Challenge SSTIC 2013

Olivier Galliez
olivier@galliez.com

8 avril 2013

Table des Matières

Résumé

Le Challenge SSTIC 2013 consiste à analyser une trace réseau pour y trouver une adresse e-mail du type @challenge.sstic.org.

Les méthodes de dissimulation et chiffrement utilisées pour protéger cette adresse e-mail sont variées. Elles vont des plus simples (César, sténographie, …) aux plus complexes (AES, coprocesseur spécialisé, …).

Plusieurs étapes ont été nécessaires pour parvenir à résoudre ce challenge. Pour chacune d'entre elles, ce document décrit les méthodes utilisées et les outils qui ont été développés.

1. Aperçu général

1.1 Poupées russes

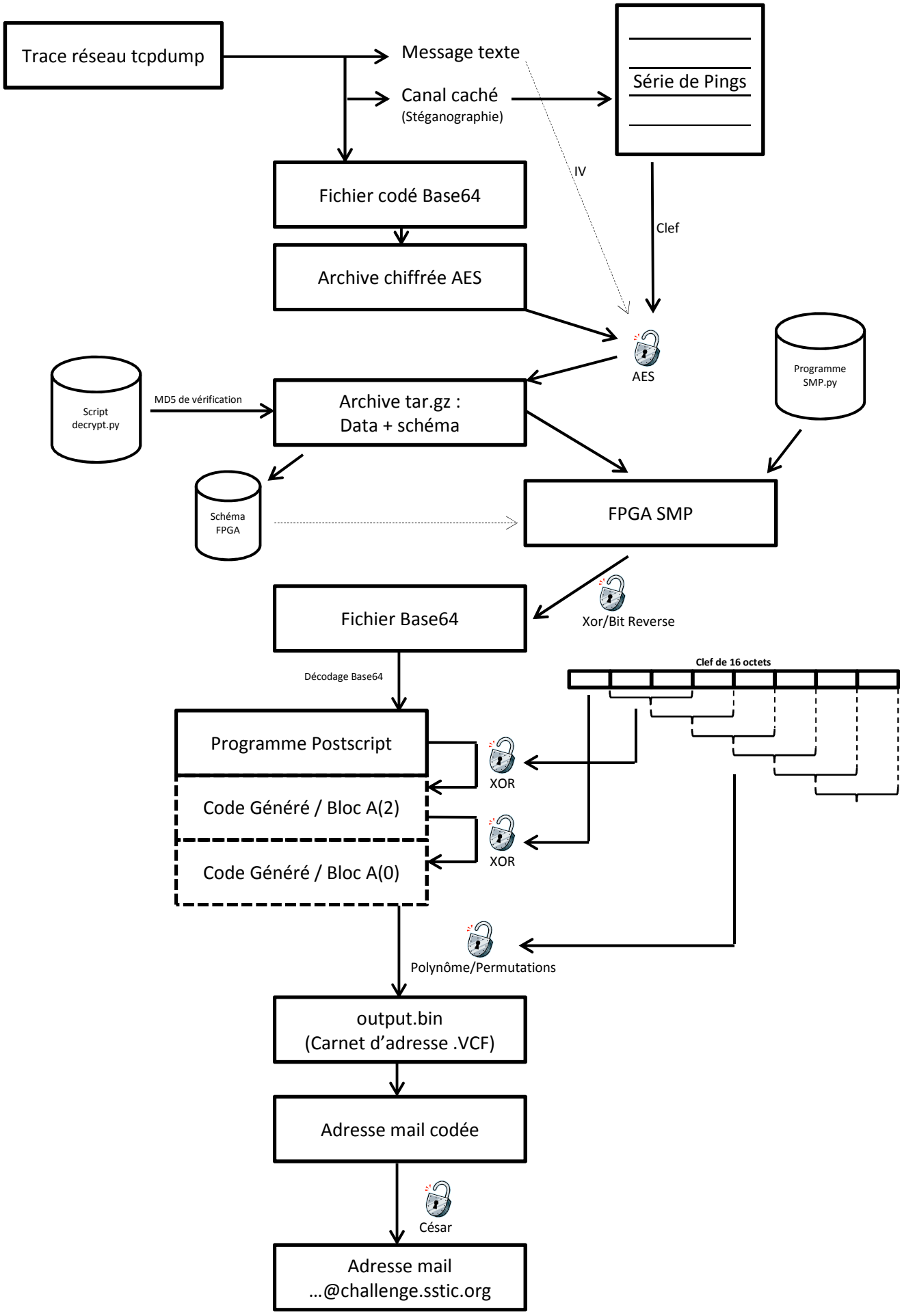
Sans rentrer dans les détails dès maintenant, le schéma ci-après présente l'organisation générale des données sur lesquelles nous allons travailler.

On peut y voit que le fichier initial transmis par FTP est chiffré en AES et codé en Base64 ; La clef AES est cachée dans une série de trames Ping.

Le fichier déchiffré est une archive compressée qui contient à son tour un programme Postscript codé en Base64 et chiffré par un coprocesseur.

Ce programme Postscript déchiffre une partie de ses propres instructions et les exécute. A son tour, ces instructions déchiffrent un autre morceau de programme qui va lui-même déchiffrer un carnet d'adresses. Bien entendu, aucune des clefs de chiffrement utilisée n'est connue…

Ce carnet d'adresses contient de nombreuses adresses mail, sauf celle recherchée qui est codée avec une méthode du type César…



Aperçu Général du Challenge SSTIC 2013

1.2 Outils utilisés

Pour résoudre ce challenge, j'ai utilisé les outils suivants :

- PC Linux (distrib mandriva), Wireshark, divers outils standards linux.
  Pour la facilité d'utilisation des outils « bas niveau » (grep, awk, tar, md5sum, base64, …)
- PC portable HP (Windows 7), Excel 2010, Visual C# 2010 Express.
  Pour la facilité et la rapidité de développement

Pour chacune des étapes, un ou plusieurs outils ont été développés. En fait, j'ai développé un seul logiciel en mode fenêtré qui intègre l'ensemble de ces outils.

J'ai choisi Visual C# pour la simplicité et la rapidité de développement mais également pour la disponibilité d'objets intéressants dans le cadre de ce Challenge (AES, MD5, …). Ce n'est peut-être pas le meilleur outil pour la rapidité d'exécution du code développé, mais je ne m'attends pas à avoir besoin d'une puissance de calcul importante. Si un « brute force » est nécessaire pour casser un chiffrage difficile (AES par exemple), c'est le matériel à ma disposition qui serait de toutes façons la première limite…

2. Téléchargement et analyse du fichier trace

2.1 Téléchargement et premiers contacts

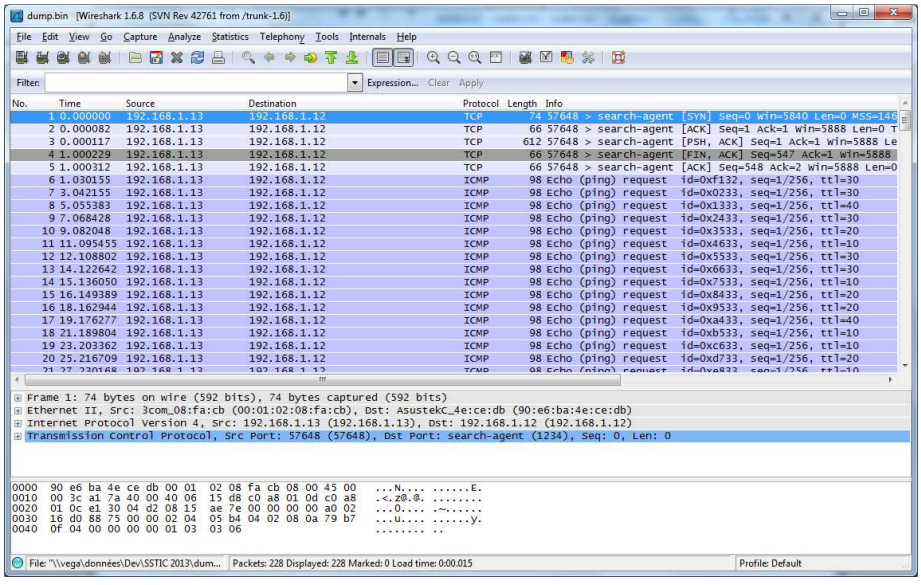
Le fichier de la trace réseau est téléchargé sur le site du SSTIC et son MD5 est vérifié :

```
# wget http://static.sstic.org/challenge2013/dump.bin
# md5sum dump.bin
968531851beed222851dbfd59140a395  dump.bin
```

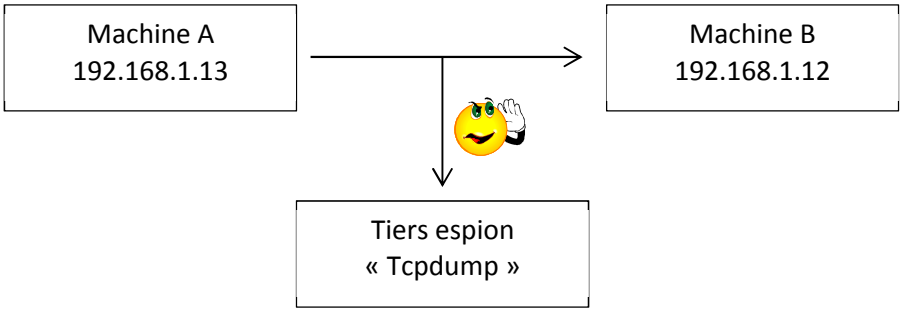
Sa taille est assez raisonnable : environ 843ko. Il s'agit d'une capture de trames réseau effectuée avec tcpdump :

```
# file dump.bin
dump.bin: tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
```

Il est donc facile de visualiser son contenu avec un outil comme Wireshark :



Ce fichier comporte 228 trames réseau échangées entre 2 machines dont les adresses IP sont respectivement 192.168.1.13 et 192.168.1.12 :



Le « tiers espion » ayant enregistré le fichier trace a certainement lancé une commande du style :

```
tcpdump -s0 -w dump.bin src host 192.168.1.13 and dst host 192.168.1.12
```

On distingue facilement 3 parties :

1. Un message texte (5 trames de session TCP)
2. Une série de 65 pings (65 trames ICMP)
3. Un transfert de fichier FTP (158 trames)

Le message texte et le transfert FTP étant faciles à interpréter, nous commencerons par eux, et nous terminerons par la série de pings qui demande un peu de recherche.

2.2 Le message texte

2.2.1 **Récupération du message**

Il a été transféré à l'aide d'une connexion TCP simple sur le port 1234.

Le contenu du message peut être visualisé avec Wireshark en utilisant la commande « Follow TCP string » :

```
Bonjour,
J'ai egare la cle pour dechiffrer mon carnet d'adresses.
Tu pourrais m'aider a la retrouver ? J'ai besoin de recuperer une adresse email a l'interieur.
Pour t'aider, je t'envoie :
- une archive chiffree en AES par FTP
- la cle AES par canaux caches
voici l'iv utilise pour AES : 76C128D46A6C4B15B43016904BE176AC
voici le checksum de l'archive pour verifier le dechiffrement :
61c9392f617290642f9a12499de6b688
merci

PS :
Indication pour les canaux caches : 1 bit de canal cache temporel
concatene a 3 bits de canal cache non temporel.
```

2.2.2 **Interprétation du message texte**

La lecture du message nous donne de précieuses informations. Elle nous indique :

- Que les données sont transférées par FTP ; cela tombe bien car la troisième partie des trames enregistrées concerne justement un transfert FTP !
- Qu'il s'agit d'un carnet d'adresses ; cela pourra éventuellement guider nos recherches plus tard ; il est toujours plus facile de trouver une clef quand le contenu du message chiffré est connu, même partiellement…
- Qu'il s'agit d'une archive ; Cela peut aussi guider nos recherches en envisageant un fichier au format « tar » ou « tar.gz »…
- Que les données sont chiffrées par AES ; l'IV (« Initialization Vecteur ») est fourni dans le message ; un MD5 est fourni pour s'assurer que le déchiffrement est correct.
- Que la clef AES est transmise en « canaux cachés » ; j'envisage déjà une raison à la présence des trames de ping…
  Une information sur la méthode de codage temporel/spatial est également fournie.

2.2.3 **Comment le message texte a été transmis**

Il peut être utile de comprendre la manière dont l'utilisateur A raisonne. Cela pourra éventuellement guider nos recherches ultérieures (plateforme et outils utilisés, niveau de chiffrement à attendre, …).

Pour effectuer le transfert du message texte, la machine B était à l'écoute du port 1234, par exemple avec une commande du type :

```
netcat -l -p 1234
```

et la machine A a envoyé son message avec une commande du type :

```
printf "Bonjour,\nJ'ai egare..." | netcat 192.168.1.12 1234
```

Comme nous le verrons plus loin l'utilisateur de la machine A a peut-être utilisé un outil de manipulation de paquets réseau comme « scapy ». Le message texte a également pu être envoyé avec cet outil à l'aide des commandes suivantes :

```
>>>sock=socket.socket()
>>>sock.connect(("192.168.1.12",1234))
>>>stream=StreamSocket(s,Raw)
>>>stream.send(Raw("Bonjour,\nJ'ai egare la clef...\n"))
```

2.3 Le transfert FTP

Les trames de transfert FTP se composent de 2 parties :

- La partie contenant les commandes FTP (login, fichier à transférer, …)
- Le fichier lui-même

2.3.1 **Commandes FTP**

Wireshark permet d'obtenir l'ensemble de ces commandes très facilement (commande « Follow TCP string » sur une des trames FTP de commande) :

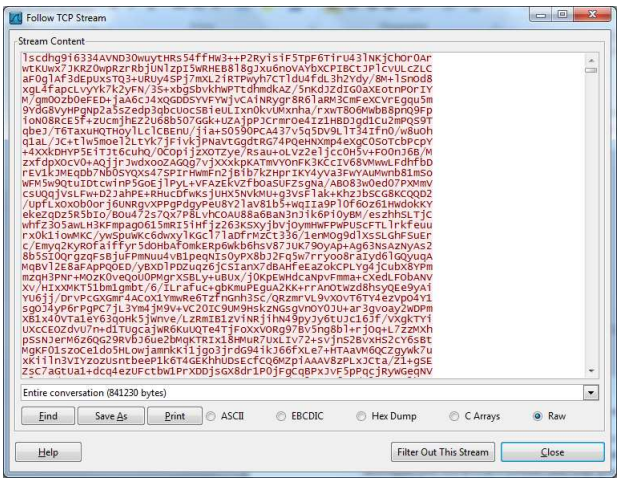
```
USER sstic
PASS sstic
PWD
FEAT
HELP SITE
CLNT NcFTP 3.2.2 linux-x86-glibc2.9
TYPE I
REST 1
REST 0
SIZE sstic.tar.gz-chiffre
MDTM sstic.tar.gz-chiffre
PASV
STOR sstic.tar.gz-chiffre
SITE UTIME sstic.tar.gz-chiffre 20130318113720 20130318113533 20130318113533 UTC
MDTM 20130318113533 sstic.tar.gz-chiffre
QUIT
```

Plusieurs informations importantes retiennent mon attention :

- L'utilisateur travaille sous linux ; nous pourrons donc nous attendre à des formats de fichiers « tar », éventuellement compressés, des fichiers codés Base64, et plus généralement, des techniques plus couramment utilisées sous linux que sous Windows.
- Le fichier transmis s'appelle « sstic.tar.gz-chiffre » ; il s'agit *à priori* d'une archive « tar » compressée et chiffrée. Cela orientera nos recherches pour le décryptage.

2.3.2 **Le fichier FTP**

La même commande Wireshark de suivi du flux TCP permet d'obtenir le fichier transféré :



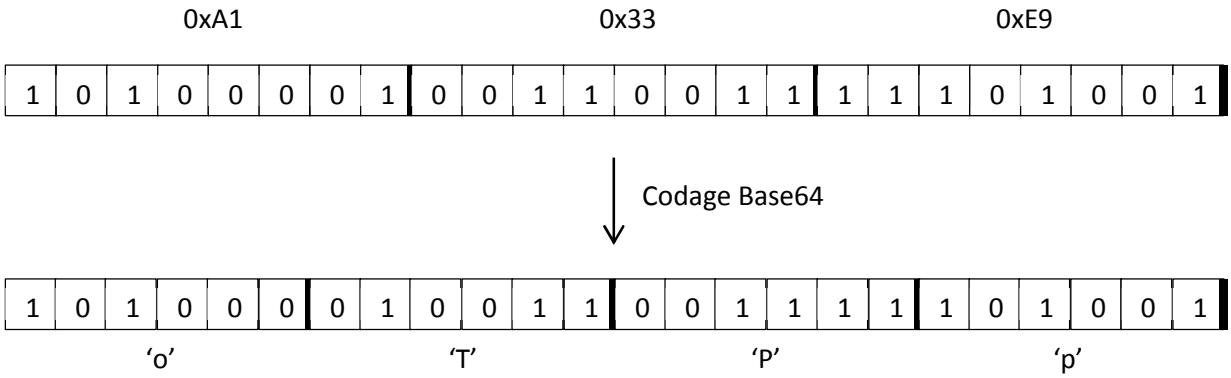
Il suffit de le sauvegarder sur le disque en « sstic.tar.gz-chiffre ».

Au passage, le jeu de caractères utilisé (alphanumérique, + et /) me fait immédiatement penser à un codage Base64.

2.3.3 **Le codage Base64**

Pour rappel, le codage Base64 est une transformation bijective qui associe 3 octets de 8 bits à 4 éléments de 6 bits. Ces 6 bits sont ensuite représentés par 64 caractères ASCII « imprimables » : 26 lettres minuscules, 26 lettres majuscules, 10 chiffres, « / » et « + ».

Par exemple, les 3 octets 0xA1, 0x33, 0xE9 seront codés en 4 caractères ‘oTPp’



L’avantage de ce codage est qu’il produit des fichiers qui ne comportent que du texte (et qui peuvent être traités par les outils qui ne travaillent qu’avec du texte, comme les mails). Le codage Mime utilisé pour les mails et le web peut utiliser le codage Base64.

Son inconvénient est qu’il produit des fichiers plus gros (dans un rapport 8/6ème)

Le lecteur intéressé par plus de détails trouvera de nombreux articles sur le web (par exemple : http://en.wikipedia.org/wiki/Base64)

2.3.4 **2.3.4 Décodage Base64**

Comme on peut le voir ci-dessous, le contenu du fichier « sstic.tar.gz-chiffre » ressemble à un codage Base64 :

```
# more sstic.tar.gz-chiffre
lscdhg9i6334AVND30wuytHRs54ffHW3++P2RyisiF5TpF6TirU43lNKjChOr0Ar
wtKUwx7JKRZ0wpRzrRbjUNlzpI5WRHEB8l8gJxu6noVAYbXCPIBCtJPlcvULcZLC
aF0glAf3dEpUxsTQ3+URUy4SPj7mXL2iRTPwyh7CTldU4fdL3h2Ydy/8M+lSnOd8
xgL4fapcLvyYk7k2yFN/3S+xbgSbvkhWPTtdhmdkAZ/5nKdJZdIG0aXEotnPOrIY
```

avec des lignes de 64 caractères :

```
# wc -L sstic.tar.gz-chiffre
64 sstic.tar.gz-chiffre
```

Pour vérifier qu’il s’agit bien d’un codage Base64, on peut tenter un décodage :

```
# base64 -d sstic.tar.gz-chiffre > sstic.tar.gz-chiffre.unbase64
```

Et pour être certain que notre supposition était correcte, il suffit de ré-encoder le fichier et de vérifier qu’il est identique à l’original :

```
# cat sstic.tar.gz-chiffre.unbase64 | base64 -w64 | diff -s sstic.tar.gz-chiffre -
Les fichiers sstic.tar.gz-chiffre et - sont identiques.
```

C’est bien le cas !

