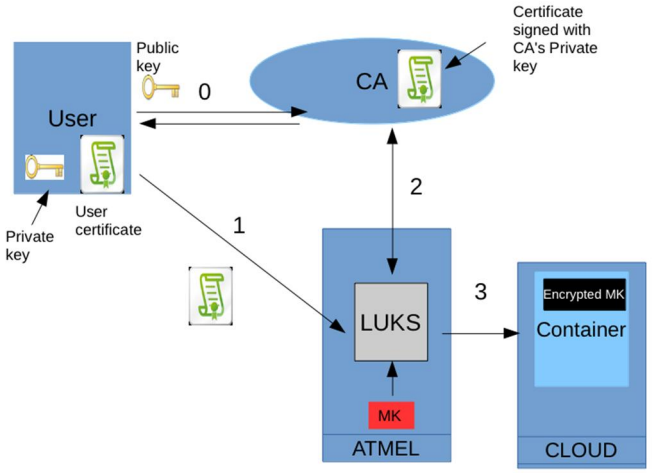
Il existe une implémentation pour LUKS utilisant un TPM (Trusted Platform Module) [6]. Ce dispositif permet le stockage de la MK en zone mémoire protégée mais ne permet pas d'effectuer des opréations cryptographiques sécurisées comme le chiffrement matériel (RSA, AES). C'est pourquoi son utilisation n'est pas comparable à un HSM intégrant l'espace mémoire protégé et les opérations cryptographiques matérielles associées. Une autre implémentation [7] basée sur eCyptfs permet d'obtenir un système de fichiers chiffrés utilisant une PKI intégré au niveau du noyau Linux. Chaque fichier et nom de fichier sont chiffrés à l'aide d'une clé unique et d'un vecteur d'initialisation comprenant une valeur TWEAK de 64 bits et un offset de 64 bits avec le mode XTS-AES. Un fichier annexe ACL contient les metadata. Contrairement à LUKS, le chiffrement ne porte pas sur le disque complet.

L'évolution proposée apporte des bénéfices qui sont déterminants dans le cas où existe un composant matériel offrant un haut niveau de sécurité, ou HSM [8] (Hardware Security Module), une oganisation qui gère une PKI [9] (Public Key Infrastructure) pour ses membres, et un service de stockage distant (par ex. Cloud). Dans ce cas le HSM peut créer un conteneur chiffré, par exemple stocké sur le Cloud, pour le compte de l'utilisateur, en utilisant la clé publique de cet utilisateur. Dans un autre cas d'usage, on pourra associer un utilisateur à un groupe dédié auquel on accorder certains droits d'accès et un certificat. Faisons l'hypothèse (raisonnable) que l'infrastructure PKI est sécurisée. La MK en clair ne quitte alors jamais le HSM, lui même considéré comme étant de confiance. La compromission du poste client n'affecte pas la sécurité du service.

2 PROPOSITION

Nous modifions Cryptsetup afin d'ajouter le chiffrement de la MK à l'aide d'une PKI : l'utilisateur sera identifié grâce à son certificat, la MK sera chiffrée à l'aide de sa clé publique et seul cet utilisateur pourra accéder au conteneur grace à sa clé privée.

Le schéma ci-dessous illustre la première étape d'initialisation du conteneur. Dans cet exemple une carte Atmel [10] joue le rôle de HSM (même si elle ne peut offrir le même niveau de confiance) et le conteneur est créé dans un Cloud.



Plus précisément :

- L'utilisateur génère une paire de clés publique/privée, et envoie une demande de certificat à son authorité CA (étape 0). Il obtient en retour un certificat signé par le CA contenant sa clé publique.
- L'utilisateur envoie son certificat au HSM (étape 1).

- Le HSM vérifie le certificat en récupérant la clé publique du CA et en recalculant la signature (étape 2).
- Le HSM crée une MK aléatoire au moyen d'un générateur d'aléas cryptographique et la chiffre avec la clé publique de l'utilisateur contenue dans son certificat. Le contenur protégé est stockée dans l'en-tête du conteneur (étape 3).

