Table des matières

1) Attaque DPA

Avant de commencer le TP je dois installer GNU octave (voir https://wiki.octave.org/Octave_for_Debian_systems)

```
sudo apt-get install octave
```

1) J'ouvre le logiciel octave et dedans j'ouvre le fichier dpa.mat

Je vois qu'on a 3 variables qui ont été ajouté :

Nom	Classe	Dimensio
CTO	uint8	200x16
PTI	uint8	200x16
traces	uint8	200x370

La variable traces contient 200 mesures contenant chacune 37000 échantillons (voir Dimensions)

2) Une valeur correspond à une ligne entière (16 octets donc 16 colonnes)

J'affiche donc les 2 premières valeurs de la variable PTI en décimal puis j'affiche en hexadécimal :

```
>> p1=PTI(1,:)
p1 =

 Columns 1 through 13:

   37  235  140   72  255  137  203  133   79  192  144  129  204
```

```
 Columns 14 through 16:

   71  237  252

>> p2=PTI(2,:)
p2 =

 Columns 1 through 13:

  134   25  178   20  254  101  146  212  139  252  234  156  157

 Columns 14 through 16:

  142   50   68

>> printHex(p1);
25eb8c48ff89cb854fc09081cc47edfc
>> printHex(p2);
8619b214fe6592d48bfcea9c9d8e3244
```

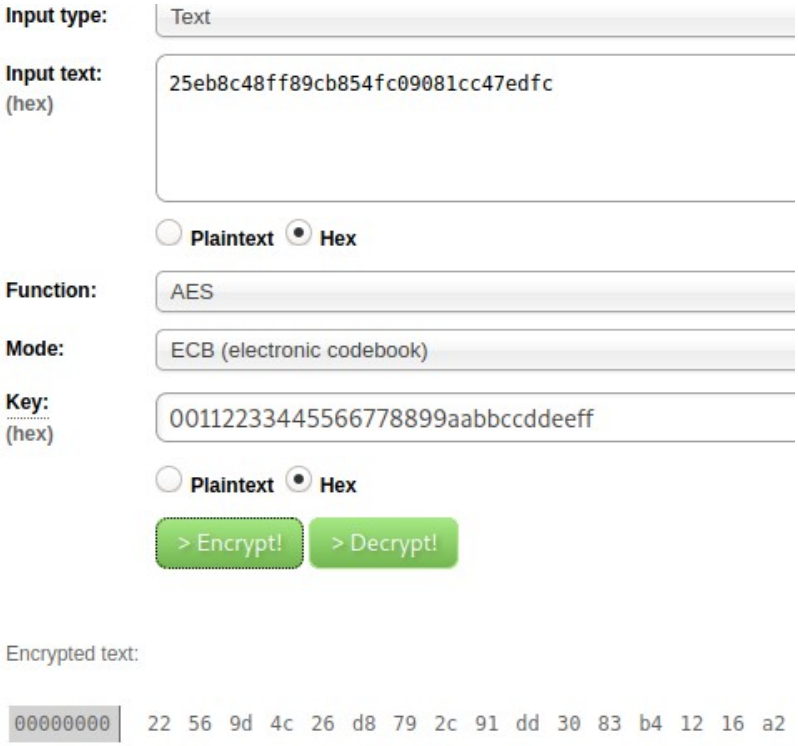
3) J'affiche les valeurs de CTO correspondants au PTI :

```
>> p1Cto=CTO(1,:)
p1Cto =

 Columns 1 through 13:

   34   86  157   76   38  216  121   44  145  221   48  131  180

 Columns 14 through 16:

   18   22  162

>> p2Cto=CTO(2,:)
p2Cto =

 Columns 1 through 13:

  207  190  144   83  168   85  225  117  181  167  248  223   45

 Columns 14 through 16:

   42   74  144
```

4) Je peux vérifier sur internet que la formule suivante fonctionne bien sur GNU octave :

