Rapport de TP :
Chiffrement Symétrique sous Linux

Module : Introduction a la Cybersécurité

Présenté par : **Ali FOUGHALI**
Présenté par : **Lucas RANDRIAMANANTENA**

16 janvier 2026

Table des matières

1 Introduction et Objectifs

Ce travail pratique a pour objectif de comprendre les mécanismes de chiffrement et déchiffrement symétrique via l'outil `openssl`. Nous explorerons la génération de clés, l'impact des longueurs de clés et la dérivation via mot de passe.

Le chiffrement symétrique utilise la même clé pour chiffrer et déchiffrer les données, ce qui le rend particulièrement efficace pour sécuriser de grandes quantités d'information. Toutefois, la gestion sécurisée de cette clé partagée constitue un défi majeur que nous aborderons dans ce rapport.

2 Partie I : Chiffrement symétrique avec OpenSSL

2.1 Vérification de l'environnement

Avant de débuter, nous vérifions la version d'OpenSSL installée sur la machine Linux afin de nous assurer de la disponibilité des fonctionnalités nécessaires.

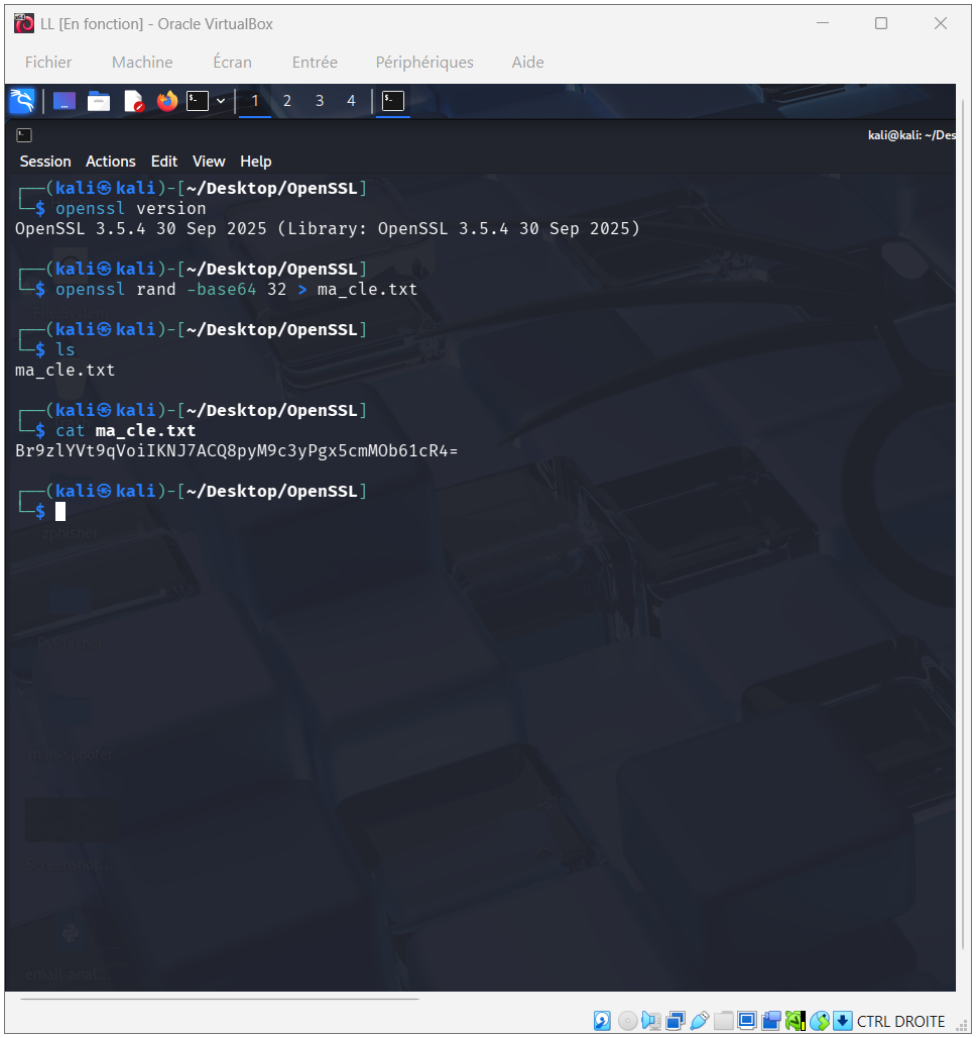

FIGURE 1 – Version d'OpenSSL installée

```
openssl version
```

Cette étape est cruciale car certaines options de sécurité avancées, comme `-pbkdf2`, ne sont disponibles que dans les versions récentes d'OpenSSL.