A partir de maintenant, nous travaillerons donc sur le fichier « sstic.tar.gz-chiffre.unbase64 ».

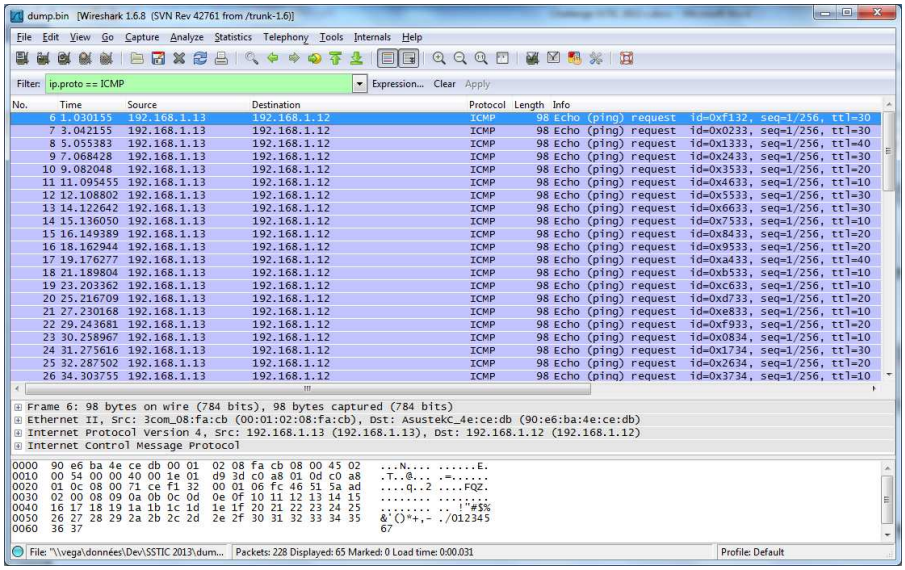
2.4 La série de pings

2.4.1 **Première analyse**

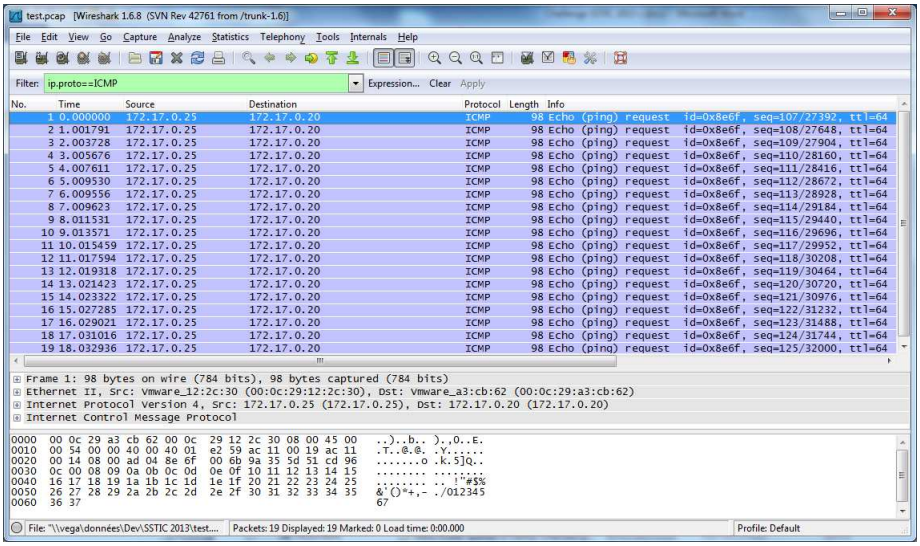
A la lecture du message texte, on se doute déjà que la clef AES qui sert à déchiffrer le fichier « sstic.tar.gz-chiffre.unbase64 » est cachée dans la série de pings enregistrée dans la capture réseau.

Il s'agit d'une technique de stéganographie : elle consiste à cacher un message de telle manière que sa présence soit insoupçonnable.

En appliquant le filtre de paquets « ip.proto == ICMP » à Wireshark, on isole les 65 pings :



A titre de comparaison, on trouvera ci-dessous la capture réseau d'un ping « standard » :



On remarque immédiatement 3 choses :

- Les pings sont espacés soit d'1 seconde, soit de 2 secondes alors qu'un ping standard génère des trames de manière régulière (1 seconde par défaut).
- Le champ TTL (Time To Live) n'est pas constant : 10, 20, 30, 40 et 1 pour la dernière trame
  Ce paramètre du protocole IP sert à limiter le nombre de passages dans des routeurs ; un ping standard utilise toujours la même valeur de TTL (64 par défaut).

- Le champ id varie de trame en trame, alors que ce n'est pas le cas pour un ping standard.

Je décide donc d'analyser ces trames de manière plus attentive.

En comparant les trames entre-elles de manière différentielle, je note les champs qui changent. La manipulation est très aisée avec Wireshark puisqu'il suffit de se déplacer de trame en trame et de noter les changements dans la fenêtre de visualisation en hexadécimal (en bas de l'écran). Par exemple, entre les paquets 7 et 8, les différences sont les suivantes :

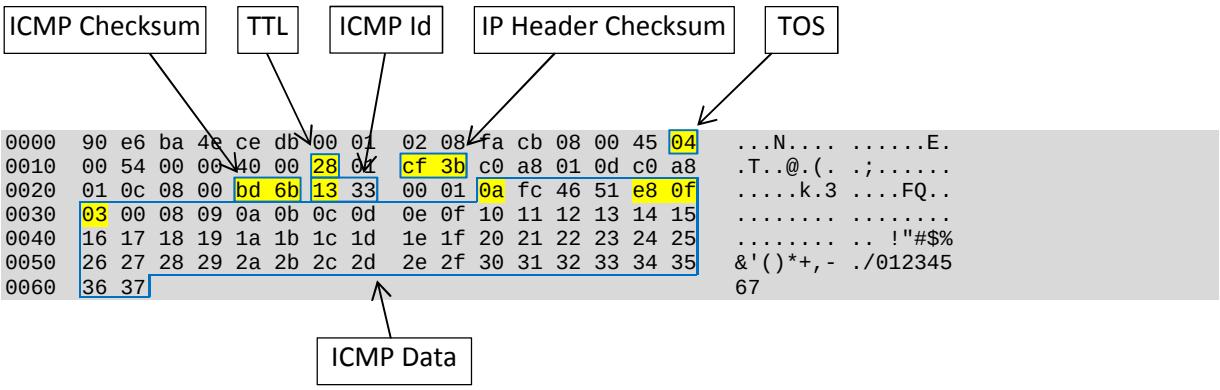
Trame n°7 :

```
0000  90 e6 ba 4e ce db 00 01  02 08 fa cb 08 00 45 02   ...N.... ......E.
0010  00 54 00 00 40 00 1e 01  d9 3d c0 a8 01 0d c0 a8   .T..@... .=......
0020  01 0c 08 00 7c 9f 02 33  00 01 08 fc 46 51 3c dc   ....|..3 ....FQ<.
0030  02 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

Trame n°8 :

```
0000  90 e6 ba 4e ce db 00 01  02 08 fa cb 08 00 45 04   ...N.... ......E.
0010  00 54 00 00 40 00 28 01  cf 3b c0 a8 01 0d c0 a8   .T..@.(. .;......
0020  01 0c 08 00 bd 6b 13 33  00 01 0a fc 46 51 e8 0f   .....k.3 ....FQ..
0030  03 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

En cliquant sur les données surlignées, Wireshark nous donne la fonction de ces champs :



Les champs « IP Header Checksum » et « ICMP Checksum » sont des sommes de contrôles. Il est donc normal qu'ils varient en fonction du contenu des trames.

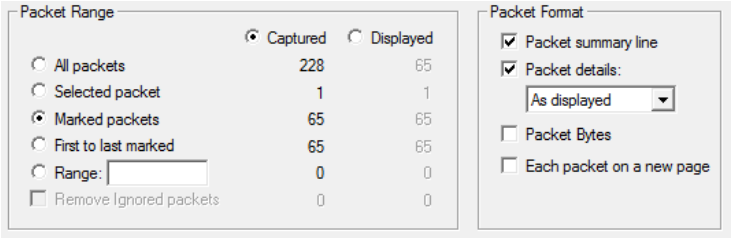
Le champ « ICMP Data » contient les données transmises par la commande ping. Il n'est pas surprenant que ces données varient de trame en trame, puisqu'un ping standard utilise ce champ pour transmettre un horodatage de la trame (« struct timeval » sur 8 octets). J'analyserai quand même ce champ en détail pour vérifier qu'il ne contient pas d'information cachée.

Par contre, les champs « TTL », « ICMP Id », « TOS » ne devraient pas varier. Je suspecte une utilisation détournée de ces champs pour transporter des données ; Associé aux variations détectées plus haut dans le séquencement des trames, je commence à entrevoir le canal caché…

2.4.2 **Analyse détaillée des champs forgés**

Je décide d'analyser le séquencement des trames et les variations des champs « TTL », « ICMP Id », « TOS » et « ICMP Data » avec un tableur.

Pour cela, les trames ICMP sont tout d'abord exportées au format texte :



Cela nous donne un fichier texte qui contient pour chaque trame les informations ci-dessous. Chacun des champs que nous souhaitons analyser a été surligné :

```
No.     Time        Source                Destination           Protocol Length Info
      6 1.030155    192.168.1.13          192.168.1.12          ICMP     98     Echo (ping)
request  id=0xf132, seq=1/256, ttl=30

Frame 6: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
    Arrival Time: Mar 18, 2013 12:35:34.175461000 Paris, Madrid
    Epoch Time: 1363606534.175461000 seconds
    [Time delta from previous captured frame: 0.029843000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 1.030155000 seconds]
Ethernet II, Src: 3com_08:fa:cb (00:01:02:08:fa:cb), Dst: AsustekC_4e:ce:db
(90:e6:ba:4e:ce:db)
Internet Protocol Version 4, Src: 192.168.1.13 (192.168.1.13), Dst: 192.168.1.12
(192.168.1.12)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x02 (DSCP 0x00: Default; ECN: 0x02: ECT(0) (ECN-Capable
Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..10 = Explicit Congestion Notification: ECT(0) (ECN-Capable Transport) (0x02)
    Total Length: 84
    Identification: 0x0000 (0)
    Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 30
    Protocol: ICMP (1)
    Header checksum: 0xd93d [correct]
        [Good: True]
        [Bad: False]
    Source: 192.168.1.13 (192.168.1.13)
    Destination: 192.168.1.12 (192.168.1.12)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x71ce [correct]
    Identifier (BE): 61746 (0xf132)
    Identifier (LE): 13041 (0x32f1)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    Data (56 bytes)

0000  06 fc 46 51 5a ad 02 00 08 09 0a 0b 0c 0d 0e 0f   ..FQZ...........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                          01234567
      Data: 06fc46515aad020008090a0b0c0d0e0f1011121314151617...
      [Length: 56]
```

Ensuite, chacun de ces champs est extrait du fichier pour être injecté dans un tableur. Chacune des lignes correspond à une trame :

```
cat ping.txt | awk '
        /192\.168\.1\.13.*ICMP/         { printf("%s\t",$1); }
        /Epoch Time/                    { printf("%s\t",$3) }
        /Differentiated Services Field/ { printf("%s\t",substr($4,4,1)) }
        /Time to live/                  { printf("%s\t",$4) }
        /Identifier \(LE\)/             { printf("%s\t",$3) }
        /Data:/                         { printf("%s\n",substr($2,1,16)) }
'

6       1363606534.175461000    2       30      13041   06fc46515aad0200
7       1363606536.187461000    2       30      13058   08fc46513cdc0200
8       1363606538.200689000    4       40      13075   0afc4651e80f0300
9       1363606540.213734000    4       30      13092   0cfc4651dc420300
10      1363606542.227354000    2       20      13109   0efc465110780300
11      1363606544.240761000    4       10      13126   10fc46516fac0300
…
```

On a maintenant un tableur qui compte 65 lignes de données brutes :

	A	B	C	D	E	F
1	N°Trame	Epoch	TOS	TTL	ICMP Id	ICMP Data
2	6	1363606534,17546	2	30	13041	06fc46515aad0200
3	7	1363606536,18746	2	30	13058	08fc46513cdc0200
4	8	1363606538,20068	4	40	13075	0afc4651e80f0300
5	9	1363606540,21373	4	30	13092	0cfc4651dc420300
6	10	1363606542,22735	2	20	13109	0efc465110780300
7	11	1363606544,24076	4	10	13126	10fc46516fac0300
8	12	1363606545,25410	2	30	13141	11fc465192e00300
9	13	1363606547,26794	2	30	13158	13fc4651a2160400
10	14	1363606548,28135	4	10	13173	14fc4651034b0400
11	15	1363606549,29469	4	20	13188	15fc46511d7f0400
12	16	1363606551,30825	4	20	13205	17fc46510fb40400
13	17	1363606552,32158	2	40	13220	18fc465126e80400
14	18	1363606554,33511	4	10	13237	1afc4651fa1c0500
15	19	1363606556,34866	4	10	13254	1cfc4651f1510500
16	20	1363606558,36201	2	20	13271	1efc465114860500

2.4.2.1 Horodatage des trames

Nous avons déjà remarqué que les trames étaient espacées de 1 ou de 2 secondes. Vérifions cela sur l'ensemble des trames, en ajoutant une colonne qui calcule la différence de temps avec la trame précédente (sauf pour la première bien-sûr) :

	A	B	C	D	E	F	G
1	N°Trame	Epoch	TOS	TTL	ICMP Id	ICMP Data	InterTrame
2	6	1363606534,17546	2	30	13041	06fc46515aad0200	
3	7	1363606536,18746	2	30	13058	08fc46513cdc0200	2
4	8	1363606538,20068	4	40	13075	0afc4651e80f0300	2
5	9	1363606540,21373	4	30	13092	0cfc4651dc420300	2
6	10	1363606542,22735	2	20	13109	0efc465110780300	2
7	11	1363606544,24076	4	10	13126	10fc46516fac0300	2
8	12	1363606545,25410	2	30	13141	11fc465192e00300	1
9	13	1363606547,26794	2	30	13158	13fc4651a2160400	2
10	14	1363606548,28135	4	10	13173	14fc4651034b0400	1
11	15	1363606549,29469	4	20	13188	15fc46511d7f0400	1
12	16	1363606551,30825	4	20	13205	17fc46510fb40400	2
13	17	1363606552,32158	2	40	13220	18fc465126e80400	1
14	18	1363606554,33511	4	10	13237	1afc4651fa1c0500	2
15	19	1363606556,34866	4	10	13254	1cfc4651f1510500	2
16	20	1363606558,36201	2	20	13271	1efc465114860500	2

On constate ainsi que c'est vrai pour toutes les trames : à 2/100ème de secondes près, elles sont toutes espacées de 1 ou 2 secondes.

Ce délai inter-trame peut donc servir à coder 1 bit de donnée (2 valeurs possibles), de manière cachée.

A partir de maintenant, seule la valeur arrondie à la seconde près sera affichée.

2.4.2.2 Le champ ICMP Id

Ce champ est utilisé par le protocole ICMP pour faire le rapprochement entre une trame émise et sa réponse.

On constate que cette valeur est strictement croissante. Il me semble donc naturel de visualiser la manière dont elle progresse. Pour cela, une colonne qui calcule la différence entre 2 trames est ajoutée (cela n'est bien-sûr pas possible pour la première trame) :

	A	B	C	D	E	F	G	H
1	N°Trame	Epoch	TOS	TTL	ICMP Id	ICMP Data	InterTrame	Delta/ICMPId
2	6	1363606534,17546	2	30	13041	06fc46515aad0200		
3	7	1363606536,18746	2	30	13058	08fc46513cdc0200	2	17
4	8	1363606538,20068	4	40	13075	0afc4651e80f0300	2	17
5	9	1363606540,21373	4	30	13092	0cfc4651dc420300	2	17
6	10	1363606542,22735	2	20	13109	0efc465110780300	2	17
7	11	1363606544,24076	4	10	13126	10fc46516fac0300	2	17
8	12	1363606545,25410	2	30	13141	11fc465192e00300	1	15
9	13	1363606547,26794	2	30	13158	13fc4651a2160400	2	17
10	14	1363606548,28135	4	10	13173	14fc4651034b0400	1	15
11	15	1363606549,29469	4	20	13188	15fc46511d7f0400	1	15
12	16	1363606551,30825	4	20	13205	17fc46510fb40400	2	17
13	17	1363606552,32158	2	40	13220	18fc465126e80400	1	15
14	18	1363606554,33511	4	10	13237	1afc4651fa1c0500	2	17
15	19	1363606556,34866	4	10	13254	1cfc4651f1510500	2	17
16	20	1363606558,36201	2	20	13271	1efc46511486050	2	17

On constate que la valeur obtenue est fortement corrélée au délai inter-trame : 15 et 17 pour des délais inter-trame respectifs de 1 et 2 secondes.

Ce champ n'apporte donc aucune information supplémentaire par rapport au délai inter-trame. Il ne contient aucune information cachée et, pour plus de lisibilité, les colonnes correspondantes seront masquées dans tout ce qui suit.

2.4.2.3 Le champ ICMP Data

Nous allons vérifier si le champ « ICMP Data » contient les données classiques du ping ou s'il peut contenir des informations utiles.

Un ping standard utilise ce champ pour horodater les trames en y stockant une structure « timeval » dont nous pouvons récupérer la définition dans « sys/time.h » :

```
struct timeval {
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* and microseconds */
};
```

Nous allons donc décoder le champ data en conséquence en ajoutant une colonne dont la formule convertit 2 mots de 4 octets (LSB en premier) en une valeur comparable à l'horodatage enregistré par « tcpdump » dans la trace réseau (Epoch) :

```
=HEXDEC(STXT(F2;7;2)&STXT(F2;5;2)&STXT(F2;3;2)&STXT(F2;1;2))+HEXDEC(STXT(F2;15;2)&STXT(F2;13;2
)&STXT(F2;11;2)&STXT(F2;9;2))/1000000
```

	A	B	C	D	F	G	H	I	J
1	N°Trame	Epoch	TOS	TTL	ICMP Data	InterTrame	Delta/ICMPId	timeval	Delta/Epoch
2	6	1363606534,17546	2	30	06fc46515aad0200			1363606534,17545	-0,00001
3	7	1363606536,18746	2	30	08fc46513cdc0200	2	17	1363606536,18745	-0,00001
4	8	1363606538,20068	4	40	0afc4651e80f0300	2	17	1363606538,20068	0,00000
5	9	1363606540,21373	4	30	0cfc4651dc420300	2	17	1363606540,21372	-0,00001
6	10	1363606542,22735	2	20	0efc465110780300	2	17	1363606542,22734	-0,00001
7	11	1363606544,24076	4	10	10fc46516fac0300	2	17	1363606544,24075	-0,00001
8	12	1363606545,25410	2	30	11fc465192e00300	1	15	1363606545,25410	0,00000
9	13	1363606547,26794	2	30	13fc4651a2160400	2	17	1363606547,26794	0,00000
10	14	1363606548,28135	4	10	14fc4651034b0400	1	15	1363606548,28135	0,00000
11	15	1363606549,29469	4	20	15fc46511d7f0400	1	15	1363606549,29468	-0,00001
12	16	1363606551,30825	4	20	17fc46510fb40400	2	17	1363606551,30824	-0,00001
13	17	1363606552,32158	2	40	18fc465126e80400	1	15	1363606552,32157	-0,00001
14	18	1363606554,33511	4	10	1afc4651fa1c0500	2	17	1363606554,33510	-0,00001
15	19	1363606556,34866	4	10	1cfc4651f1510500	2	17	1363606556,34866	0,00000
16	20	1363606558.36201	2	20	1efc465114860500	2	17	1363606558.36200	-0.00001

En ajoutant une colonne qui calcule la différence entre la valeur « timeval » ci-dessus et « Epoch », on constate que ces 2 champs sont corrélés. Il n'y a donc aucune information cachée dans le champ « ICMP Data ».