AES128-ECB(KEY,PTI)=CTO

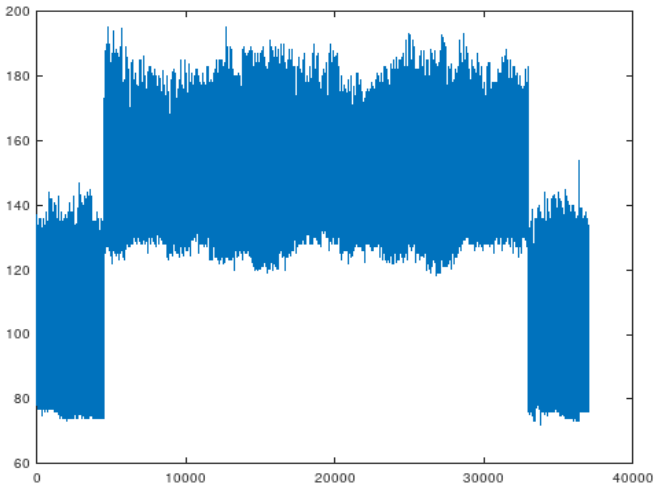
Ici je fais le test pour la première valeur. Je rentre la première valeur du PTI en hexadécimal puis je donne la clé de cryptage et je dois retrouver la première valeur du CTO



5) Maintenant j(affiche le graphique de la première mesure :

```
>> t1=traces(1,:);
>> plot(t1);
```

On obtient ceci :



En regardant le graphique je détermine approximativement le début du premier round ainsi que sa longueur

Ici il commence environ à 4626,7 et sa longueur est de 4325,9 à 33 014. Le cryptage AES ayant 10 rounds il faut simplement diviser par 10 pour avoir la longueur d'un round (on a une longueur d'environ 28654,1)

6) Je paramètre ces valeurs dans le programme ex1_dpa.m et je l'exécute. Je dois retrouver au moins 75% des octets de la clé sinon c'est que je n'ai pas encore de valeurs assez proches des vraies valeurs

```
>> ex1_dpa

Nombre de traces: 200
Nombre d'échantillons par trace: 37000
Analyse à partir de l'échantillon 4325.9 sur 2865.41 valeurs
warning: non-integer range used as index
warning: called from
    ex1_dpa at line 24 column 7
Début du calcul DPA...
K(01): 0x00
K(02): 0x2a
K(03): 0x16
K(04): 0x33
K(05): 0x44
K(06): 0x55
K(07): 0x66
K(08): 0x77
K(09): 0x0c
K(10): 0x99
K(11): 0xaa
K(12): 0xbb
K(13): 0xcc
K(14): 0xdd
K(15): 0xee
K(16): 0xff
Calcul terminé (122 s) => 7.6 s par octet
```

Les erreurs sont surlignées en vert (on ne peut pas avoir une précision de 100%) mais on retrouve à peu près notre clé

7)

2) Attaque CPA

1) Je test plusieurs fois le programme ex2_2a.m :

```
>> ex2_2a
X: 0xe3 (11100011)
Distance de Hamming: 5
>> ex2_2a
X: 0x90 (10010000)
Distance de Hamming: 2
>> ex2_2a
X: 0x5e (01011110)
Distance de Hamming: 5
>> ex2_2a
X: 0x8e (10001110)
Distance de Hamming: 4
>> ex2_2a
X: 0x09 (00001001)
Distance de Hamming: 2
```

Le programme compte bien le bon nombre de bits pour un nombre binaire généré aléatoirement

2) Je crée maintenant le script ex2_cpa.m et j'y ajoute la formule de calcul de la distance de Hamming de v à la fin de la première boucle après le S-BOX :

```
for b=byteStart:byteEnd % Pour tous les octets de 1 à 16
  % Calcul de la sortie SubByte du premier round = XOR+SBOX
  p=double(PTI(:,b))*ones(1,nbKey);
  v=SBOX(bitxor(p,keyMat)+1);
  v=HammingWeight(v+1);
```

3) Je supprime maintenant toute la boucle différentielle et je la remplace par la calcul de la matrice corrélation entre v et traces :

```
matrice=matCorr(v,traces);
```