2.2 Génération de la clé

Nous générons une clé de 32 octets (256 bits) encodée en Base64. Cette longueur correspond à la taille recommandée pour l'algorithme AES-256.

```
openssl rand -base64 32 > ma_cle.txt
```

Cette commande utilise l'option `rand` pour générer des données aléatoires cryptographiquement sûres et `-base64` pour encoder le résultat dans un format textuel facilement manipulable.

Points importants :

— La qualité de l'aléa est essentielle pour la sécurité de la clé
— L'encodage Base64 permet de stocker des données binaires dans un fichier texte
— La longueur de 32 octets correspond à 256 bits de sécurité

2.3 Chiffrement et Déchiffrement

Nous procédons au chiffrement du fichier `file1.txt` avec l'algorithme AES-256-CBC (Advanced Encryption Standard en mode Cipher Block Chaining).

```
openssl enc -aes-256-cbc -salt -in file1.txt -out file1_encrypted.
   txt -pass file:ma_cle.txt
```

Analyse du Warning : OpenSSL affiche un avertissement concernant la dérivation de clé (*deprecated key derivation*). Cela signifie que l'ancienne méthode par défaut est jugée peu sûre face aux attaques par force brute modernes. Il est fortement recommandé d'utiliser l'option `-pbkdf2` pour renforcer la sécurité de la dérivation de clé.

Déchiffrement :

```
openssl enc -aes-256-cbc -d -in file1_encrypted.txt -out
   file1_decrypted.txt -pass file:ma_cle.txt
```

L'option `-d` indique qu'il s'agit d'une opération de déchiffrement.

3 Partie II : Utilisation de différentes longueurs de clé

Dans cette partie, nous comparons le chiffrement d'un même texte avec des clés de longueurs différentes afin de comprendre l'impact sur la taille du fichier chiffré et sur la sécurité.

[Insérer ici : Capture d'écran 152308 montrant les fichiers de 80 octets]

3.1 Analyse de la taille des fichiers chiffrés

Pourquoi les tailles sont-elles identiques (80 octets) ? Dans cette partie nous avons eu 2 versions, l'un a réaliser des clés en fonction des octets et l'autre en fonction des bits , ce qui a donné des résultats différents Bien que les clés changent de longueur, la taille du fichier chiffré reste identique pour plusieurs raisons :