Le second schéma illustre l'accès au conteneur :

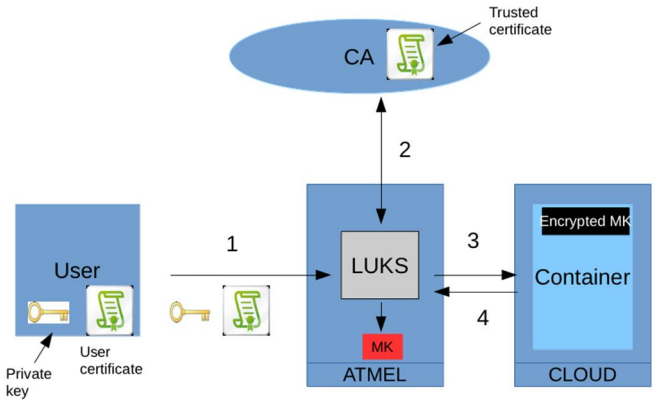


Figure 2 : Accès au conteneur avec la nouvelle archi-

Cet exemple met en évidence les bénéfices de l'approche :

- Toutes les opérations sensibles et la manipulation de la MK sont gérées dans un HSM. La compromission du poste client n'affecte pas (ou peu) la sécurité du système (par ex. l'analyse de la RAM du poste client ne peut pas compromettre la MK) ;
- Grâce au HSM, la MK peut être générée à l'aide d'un générateur d'aléas cryptographique ce qui est essentiel ;
- Elle réutilise la PKI de l'organisation afin de gérer l'accès au contenu, sans coût supplémentaire. La sécurité est bien évidemment un prérequis, mais ceci est intrinsèque à toute PKI.

3 VALIDATION FONCTIONNELLE

Montrons que l'approche est valide au moyen de tests menés sur la carte ATMEL directement, sans poste client. Pour rappel, le but de cette validation est de montrer que les opérations sensibles sur la MK peuvent être effectuées via un module externe. Dans notre cas ce module est une carte ATMEL mais à terme le module sera un composant cryptographique certifié (HSM).

(NB : Dans une configuration complète avec poste client, une fois le conteneur ouvert, il suffira de permettre un montage distant sécurisé de son contenu, par exemple via sshfs.)

3.1 Création du conteneur avec cipher-null

Nous créons tout d'abord le conteneur :

```
#!/bin/sh

CONTAINER="myMiniContainer"

sudo dd if=/dev/zero of=$CONTAINER bs=1M
count=50
```

La commande cryptsetup prend pour paramètres :

- luksFormat : pour initialiser le conteneur ;
- -c cipher_null-ecb : pour ne pas chiffrer les données, la MK chiffrée étant stockée telle que dans le conteneur ;
- $CONTAINER : spécifie le conteneur ;
- my_key : contient la clé utilisateur (qui peut aussi être une passphrase) ;
- --master-key-file : correspond à la MK imposée par l'utilisateur

Utiliser cipher-null permet dans ce cas de pouvoir vérifier que la MK chiffrée est stockée dans le conteneur, autrement l'entête du conteneur est lui même chiffré, y compris la MK chiffrée. Ceci est fait dans le but de valider l'approche seulement.

Après exécution du script nous analysons son contenu :

```
00000000  4c 55 4b 53 ba be 00 01  63 69 70 68 65 72 5f 6e  |LUKS....cipher_n|
00000010  75 6c 6c 00 00 00 00 00  00 00 00 00 00 00 00 00  |ull.............|
00000020  00 00 00 00 00 00 00 00  65 63 62 00 00 00 00 00  |........ecb.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  73 68 61 31 00 00 00 00  |........sha1....|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 10 00 00 00 00 20  |............... |
00000070  d3 e6 0f 8a bb ea 77 25  49 1f 44 6f ff e5 2a 0d  |......w%I.Do..*.|
00000080  54 f3 71 9c da 85 50 1e  bf 71 5a e4 36 6c b7 7c  |T.q...P..qZ.6l.||
00000090  a0 6a d1 23 4f 8a 3b 8c  be b2 59 a7 56 81 57 68  |.j.#O.;...Y.V.Wh|
000000a0  51 2a d2 84 00 01 24 f8  35 63 34 35 32 37 61 39  |Q*....$.5c4527a9|
000000b0  2d 66 36 37 37 2d 34 30  38 38 2d 39 61 65 62 2d  |-f677-4088-9aeb-|
000000c0  39 32 38 66 30 30 62 35  37 32 62 32 00 00 00 00  |928f00b572b2....|
000000d0  00 ac 71 f3 00 04 a6 79  53 dd ed 30 f4 88 9c a2  |..q....yS..0....|
000000e0  c9 1e 0c 4b 28 47 41 37  b8 8f 49 27 a0 2f db 8e  |...K(GA7..I'./..|
000000f0  21 93 46 18 a8 fe 00 75  00 00 00 08 00 00 0f a0  |!.F....u........|
00000100  00 00 de ad 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 01 08 00 00 0f a0  |................|
00000130  00 00 de ad 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Nous remarquons le cipher null et le type de container LUKS. Puis quelques lignes plus bas nous avons le chiffré de la MK :