Le programme modifié donne ceci (on a juste modifié une boucle) :

```
for b=byteStart:byteEnd % Pour tous les octets de 1 à 16
  % Calcul de la sortie SubByte du premier round = XOR+SBOX
  p=double(PTI(:,b))*ones(1,nbKey);
  v=SBOX(bitxor(p,keyMat)+1);
  v=HammingWeight(v+1);
  matrice=matCorr(v,traces);

  % Recherche de la position du maximum (pMax)
  [m,pMax]=max(max(matrice,[],2));
```

```
  printf('K(%02d): 0x%02x\n',b,pMax-1);
end
```

Maintenant si je le lance je dois retrouver la clé :

```
>> ex2_cpa

Nombre de traces: 200
Nombre d'échantillons par trace: 37000
Analyse à partir de l'échantillon 4325.9 sur 2865.41 valeurs
warning: non-integer range used as index
warning: called from
    ex2_cpa at line 19 column 7
Début du calcul DPA...
K(01): 0x00
K(02): 0x11
K(03): 0x22
K(04): 0x33
K(05): 0x44
K(06): 0x55
K(07): 0x66
K(08): 0x77
K(09): 0x88
K(10): 0x99
K(11): 0xaa
K(12): 0xbb
K(13): 0xcc
K(14): 0xdd
K(15): 0xee
K(16): 0xff
Calcul terminé (0.539 s) => 0.0337 s par octet
```

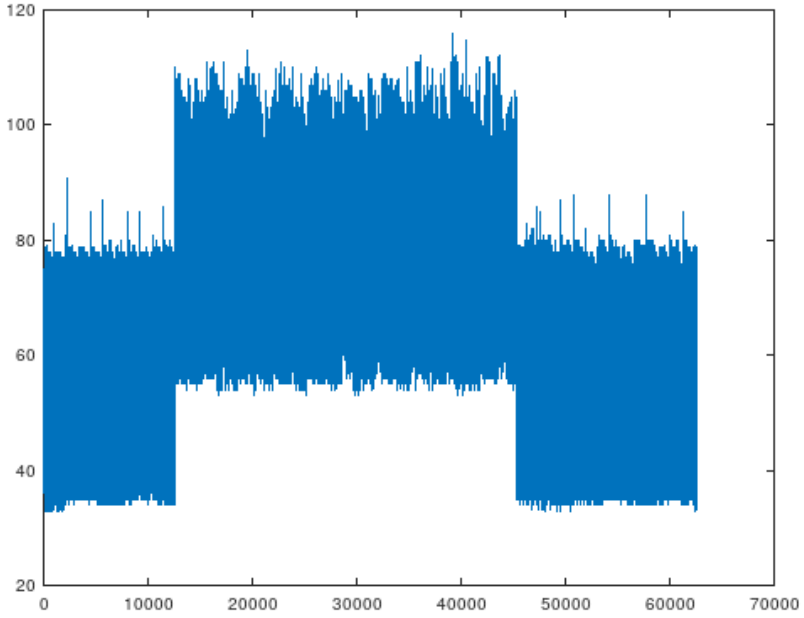
Le programme a bien retrouvé la clé sans aucunes erreurs et en très peu de temps (0,5s)

4) Maintenant que le programme fonctionne et que je récupère bien la clé prédéfini au début du TP je peux le modifier afin de retrouver une clé que je ne connais pas

Pour se faire je dois d'abord charger le fichier cpa.mat et afficher le graphique de la première valeur :

```
>> load cpa.mat
>> PTI(1,:)
ans =

 Columns 1 through 13:

   41   35  190  132  225  108  214  174   82  144   73  241  241
```

```
 Columns 14 through 16:

  187  233  235

>> t1=traces(1,:);
>> plot(t1);
```



Maintenant je fais comme au début du TP. Je trouve approximativement la valeur de départ du 1er round et la longueur d'un round :

Valeur round 1 : 12613

Valeur round 10 : 45516

Longueur de 1 round : 3290,3

Maintenant je dois reporter ces valeurs dans le script et faire attention à bien charger le bon fichier (cpa.mat) :

```
R1offset=12613;
R1length=3290.4;
aes_utils;
byteStart=1;
byteEnd=16;
keyStart=1;
keyEnd=256;
nbKey=keyEnd-keyStart+1;
KEY='00112233445566778899aabbccddeeff';
```

```
for b=byteStart:byteEnd
  key(b)=hex2dec(KEY(2*b-1:2*b));
end
load cpa.mat
[nbTraces,nbBytes]=size(PTI);
printf('Nombre de traces: %d\n',nbTraces);
...
```