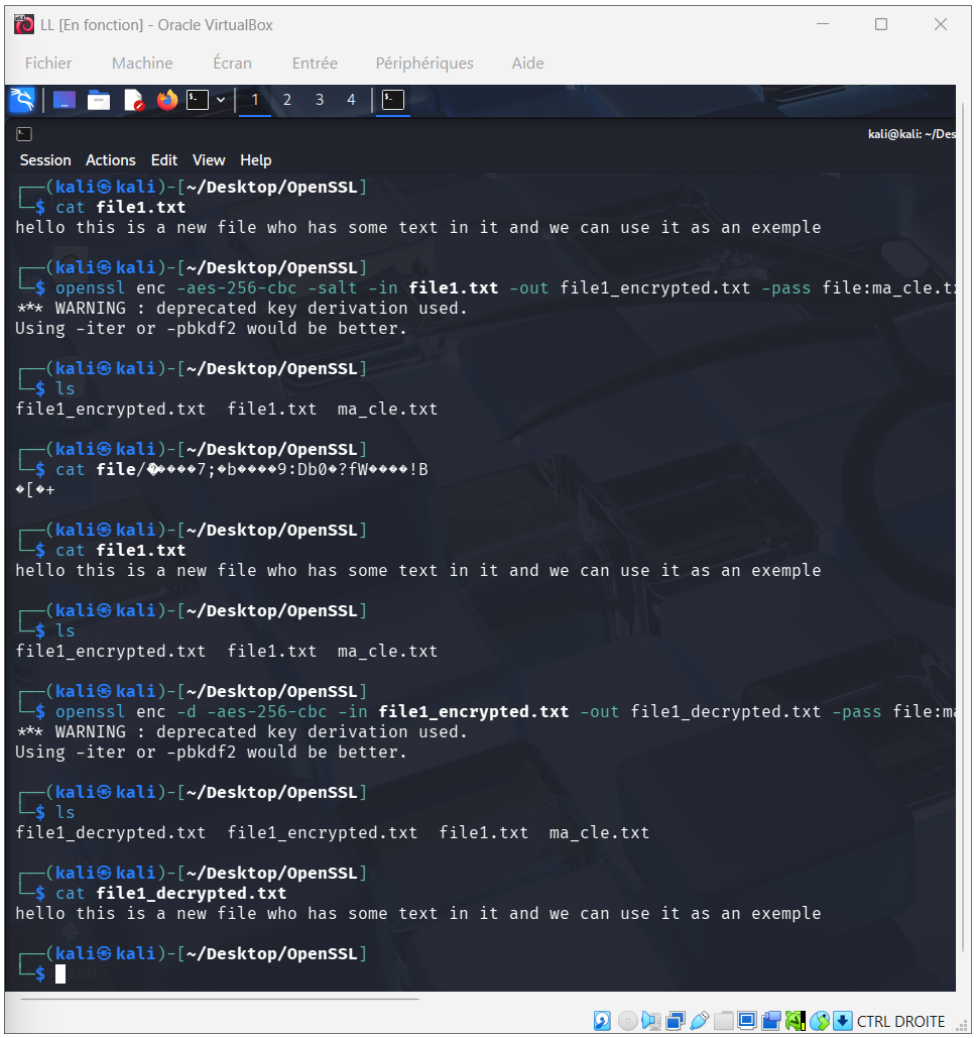

FIGURE 2 –

1. **Chiffrement par blocs :** AES est un algorithme de chiffrement par blocs qui traite les données par segments de 16 octets, quelle que soit la longueur de la clé utilisée.
2. **Padding :** Si le texte clair n'est pas un multiple exact de 16 octets, il sera complété par du *padding* (rembourrage) pour atteindre cette taille.
3. **En-tête fixe :** L'en-tête "Salted__" occupe une taille fixe de 16 octets (8 octets pour la chaîne magique, 8 octets pour le sel).
4. **Conclusion :** La taille du fichier chiffré dépend principalement de la taille du texte source et non de la longueur de la clé.

3.2 Relation entre longueur de clé et sécurité

Impact sur la sécurité :

Plus la clé est longue, plus le nombre de combinaisons possibles augmente de manière exponentielle :

— **AES-128 :** 2^{128} combinaisons possibles (environ 3.4×10^{38})
— **AES-256 :** 2^{256} combinaisons possibles (environ 1.1×10^{77})

FIGURE 3 –

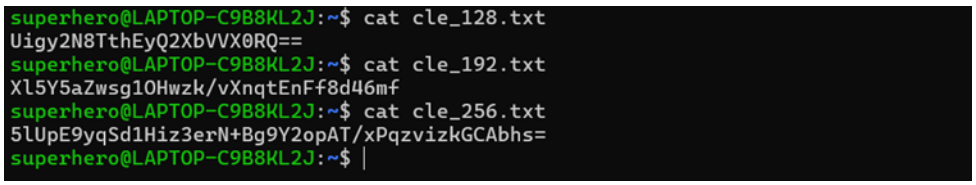

FIGURE 4 –

Avec la puissance de calcul actuelle, même les supercalculateurs ne pourraient pas tester toutes les combinaisons d'une clé AES-256 en un temps raisonnable, ce qui rend les attaques par force brute pratiquement impossibles.