A partir de maintenant, et pour plus de clarté, les colonnes « Epoch », « ICMP Data », « timeval » et « Delta/Epoch » seront masquées puisqu'elles nous sont inutiles.

2.4.2.4 Le champ TOS

Ce champ est utilisé par le protocole IP pour gérer le type de service transporté (Type Of service). Pour faire simple, il peut être utilisé pour gérer la priorité entre différentes trames. Son usage a été étendu au fil du temps pour gérer les notifications de congestion et les classes de services (data, voix, …).

Il n'y a aucune raison valable pour que sa valeur change entre les différents pings. Je suis donc certain que ce champ véhicule une information cachée.

Sa valeur étant « 2 » ou « 4 » (sauf pour la dernière trame), il ne peut coder qu'un seul bit.

2.4.2.5 Le champ TTL

Ce champ du protocole IP sert à limiter le nombre de passages dans des routeurs ; un ping standard utilise toujours la même valeur de TTL (64 par défaut). On a donc la certitude que ce champ véhicule une information cachée.

Les valeurs de ce champ étant 10, 20, 30 ou 40 sauf pour la dernière trame (1), il code certainement 2 bits.

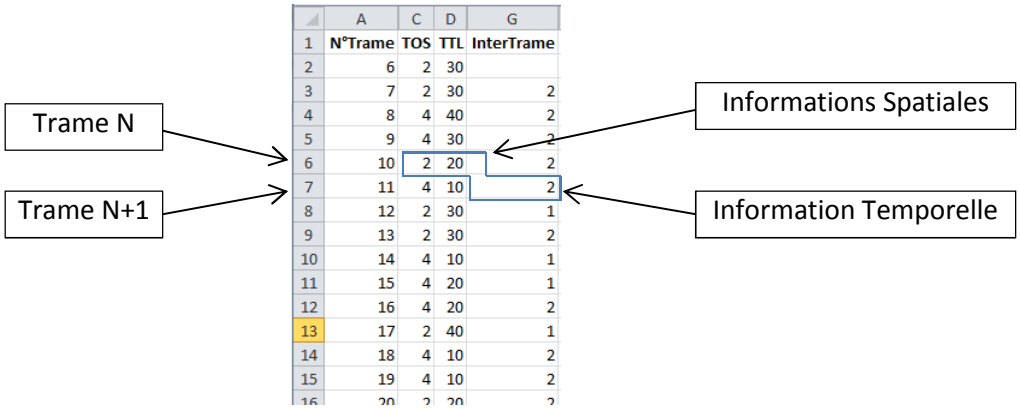
2.4.3 Le canal caché

Nous venons de voir que des informations sont véhiculées par les pings de manière cachée de la manière suivante :

- 1 bit codé par le délai entre les trames (information temporelle)
- 1 bit codé dans le champ TOS (information spatiale)
- 2 bits codés dans le champ TTL (information spatiale)

Il est clair que la première trame ne peut pas encore contenir d'information temporelle puisqu'il n'y a pas de ping qui la précède. A contrario, les informations « spatiales » peuvent commencer dès la première trame.

Il y a donc certainement un « déphasage » d'une trame entre l'information temporelle et les informations « spatiales ».



	A	C	D	G
1	N°Trame	TOS	TTL	InterTrame
2	6	2	30	
3	7	2	30	2
4	8	4	40	2
5	9	4	30	2
6	10	2	20	2
7	11	4	10	2
8	12	2	30	1
9	13	2	30	2
10	14	4	10	1
11	15	4	20	1
12	16	4	20	2
13	17	2	40	1
14	18	4	10	2
15	19	4	10	2
16	20	2	20	2

Cela explique également pourquoi les informations « spatiales » ne sont pas présentes dans la dernière trame (valeur 0 pour TOS et 1 pour TTL).

Nos 65 trames donnent donc 64 * 4 bits d'information. La clef AES ferait donc 256 bits ce qui semble cohérent.

Par contre, nous ne connaissons pas la correspondance symboles/données :

- Pour le délai inter-trame, 1 seconde code-t-elle un 0 ou un 1 ? (2 possibilités)
- Pour le champ TOS, '2' code-t-il un 0 ou un 1 ? (2 possibilités)
- Pour le champ TTL, comment les valeurs 10, 20, 30, 40 codent-elles 2 bits d'information ? (Factorielle 4 = 24 permutations possibles)

De la même manière, nous ne connaissons pas l'ordre dans lequel ces informations doivent être rangées (Factorielle 4 = 24 permutations de 4 bits sont possibles).

Nous allons donc partir sur l'hypothèse suivante, et tester plus loin les différentes possibilités :

- Délai inter-trame : 1 seconde => 0 ; 2 secondes => 1
- Champ TOS : '2' => 0 et '4'=>1
- Champ TTL : '10'=>1, '20'=>2, '30'=>3 et '40'=>0

A partir de cela, nous modifions notre tableur pour afficher les 4 bits correspondant à chaque ligne en ajoutant les formules nécessaires. Une colonne présente également le résultat en hexadécimal. Par exemple, pour la ligne n°3 :

```
Bit-délai       =SI(G3=1; 0; 1)
Bit-TOS         =SI(C3=2;0;1)
Bits-TTL        ==CHOISIR(D3/10;"01";"10";"11";"00")
Hexa            ==BINHEX(L4&M3&N3)
```

	A	C	D	G	K	L	M	N	O
1	N°Trame	TOS	TTL	InterTrame		bit-Temporel	bit-TOS	bits-TTL	Hexa
2	6	2	30				0	11	B
3	7	2	30	2		1	0	11	B
4	8	4	40	2		1	1	00	C
5	9	4	30	2		1	1	11	F
6	10	2	20	2		1	0	10	A
7	11	4	10	2		1	1	01	5
8	12	2	30	1		0	0	11	B
9	13	2	30	2		1	0	11	3
10	14	4	10	1		0	1	01	5
11	15	4	20	1		0	1	10	E
12	16	4	20	2		1	1	10	6
13	17	2	40	1		0	0	00	8
14	18	4	10	2		1	1	01	D
15	19	4	10	2		1	1	01	D
16	20	2	20	2		1	0	10	A

1010 = 0xA

Afin que les données de ce canal caché soient facilement exploitables dans un programme, nous ajoutons une formule qui concatène l'ensemble de ces données hexadécimales avec un format approprié :

```
="0x"&02&03&","&"0x"&04&05&","&"0x"&06&07&","&"0x"&08&09&","&"0x"&010&011&","&"0x"&012&013&","&"0x"&014&015&","&"0x"&016&017&","&"0x"&018&019&","&"0x"&020&021&","&"0x"&022&023&","&"0x"&024&025&","&"0x"&026&027&","&"0x"&028&029&","&"0x"&030&031&","&"0x"&032&033&","&"0x"&034&035&","&"0x"&036&037&","&"0x"&038&039&","&"0x"&040&041&","&"0x"&042&043&","&"0x"&044&045&","&"0x"&046&047&","&"0x"&048&049&","&"0x"&050&051&","&"0x"&052&053&","&"0x"&054&055&","&"0x"&056&057&","&"0x"&058&059&","&"0x"&060&061&","&"0x"&062&063&","&"0x"&064&065
```

Ce qui nous permet d'obtenir une chaîne de 32 octets directement utilisable dans un tableau en C !

64	68	2	40	2	1	0	00	8
65	69	4	30	2	1	1	11	7
66	70	0	1	1	0			
67								
68	0xB8,0xCF,0xA5,0xB3,0x5E,0x68,0xDD,0xA9,0x21,0x7E,0x1D,0xFB,0x4E,0x7D,0x68,0xEC,0x1E,0xB8,0x19,0xF7,0xC2,0x4E,0xFB,0x4B,0x39,0x6A,0xDD,0xBC,0x6D,0x61,0xDE,0x87				

En conclusion, la série de trame de pings nous a permis d'obtenir la clef AES annoncée dans le message texte. Toutefois, à ce stade, cette clef n'est valable qu'avec les limites suivantes :

- Ordre des bits à trouver : 24 possibilités différentes.
- Affectation des symboles à trouver : 2 possibilités pour le bit temporel, 2 pour TOS et 24 pour TTL (en fait, pour être précis, il n'y a réellement que 12 permutations possibles pour TTL puisque les permutations des bits du quartet mentionnées ci-dessus couvrent déjà la moitié des cas de ce champ ; par souci de simplicité les 24 permutations seront testées).

Ce qui donne un ensemble de 24*2*24 = 1152 clefs possibles.

Pour trouver la clef réellement utilisée, nous allons tester toutes ces possibilités dans le chapitre suivant.

2.4.4 **Comment les pings ont-ils été générés ?**

Nous avons vu que les informations concernant la clef AES étaient cachées dans les trames « ping », de manière temporelle et spatiale.

Pour coder chacun des 64 quartets de la clef, l'utilisateur a certainement utilisé la commande ping de la manière suivante (les champs TTL et TOS sont passés en paramètres) :

```
ping 172.17.0.20 -c 1 -t 30 -Q 2
```

Si l'utilisateur A avait voulu utiliser d'autres champs dans les trames, il aurait également pu utiliser un outil comme « scapy ». Cet outil permet de modifier les trames en profondeur. Par exemple, pour générer des trames identiques à celles du Challenge, on peut utiliser la commande suivante :

```
>>>class timeval(Packet): fields_desc = [ IntField("sec", 0), IntField("micro", 0) ]
...
>>>send(IP(dst="172.17.0.20",tos=2,ttl=30)/ICMP(id=13041)/
       timeval(sec=int(time.time()),micro=int((time.time()-int(time.time()))*1000000))/
       "\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a
       \x1b\x1c\x1d\x1e\x1f 1234567")
```

Que l'on pourrait simplifier de la manière suivante en supprimant les champs superflus (ICMP Id et ICMP Data) :

```
>>>send(IP(dst="172.17.0.20",tos=2,ttl=30)/ICMP())
```

Pour envoyer la totalité de la clef, et reproduire à l'identique l'ensemble des trames du Challenge, on peut utiliser le script ci-dessous. Pour ce script, c'est la clef finale, telle que trouvée dans le chapitre suivant qui est utilisée. De cette manière le script est conforme aux données du Challenge :

```
#!/bin/bash

Key="DD8CF2D52E69AAFB734E3ACD0E4A69E83ED93BC4870ECD0D5B6FAAD86A63AE94"
IP=192.168.1.13

# Verification de la longueur de la clef
if [ ${#Key} != 64 ]; then
        printf "La clef doit comporter 64 caracteres !\n"
        exit
fi

# Parcourt tous les quartets de la clef
for i in {0..63}
do
        # Isole le quartet a envoyer
        hex=${Key:$i:1}
        dec=$((0x$hex))

        # Extraction des bits a envoyer et preparation des parametres
        # Bit 3 : Delai inter-trame
        if [ $(($dec & 8)) -eq 8 ]; then
                delai=2
        else
                delai=1
        fi

        # Bit 1 et 2 : TTL
        case $(($dec & 6)) in
                0) ttl=40;;
                2) ttl=10;;
                4) ttl=30;;
                6) ttl=20;;
        esac

        # Bit 0 : TOS
        if [ $(($dec & 1)) -eq 1 ]; then
                tos=2
        else
                tos=4
        fi

        # Affichage message de progression
        printf "Envoi du quartet %s : " $hex
        printf "delai=%d ttl=%d tos=%d\n" $delai $ttl $tos

        # Envoi de l'information spatiale
        ping $IP -t $ttl -Q $tos -c 1 >/dev/null
        # Envoi de l'information temporelle
        sleep $delai
done

# Envoi de la trame de fin (ttl=1 et tos=0)
ping $IP -c 1 -t 1 -Q 0
```

3. Décryptage du fichier AES

3.1 Informations connues et méthode

A ce stade, nous disposons des informations suivantes :

- Un fichier « sstic.tar.gz-chiffre.unbase64 » qui est à priori une archive compressée ;
- Un vecteur d'initialisation AES : 76C128D46A6C4B15B43016904BE176AC ;
- Une clef de 64*4 bits aux permutations de symboles près, ce qui représente en fait un ensemble de 1152 clefs ;

Comme nous ne connaissons pas encore la clef de manière précise, nous parlons de « décryptage ».

Pour trouver la bonne clef AES, **nous allons tester toutes les permutations** évoquées précédemment pour trouver celles qui nous donnent un fichier « tar.gz ».

Un fichier « .gz » est un fichier compressé dont l'en-tête est parfaitement connu : les 2 premiers octets sont 0x1f, 0x8b et probablement 0x08 pour le troisième (méthode de compression). Le lecteur intéressé par plus de détails pourra consulter la page http://www.gzip.org/zlib/rfc-gzip.html#header-trailer

La méthode de chiffrement AES est telle qu'il est inutile de chiffrer l'ensemble du fichier pour vérifier si avons trouvé le bon en-tête : nous n'allons travailler qu'avec les 256 premiers octets du fichier chiffré.

3.2 Le chiffrement AES

Cette méthode de chiffrement a été conçue en 2000 à partir des travaux réalisés par 2 experts en cryptographie : Joan Daemen et Vincent Rijmen.
En simplifiant à l'extrême, elle consiste à réaliser des permutations et substitutions sur des blocs de données.

A ce jour, cette méthode de chiffrement n'a pas été cassée, sauf dans certains cas très précis.

Le vecteur d'initialisation (IV) permet de démarrer le chiffrement du premier bloc de données. Il permet de rajouter une dose de hasard dans le chiffrement, mais surtout, il permet de faire en sorte que les mêmes données ne donneront pas le même résultat après chiffrement dès lors que l'IV est différent. Pour cette raison, il est plus prudent d'utiliser un IV différent à chaque chiffrement. Cet IV peut être transmis en clair, ce qui est d'ailleurs le cas pour le Challenge.

Il existe différentes sous-méthodes AES pour la relation entre les différents blocs de données. Certaines chiffrent les données par bloc indépendants (2 blocs identiques produiront le même résultat) alors que d'autres utilisent des chainages entre les blocs. Pour commencer, nous utiliserons la méthode la plus courante : le Cypher Bloc Chaining (CBC).

Le lecteur intéressé par plus de détails pourra consulter l'article suivant :
http://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_%28cryptographie%29#IV_.E2.80.93_Vecteur_initial

Un autre paramètre du chiffrement AES concerne la manière de traiter le dernier bloc de données. Si ce bloc n'est pas complet (cela dépend de la taille du fichier à chiffrer), il faut utiliser une méthode de remplissage. Comme cela ne concerne que la fin du fichier, nous rechercherons la méthode de remplissage utilisée en dernier.

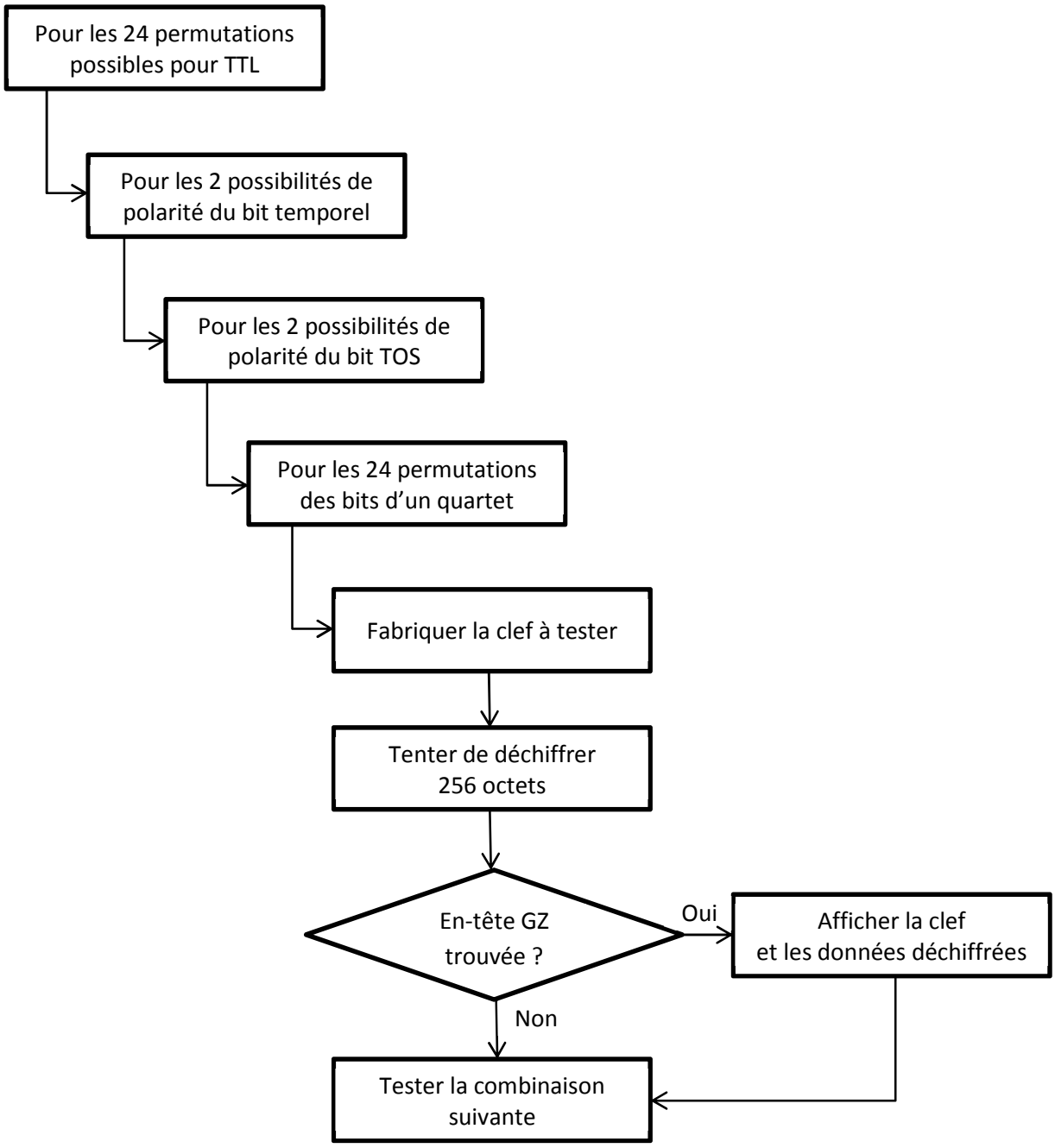
Déchiffrer des données en C# est assez simple ; un objet « AesCryptoServiceProvider » existe et le code suivant peut être utilisé. Il suffit de fournir à cette fonction les données, la clef et l'IV :

```
public static byte[] AesDechiffre(byte[] DataToDecrypt, byte[] Key, byte[] IV)
{
        AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
        Provider.Key = Key;
        Provider.Mode = CipherMode.CBC;
        Provider.Padding = PaddingMode.None;
        Provider.IV = IV;

        MemoryStream ms = new MemoryStream(DataToDecrypt);
        CryptoStream cs = new CryptoStream(ms, Provider.CreateDecryptor(Provider.Key,
                                                Provider.IV), CryptoStreamMode.Read);
        byte[] DecryptedData = new byte[DataToDecrypt.Length];
        var byteCount = cs.Read(DecryptedData, 0, DataToDecrypt.Length);
        Array.Resize(ref DecryptedData, byteCount);
        return DecryptedData;
}
```

3.3 Recherche de la clef

3.3.1 **Algorithme**

La méthode de recherche évoquée plus haut nous donne l'algorithme de recherche suivant :



3.3.2 **Permutations sur le champ TTL**

Le champ TTL peut contenir les symboles 10, 20, 30, 40, mais nous ne savons pas à quelle valeur correspond chacun de ces symboles. La clef que nous avons constituée dans le premier chapitre fait une hypothèse mais il y a peu de chances que ce soit la bonne… Nous devons donc essayer toutes les combinaisons possibles. Par exemple :

- Symboles 10,20,30,40 => Valeurs 0,1,2,3
- Symboles 10,20,30,40 => Valeurs 0,1,3,2

- Symboles 10,20,30,40 => Valeurs 0,2,1,3
- …

Le code suivant est utilisé pour tester toutes ces permutations. Il y en a 24 (Factorielle 4) Cette fonction accepte en entrée la clef à modifier et le numéro de la permutation à appliquer à la clef (de 0 à 23) :

```
private void TestPermutationTTL(ref Byte[] Key, Byte iPermut)
{
    Byte[][] Swaps = new Byte[][]
    {
        new Byte[4] {0,1,2,3},
        new Byte[4] {0,1,3,2},
        new Byte[4] {0,2,1,3},
        new Byte[4] {0,2,3,1},
        new Byte[4] {0,3,1,2},
        new Byte[4] {0,3,2,1},

        new Byte[4] {1,0,2,3},
        new Byte[4] {1,0,3,2},
        new Byte[4] {1,2,0,3},
        new Byte[4] {1,2,3,0},
        new Byte[4] {1,3,0,2},
        new Byte[4] {1,3,2,0},

        new Byte[4] {2,0,1,3},
        new Byte[4] {2,0,3,1},
        new Byte[4] {2,1,0,3},
        new Byte[4] {2,1,3,0},
        new Byte[4] {2,3,1,0},
        new Byte[4] {2,3,0,1},

        new Byte[4] {3,0,1,2},
        new Byte[4] {3,0,2,1},
        new Byte[4] {3,1,0,2},
        new Byte[4] {3,1,2,0},
        new Byte[4] {3,2,1,0},
        new Byte[4] {3,2,0,1}
    };

    // Pour tous les octets de la clef
    for (int i = 0; i < Key.Length; i++)
    {
        // Pour chacun des quartets
        Key[i] = (Byte)((Key[i] & ~0x03) | Swaps[iPermut][Key[i] & 0x03]);
        Key[i] = (Byte)((Key[i] & ~0x30) | (Swaps[iPermut][(Key[i]>>4) & 0x03]<<4));
    }
}
```

3.3.3 **Polarité du bit temporel et du bit TOS**

Nous ne connaissons pas la polarité du bit temporel et du bit TOS. Nous allons donc tester les 2 valeurs possibles pour chacun de ces bits.