Je n'ai plus qu'à lancer le programme et trouver la clé :

```
>> ex2_cpa

Nombre de traces: 200
Nombre d'échantillons par trace: 62500
Analyse à partir de l'échantillon 12613 sur 3290.4 valeurs
Début du calcul DPA...
K(01): 0xde
K(02): 0xad
K(03): 0xbe
K(04): 0xef
K(05): 0x80
K(06): 0x35
K(07): 0x07
K(08): 0x16
K(09): 0x2b
K(10): 0xa7
K(11): 0x84
K(12): 0x56
K(13): 0x62
K(14): 0x3a
K(15): 0x45
K(16): 0x2f
Calcul terminé (0.568 s) => 0.0355 s par octet
```

clé : deadbeef803507162ba78456623a452f

5) On peut vérifier que la clé est bonne sur l'ordinateur avec les valeurs du PTI et du CTO :

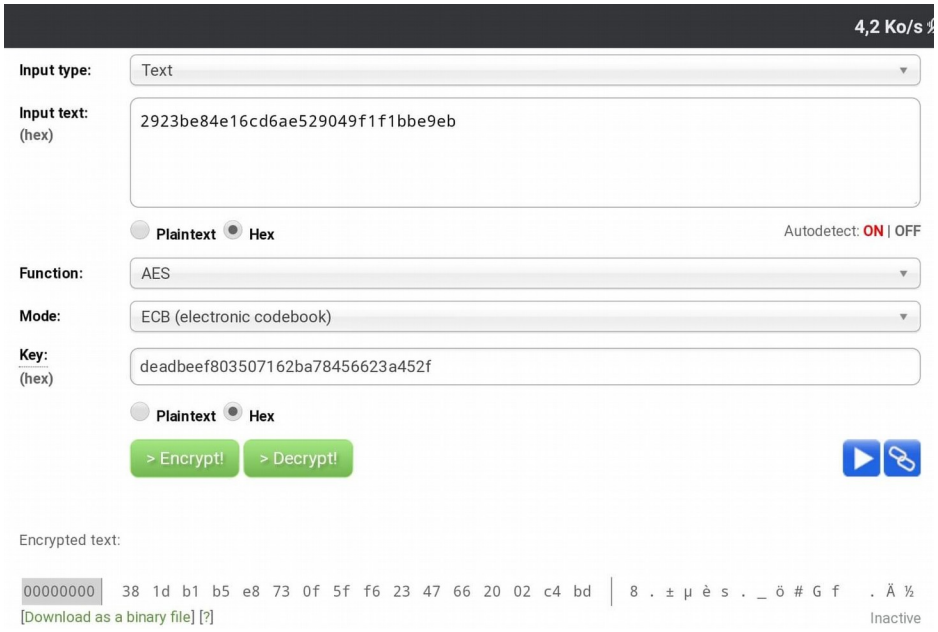
On prend le PTI de la première valeur en hexa :

```
>> p1=PTI(1,:);
>> printHex(p1);
2923be84e16cd6ae529049f1f1bbe9eb
```

On affiche aussi le CTO de la première valeur en hexa :

```
 >> p1Cto=CTO(1,:);
>> printHex(p1Cto);
381db1b5e8730f5ff62347662002c4bd
```

Maintenant sur le site en mettant la clé qu'on a trouvé et le PTI (message clair) on devrait retrouver le CTO (message crypté) en sortie :



On retrouve bien le bon CTO en utilisant la clé qu'on a trouvé

3) Attaque CEMA

Dans cette partie on va devoir retrouver la clé initiale (4c8cdf23b5c906f79057ec7184193a67) en se servant de la clé K10 qu'on va récupérer

1) J'exécute le script3_cema.m :

