Solution à : Saturn Cipher.

La console demande l'entrée d'un bloc en hexadécimal de seize octets (32 caractères).

Première option :

Un message libre d'entrée est chiffré en clair connu de 16 octets (clé 128 bits) ; le résultat est restitué.

L'opération peut être répétée autant de fois qu'on veut.

Deuxième option :

le programme renvoie la version encryptée du flag est retourné : le calcul est basé sur une clé aléatoire d'une longueur égale à seize octets qui est donc indéchiffrable en « temps réel ».

L'encryptage réalise des opérations complémentaires de rotation et de permutation.

Les opérations dépendent a priori de la clé. Une faille de l'algo est repérée avec le clair connu. Si un caractère seul change en entrée, la chaîne encryptée est quasi identique. Cela signifie qu'il n'y a pas de diffusion. En plus, dans une même session, si un même caractère change en entrée, on obtient : une chaîne cryptée qui se modifie toujours au même endroit. Le challenge est sensible à une attaque octet par octet (caractère par caractère).

Par exemple, on part d'un clair connu à « 00000000000000000000000000000000 ».

```
┌──(kali㉿kali)-[/mnt/Share/404CTF2025]
└─$ nc challenges.404ctf.fr 30169
Saturne a tout pour être sur, bonne chance pour déchiffrer ...
Veux-tu chiffrer (1) ou obtenir le flag (2) ? > 1
Donne moi un bloc en hexadécimal à chiffrer > 00000000000000000000000000000000
Et voilà le chiffré > 648b1342da994e0c74551c24b0d5fea0
Veux-tu chiffrer (1) ou obtenir le flag (2) ? > 2
Bon courage ... > 75fa061b5a8e80a11aa6813662432e129d28a5d962d86bc3b4bb1ade556889ec489366b99a08809cd59bbf6bf701d6dd
Veux-tu chiffrer (1) ou obtenir le flag (2) ? >
```

On change un « 0 » en position n. On voit ce qui change dans la chaîne chiffrée. Quand on modifie le n ième caractère de la chaîne de départ, cela nous modifie le p ième caractère de la chaîne d'arrivée.

****

Ce n'est pas l'algo d'AES ici :

Transformation et diffusion

Supposons que seul l'octet en position $\mathbf{b_0}$ (coin supérieur gauche) est non nul dans l'état initial. Après plusieurs tours de transformations AES, cet octet influencera d'autres positions dans l'état. Par exemple, après deux tours, l'influence de $\mathbf{b_0}$ peut se propager à la position $\mathbf{b_{10}}$.

Voici une représentation simplifiée de cette diffusion :

État initial : État après transformations :

$[\ b_0\ \ b_4\ \ b_8\ \ b_{12}\ ]$ $[\ *\ \ *\ \ *\ \ *\ ]$

$[\ b_1\ \ b_5\ \ b_9\ \ b_{13}\ ]$ $[\ *\ \ *\ \ *\ \ *\ ]$

$[\ b_2\ \ b_6\ \ b_{10}\ b_{14}\ ]$ $[\ *\ \ *\ \ b_{10}\ *\ ]$

$[\ b_3\ \ b_7\ \ b_{11}\ b_{15}\ ]$ $[\ *\ \ *\ \ *\ \ *\ ]$

Dans cet exemple, l'octet $\mathbf{b_0}$ a une influence sur l'octet $\mathbf{b_{10}}$ après les transformations. Cela illustre la propriété de diffusion, où la modification d'un seul octet d'entrée affectera plusieurs octets de sortie.

****

On change maintenant le n ième caractère de la chaîne de départ par d'autres valeurs et on boucle : jusqu'à la valeur à la p ième position correspondante à la chaîne du flag (alignement sur n positions).

On a ainsi retrouvé la valeur du flag en position n. Il n'y a plus qu'à changer la valeur à la position n jusqu'à ce que la valeur donne le flag.

Faire tourner les valeurs du départ : 00000000000000000000000000000001 pour obtenir un seul caractère changé à l'arrivée en chaîne.

On peut donc tester donc, un à un, par caractère, si c'est le bon : ce n'est pas une diffusion où, si l'on changeait un caractère, toute la chaîne ou grande partie de celle-ci, serait changée en bout de course.

A chaque redémarrage de serveur (lancement d'une session : netcat), la clé de session est changée.

La dernière étape est de rejouer ce script sur le serveur netcat pour obtenir ainsi le vrai flag en HEXA.

mnt > Share > 404CTF2025 > publications_git > 404CTF2025 > Cryptanalyse > Saturn Cipher > SaturnCipherExploit.py > main

```
 94  def main():
121      padded_flag = pad(FLAG, 16)
122      flag_blocks = [padded_flag[i:i+16] for i in range(0, len(padded_flag), 16)]
123      enc_blocks = list(map(s.Encrypt, flag_blocks))
124      flagCrypte =  b"".join(enc_blocks)
125      print("Flag crypté >", flagCrypte.hex())
126
127      for p in range(15):
128          c = []
129          for b in range(2):
130              test = "00" * p + bytes([b]).hex() + "00" * (16 - p)  # 1 byte fixé, le reste à 0
131              block = bytes.fromhex(test)
132              ciphertext = s.Encrypt(block)
133              c.append(ciphertext)
134              print(test, "-", block, " - ", ciphertext.hex())
135
136          print("\nDifférences byte par byte :")
137          for i, (b1, b2) in enumerate(zip(c[0], c[1])):
138              if b1 != b2:
139                  print(f"Byte pos {p} -> #{i}: {b1:02x} → {b2:02x}  |  flagCrypte[{i}] = {flagCrypte[i
140                  for b in range(0, 257):
141                      test = "00" * p + bytes([b]).hex() + "00" * (16 - p)  # 1 byte fixé, le reste à 0
142                      block = bytes.fromhex(test)
143                      ciphertext = s.Encrypt(block)
144                      ci = ciphertext[i]
145                      if ci == flagCrypte[i]:
146                          print(f"✔️  flag[{p}] = {b:#02x} ('{chr(b)}')")
147                          break
```

mnt > Share > 404CTF2025 > publications_git > 404CTF2025 > Cryptanalyse > Saturn Cipher > SaturnCipherNetcat.py > ...

```
4  HOST = "challenges.404ctf.fr"
5  PORT = 30169
6
7
8  def getFlagEncored(p):
9      # === Lire jusqu'à l'invite "(2) ? >" et envoyer "2" pour obtenir le flag chiffré ===
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[+] Chiffré : 00000000000000000000000000000000 - 90e384358641c03c7220e1a710c98f13
[+] Chiffré : 00000000000000000000000000000001 - 90e384358641c0c37220e1a710c98f13

Différences byte par byte :
Byte pos 15 -> #7: 3c → c3  |  flagCrypte[7] = 03
✔️ flag[15] = 0xa ('
') : 404CTF{S4turn3d_d1fus10n_1s_1mp0rt4nt}
```

```
Changement de bloc : 3
[*] Closed connection to challenges.404ctf.fr port 30169
[+] Opening connection to challenges.404ctf.fr on port 30169: Done
```

Ln 7, Col 1 Spaces: 4 UTF-8 LF {} Python