La fonction suivante est utilisée pour appliquer un opérateur XOR à chacun des octets de la clef. Pour inverser le bit temporel, il suffit de lui passer un masque Xor égal à 0x88. De cette manière les bits des 2 quartets seront changés en même temps. Pour TOS, il faut utiliser la valeur 0x44.

```
private void TestPolarite(ref Byte[] Key, Byte XorMask)
{
    // Applique le mask Xor sur chacun des quartets de la clef
    for (int i = 0; i < Key.Length; i++)
        Key[i] = (Byte)(Key[i] ^ XorMask);
}
```

3.3.4 **Permutations sur les bits de chaque quartet de la clef**

Chaque trame ping transmet 4 bits. Nous ne savons cependant pas comment les ordonner.

Par exemple, on pourrait avoir les possibilités suivantes pour chaque quartet :

Bit Temporel	Bit TOS	Bit TTL 1	Bit TTL 1

= 1 quartet de la clef

Mais également :

Bit TOS	Bit TTL 1	Bit TTL 0	Bit Temporel

= 1 quartet de la clef

La fonction suivante va nous permettre de tester les 24 possibilités (Factorielle 4 permutations possibles sur 4 bits) :

```
private void TestPermutationBits(ref Byte[] Key, Byte iPermut)
{
    UInt16[] Swaps=new UInt16[24]
    {
        0x3210, // Pas de permutation
        0x3201,
        0x3120,
        0x3102,
        0x3021, // Bit2->Bit0, Bit1->Bit2, Bit0->Bit1
        0x3012,
        0x2310,
        0x2301,
        0x2130,
        0x2103,
        0x2021,
        0x2012,
        0x1320,
        0x1302,
        0x1230,
        0x1203,
        0x1032,
        0x1023,
        0x0321,
        0x0312,
        0x0231,
        0x0213,
        0x0132,
        0x0123 // Bit3->Bit0, Bit2->Bit1, Bit1->Bit2, Bit0->Bit3
    };
    // Permute les bits de chacun des quartets de la clef
    UInt16 sw = Swaps[iPermut];
    for (int i = 0; i < Key.Length; i++)
    {
        Byte c = 0;
        for (Byte iSrcBit = 0; iSrcBit < 4; iSrcBit++)
        {
            Byte iDestBit = (Byte)((sw >> (iSrcBit * 4)) & 0xf);
            // Quartet de poids faible
            if ((Key[i] & (1 << iSrcBit))!=0) c |= (Byte)(1 << iDestBit);
            // Quartet de poids fort
            if ((Key[i] & (1 << (iSrcBit + 4))) != 0) c |= (Byte)(1 << (iDestBit + 4));
        }
        Key[i] = c;
    }
}
```

3.3.5 **Programme réalisé**

Nous allons réaliser un programme unique pour l'ensemble des étapes du Challenge. Pour chacune de ces étapes, nous ajouterons un ou plusieurs boutons dans son interface graphique.

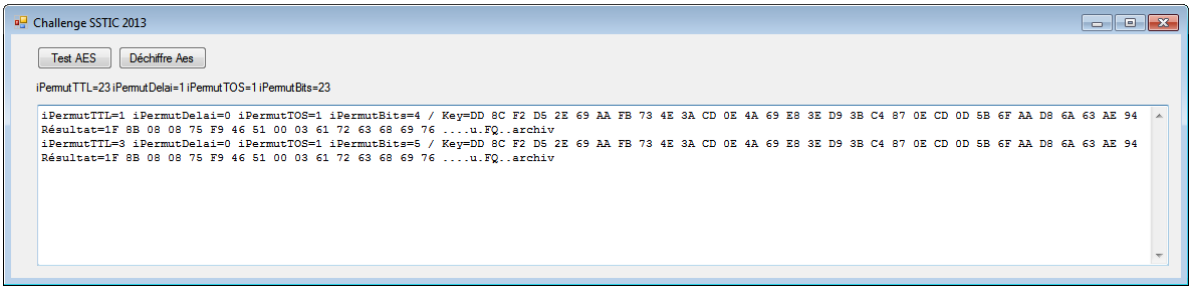
Le code suivant correspond à l'organigramme donné plus haut. Il est déclenché par un appui sur le bouton 'Test AES' :

```
byte[] IV = new byte[16] { 0x76, 0xC1, 0x28, 0xD4, 0x6A, 0x6C, 0x4B, 0x15, 0xB4, 0x30, 0x16,
0x90, 0x4B, 0xE1, 0x76, 0xAC };
private void BtnTestAes_Click(object sender, EventArgs e)
{
    byte[] Key = new byte[] {
       0xBB,0xCF,0xA5,0xB3,0x5E,0x68,0xDD,0xA9,0x21,0x7E,0x1D,0xFB,0x4E,0x7D,0x68,0xEC,
       0x1E,0xB8,0x19,0xF7,0xC2,0x4E,0xFB,0x4B,0x39,0x6A,0xDD,0xBC,0x6D,0x61,0xDE,0x87 };
    byte[] EncryptedData = { // 256 premiers octets du fichier sstic.tar.gz-chiffre
        0x96,0xc7,0x1d,0x86,0x0f,0x62,0xeb,0x7d,0xf8,0x01,0x53,0x43,0xdf,0x4c,0x2e,0xca,
        0xd1,0xd1,0xb3,0x9e,0x1f,0x7c,0x75,0xb7,0xfb,0xe3,0xf6,0x47,0x28,0xac,0x88,0x5e,
        0x53,0xa4,0x5e,0x93,0x8a,0xb5,0x38,0xde,0x53,0x4a,0x8c,0x28,0x4e,0xaf,0x40,0x2b,
        0xc2,0xd2,0x94,0xc3,0x1e,0xc9,0x29,0x16,0x74,0xc2,0x94,0x73,0xad,0x16,0xe3,0x50,
        0xd9,0x73,0xa4,0x8e,0x56,0x44,0x71,0x01,0xf2,0x5f,0x20,0x27,0x1b,0xba,0x9e,0x85,
        0x40,0x61,0xb5,0xc2,0x3c,0x80,0x42,0xb4,0x93,0xe5,0x72,0xf5,0x0b,0x71,0x92,0xc2,
        0x68,0x5d,0x20,0x94,0x07,0xf7,0x74,0x4a,0x54,0xc6,0xc4,0xd0,0xdf,0xe5,0x11,0x53,
        0x2e,0x12,0x3e,0x3e,0xe6,0x5c,0xbd,0xa2,0x45,0x33,0xf0,0xca,0x1e,0xc2,0x4e,0x57,
        0x54,0xe1,0xf7,0x4b,0xde,0x1d,0x98,0x77,0x2f,0xfc,0x33,0xe9,0x52,0x9c,0xe7,0x7c,
        0xc6,0x02,0xf8,0x7d,0xaa,0x5c,0x2e,0xfc,0x98,0x93,0xb9,0x36,0xc8,0x53,0x7f,0xdd,
        0x2f,0xb1,0x6e,0x04,0x9b,0xbe,0x48,0x56,0x3d,0x3b,0x5d,0x86,0x67,0x64,0x01,0x9f,
        0xf9,0x9c,0xa7,0x49,0x65,0xd2,0x06,0xd1,0xa5,0xc4,0xa2,0xd9,0xcf,0x3a,0xb2,0x18,
        0x33,0xf8,0x26,0xd0,0xec,0xdb,0xd1,0xe1,0x44,0x0f,0xe8,0xda,0x03,0xa7,0x09,0xe3,
        0x14,0x06,0x0c,0x34,0x98,0x54,0x56,0x16,0x8e,0xf0,0x80,0x88,0xd4,0x72,0x82,0xbf,
        0x11,0xea,0x56,0x91,0x33,0x70,0xa6,0x15,0xe5,0xc2,0x56,0xb1,0x20,0xaa,0xee,0x66,
        0xf5,0x87,0x46,0xf1,0x5c,0x87,0x3e,0x03,0x69,0xd9,0xae,0x6c,0x65,0xe7,0x69,0xde};

    Byte[] TestKey = new Byte[32];
    for (Byte iPermutTTL = 0; iPermutTTL < 24; iPermutTTL++)
    {
        for (Byte iPermutBitDelai = 0; iPermutBitDelai < 2; iPermutBitDelai++)
        {
            for (Byte iPermutBitTOS = 0; iPermutBitTOS < 2; iPermutBitTOS++)
            {
                for (Byte iPermutBits = 0; iPermutBits < 24; iPermutBits++)
                {
                    labelInfos.Text =   "iPermutTTL="+iPermutTTL.ToString()+
                                        " iPermutDelai=" + iPermutBitDelai.ToString() +
                                        " iPermutTOS=" + iPermutBitTOS.ToString() +
                                        " iPermutBits=" + iPermutBits.ToString();
                    labelInfos.Update();

                    // Préparation de la clef à tester
                    for (int i = 0; i < TestKey.Length; i++) TestKey[i] = Key[i];

                    // Permutation des symboles du TTL
                    TestPermutationTTL(ref TestKey, iPermutTTL);

                    // Polarité des bits Délai et TOS
                    Byte XorMask = (Byte)(((iPermutBitDelai==1)
                                  ? 0x88:0x00) | ((iPermutBitTOS==1) ? 0x44:0x00));
                    TestPolarite(ref TestKey, (Byte)XorMask);

                    // Permutation des bits du quartet
                    TestPermutationBits(ref TestKey, iPermutBits);

                    // Tentative de décryptage
                    AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
                    Provider.Key = TestKey;
                    Provider.Mode = CipherMode.CBC;
                    Provider.Padding = PaddingMode.None;
                    Provider.IV = IV;
                    MemoryStream ms = new MemoryStream(EncryptedData);
                    CryptoStream cs = new CryptoStream(ms,
                                  Provider.CreateDecryptor(Provider.Key, Provider.IV),
                                  CryptoStreamMode.Read);
                    byte[] DecryptedData = new byte[EncryptedData.Length];
                    var byteCount = cs.Read(DecryptedData, 0, EncryptedData.Length);
                    Array.Resize(ref DecryptedData, byteCount);

                    // En-tête GZIP trouvée ?
                    if (DecryptedData[0] == 0x1f && DecryptedData[1] == 0x8b)
                    {
                      // Affichage de la clef
                      textResult.Text+=labelInfos.Text+" / Key="+Dump(TestKey, 32, false);
                      textResult.Text+="\r\nRésultat="+Dump(DecryptedData, 16, true)+"\r\n";
                    }
                }
            }
        }
    }
}
```

La fonction Dump() est une fonction qui sera également utilisée par la suite ; elle permet d'afficher un tableau d'octet à la manière d'un « hexdump » :

```
string Dump(Byte[] Data, int Len, Boolean bAscii)
{
    string s = "";
    for (int i = 0; i < Len; i++) s += Data[i].ToString("X2")+" ";
    if (bAscii)
        for (int i = 0; i < Len; i++)
            s+=(Data[i] <= 0x20 || Data[i] > 0x7f) ?
                 "." : Encoding.UTF8.GetString(new Byte[] {Data[i]});
    return s;
}
```

On lance le programme en cliquant sur "Test AES". Il parcourt toutes les possibilités de clef à la recherche d'un fichier au format gzip. Moins de 2 secondes plus tard, il a trouvé la clef ! Les données affichées sont encourageantes puisqu'on voit apparaître le mot « archiv » en clair.



La clef trouvée est :

```
0xDD,0x8C,0xF2,0xD5,0x2E,0x69,0xAA,0xFB,0x73,0x4E,0x3A,0xCD,0x0E,0x4A,0x69,0xE8,
0x3E,0xD9,0x3B,0xC4,0x87,0x0E,0xCD,0x0D,0x5B,0x6F,0xAA,0xD8,0x6A,0x63,0xAE,0x94
```

Le lecteur attentif aura remarqué que le programme a trouvé 2 fois la même clef : cela est dû au fait que l'on a testé les 24 possibilités de permutations pour TTL alors que la moitié d'entre-elles sont déjà testées par les permutations sur le quartet.

3.3.6 **Format final du canal caché**

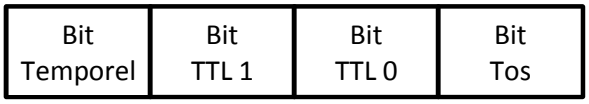
Le programme réalisé nous apprend également que cette clef correspond aux éléments suivants du canal caché :

Délai inter-Trame	**Bit temporel**
1 seconde	0
2 secondes	1

Valeur du TTL	**Bits 'TTL'**
10	01
20	11
30	10
40	00

Valeur du champ TOS	**Bit 'Tos'**
2	0
4	1

Utilisés dans l'ordre suivant :



= 1 quartet de la clef

Le message apportait une précision sur le canal caché : « 1 bit de canal caché temporel concaténé a 3 bits de canal caché non temporel ». En fait, cette information était superflue pour 2 raisons :

- Comme nous l'avons vu, la même information que celle du bit temporel était également véhiculée par l'ICMP Id. Ainsi, même si nous n'avions pas remarqué l'information temporelle, nous avions une 2ème chance avec l'ICMP Id…
- Nous avons trouvé 4 bits d'informations spatiales (en incluant l'ICMP Id) dans 64 trames (la 65ème trame ne comporte pas les informations spatiales). Nous pouvions en déduire la longueur de clef...

3.4 Déchiffrement, vérification et contenu du fichier AES

3.4.1 Déchiffrement du fichier AES

Maintenant que nous connaissons la clef, la fonction suivante est chargée de déchiffrer l'ensemble du fichier :

```
private void btnDechiffreAes_Click(object sender, EventArgs e)
{
    Byte[] Key = new Byte[32]
        {
          0xDD,0x8C,0xF2,0xD5,0x2E,0x69,0xAA,0xFB,0x73,0x4E,0x3A,0xCD,0x0E,0x4A,0x69,0xE8,
          0x3E,0xD9,0x3B,0xC4,0x87,0x0E,0xCD,0x0D,0x5B,0x6F,0xAA,0xD8,0x6A,0x63,0xAE,0x94
        };

    AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
    Provider.Key = Key;
    Provider.Mode = CipherMode.CBC;
    Provider.Padding = PaddingMode.ISO10126;
    Provider.IV = IV;

    FileStream InFs = new FileStream(@"sstic.tar.gz-chiffre.unbase64",
                FileMode.Open, FileAccess.Read);
    CryptoStream cs = new CryptoStream(InFs,
                Provider.CreateDecryptor(Provider.Key, Provider.IV), CryptoStreamMode.Read);
    Byte[] DecryptedData = new Byte[InFs.Length];
    var byteCount = cs.Read(DecryptedData, 0, DecryptedData.Length);
    InFs.Close();

    FileStream OutFs = new FileStream("sstic.tar.gz-chiffre.dechiffre",
                                                    FileMode.Create, FileAccess.Write);
    OutFs.Write(DecryptedData, 0, byteCount);
    OutFs.Close();
}
```

Comme indiqué plus avant, nous ne connaissons pas encore la méthode de remplissage des paquets (padding). Nous n'avons utilisé aucun remplissage pour la recherche de la clef puisque cela n'est utile que pour la fin du fichier où il faut remplir le dernier bloc à déchiffrer (pour qu'il soit plein).

Vu le faible nombre de méthodes de remplissage (5), je décide de les tester manuellement. Pour cela, un déchiffrage est tenté avec une méthode, et le fichier est vérifié comme indiqué dans le paragraphe suivant.

2 essais sont nécessaires pour obtenir la bonne méthode de remplissage : ISO10126

3.4.2 **Vérification du fichier déchiffré**

Le message texte inclus dans les trames nous donne le code MD5 du fichier déchiffré.

Pour le vérifier, il suffit de faire :

```
# md5sum sstic.tar.gz-chiffre.dechiffre
61c9392f617290642f9a12499de6b688  sstic.tar.gz-chiffre.dechiffre
```

et de comparer le résultat avec la valeur donnée par l'utilisateur de la machine A dans son message texte.

3.4.3 **Contenu du fichier AES**

Nous n'avons plus qu'à désarchiver le fichier :

```
# tar xvzf sstic.tar.gz-chiffre.dechiffre
archive/
archive/smp.py
archive/data
archive/s.ngr
archive/decrypt.py
```

Le contenu du répertoire obtenu est :

```
# ll
total 3208
-rw-rw-r-- 1 1000 1000   44053 2013-03-18 12:21 data
-rw-rw-r-- 1 1000 1000    1035 2013-03-18 12:21 decrypt.py
-rw-rw-r-- 1 1000 1000    1394 2013-03-18 12:21 smp.py
-rw-r--r-- 1 1000 1000 3226052 2013-03-18 12:21 s.ngr
```

Nous sommes arrivés au bout de cette première étape !