```
00001000  28 65 6e 63 2d 76 61 6c  20 0a 20 28 72 73 61 20  |(enc-val . (rsa |
00001010  0a 20 20 28 61 20 23 31  41 42 33 34 43 36 45 44  |.  (a #1AB34C6ED|
00001020  44 38 46 44 32 45 31 39  45 35 31 34 31 32 36 30  |D8FD2E19E5141260|
00001030  33 38 37 39 36 30 32 33  42 42 38 44 37 32 45 46  |38796023BB8D72EF|
00001040  42 41 44 44 43 34 46 36  30 36 38 36 34 36 31 38  |BADDC4F606864618|
00001050  39 41 42 43 35 33 33 33  37 34 33 45 38 35 36 31  |9ABC5333743E8561|
00001060  44 38 46 36 41 33 33 46  44 35 31 33 39 35 33 34  |D8F6A33FD5139534|
00001070  34 42 39 33 33 34 39 36  39 32 30 43 35 34 46 35  |4B933496920C54F5|
00001080  39 43 32 33 44 31 43 31  36 39 43 39 30 37 31 41  |9C23D1C169C9071A|
00001090  41 35 46 46 30 44 44 44  46 31 35 31 44 37 39 41  |A5FF0DDDF151D79A|
000010a0  31 33 37 31 45 42 37 45  43 42 45 34 46 39 34 45  |1371EB7ECBE4F94E|
000010b0  31 36 34 42 39 33 42 43  45 31 43 42 34 45 45 39  |164B93BCE1CB4EE9|
000010c0  42 34 31 31 31 39 31 38  37 36 39 31 46 36 38 45  |B41119187691F68E|
000010d0  36 39 38 31 46 41 37 38  37 33 41 33 33 45 31 32  |6981FA7873A33E12|
000010e0  35 30 34 45 34 39 37 36  39 38 34 33 43 45 44 34  |504E49769843CED4|
000010f0  42 37 42 35 41 34 43 32  42 42 33 33 35 42 38 43  |B7B5A4C2BB335B8C|
00001100  38 44 32 41 36 33 39 41  35 30 36 41 44 39 35 32  |8D2A639A506AD952|
00001110  41 35 33 43 46 31 36 30  42 33 33 35 31 33 41 45  |A53CF160B33513AE|
00001120  32 30 31 32 46 46 36 43  43 43 39 36 34 36 31 32  |2012FF6CCC964612|
00001130  42 38 36 46 36 39 30 41  44 35 35 31 34 42 31 46  |B86F690AD5514B1F|
00001140  42 37 44 37 41 32 44 37  41 36 41 35 34 34 35 37  |B7D7A2D7A6A54457|
00001150  37 33 31 46 41 31 32 45  35 37 37 39 35 42 38 30  |731FA12E57795B80|
00001160  39 41 36 32 44 32 33 30  46 32 39 37 45 38 33 32  |9A62D230F297E832|
00001170  33 42 32 35 42 31 44 33  34 32 33 38 31 39 45 38  |3B25B1D3423819E8|
00001180  30 30 41 33 42 45 33 32  30 45 43 43 43 46 38 38  |00A3BE320ECCCF88|
00001190  36 41 39 39 38 33 32 37  46 39 45 30 35 43 36 39  |6A998327F9E05C69|
000011a0  36 38 44 46 45 45 35 33  33 35 30 46 38 31 35 45  |68DFEE53350F815E|
000011b0  45 46 34 46 42 46 41 32  45 46 42 31 35 44 42 44  |EF4FBFA2EFB15DBD|
000011c0  30 37 37 37 31 38 33 33  31 34 46 34 41 35 44 31  |0777183314F4A5D1|
000011d0  43 33 36 35 33 41 34 43  43 42 46 42 38 46 31 30  |C3653A4CCBFB8F10|
000011e0  35 45 36 35 36 36 39 43  44 33 34 46 41 37 39 44  |5E65669CD34FA79D|
000011f0  46 33 36 30 34 43 45 31  39 33 32 32 46 34 42 43  |F3604CE19322F4BC|
00001200  30 37 30 30 31 35 33 38  39 43 35 39 37 46 32 33  |070015389C597F23|
00001210  42 39 43 46 46 38 35 23  29 0a 20 20 29 0a 20 29  |B9CFF85#).  ). )|
00001220  0a 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Ainsi la MK est protégée par un chiffrement RSA, ce processus peut être automatisé par la suite et intégrer dans une infrastructure gérant les certificats.

