TP : Double chiffrement AES par blocs

L'AES est un protocole de chiffrement par blocs : Un bloc de donnée de 16 octets (AES-128 bits) ou 32 octets (AES-256-bits) est manipulé pour être chiffré. Pour chiffrer un message plus long, il faut découper la donnée en plusieurs blocs. Il existe de nombreuses manières de découper la donnée et de manipuler plusieurs blocs, dont certaines ont des limitations.

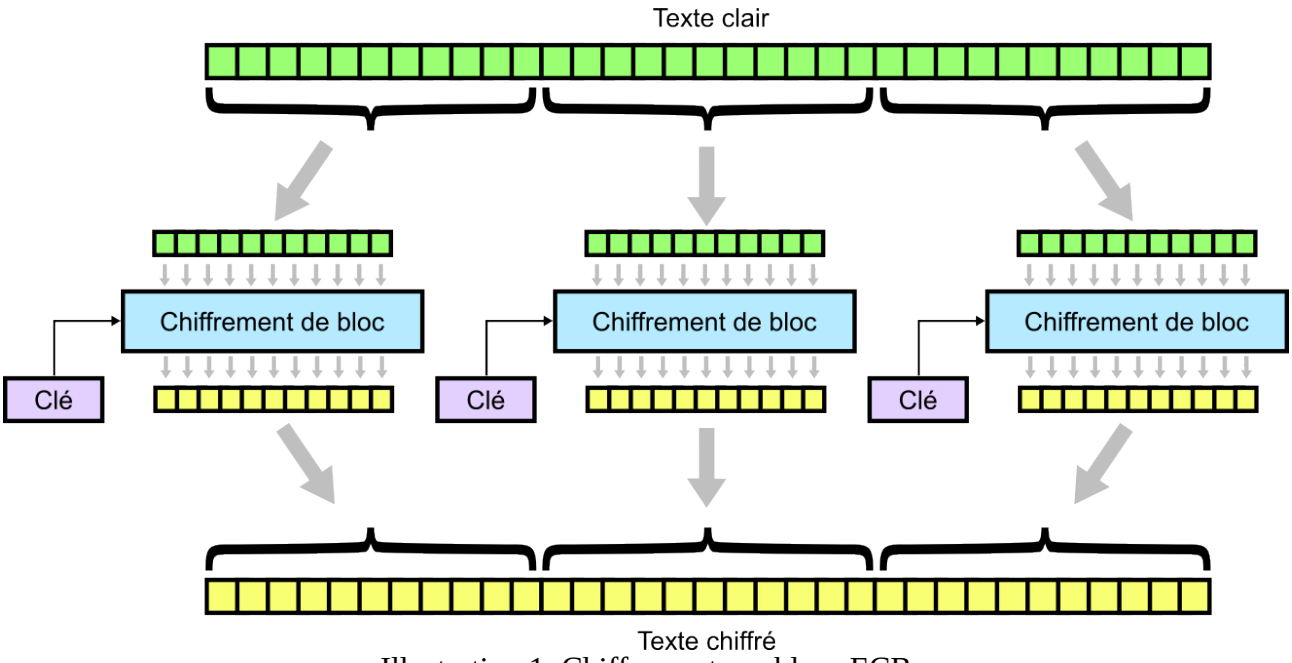

Illustration 1: Chiffrement par blocs ECB

A. Chiffrer par blocs avec Openssl.

• Openssl est une boîte à outils open-source qui fournit tout le nécessaire pour chiffrer/déchiffrer des messages à l'aide de différents protocoles cryptographiques. Si ce n'est pas déjà fait, installez openssl :

```
sudo apt-get install -y openssl
```

• Recherchez et téléchargez sur votre ordinateur l'image d'un **logo** avec l'extension **.bmp** (les résultats d'une recherche google peuvent être filtrés en ajoutant à la recherche "filetype:bmp").

• Chiffrez votre logo en utilisant openssl et le protocole AES en mode ECB:

```
openssl enc -aes-256-ecb -in FileName.bmp -out FileName_crypt.bmp
```

Il vous est demandé de saisir un mot de passe utilisé pour chiffrer/déchiffrer votre fichier. Choisissez n'importe lequel.