4. Décryptage SMP

4.1 Les données dont nous disposons

A ce stade, nous disposons du répertoire « archive » :

```
# ll
total 3208
-rw-rw-r-- 1 1000 1000   44053 2013-03-18 12:21 data
-rw-rw-r-- 1 1000 1000    1035 2013-03-18 12:21 decrypt.py
-rw-rw-r-- 1 1000 1000    1394 2013-03-18 12:21 smp.py
-rw-r--r-- 1 1000 1000 3226052 2013-03-18 12:21 s.ngr
```

Il contient apparemment 2 fichiers de données (data et s.ngr) et 2 scripts Python.

Ce sont les 2 scripts Python que nous allons analyser en premier

4.2 Analyse des scripts

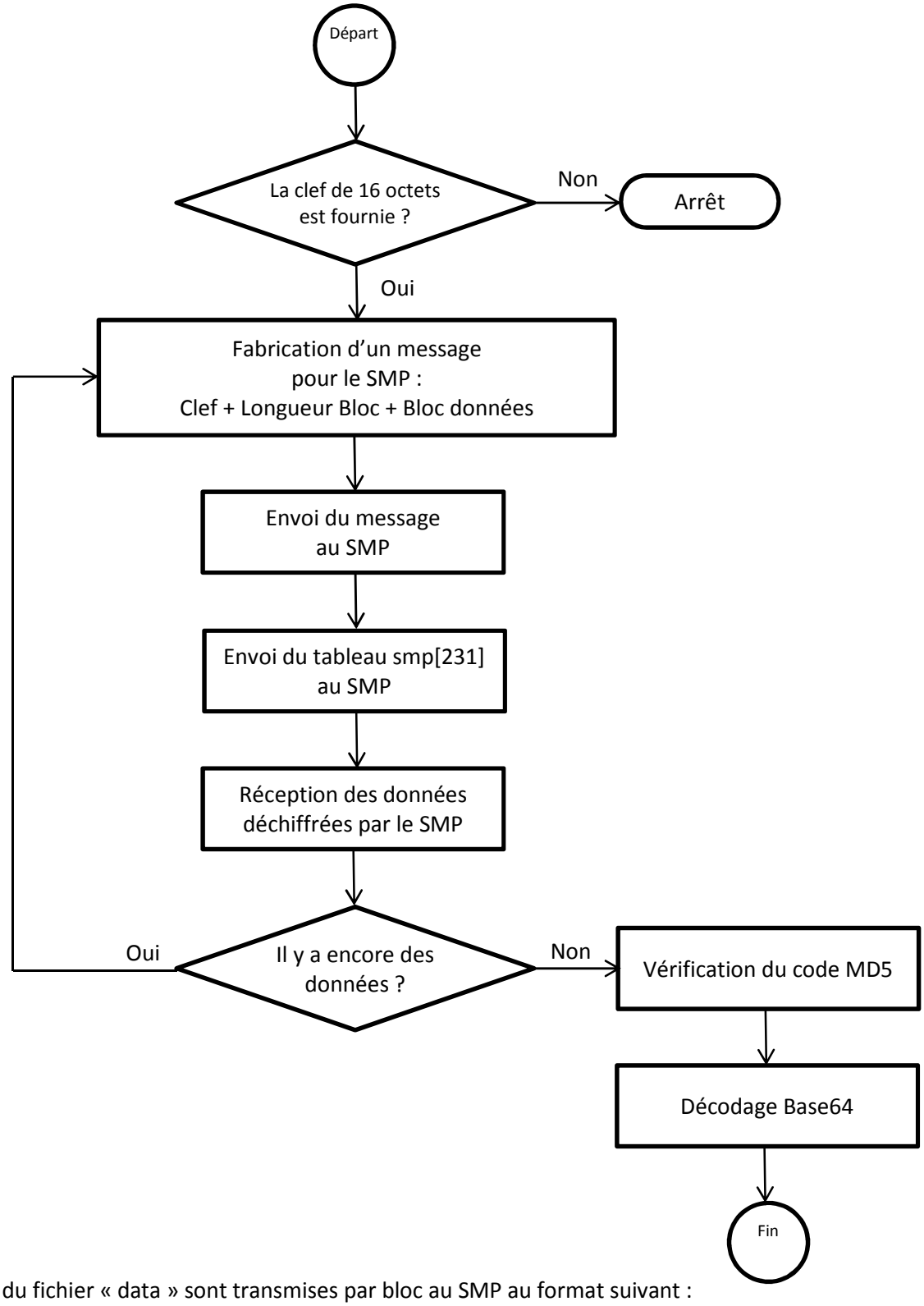
Le script « decrypt.py » n'est pas très long :

```
#!/usr/bin/python

import sys
import base64
import md5

import dev
import smp

if len(sys.argv) != 2:
  print("usage: %s key" % sys.argv[0])
  sys.exit(1)

key = int(sys.argv[1], 16)
key = [(key >> (i * 8)) & 0xff for i in range(16)]

result = []
d = open("data", "rb").read()
dev.init("sp.ngr")
for i in range(0, len(d), 224):
  smd = d[i : (i + 224)]
  smd = (key, len(smd), smd)
  dev.send_smd(smd)
  dev.send_smp(smp.smp)
  dev.start()
  dev.wait_finished()
  result = result + dev.get_data()

print "".join(result)
result_md5 = md5.new()
result_md5.update("".join(result))
result_md5 = result_md5.digest()
result_md5 = [ord(x) for x in result_md5]
target_md5 = "6c0708b3cf6e32cbae4236bdea062979"
target_md5 = [int(target_md5[x:(x + 2)], 16) for x in range(0, len(target_md5), 2)]
print(["%02x" % x for x in target_md5])
print(["%02x" % x for x in result_md5])
if result_md5 != target_md5:
  print("Bad key...")
  sys.exit(1)

result = base64.b64decode("".join(result))
d = open("atad", "wb")
d.write(result)
d.close()
sys.exit(0)
```

et on comprend rapidement qu'il est chargé de déchiffrer le fichier « data » à l'aide d'un dispositif spécialisé. Il vérifie également le code MD5 du fichier obtenu et effectue un décodage Base64 pour

terminer. Nous ne savons pas, à ce stade de quoi est constitué ce « dispositif spécialisé » mais nous l'appellerons « SMP », comme dans le script Python.

Le fichier « smp.py » définit juste un tableau « smp[] » de 231 octets qui est transmis au SMP.



Les données du fichier « data » sont transmises par bloc au SMP au format suivant :

Clef sur 16 octets	Longueur bloc de données sur 2 octets	Bloc de données (max 224 octets)

En retour, le SMP renvoie les données déchiffrées (sans enrobage) et ces données sont simplement concaténées par le script « decrypt.py ».

4.3 Le FPGA Xilinx

4.3.1 **Utilisation d'un FPGA**

Le répertoire « archive » contient le fichier « s.ngr ». Je décide de visualiser son contenu pour savoir à quoi il correspond :

```
hexdump -n 64 -C s.ngr
00000000  58 49 4c 49 4e 58 2d 58  44 42 20 30 2e 31 20 53  |XILINX-XDB 0.1 S|
00000010  54 55 42 20 30 2e 31 20  41 53 43 49 49 0a 58 49  |TUB 0.1 ASCII.XI|
00000020  4c 49 4e 58 2d 58 44 4d  20 56 31 2e 36 65 0a 24  |LINX-XDM V1.6e.$|
00000030  37 35 34 3d 7e 32 3f 33  26 62 64 61 68 21 6a 61  |754=~2?3&bdah!ja|
```

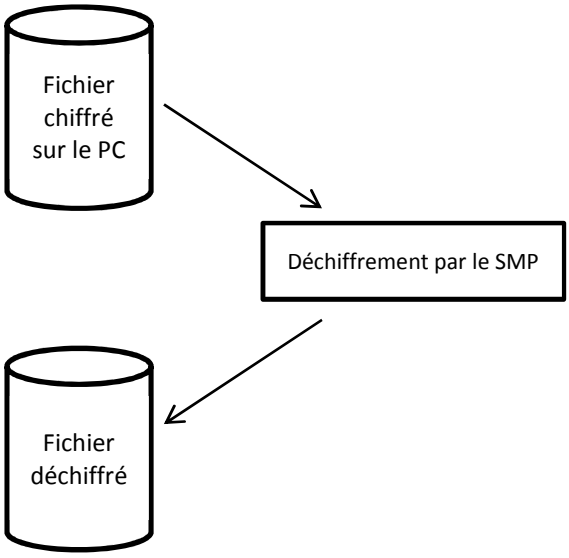
Le nom Xilinx fait immédiatement penser au fabricant de FPGA…

Un FPGA (Field-Programmable Gate Array) est un circuit électronique qui, en simplifiant à l'extrême, peut être vu comme une matrice dans laquelle on peut câbler des blocs logiques unitaires (portes logiques, bascules, mémoires, …). Les connexions entre les blocs logiques unitaires sont configurées par l'utilisateur (concepteur d'une carte qui intègre un FPGA par exemple), ce qui confère une grande souplesse à ce type de circuit.

Pour simplifier le travail des concepteurs, des bibliothèques de blocs prédéfinis plus complexes peuvent également être intégrés : ils comportent déjà les interconnexions entre les blocs élémentaires. Certains FPGA permettent par exemple de définir facilement un processeur complet.

Dans le cadre du Challenge, l'utilisation d'un FPGA prend tout son sens. Ce FPGA est en fait un circuit spécialisé en déchiffrement. L'intérêt est de décharger le PC d'une tâche éventuellement couteuse en temps CPU en faisant appel à un dispositif électronique externe. L'appellation SMP signifie très certainement « Symetric Multi Processing » : le traitement du déchiffrement se fait en parallèle du PC.

Un tel dispositif externe est généralement très performant du fait de sa spécialisation.



Dans le cadre du Challenge, l'utilisation d'un tel dispositif spécialisé laisse imaginer, *à priori*, à une méthode de chiffrement complexe.

4.3.2 **Récupération des outils Xilinx**

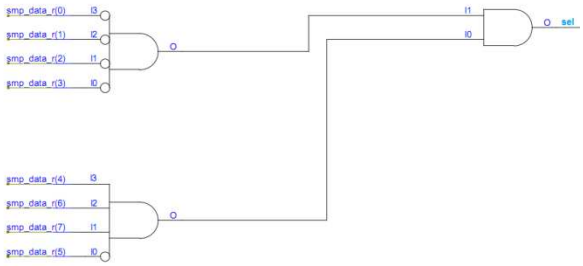
Quelques recherches sur le web permettent de comprendre que le fichier « s.ngr » est en fait un fichier comportant le schéma des blocs et interconnexions du FPGA utilisé.

Un tel fichier ne comporte pas le « source » de conception, ce qui va certainement rendre la tâche un peu plus compliquée.

En effet, lorsque l'on utilise un FPGA, on définit généralement sa configuration à partir d'équations faciles à manipuler ; par exemple (avec un langage fictif) :

```
If smp_data=0xD0 then sel=1 else sel=0
```

produira l'utilisation des blocs suivants avec les bonnes interconnexions :



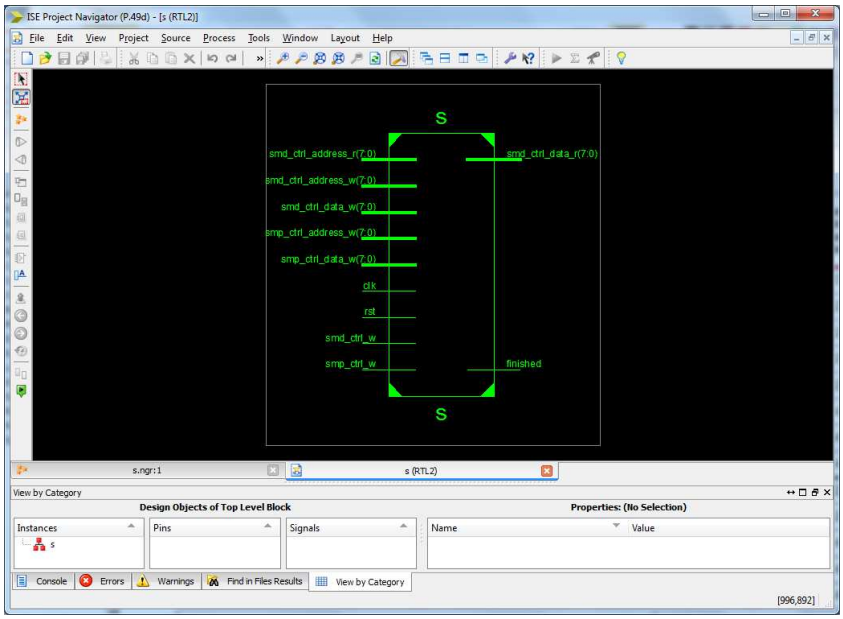
Trouver l'équation de départ à partir du schéma n'est pas très compliqué dans ce cas, mais peut le devenir rapidement pour un circuit plus complexe. Il faut par ailleurs être très vigilant : vous remarquerez dans le schéma ci-dessus que l'ordre des signaux qui composent le bus smp_data n'est pas continu (0,1,2,3,**4,6,7,5**)…

Les outils Xilinx permettant de lire les schémas au format « .ngr » sont disponibles gratuitement sur le site du fabricant sous la forme de produit d'évaluation (produit complet sur une durée de 30j).

Je décide donc de télécharger la suite ISE (fichier Xilinx_ISE_DS_Win_14.4_P.49d.3.0.tar sur http://www.xilinx.com/support/download/index.htm). Avec ma connexion internet, cela prend près de 10h pour récupérer l'archive de plus de 6 Go…

4.3.3 **Visualisation du schéma électronique**

Après avoir installé la suite Xilinx ISE, on peut ouvrir le fichier « s.ngr » :

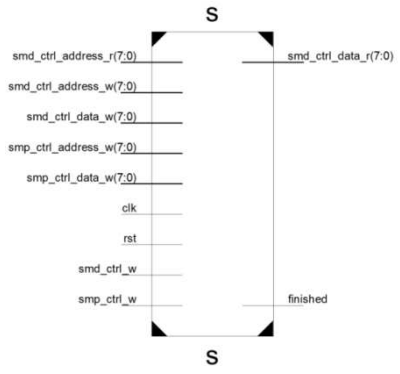


Nous avons maintenant la confirmation que le dispositif SMP est à base de FPGA. ISE nous apprend que le FPGA utilisé est un « Spartan-6 xc6slx45t-3-fgg484 ».

Les spécifications de ce circuit alimentent mes craintes sur la complexité du système : il peut comporter 43661 blocs logiques…

Device	Logic Cells[1]	Configurable Logic Blocks (CLBs)			DSP48A1 Slices[3]	Block RAM Blocks		CMTs[5]	Memory Controller Blocks (Max)[6]	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices[2]	Flip-Flops	Max Distributed RAM (Kb)		18 Kb[4]	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Heureusement, le schéma général du circuit semble assez simple :

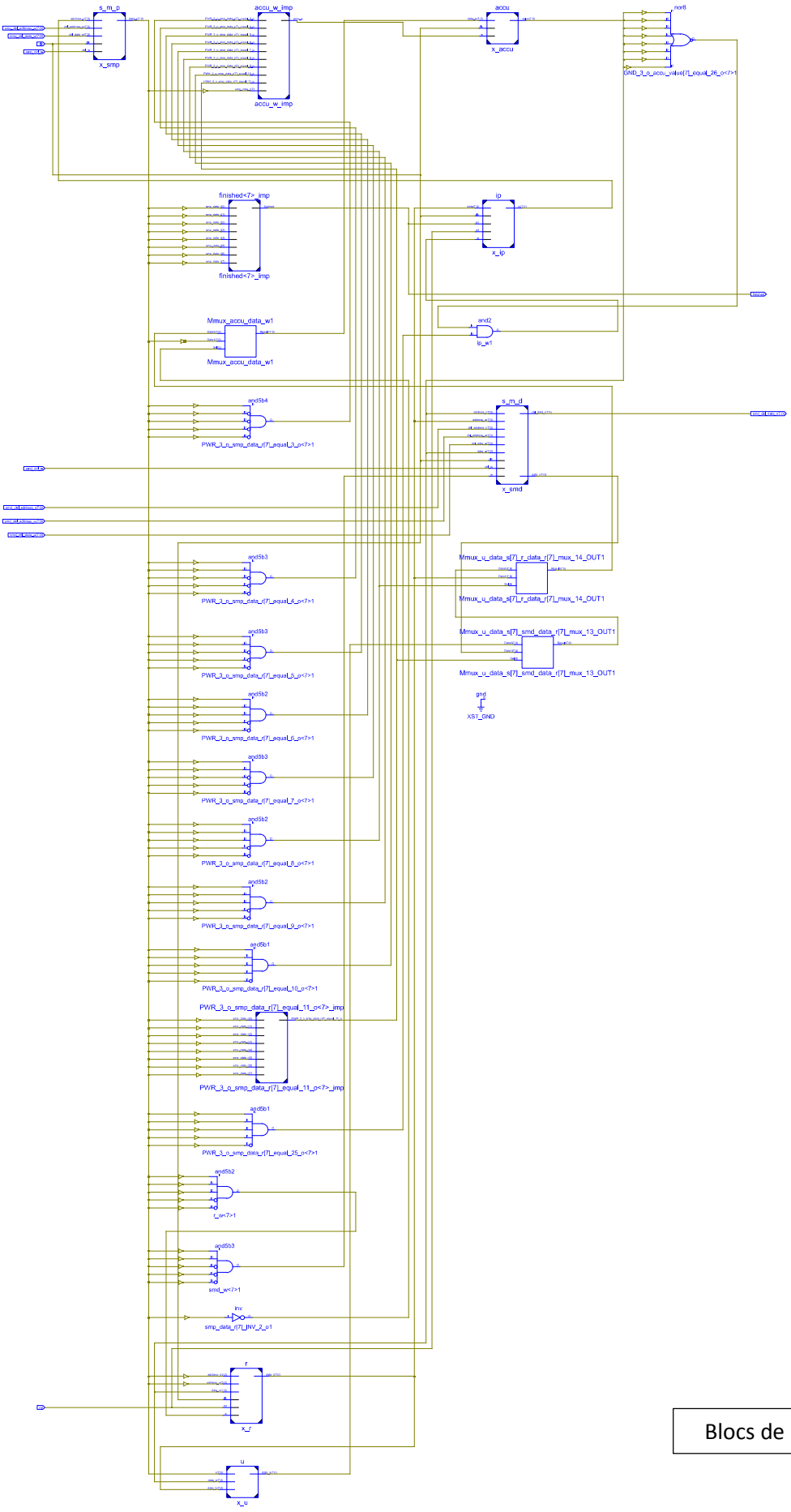


On y reconnait des signaux de contrôle (clk, rst, smd_ctrl_w, spm_ctrl_w), 3 bus d'adresse 8 bits (smd_ctrl_address_r, smd_ctrl_address_w et smp_ctrl_address_w) et 2 bus de données 8 bits (smp_ctrl_data_w, smd_ctrl_data_w et smd_ctrl_data_r).

Pour comprendre son fonctionnement, il faut rentrer dans le détail et afficher son contenu.

Le schéma des blocs de premier niveau est affiché ci-dessous ; il fait apparaitre des sous-blocs fonctionnels que nous allons également analyser dans ce qui suit.

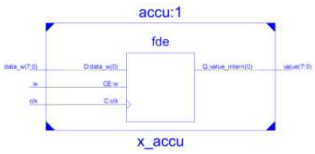
Dans le reste de ce document, j'ai annoté les schémas avec des textes en rouge pour expliquer leur fonctionnement.