L'AES est un chiffrement par blocs, il découpe le fichier par morceau de taille fixe. Donc on en conclut que la sécurité augmente (126->192->256) Les fichiers ont tous une taille de 64 octets Les tailles sont identiques pourquoi ? c'est dû aux mécanismes de l'algorithme AES. 1) Le chiffrement par blocs : si le message ne remplit pas exactement le bloc, OpenSSL ajoute du Padding donc il remplit pour compléter le bloc 2) La structure d'OpenSSL : Les fichiers chiffré avec l'option -salt commencent par un en tete de 16 octets On en conclut que la longueur de clé quel qu'elle soit, n'influence pas la taille du fichier final. Donc qu'est ce qui l'influence ? La structure des blocs et la longueur du message clair.

Avantages et inconvénients :

128 bits : Suffisante pour usage courant, taille de fichier identique, très rapide (performance) 192 bits : Niveau de sécurité intermédiaire, taille de fichier identique, Equilibré (performance) 256 bits : Niveau de sécurité maximale, taille de fichier identique aux autres, plus lent (performance) car il consomme plus de ressources processeur (CPU)

La relation entre la longueur de clé et la sécurité du chiffrement symétrique :

Concernant l'AES, il n'existe pas de méthode mathématique connu pour contourner l'algorithme, la seule solution est de tester toutes les clés possibles de manière brute.

Comme nous l'avons mentionné plus en haut, plus on rajoute de bits à la longueur d'une clé, plus on augmente la sécurité car on augmente le nombre de combinaisons totales

Donc la sécurité de chiffrement symétrique est proportionnelle aux nombres de combinaisons. Plus la clé est longue et plus la difficulté de l'attaque est multipliée.

4 Partie III : Chiffrement avec mot de passe (PBKDF2)

L'objectif de cette partie est d'utiliser une fonction de dérivation de clé sécurisée (PBKDF2 - Password-Based Key Derivation Function 2) pour transformer un mot de passe humain en clé cryptographique robuste.

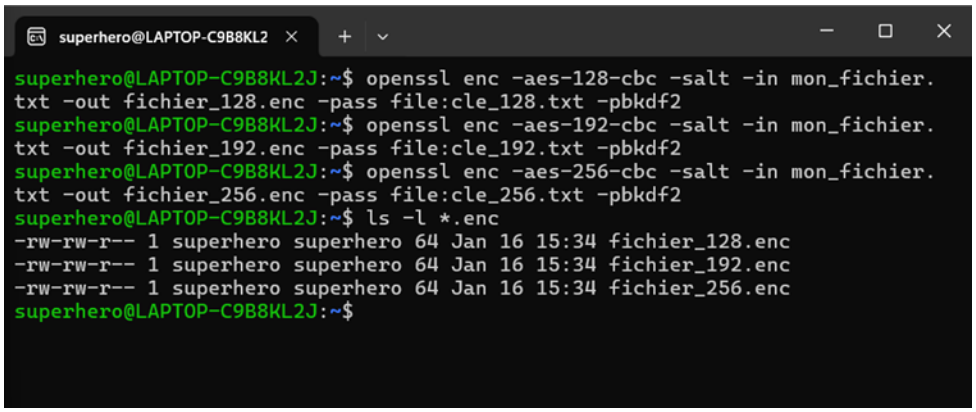

FIGURE 5 –

4.1 Pourquoi PBKDF2 ?

Les mots de passe choisis par les humains sont généralement faibles et prévisibles. PBKDF2 applique une fonction de hachage de manière itérative (par défaut 10 000 fois) pour :

— Ralentir considérablement les attaques par force brute
— Augmenter le coût computationnel des attaques
— Dériver une clé de longueur appropriée à partir d'un mot de passe court

4.2 Utilisation du Sel (Salt)

Le sel est une valeur aléatoire ajoutée au mot de passe avant la dérivation. Son rôle est crucial pour la sécurité.