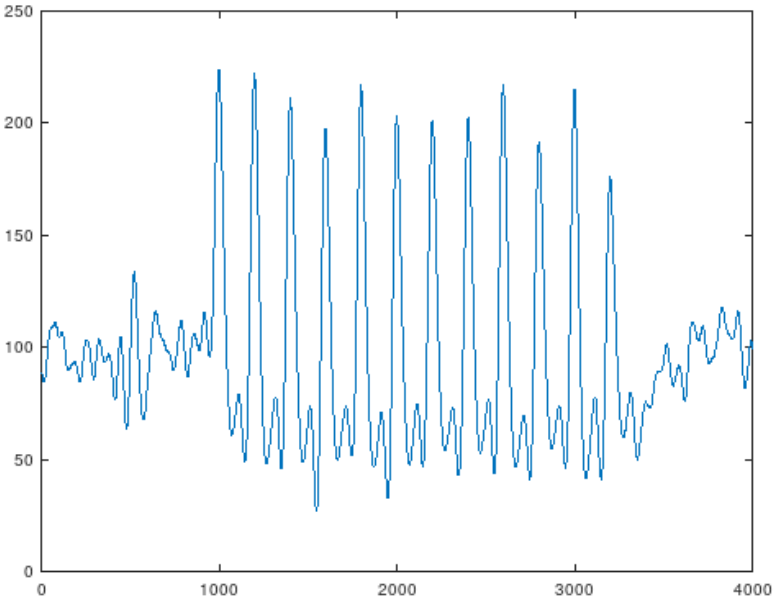
```
>> ex3_cema

Nombre de traces: 2500
Nombre d'échantillons par trace: 4000
Analyse à partir de l'échantillon 0 sur 100 valeurs
Calculate AES key schedule round: 01
Calculate AES key schedule round: 02
Calculate AES key schedule round: 03
Calculate AES key schedule round: 04
Calculate AES key schedule round: 05
Calculate AES key schedule round: 06
```

```
Calculate AES key schedule round: 07
Calculate AES key schedule round: 08
Calculate AES key schedule round: 09
Calculate AES key schedule round: 10
Debut du calcul de Hamming pour 2500 traces...
  Hamming computed for 100 traces...
  Hamming computed for 200 traces...
  Hamming computed for 300 traces...
  Hamming computed for 400 traces...
  Hamming computed for 500 traces...
  Hamming computed for 600 traces...
  Hamming computed for 700 traces...
  Hamming computed for 800 traces...
  Hamming computed for 900 traces...
  Hamming computed for 1000 traces...
  Hamming computed for 1100 traces...
  Hamming computed for 1200 traces...
  Hamming computed for 1300 traces...
  Hamming computed for 1400 traces...
  Hamming computed for 1500 traces...
  Hamming computed for 1600 traces...
  Hamming computed for 1700 traces...
  Hamming computed for 1800 traces...
  Hamming computed for 1900 traces...
  Hamming computed for 2000 traces...
  Hamming computed for 2100 traces...
  Hamming computed for 2200 traces...
  Hamming computed for 2300 traces...
  Hamming computed for 2400 traces...
  Hamming computed for 2500 traces...
Calcul de Hamming terminé (37.7 s - 1.51 s pour 100 traces)
```

Il y a 25 mesures de 100 échantillons

2) Je dois afficher le graphique de la première valeur :

```
>> load cema.mat
>> t1=traces(1,:);
>> plot(t1);
```



Valeur 1er round : 951,15

Valeur dernier round : 3273,7

Longueur de 1 round : 232,255

Valeur dernier round : 3041,445

3) Il faut vérifier les valeurs de KEY, PTI et CTO comme à la question 1.4 :