Blocs de 1er niveau

4.3.4 **Analyse des blocs fonctionnels**

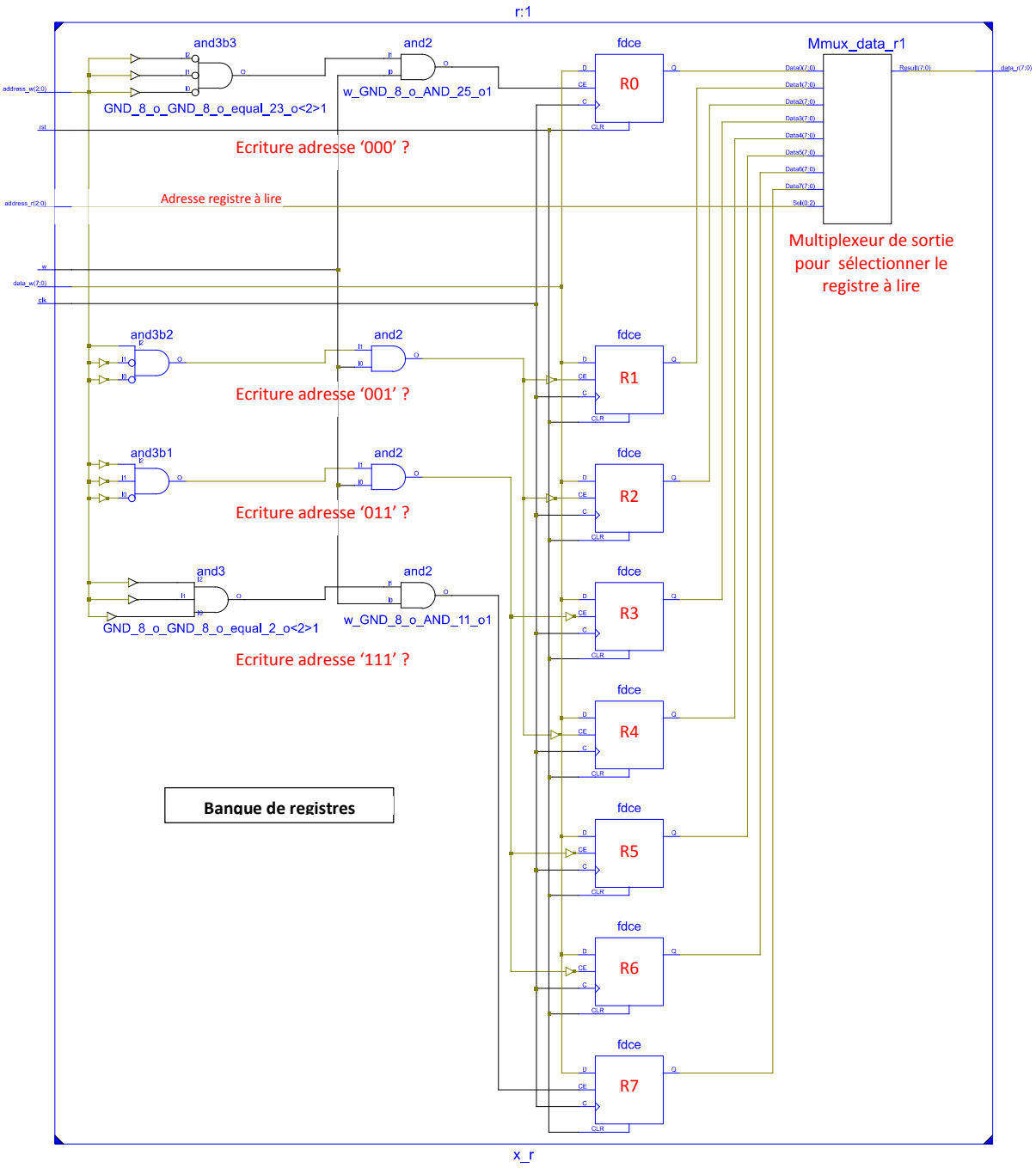
4.3.4.1 x_accu – Accumulateur



Ce bloc est très simple : il s'agit de 8 bascules D qui représentent une cellule mémoire. Le nom de ce bloc fait penser à un « accumulateur » pour un micro-processeur. Le FPGA intègrerait-il un cœur de micro-processeur ?

4.3.4.2 x_r – Banque de registres

Le bloc « x_r » se compose ainsi :

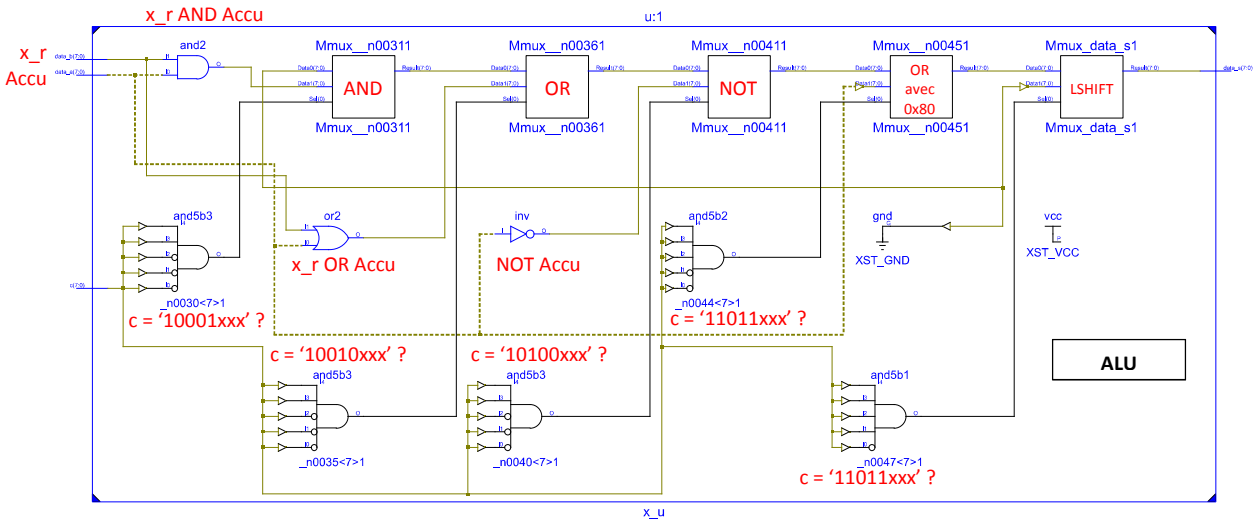


Ce bloc est une banque de 8 registres de 8 bits accessibles en lecture et écriture. On notera que le fichier « s.ngr » semble légèrement corrompu à ce niveau puisque le décodage d'adresse pour l'écriture des registres semble erroné (plusieurs décodeurs d'adresse se superposent sans possibilité de les distinguer). L'adressage du n° de registre se fait avec les 3 bits de poids faible des instructions.

Il semble de plus en plus clair que ce FPGA implémente un cœur de micro-processeur.

4.3.4.3 x_u – Unité arithmétique et logique (ALU)

Le schéma du bloc « x_u » est le suivant :



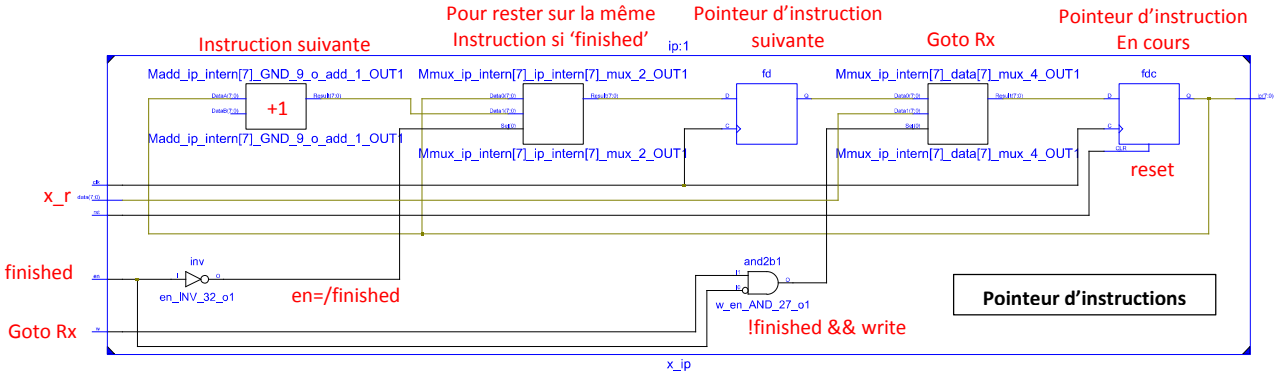
Il s'agit d'une unité de calcul, couramment appelée « Unité Arithmétique et Logique » (ALU) dans un micro-processeur.

En fonction des instructions en entrée qui lui sont données par le bus « c », elle effectue les calculs suivants :

Instruction	Action	Résultat en sortie
1000.1xxx	ET logique	Accu & x_r
1001.0xxx	OU logique	Accu | x_r
1010.0xxx	Complément à 1	~Accu
1110.0xxx	Force le bit 7 à 1	Accu | 0x80
1101.1xxx	Décalage à gauche	Accu<<1

4.3.4.4 x_ip – Pointeur d'instruction (IP)

Maintenant que l'on a compris que le FPGA implémentait un cœur de micro-processeur, le nom de ce bloc fonctionnel fait penser à un « Pointeur d'Instruction » (IP) :



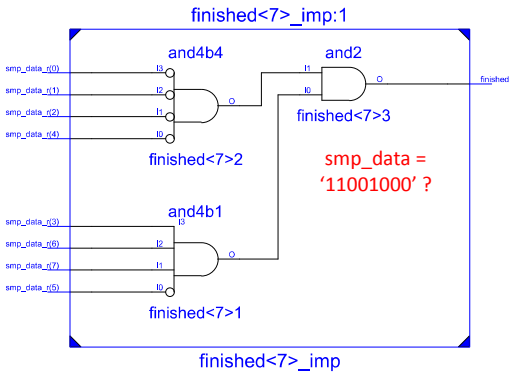
Ce bloc gère un « pointeur d'instruction » :

- Il mémorise la valeur courante du pointeur
- Il incrémente ce pointeur à chaque tick d'horloge Clk
- Il gère l'instruction « goto Rx » qui force le pointeur à l'adresse donnée par le bloc « x_r »
- Il initialise le pointeur à 0 lors d'un reset
- Il arrête l'incrémentation du pointeur lorsque l'entrée « finished » est active ; il s'agit certainement de la gestion de fin de programme.

4.3.4.5 finished – Instruction "Halt"

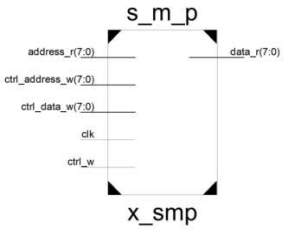
Ce bloc est juste un décodeur d'instruction. Lorsque « smp_data » vaut 0xC8, la sortie « finished » est activée. Nous avons vu précédemment que cela avait pour effet d'arrêter le pointeur d'instruction.

Cette instruction force donc l'arrêt du programme (instruction « Halt »).

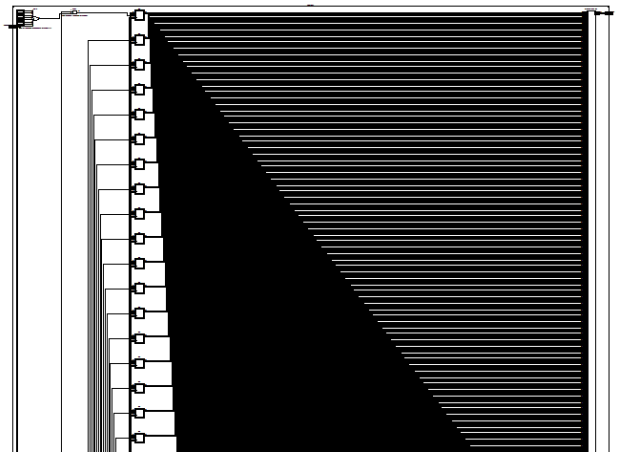


4.3.4.6 x_smp – Mémoire Programme double accès

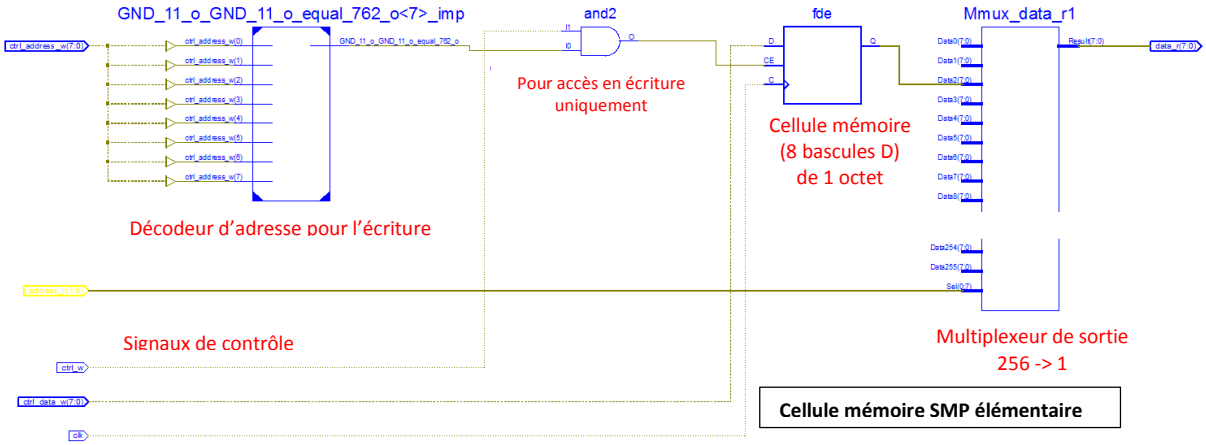
Ce bloc fonctionnel dispose de bus d'adresses et de données et de signaux de contrôle :



Son contenu symétrique évoque une mémoire :



Pour en être certain et comprendre son fonctionnement, je décide d'isoler une cellule élémentaire :



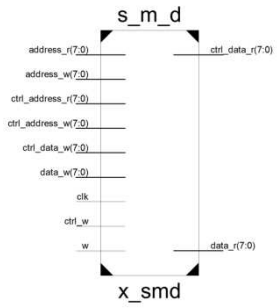
On comprend maintenant qu'il s'agit d'une mémoire double accès de 256 octets :

- En écriture uniquement côté contrôleur (certainement le PC)
- En lecture uniquement côté interne du FPGA

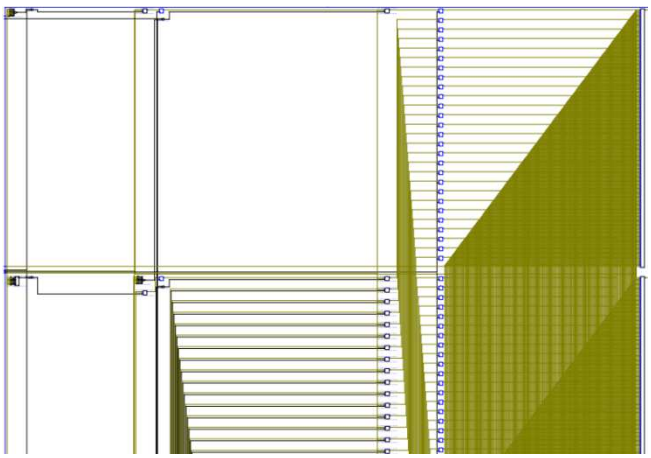
Vu ce que nous avons analysé auparavant, on peut affirmer qu'il s'agit d'une mémoire programme (l'instruction Halt est par exemple décodée par le bloc « finished »). Le programme peut être écrit dans cette mémoire par le PC et exécuté par le FPGA.

4.3.4.7 x_smd – Mémoire Données double accès

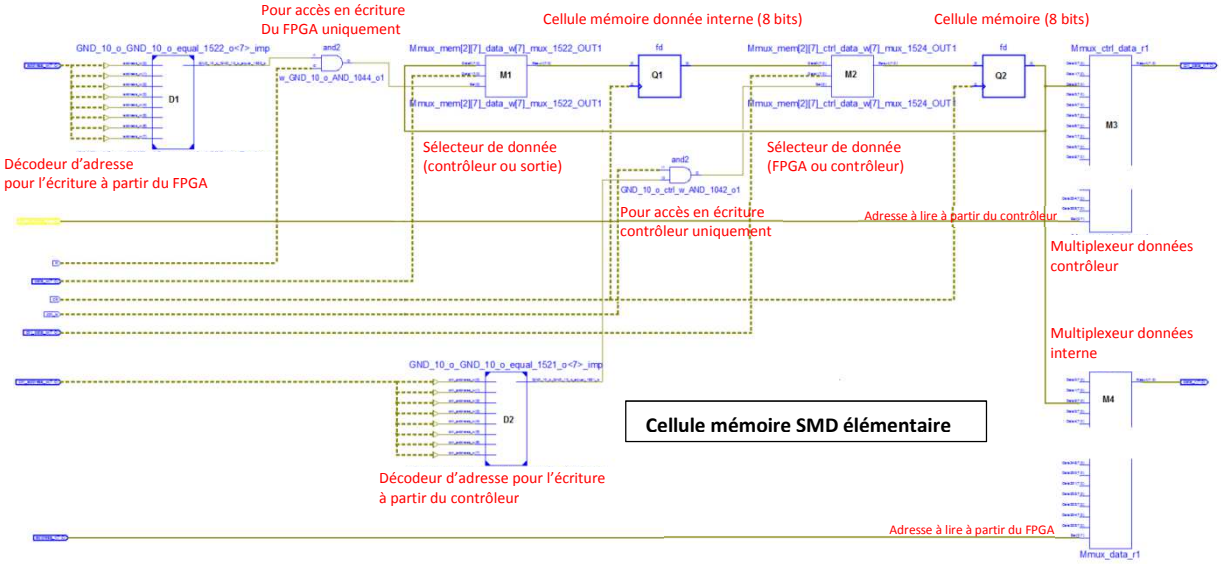
Ce bloc dispose également de bus d'adresses et de données et de signaux de contrôle :



La structure symétrique évoque également une mémoire. Elle semble néanmoins un peu plus complexe :



Comme précédemment, il est nécessaire d'isoler une cellule élémentaire pour comprendre son fonctionnement :

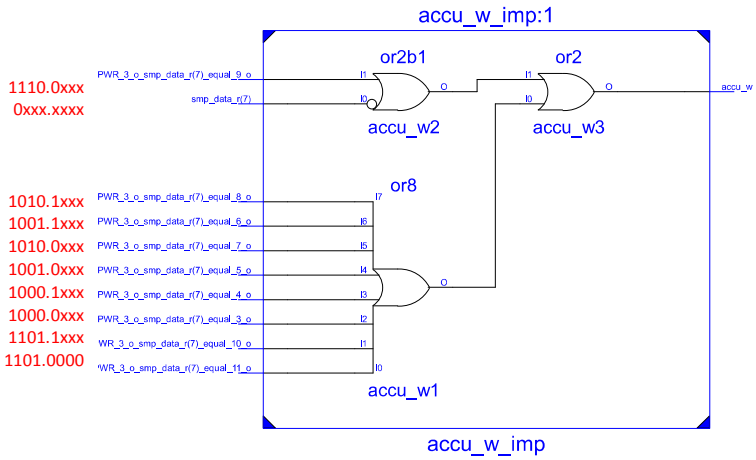


Ce bloc est une mémoire double accès de 256 octets :

- En lecture/écriture côté contrôleur (certainement le PC)
- En lecture/écriture côté interne du FPGA