3.2 Ajout de données

Ouvrons le conteneur dans un device virtuel « coffre ». La première fois, nous créons un systeme de fichier

```
#!/bin/sh

CONTAINER="myMiniContainer"

sudo cryptsetup --debug luksOpen --key-file
../my_key $CONTAINER coffre

sudo mkfs.ext4 /dev/mapper/coffre

sudo mount /dev/mapper/coffre /mnt/target
```

ext4fs (ou autre) et nous montons le coffre dans un dossier local :

Ajoutons un simple fichier de test, par exemple :

```
$ more test.txt

aaaaaabbbbbbbbb
```

Nous observons à nouveau le contenu du conteneur pour constater que le fichier est présent (ceci est

```
00a088a0  c7 00 00 00 d7 00 00 00  e2 05 00 00 3e 1f 00 08  |............>...|
00a088b0  00 00 01 00 00 00 00 00  00 00 00 00 00 08 71 c3  |..............q.|
00a088c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00a08400  61 61 61 61 61 61 62 62  62 62 62 62 62 62 62 0a  |aaaaaabbbbbbbbb.|
00a08410  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
01200400  c0 3b 39 98 00 00 00 04  00 00 00 00 00 00 04 00  |.;9.............|
01200410  00 00 10 00 00 00 00 01  00 00 00 02 00 00 00 01  |................|
```

possible car nous avons choisi cipher-null). Cryptsetup fonctionne !

3.3 Conteneur avec un cipher AES-XTS

Recommençons les étapes précédentes en utilisant cette fois ci un cipher AES en mode XTS comme il se doit : création du conteneur, initialisation avec AES, ajout du même fichier.

L'entête nous donne des informations sur le chiffrement :

```
          0000000  4c 55 4b 53 ba be 00 01  61 65 73 00 00 00 00 00  |LUKS....aes.....|
1         0000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
2         0000020  00 00 00 00 00 00 00 00  78 74 73 2d 70 6c 61 69  |........xts-plai|
3         0000030  6e 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |n...............|
4         0000040  00 00 00 00 00 00 00 00  73 68 61 31 00 00 00 00  |........sha1....|
5         0000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
6         0000060  00 00 00 00 00 00 00 00  00 00 10 00 00 00 00 20  |............... |
7         0000070  a1 2e 2b 21 6a a6 4d 26  6b b9 3f 2a 98 4b ec 9b  |..+!j.M&k.?*.K..|
8         0000080  75 8f 7b b5 fe 5d 56 1c  93 e9 ce bf 3a eb dc 80  |u.{..]V.....:...|
```

Nous retrouvons la mention AES. Cherchons maintenant la chaine de caractère correspondant au contenu du fichier. Comme attendu, le pattern « aaaaaa » n'est pas trouvé dans le conteneur.

4 Conclusion

Cet article discute de l'intérêt que peut avoir l'usage d'une infrastructure PKI au sein du service Cryptsetup de Linux. Nous avons expliqué la proposition, ses motivations, et présenté une preuve de concept.

La solution proposée sera pertinente dans certaines situations, par exemple dans le cas où les traitements cryptographiques sont effectués dans un HSM : la totalité des opérations sensibles est alors effectuée dans un environnement de confiance.

Nous avons illustré notre proposition au moyen d'un service de stockage distant, le client y accédant au moyen du HSM, après avoir présenté son certificat et sa clé privée au travers d'un canal sécurisé. D'autres solutions sont possibles pour éviter l'envoi de la clé privée. Par exemple un CA interne à l'organisation pourrait stocker la paire de clés publique/privée de tous les utilisateurs. Un canal sécurisé entre HSM et CA permettrait alors la récupération de ces clés sans que l'utilisateur n'ait à la transmettre. Ce dernier serait juste authentifié via un mécanisme adapté (par ex. à l'aide d'un token). D'autres cas d'usages faisant usage d'un HSM pourront bénéficier de cette évolution.

5 REFERENCES

[1] «Cryptsetup,» [En ligne]. Available: https://gitlab.com/cryptsetup/cryptsetup.

[2] S. C., «dm-crypt: a device-mapper crypto targe,» 2014. [En ligne]. Available: http://www. saout. de/misc/dm-crypt.

[3] F. C., «Hard disk encryption with DM-Crypt, LUKS, and cryptsetup,» *ISSUE,* vol. 61, pp. 65-71, 2005.

[4] C. Fruhwirth, «LUKS On-Disk Format Specification Version 1.2,» 2011.

[5] F. C., «TKS1-An anti-forensic, two level, and iterated key setup scheme.,» 2004.

[6] S. C. Kühn U, «User-friendly and secure tpm-based hard disk key management. In Future of Trust in Computing,» pp. 171-177, 2009.

[7] K. S. Rawat US, «ECFS: An Enterprise-Class Cryptographic File System for Linux,» *International Journal of Information Security and Privacy (IJISP),* vol. 6, n° %12, pp. 53-63, 2012.

[8] «HSM wikipedia,» [En ligne]. Available: https://en.wikipedia.org/wiki/Hardware_security_module.

[9] T. P. Russ Housley, Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure (1st ed.), New York: John Wiley & Sons, 2001.

[10] ATMEL, «SAMA5D36-EK,» [En ligne]. Available: http://www.atmel.com/tools/SAMA5D36-EK.aspx.