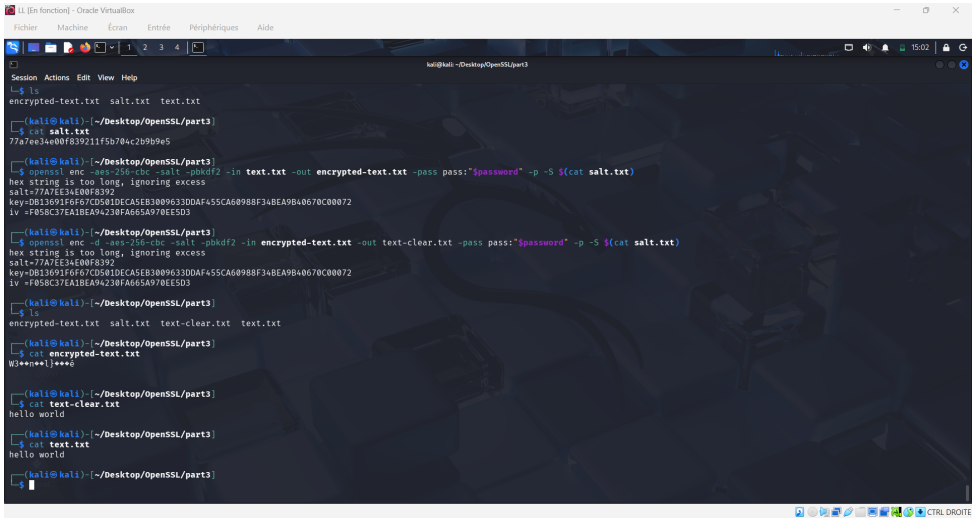

FIGURE 6 –

```
openssl rand -hex 16 > salt.txt
openssl enc -aes-256-cbc -salt -pbkdf2 -in text.txt -out encrypted-
   text.txt -pass pass:"$password" -p -S $(cat salt.txt)
```

Avantages du sel :

— Comme observé dans les captures, même avec un mot de passe identique, un sel différent produira un texte chiffré totalement différent
— Empêche les attaques par "Rainbow Tables" (tables précalculées de hachages)
— Garantit l'unicité de chaque chiffrement, même pour des données identiques

L'option `-p` affiche les paramètres utilisés (clé, vecteur d'initialisation, sel), ce qui est utile pour le débogage et la compréhension du processus.

5 Partie IV : Sécurité et stockage des clés

La sécurité d'un système de chiffrement symétrique repose entièrement sur la confidentialité de la clé. Si la clé est compromise, toutes les données chiffrées avec cette clé le sont également.

5.1 Protection des fichiers de clés sous Linux

Nous utilisons la commande `chmod` pour restreindre les permissions d'accès aux fichiers contenant les clés.