4.3.4.8 accu_w_imp – Ecriture dans l'accumulateur

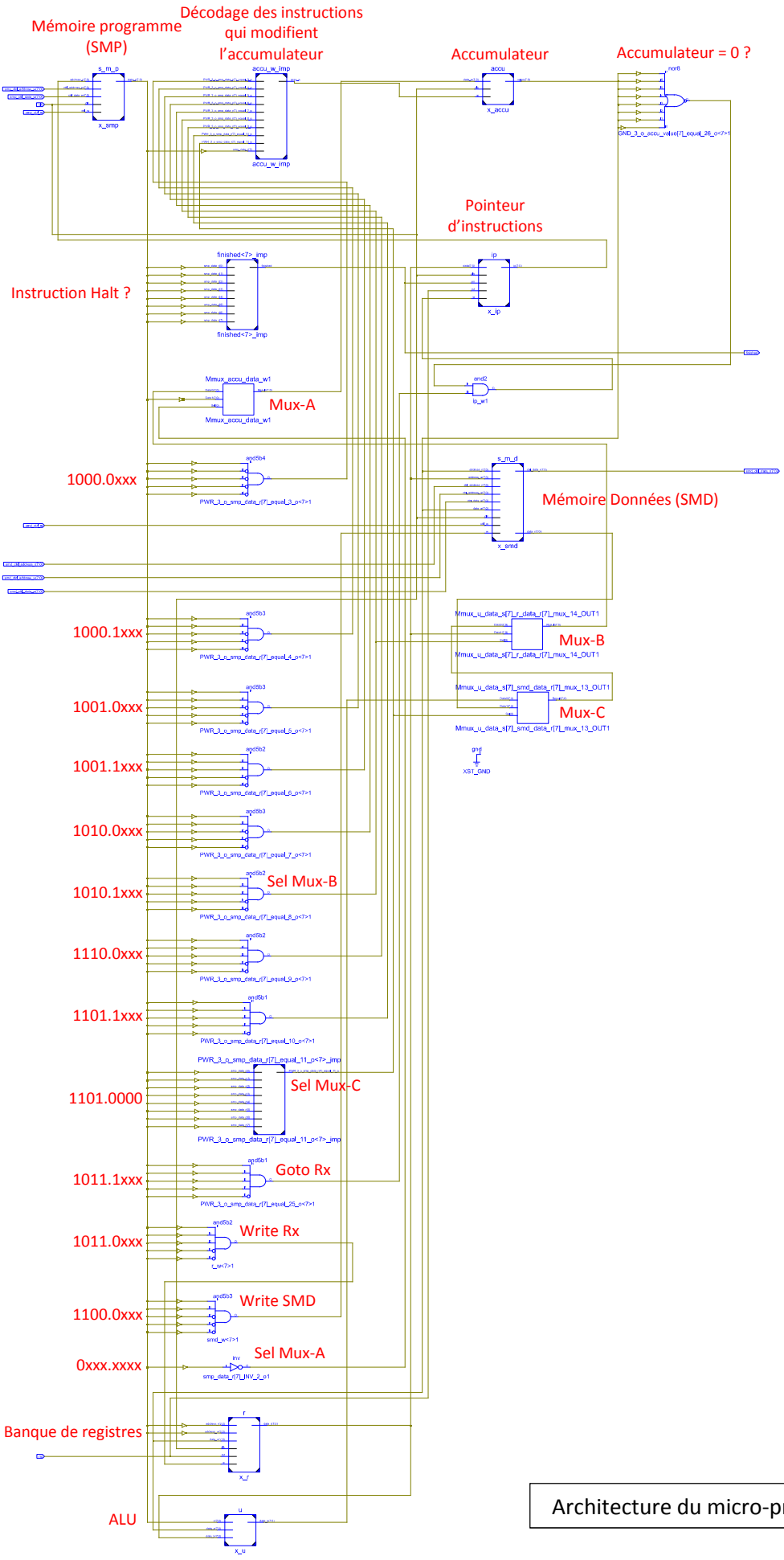
Ce bloc est un décodeur d'instruction qui pilote le signal d'écriture dans l'accumulateur.



La plupart des entrées de ce décodeur proviennent elles-mêmes des décodeurs présents sur le schéma général que nous allons voir ci-dessous (sauf celle qui décode 0xxxxxxx).

4.3.5 Architecture du micro-processeur

Maintenant que nous comprenons mieux les blocs fonctionnels, il est temps de reprendre et d'annoter le schéma général :



Architecture du micro-processeur SMP

Il reste à analyser le fonctionnement des multiplexeurs Mux-A, Mux-B et Mux-C :

- Mux-A permet de choisir entre Mux-B et la mémoire SMP
- Mux-B permet de choisir entre Mux-C et la banque de registre
- Mux-C permet de choisir entre l'ALU et la mémoire SMD

Ainsi, en fonction de l'instruction en cours, l'entrée de l'accumulateur est la suivante :

Instruction	**Entrée de l'accumulateur**
0xxxx.xxxx	Mémoire SMP
1010.1xxx	Banque de registre
1101.0000	Mémoire SMD
Autres	Unité ALU

Pour les instructions concernées par la banque de registre, le numéro de registre se trouve dans les 3 bits de poids faible.

4.3.6 **Jeu d'instructions du micro-processeur**

En fonction de tout ce qui précède, on déduit le jeu d'instructions du micro-processeur implémenté dans ce FPGA :

Instruction	**Modifie Accu ?**	**Entrée de Accu**	**Sortie de ALU**	**Instruction**
0xxx.xxxx	Oui	**Mémoire SMP**	0	**A=[SMP]**
1000.0xxx	Oui	ALU	0	
1000.1rrr	Oui	ALU	**A & Rx**	**A = A & Rx**
1001.0rrr	Oui	ALU	**A | Rx**	**A = A | Rx**
1001.1xxx	Oui	ALU	0	
1010.0xxx	Oui	ALU	**~A**	**A = ~A**
1010.1rrr	Oui	**Banque registres**	0	**A = Rx**
1011.0rrr		ALU	0	**Rx = A**
1011.1rrr		ALU	0	**IF A==0 Goto Rx**
1100.0rrr		ALU	0	**SMD[Rx]=A**
1100.1000		ALU	0	**Halt**
1100.1001 à 1100.1111		ALU	0	
1101.0000	Oui	**Mémoire SMD**	0	**A=SMD[A]**
1101.0001 à 1101.1111	Oui	ALU	0	
1101.1xxx	Oui	ALU	**A<<1**	**LSHIFT A (A<<=1)**
1110.0xxx	Oui	ALU	**A|0x80**	**A = A|0x80**
1110.1xxx 1111.xxxx		ALU	0	

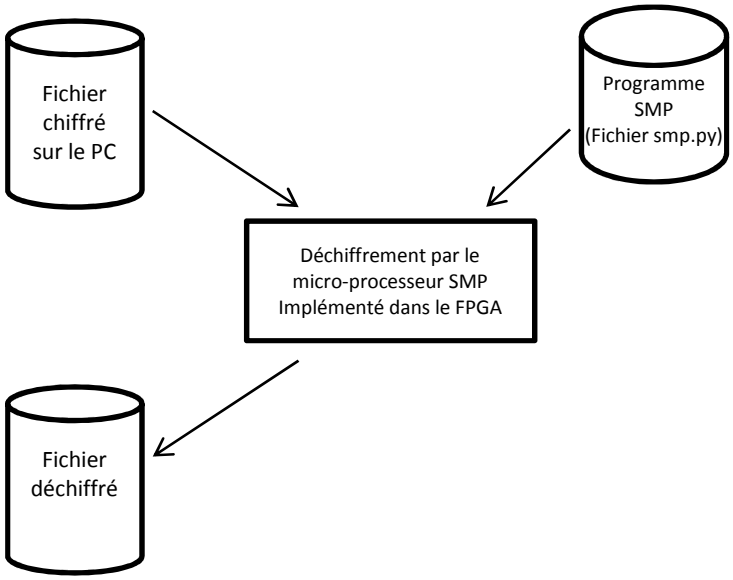
Notes :

- Dans les instructions ci-dessus, A représente l'accumulateur
- Pour chacune des instructions concernées par un registre Rx, le numéro de registre est indiqué dans les 3 bits de poids faible du code de l'instruction (noté 'rrr' ci-dessus)
- Pour l'instruction A=[SMP], il s'agit d'un mode adressage immédiat (on charge dans l'accumulateur une valeur qui se trouve dans l'instruction elle-même). Cette valeur est

limitée à 7 bits puisque le 8ème bit sert à reconnaitre l'instruction… Pour pallier cette limitation, l'instruction « A|=0x80 » permet de forcer rapidement le bit 7 à 1. Ainsi :
    - 0x27 correspond à l'instruction A=0x27
    - 0x27,0xE0 correspondent aux instructions A=0x27 et A|=0x80, soit A=0xA7
- Il n'y a qu'une seule instruction de branchement ; il est conditionnel et indirect par un registre :
    - IF A==0 Goto Rx / Si A est non nul, on branche à l'adresse contenu dans Rx, sinon on passe à l'instruction suivante
- Il y a 2 instructions d'accès à la mémoire SMD :
    - En lecture : A=SMD[A] / Adressage indirect par A et résultat dans A
    - En écriture : SMD[Rx]=A / Adressage indirect par Rx et valeur à écrire dans A

4.4 Désassemblage du programme SMP

Nous avons maintenant une vision plus précise de la manière dont le fichier « data » est déchiffré par le FPGA :



Pour comprendre la méthode de chiffrement, et ainsi espérer décrypter les données, nous devons désassembler le programme exécuté par le SMP.

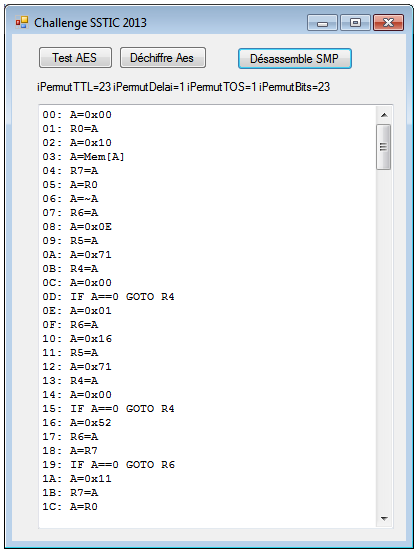
Pour cela, le contenu du fichier SMP.PY est converti en C# :

```
// Programme SMP
Byte[] smp = new Byte[231]
        {
            0x00,0xb0,0x10,0xd0,0xb7,0xa8,0xa0,0xb6,0x0e,0xb5,0x71,0xb4,0x00,0xbc,0x01,0xb6,
            0x16,0xb5,0x71,0xb4,0x00,0xbc,0x52,0xb6,0xaf,0xbe,0x11,0xb7,0xa8,0xb6,0x24,0xb5,
            0x71,0xb4,0x00,0xbc,0xaf,0xb1,0xa9,0xd0,0xb6,0x0f,0x88,0xd0,0x96,0xb6,0xa9,0xd0,
            0xb7,0x0f,0x88,0xd0,0x8f,0xb7,0xaf,0xa0,0x8e,0xb7,0x40,0xb6,0x53,0xb5,0x00,0xbd,
            0xaf,0xc1,0x01,0xb6,0xa8,0xb7,0x4c,0xb5,0x71,0xb4,0x00,0xbc,0xaf,0xb0,0x02,0xb6,
            0x00,0xbe,0xc8,0x01,0xb5,0x00,0xb4,0x6d,0xb3,0xad,0xbb,0x66,0xb3,0xac,0xd8,0xb4,
            0xad,0x8f,0xbb,0x01,0x94,0xb4,0xad,0xd8,0xb5,0x57,0xb3,0x00,0xbb,0xac,0xb7,0x00,
            0xbe,0x00,0xb1,0xb3,0x01,0xb2,0x63,0xe0,0xb4,0xaa,0xbc,0x2c,0xe0,0xb4,0xaf,0x8a,
            0xbc,0x16,0xe0,0xb4,0xae,0x8a,0xbc,0x0f,0xe0,0xb4,0xa9,0xbc,0xab,0x92,0xb3,0xaa,
            0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x22,0xe0,0xb4,0xa9,0xbc,0xaa,0xb1,0x59,0xe0,0xb4,
            0x00,0xbc,0xab,0x92,0xb3,0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x48,0xe0,0xb4,0xae,
            0x8a,0xbc,0x3e,0xe0,0xb4,0xa9,0xbc,0xaa,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0xaa,0x93,
            0xb3,0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x57,0xe0,0xb4,0xa9,0xbc,0xaa,0x93,0xb3,
            0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x00,0xb1,0xaa,0xd8,0xb2,0xa9,0xd8,0xb1,0x76,
            0xb4,0x00,0xbc,0xab,0xb7,0x00,0xbd
        };
```

Une fonction de désassemblage est écrite. Son fonctionnement est très simple : elle parcourt toutes les instructions du programme ci-dessus, et les convertit en texte à l'aide du jeu d'instructions décrit dans le chapitre précédent. Pour utiliser cette fonction, un bouton « Désassemble SMP » est ajouté à l'interface graphique.

```
private void btnDesassemble_Click(object sender, EventArgs e)
{
    // Parcours toutes les instructions du programme SMP
    for (Byte ip = 0; ip < smp.Length; ip++)
    {
        // Lecture de l'instruction
        Byte Instruction = smp[ip];
        // Isole le n° de registre
        Byte Register = (Byte)(Instruction & 0x7);
        // Affiche l'adresse
        string s = ip.ToString("X2") + ": ";
        // Affiche l'opcode
        s += smp[ip].ToString("X2");
        // Display instruction
        if ((Instruction >= 0x00) && (Instruction <= 0x7F))
            s += "A=0x" + Instruction.ToString("X2");
        else if ((Instruction >= 0x88) && (Instruction <= 0x8F))
            s += "A=A&R" + Register.ToString();
        else if ((Instruction >= 0x90) && (Instruction <= 0x97))
            s += "A=A|R" + Register.ToString();
        else if ((Instruction >= 0xA0) && (Instruction <= 0xA7))
            s += "A=~A";
        else if ((Instruction >= 0xA8) && (Instruction <= 0xAF))
            s += "A=R" + Register.ToString();
        else if ((Instruction >= 0xB0) && (Instruction <= 0xB7))
            s += "R" + Register.ToString() + "=A";
        else if ((Instruction >= 0xB8) && (Instruction <= 0xBF))
            s += "IF A==0 GOTO R" + Register.ToString();
        else if ((Instruction >= 0xC0) && (Instruction <= 0xC7))
            s += "Mem[R" + Register.ToString() + "]=A";
        else if (Instruction == 0xC8)
            s += "HALT";
        else if (Instruction == 0xD0)
            s += "A=Mem[A]";
        else if ((Instruction >= 0xD8) && (Instruction <= 0xDF))
            s += "LSHIFT A";
        else if ((Instruction >= 0xE0) && (Instruction <= 0xE7))
            s += "A=A|0x80";
        else
            s += "Illegal OpCode !";
        // Affiche l'instruction
        textResult.Text += s + "\r\n";
    }
}
```

L'exécution de ce désassemblage donne le résultat suivant :



Le listing complet du désassemblage est :

```
00: A=0x00
01: R0=A
02: A=0x10
03: A=Mem[A]
04: R7=A
05: A=R0
06: A=~A
07: R6=A
08: A=0x0E
09: R5=A
0A: A=0x71
0B: R4=A
0C: A=0x00
0D: IF A==0 GOTO R4
0E: A=0x01
0F: R6=A
10: A=0x16
11: R5=A
12: A=0x71
13: R4=A
14: A=0x00
15: IF A==0 GOTO R4
16: A=0x52
17: R6=A
18: A=R7
19: IF A==0 GOTO R6
1A: A=0x11
1B: R7=A
1C: A=R0
1D: R6=A
1E: A=0x24
1F: R5=A
20: A=0x71
21: R4=A
22: A=0x00
23: IF A==0 GOTO R4
24: A=R7
25: R1=A
26: A=R1
27: A=Mem[A]
28: R6=A
29: A=0x0F
2A: A=A&R0
2B: A=Mem[A]
2C: A=A|R6
2D: R6=A
2E: A=R1
2F: A=Mem[A]
30: R7=A
31: A=0x0F
32: A=A&R0
33: A=Mem[A]
34: A=A&R7
35: R7=A
36: A=R7
37: A=~A
38: A=A&R6
39: R7=A
```

```
3A: A=0x40
3B: R6=A
3C: A=0x53
3D: R5=A
3E: A=0x00
3F: IF A==0 GOTO R5
40: A=R7
41: Mem[R1]=A
42: A=0x01
43: R6=A
44: A=R0
45: R7=A
46: A=0x4C
47: R5=A
48: A=0x71
49: R4=A
4A: A=0x00
4B: IF A==0 GOTO R4
4C: A=R7
4D: R0=A
4E: A=0x02
4F: R6=A
50: A=0x00
51: IF A==0 GOTO R6
52: HALT
53: A=0x01
54: R5=A
55: A=0x00
56: R4=A
57: A=0x6D
58: R3=A
59: A=R5
5A: IF A==0 GOTO R3
5B: A=0x66
5C: R3=A
5D: A=R4
5E: LSHIFT A
5F: R4=A
60: A=R5
61: A=A&R7
62: IF A==0 GOTO R3
63: A=0x01
64: A=A|R4
65: R4=A
66: A=R5
67: LSHIFT A
68: R5=A
69: A=0x57
6A: R3=A
6B: A=0x00
6C: IF A==0 GOTO R3
6D: A=R4
6E: R7=A
6F: A=0x00
70: IF A==0 GOTO R6
71: A=0x00
72: R1=A
73: R3=A
```

```
74: A=0x01
75: R2=A
76: A=0x63
77: A=A|0x80
78: R4=A
79: A=R2
7A: IF A==0 GOTO R4
7B: A=0x2C
7C: A=A|0x80
7D: R4=A
7E: A=R7
7F: A=A&R2
80: IF A==0 GOTO R4
81: A=0x16
82: A=A|0x80
83: R4=A
84: A=R6
85: A=A&R2
86: IF A==0 GOTO R4
87: A=0x0F
88: A=A|0x80
89: R4=A
8A: A=R1
8B: IF A==0 GOTO R4
8C: A=R3
8D: A=A|R2
8E: R3=A
8F: A=R2
90: R1=A
91: A=0x59
92: A=A|0x80
93: R4=A
94: A=0x00
95: IF A==0 GOTO R4
96: A=0x22
97: A=A|0x80
98: R4=A
99: A=R1
9A: IF A==0 GOTO R4
9B: A=R2
9C: R1=A
9D: A=0x59
9E: A=A|0x80
9F: R4=A
A0: A=0x00
A1: IF A==0 GOTO R4
A2: A=R3
A3: A=A|R2
A4: R3=A
A5: A=0x00
A6: R1=A
A7: A=0x59
A8: A=A|0x80
A9: R4=A
AA: A=0x00
AB: IF A==0 GOTO R4
AC: A=0x48
AD: A=A|0x80
```

```
AE: R4=A
AF: A=R6
B0: A=A&R2
B1: IF A==0 GOTO R4
B2: A=0x3E
B3: A=A|0x80
B4: R4=A
B5: A=R1
B6: IF A==0 GOTO R4
B7: A=R2
B8: R1=A
B9: A=0x59
BA: A=A|0x80
BB: R4=A
BC: A=0x00
BD: IF A==0 GOTO R4
BE: A=R2
BF: A=A|R3
C0: R3=A
C1: A=0x00
C2: R1=A
C3: A=0x59
C4: A=A|0x80
C5: R4=A
C6: A=0x00
C7: IF A==0 GOTO R4
C8: A=0x57
C9: A=A|0x80
CA: R4=A
CB: A=R1
CC: IF A==0 GOTO R4
CD: A=R2
CE: A=A|R3
CF: R3=A
D0: A=0x00
D1: R1=A
D2: A=0x59
D3: A=A|0x80
D4: R4=A
D5: A=0x00
D6: IF A==0 GOTO R4
D7: A=0x00
D8: R1=A
D9: A=R2
DA: LSHIFT A
DB: R2=A
DC: A=R1
DD: LSHIFT A
DE: R1=A
DF: A=0x76
E0: R4=A
E1: A=0x00
E2: IF A==0 GOTO R4
E3: A=R3
E4: R7=A
E5: A=0x00
E6: IF A==0 GOTO R5
```