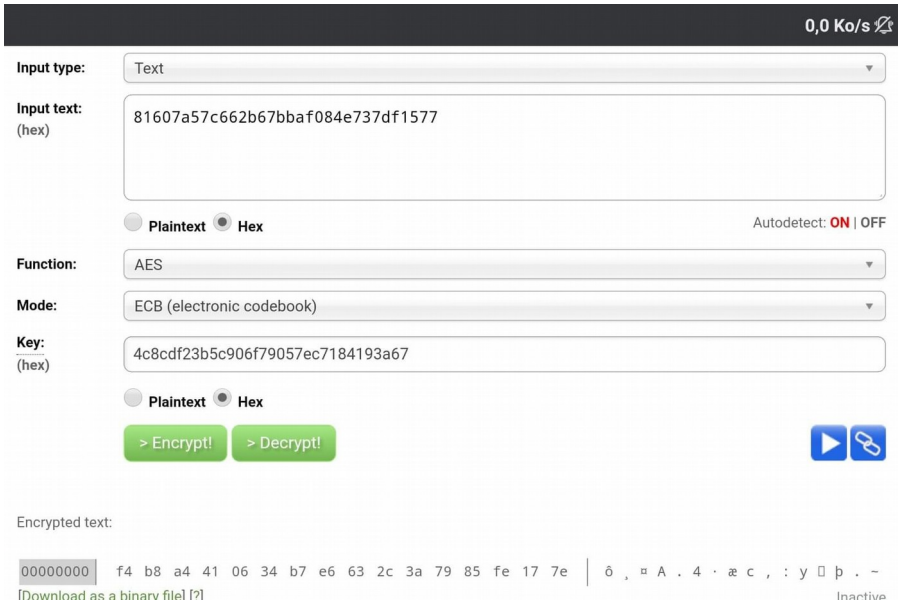
PTI :

```
>> p1=PTI(1,:);
>> printHex(p1);
81607a57c662b67bbaf084e737df1577
```

CTO :

```
>> p1Cto=CTO(1,:);
>> printHex(p1Cto);
f4b8a4410634b7e6632c3a7985fe177e
```

On vérifie sur internet qu'on retrouve bien le même CTO en utilisant le PTI et la clé donnée au début du paragraphe :



Les valeurs sont bien vérifiées

4) Je vérifie les valeurs d'expansions de la clé AES

J'affiche les valeurs du round 10 en hexadécimal :

```
>> printHex(sub_keys(10+1,:));
3c47400140d680924d3800b925fb78b0
```

Je compare avec un site (https://www.cryptool.org/en/cto/aes-step-by-step) :

Key

```
4c8cdf23 b5c906f7 9057ec71 84193a67
```

Expanded Key

```
4c8cdf23 b5c906f7 9057ec71 84193a67 990c5a7c 2cc55c8b bc92b0fa 388b8a9d a672047b 8ab758f0 3625e80a
0eae6297 46d88cd0 cc6fd420 fa4a3c2a f4e45ebd 2780f66f ebef224f 11a51e65 e54140d8 b48997b6 5f66b5f9
4ec3ab9c ab82eb44 87608cd4 d806392d 96c592b1 3d4779f5 67d66af3 bfd053de 2915c16f 1452b89a e7bad209
586a81d7 717f40b8 652df822 24fb4144 7c91c093 0dee802b 68c37809 3c474001 40d68092 4d3800b9 25fb78b0
```

Je retrouve bien la même valeur

5) Je décommente la fin du fichier pour activer la corrélation et retrouver la clé initiale :

```
n=0;
for b=1:16
  c=matCorr(double(squeeze(cbHamming(:,:,b))),traces(:,:));
  % Recherche de la position du maximum (pMax)
  [m,pMax]=max(max(c,[],2));
  printf('K(%02d): 0x%02x (0x%02X)\n',b,pMax-1,sub_keys(11,b));
  if pMax-1==sub_keys(11,b)
    n=n+1;
  end
end
printf('Found %d/%d bytes\n',n,nbBytes);
```

Je lance le programme et je dois retrouver ma clé du 10ème round :

```
>> ex3_cema

R10offset =  3041
Nombre de traces: 2500
Nombre d'échantillons par trace: 4000
Analyse à partir de l'échantillon 3041 sur 232.255 valeurs
K(01): 0x3c (0x3C)
K(02): 0x47 (0x47)
K(03): 0x40 (0x40)
K(04): 0x01 (0x01)
K(05): 0x40 (0x40)
K(06): 0xd6 (0xD6)
K(07): 0x80 (0x80)
K(08): 0x92 (0x92)
K(09): 0x4d (0x4D)
K(10): 0x38 (0x38)
K(11): 0x00 (0x00)
K(12): 0xb9 (0xB9)
K(13): 0x25 (0x25)
K(14): 0xfb (0xFB)
K(15): 0x78 (0x78)
K(16): 0xb0 (0xB0)
Found 16/16 bytes
```

6)