• Déchiffrez le fichier que vous venez de chiffrer en réutilisant le même mot de passe et vérifiez que le résultat est identique à l'image initiale.

```
openssl enc -aes-256-ecb -d -in FileName_crypt.bmp -out FileName_decrypt.bmp
```

• Votre ordinateur a besoin de certaines informations pour interpréter un fichier comme une image, malheureusement ces informations ont été chiffré et vous ne pouvez pas visualiser le fichier "FileName_crypt.bmp", pour remédier à ce problème, nous pouvons copier l'en-tête de l'image initiale dans notre fichier chiffré :

```
dd if=FileName.bmp of=FileName_crypt.bmp bs=54 count=1 conv=notrunc
```

• Ouvrez le fichier chiffré à l'aide d'une visionneuse d'image. Que constatez vous ? Quelle est la limite du chiffrement en mode ECB ? Qu'est-ce qu'un vecteur d'initialisation (IV) ?

• Chiffrez à nouveau votre image cette fois en mode CBC. Que constatez vous ?

• Openssl peut également être utilisé pour chiffrer directement des messages dans un terminal :

```
echo "Ceci est un message" | openssl enc -aes-256-ecb
```

• Chiffrez un message en base64 (utiliser la commande openssl enc -help pour avoir plus d'informations). Qu'est ce que l'encodage base64 ? À quoi le reconnait-on ?

B. Double chiffrement AES

Un message a été chiffré par le protocole AES en mode ECB à l'aide d'une clé et le résultat a lui-même été chiffré en utilisant une clé différente. Votre mission, si vous l'acceptez est de retrouver les clés de chiffrement utilisées par attaque bruteforce en utilisant python, et de déchiffrer le message secret fourni ci-dessous, qui a également été chiffré avec la même paire de clé.

```
plain_text: QnJhdm8gISBWb3VzIGV0ZXMgcGFydmVudSBhIGRlY29kZXIgdW4gbWVzc2FnZSBlbiBiYXNlNjQgISAgICAgIA==

double_cipher: I2vizvsCLxYmTQOW/+swdmf1/0V1lDWgx989YPW5H3qL4PycVyFFsV8DfOvZIQti0t+zSWVtaoFBOkA8vJmySw==

secret_cipher: dwfuRvQvBc32b5KZbtf3gfL0yNMpR/cr1VudpW9V+YohPnjt6WLrMLo4EfMHGSjbQDieLFTsDq2ECRZBhI+7Nh9W0RVg3hDF3jC04L/tJUvIrFw94veCWuRXNisuTtmg
```

Indication : Les deux clés AES-256 sont de taille 32 octets et les 29 octets de poids fort valent 0.

Aide :

• **AES**

La librairie pycryptodome permet d'utiliser le protocole AES en python. Si ce n'est pas déjà fait, installez là.

```
pip install pycryptodome
```

Vous pouvez chiffrer et déchiffrer des messages de la manières suivante :

```
from Crypto.Cipher import AES

cipher = AES.new(key, AES.MODE_ECB).encrypt(b"message")
AES.new(key, AES.MODE_ECB).decrypt(cipher)
b"message"
```

• **Base64**

Le module AES présenté ci-dessus requiert une clé et un message sous forme de séquence d'octets. vous pouvez utiliser le module base64 pour réaliser des conversions entre le base64 et les séquences d'octets :

```
import base64

base64.b64encode(b"Ceci est une sequence d'octets")
b'Q2VjaSBlc3QgdW5lIHNlcXVlbmNlIGQnb2N0ZXRz'

base64.b64decode("Q2VjaSBlc3QgdW5lIHNlcXVlbmNlIGQnb2N0ZXRz")
b"Ceci est une sequence d'octets"
```

• **Bit-shift**

On peut réaliser un shift de plusieurs bits sur un entier via l'opérateur "<<" :

```
x = 1328

bin(x)
0b10100110000

x = x << 4

bin(x)
0b101001100000000
```

• **Conversion d'entier**

On peut convertir un entier en séquence d'octets grâce à la méthode "to_bytes" :

```
x = 341039

x.to_bytes(6, "big")
b'\x00\x00\x00\x054/'
```

• **Dictionnaire**

Un dictionnaire est une structure de donnée dont les valeurs sont indexées par des clés qui peuvent être de n'importe quel type :

```
message_dict = {}

message_dict ["aucune"] = "idee"

message_dict
{'aucune': 'idee'}

message_dict["aucune"]
'idee'
```

Il est possible de tester simplement si un élément appartient à l'ensemble des indexes du dictionnaire ou à ses valeurs :

```
"aucune" in message_dict
True

"idee" in message_dict.values()
True
```