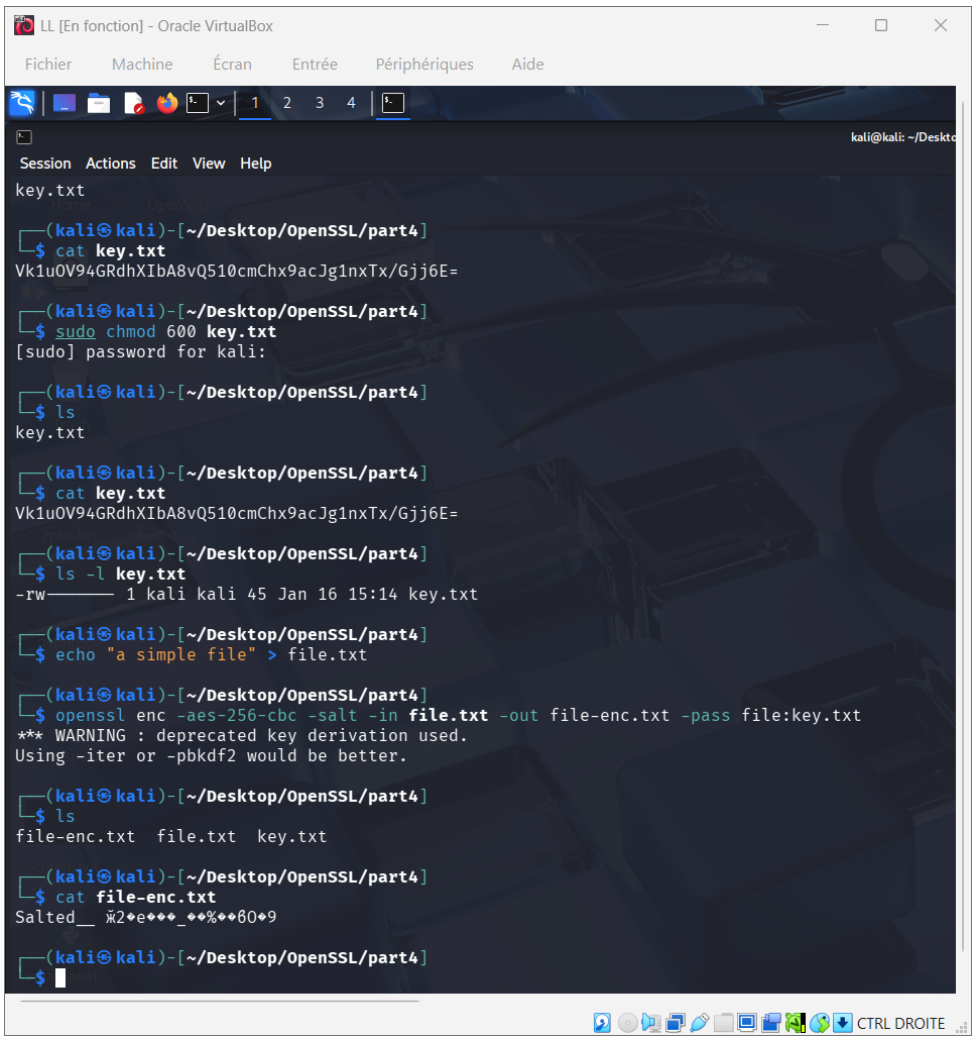

FIGURE 7 –

```
sudo chmod 600 key.txt
```

Explication des permissions 600 :
— **6** pour le propriétaire : lecture (4) + écriture (2) = 6
— **0** pour le groupe : aucun droit
— **0** pour les autres utilisateurs : aucun droit

Ainsi, seul le propriétaire du fichier peut lire et modifier la clé, ce qui limite considérablement les risques d'accès non autorisé.

5.2 Bonnes pratiques de gestion des clés

Synthèse des meilleures pratiques :
— **Ne jamais stocker la clé en clair** sur un support non protégé ou dans le code source d'une application
— **Utiliser des gestionnaires de secrets** tels que :
  — HashiCorp Vault
  — AWS Secrets Manager
  — Azure Key Vault
  — Keypass ou autres gestionnaires de mots de passe
— **Privilégier les HSM** (Hardware Security Modules) pour les environnements hautement sécurisés, car ils stockent les clés dans des composants matériels inviolables
— **Pratiquer la rotation régulière des clés** pour limiter l'impact d'une compromission éventuelle
— **Chiffrer les clés au repos** en utilisant une clé maîtresse (Key Encryption Key)
— **Implémenter une séparation des privilèges** : les personnes qui gèrent les clés ne doivent pas nécessairement avoir accès aux données chiffrées
— **Journaliser l'accès aux clés** pour détecter les utilisations anormales
— **Prévoir un plan de récupération** en cas de perte de clé, tout en maintenant un niveau de sécurité élevé

6 Conclusion

Ce travail pratique nous a permis d'explorer en profondeur les mécanismes du chiffrement symétrique avec OpenSSL. Nous avons appris que la sécurité d'un système cryptographique ne repose pas uniquement sur la robustesse de l'algorithme, mais également sur :
— La qualité de la génération des clés
— L'utilisation appropriée des sels et des fonctions de dérivation
— Les bonnes pratiques de gestion et de stockage des clés
— La mise à jour régulière des outils cryptographiques

L'outil OpenSSL, bien que puissant, nécessite une compréhension approfondie de ses options pour être utilisé de manière sécurisée. Les avertissements concernant les méthodes dépréciées doivent être pris au sérieux et les recommandations (comme l'utilisation de PBKDF2) doivent être systématiquement appliquées.

Enfin, il est essentiel de rappeler que le chiffrement symétrique, malgré son efficacité, pose le problème fondamental de la distribution sécurisée de la clé partagée

entre les parties communicantes. Ce défi a conduit au développement du chiffrement asymétrique, qui fait l'objet d'études complémentaires.