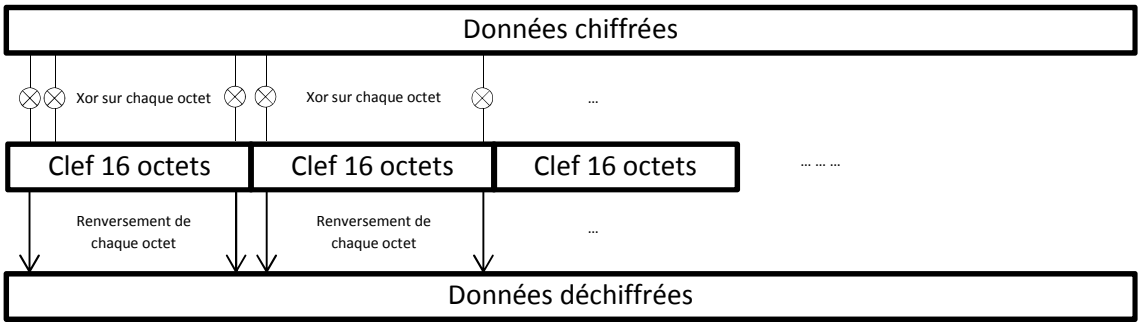
Mais la simplicité des instructions rend la lecture difficile. Je décide donc de l'écrire de telle manière que le programme soit plus facilement lisible. Les adresses d'instruction ont été conservées en commentaire afin de faciliter le lien avec le listing de désassemblage précédent.

```
Main()
{
    // Hard reset
    r0 = 0; r1 = 0; r2 = 0; r3 = 0; r4 = 0; r5 = 0; r6 = 0; r7 = 0;

    //0x02
    while (true)
    {
        //0x04
        r7 = Mem[0x10]; // Taille des données
        // Soustrait le compteur
```

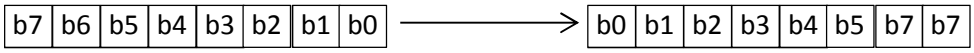
```
        AddToR7((Byte)~r0);

        //0x0e
        AddToR7(1);

        //0x16
        if (r7 != 0) // Plus de données ?
            return; //0x52 : Halt

        r7 = 0x11;
        AddToR7(0);

        // Déchiffrement d'un octet :
        // Effectue un Xor de la donnée avec la clef
        //0x24
        r1 = r7;
        r7 = (Byte)(Mem[r1] ^ Mem[r0 & 0x0f]);
        // Renverse les bits de la donnée
        //0x3a
        RenversementBits();

        // Sauvegarde de la donnée
        //0x40
        Mem[r1] = r7;

        // Incrémente le compteur
        r7 = r0;
        AddToR7(1);
        //0x4c
        r0 = r7;
    }
}

//0x53
void RenversementBits()
{
    // r7 contient la donnée à 'renverser'
    // r4 contient le résultat du renversement
    // r5 est le masque qui est décalé à gauche de 0x01 à 0x80
    for (r4 = 0, r5 = 1; r5 != 0; r5 <<= 1)
    {
        r4 <<= 1;
        //0x62
        // Teste un bit de la donnée et le transfert dans r4
        if ((r5 & r7) != 0) r4 |= 1;
        // 0x6c
    }
    // 0x6d
    // Retourne la donnée renversée
    r7 = r4;
}

//0x71
void AddToR7(Byte r6)
{
    r3 = 0;

    for (r1 = 0, r2 = 0; r2 != 0; r1 <<= 1, r2 <<= 1)
    {
        //0x80
        if ((r2 & r7) == 0)
        {
            //0xac
            if ((r2 & r6) == 0)
            {
                //0xc8
                if (r1 != 0)
                {
                    // 0xcd
                    r3 |= r2;
                    r1 = 0;
                }
                // 0xd7
                r1 = 0;
            }
            else
            {
```

```
                if (r1 == 0) // 0xb6
                {
                    //0xbe:
                    r3 |= r2;
                    r1 = 0;
                }
                else
                {
                    //0xb7
                    r1 = r2;
                }
            }
        }
        else
        {
            // 0x86
            if ((r2 & r6) != 0)
            {
                //0x87
                if (r1 != 0)
                {
                    //0x8c
                    r3 |= r2;
                }
                //0x8f
                r1 = r2;
            }
            else
            {
                //0x96
                if (r1 != 0)
                {
                    //0x9b
                    r1 = r2;
                }
                else
                {
                    //0xa2
                    r3 |= r2;
                    r1 = 0;
                }
            }
        }
        //0xd9
    }

    //0xE3
    r7 = r3;
}
```

On comprend cette fois-ci que le programme parcourt les données pour faire un Xor avec chaque octet de la clef et un renversement de bit :



Le renversement de chaque octet consiste à le transformer de la manière suivante :



Dans ce qui suit, nous écrirons désormais :

DonnéesDéchiffrées = Renverse(DonnéesChiffrées ^ Clef)

Les 2 fonctions Renverse et Xor (^) étant involutives, on a également :

DonnéesChiffrées = Renverse(DonnéesDéchiffrées) ^ Clef

Les fonctions de déchiffrement, mais également de chiffrement, peuvent maintenant être écrites en C# :

```
private void SmpDechiffre(ref Byte[] Key, ref Byte[] Data)
{
    for (int i = 0; i < Data.Length; i++)
    {
        // Xor avec la clef
        Byte v1 = (Byte)(Data[i] ^ Key[i & 0x0f]), v2 = 0;
        // Inversion des bits
        for (int j = 0; j < 8; j++)
        {
            v2 <<= 1;
            v2 |= (Byte)(v1 & 1);
            v1 >>= 1;
        }
        Data[i] = v2;
    }
}

private void SmpChiffre(ref Byte[] Key, ref Byte[] Data)
{
    for (int i = 0; i < Data.Length; i++)
    {
        Byte v1 = Data[i], v2 = 0;
        // Inversion des bits
        for (int j = 0; j < 8; j++)
        {
            v2 <<= 1;
            v2 |= (Byte)(v1 & 1);
            v1 >>= 1;
        }
        // Xor avec la clef
        Data[i] = (Byte)(v2 ^ Key[i & 0x0f]);
    }
}
```

Maintenant que nous maitrisons l'algorithme de chiffrement/déchiffrement SMP, nous avons tous les outils pour casser la clef du fichier « data ».

4.5 Recherche de la clef SMP

4.5.1 **Ce que nous savons**

A ce stade nous savons :

- Comment fonctionne l'algorithme de chiffrement/déchiffrement SMP
- Que d'après le script « decrypt.py », le fichier déchiffré est codé en Base64 ; son jeu de caractères est donc connu (alphanumérique, + et /) mais il est également fort probable qu'il soit constitué d'enregistrements de taille fixe séparés par des sauts de lignes (code Ascii 0x0A). Je vais me servir de cette propriété pour trouver la clef. Néanmoins, nous ne connaissons pas la taille des enregistrements ; elle est généralement de 64 caractères, mais cela n'est pas garanti…

4.5.2 **Propriété du chiffrement SMP**

On sait que :

DonnéesChiffrées = Renverse(DonnéesDéchiffrées) ^ Clef »

En appliquant cette formule à un texte connu (TextConnu), on a :

TexteConnuChiffré = Renverse(TextConnu) ^Clef

Et en utilisant le fait que la fonction Xor est involutive, on peut écrire :

Clef = Renverse(TexteConnu) ^ TexteConnuChiffré

Ainsi, si nous connaissons une partie du message en clair et sa version chiffrée, nous en déduisons la clef. C'est cette propriété que nous allons utiliser.

Pour cela, nous considérons que notre message en clair est constitué d'enregistrements (de longueur « Len ») séparés par des sauts de ligne (0x0a) ; la version chiffrée de ces sauts de ligne se retrouve aux mêmes emplacements (que les sauts de lignes) dans le fichier « data ». Par ailleurs, des données identiques se trouvant à une même adresse modulo 16 sont chiffrées de la même manière.

Nous pouvons donc constituer notre « TexteConnu » avec 16 octets 0x0A ; la version chiffrée de ce message est fabriquée en prenant un octet tous les « Len » octets dans le fichier chiffré « data » (on ignore le premier enregistrement puisqu'il n'est pas précédé par un séparateur).

Cela nous donne une clef que nous pouvons tester. Pour la vérifier, il suffit de tenter de déchiffrer le fichier « data » avec cette clef, et de vérifier si nous obtenons un fichier Base64 (avec uniquement des caractères alphanumériques, + et /). Si c'est le cas, nous pouvons également vérifier le code MD5 que nous avons trouvé dans le script « decrypt.py ».

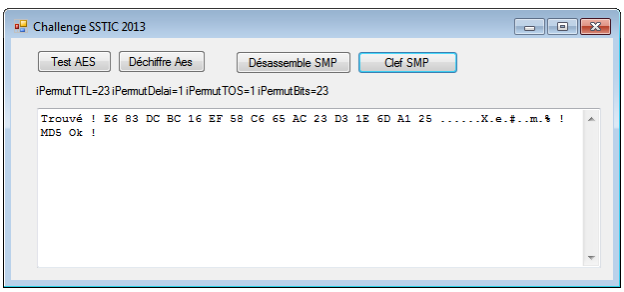
La seule limite à cette méthode est qu'elle ne fonctionne pas si la longueur de chaque enregistrement (séparateur inclus) est un multiple de 16 : les séparateurs se retrouveraient tous à la même adresse modulo 16 et seraient tous déchiffrés de la même manière. Néanmoins, cette méthode vaut le coup d'être essayée !

4.5.3 **Programme de recherche de la clef SMP**

Avec la méthode décrite ci-dessus, le programme suivant est réalisé :

```
private void btnSmpKey_Click(object sender, EventArgs e)
{
    string sOut;
    FileStream InFs = new FileStream("data", FileMode.Open, FileAccess.Read);
    FileStream OutFs = new FileStream(sOut="data.dechiffre", FileMode.Create,
                                                            FileAccess.Write);
    // Lecture du fichier
    int Max = (int)InFs.Length;
    Byte[] Data = new Byte[Max];

    for (int Base64RecordLength = 16; Base64RecordLength < 1025; Base64RecordLength++)
    {
        // Lecture du fichier
        InFs.Seek(0, SeekOrigin.Begin);
        int Count = InFs.Read(Data, 0, Max);

        // Composition d'une clef à la recherche des séparateurs
        Byte[] WhatWeExpect = new Byte[16] { 0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a,
                                             0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a, 0x0a };
        Byte[] Tmp = new Byte[16];
        for (int i = 0; i < 16; i++) Tmp[i] = Data[(i + 1) * Base64RecordLength - 1];

        // Encodage de ce qu'on recherche
        SmpChiffre(ref Tmp, ref WhatWeExpect);
        // On s'en sert de nouvelle clef
        SmpDechiffre(ref WhatWeExpect, ref Data);

        // Base64 trouvé ?
        Boolean bFound = true;
        for (int i = 0; i < 16; i++)
        {
            if ((Data[i] >= 'A' && Data[i] <= 'Z') || (Data[i] >= 'a' && Data[i] <= 'z') ||
                (Data[i] >= '0' && Data[i] <= '9') || (Data[i] == '+') || (Data[i] == '/'))
                continue;
            bFound = false; break;
        }
        if (bFound)
        {
            // Clef trouvée !
            textResult.Text += "Trouvé ! " + Dump(WhatWeExpect, 16, true) + " !\r\n";
            OutFs.Write(Data, 0, Count);
            OutFs.Close(); InFs.Close();

            // On vérifie le MD5
            string sExpectedMD5 = "6c0708b3cf6e32cbae4236bdea062979", sComputedMD5 = "";
            MD5 md5 = MD5.Create();
            InFs = new FileStream(sOut, FileMode.Open, FileAccess.Read);
            sComputedMD5 = BitConverter.ToString(
                                        md5.ComputeHash(InFs)).Replace("-", "").ToLower();
            InFs.Close();
            if (sComputedMD5 == sExpectedMD5) textResult.Text += "MD5 Ok !\r\n";
            break;
        }
    }
}
```

Et il trouve immédiatement la clef : E683DCBC16EF58C665AC23D31E6DA125



4.6 Déchiffrement final du fichier SMP

En même temps qu'il a trouvé la clef, le programme a généré le fichier « data.dechiffre » :

```
# more data.dechiffre
L0kxIGN1cnJlbnRmaWxlIDAgKGNhZmViYWJlKSAvU3ViRmlsZURlY29kZSBmaWx0ZXIgL0FTQ0lJ
SGV4RGVjb2RlIGZpbHRlciAvUmV1c2FibGVTdHJlYW1EZWNvZGUgZmlsdGVyIGNmNzYwYmM3N2Ri
MWYyODJlODgxZWRlOWExMDEyMmIyMjA4ODc0NjZiOTczYjg1NDIxOGI4NWMyMzBkNjczM2FiNDU5
```

On retrouve bien un fichier Base64 ; il est inutile de vérifier son code MD5 puisqu'il a déjà été vérifié par le programme de recherche de clef.

Ce fichier peut maintenant être décodé Base64 :

```
# base64 -d data.dechiffre >data.dechiffre.unbase64
```

Ce fichier est un fichier texte :

```
# hexdump -n 64 -C data.dechiffre.unbase64
00000000  2f 49 31 20 63 75 72 72  65 6e 74 66 69 6c 65 20  |/I1 currentfile |
00000010  30 20 28 63 61 66 65 62  61 62 65 29 20 2f 53 75  |0 (cafebabe) /Su|
00000020  62 46 69 6c 65 44 65 63  6f 64 65 20 66 69 6c 74  |bFileDecode filt|
00000030  65 72 20 2f 41 53 43 49  49 48 65 78 44 65 63 6f  |er /ASCIIHexDeco|
```

En visualisant la fin de ce fichier, on reconnaît quelques commandes Postscript (pop, bind, def, …) :

```
# tail -c 256 data.dechiffre.unbase64
roll putinterval exch pop } for 0 1 1 {pop pop} for cvx exec } if false } { (no key
provided\n) error true } ifelse } { (missing '--' preceding script file\n) error true } ifelse
{ (usage: gs -- script.ps key\n) error flush } if } bind def main clear quit
```

En conclusion de ce chapitre, nous avons réussi à casser la clef du chiffrement SMP et nous avons obtenu un fichier Postscript.

5. Fichier Postscript

5.1 Impression du fichier Postscript

5.1.1 **Tentative d'impression**

Je me dis que la résolution du Challenge est proche : j'ai obtenu un fichier Postscript. En l'imprimant, j'obtiendrai certainement une page sur laquelle figurera l'adresse mail à trouver !

Je décide donc d'envoyer ce fichier à mon imprimante Konica ; ça tombe bien, elle est Postscript :

```
# lpr data.dechiffre.unbase64
```

L'imprimante se met en chauffe, mais elle n'imprime rien. Je vérifie sa configuration, fais quelques essais, mais toujours rien…

5.1.2 **Utilisation de Ghostscript**

Ghostscript est un interpréteur de programme Postscript. Il est utilisé principalement pour visualiser le résultat des fichiers qui décrivent une page, ou pour les convertir afin de les imprimer sur des imprimantes non compatibles Postscript.

Ce programme est sous licence Gnu GPL : il peut donc être utilisé librement.

Comme il est déjà installé sur ma machine, je tente une exécution directe avec la commande suivante :

```
# gs data.dechiffre.unbase64
GPL Ghostscript 9.05 (2012-02-08)
Copyright (C) 2010 Artifex Software, Inc.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
missing '--' preceding script file
usage: gs -- script.ps key
```

Le programme me demande une clef… Je retente la commande en passant un argument :

```
# gs -- data.dechiffre.unbase64 0000
GPL Ghostscript 9.05 (2012-02-08)
Copyright (C) 2010 Artifex Software, Inc.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
```

Cette fois-ci, aucune réponse. Il est maintenant clair que ce fichier n'est pas une simple description de page à imprimer… Il s'agit vraisemblablement d'un programme un peu plus complexe.

5.2 Programme Postscript

5.2.1 **Le langage Postscript**

Le Postscript est un langage de programmation. Il est principalement utilisé pour l'impression de documents : le langage sert dans ce cas à décrire une page à imprimer et cette description est exécutée par une imprimante compatible pour dessiner la page.

Dans le cas de documents à imprimer, la génération du programme Postscript n'est pas manuelle… elle est très souvent réalisée par le pilote de d'imprimante.

Néanmoins, ce langage peut être utilisé pour réaliser quasiment n'importe quel type de programme. Il fonctionne avec une pile et un dictionnaire ; il utilise la notation « polonaise inverse » pour toutes ses opérations, et la pile est utilisée pour traiter tout type d'information (nombres, chaînes, code, …).

Opération à réaliser	**En Postscript**
1 + 3	1 3 add
if (x==3) { … }	3 eq { … } if
for (i=0 ; i<10 ; i++) { }	0 1 10 { …} for

Ce mode de fonctionnement rend les programmes Postscript assez difficile à lire…

Il y a quelques années, je m'étais intéressé au fonctionnement du langage Postscript, mais mes seuls souvenirs sont les commandes « pop », « dup » et « stack », et les les lignes de commentaires qui commencent par « % »…

5.2.2 **Visualisation du programme Postscript**

Pour ne pas modifier mon fichier original, je décide de travailler sur une première copie :

```
# cp data.dechiffre.unbase64 p1.ps
```

Le Postscript étant un langage écrit en texte clair, je décide d'ouvrir le fichier avec un éditeur de texte. Malheureusement, il n'y a aucun retour à la ligne : la seule ligne du fichier fait près de 32ko.

Le premier travail consiste à ajouter des retours à la ligne pour que ce programme devienne lisible. J'en profite pour localiser la gestion de l'argument passé en ligne de commande et je remarque également la présence de 2 blocs de code quasi-identiques, à une valeur près (0 ou 2). J'appelle ce bloc « A(x) » où x est la valeur 0 ou 2. Je rajoute quelques commentaires à ce sujet :

```
/I1
currentfile 0 (cafebabe)
/SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode filter
cf760bc77db1f282e881ede9a10122b220887466b973b854218b85c230d6733ab459fda9a879973664130312d5f…
cafebabe
def

/I2
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode
filter
c598d7cc5b5354440b0613490c45483d4e7b0f6c0368120f1001070501030202120914031508150202030504050…
cafebabe
def

/I3
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode
filter
338f25667eb4ec47763dab51c3fa41cba329e18536b83159b3a690a0265ec519aae94f0e715376c4f087bcccdd0…
cafebabe
def

/I4
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode
filter
8ae98ae900000000020002000316121411455556000855540012 4e5245114e011d1b48454c430f4540490911111…
cafebabe def

/error { (%stderr)(w) file exch writestring } bind def
errordict
/handleerror { quit } put
```

```
/main
{
	mark
	shellarguments
	{
		% Vérifie qu'on a bien un argument
		counttomark 1 eq
		{
			dup length exch /ReusableStreamDecode filter exch 2 idiv
			string readhexstring pop dup

			% Vérifie que la clef passée en argument fait bien 16 caractères
			length 16 eq
			{
				I1 32 exch mark 1 index resetfile 1 index
				{
					counttomark 1 sub index counttomark 2 add index
					4 mul string readstring pop dup () eq {pop exit} if
				} loop
				counttomark -1 roll counttomark 1 add 1 roll ] 4 1 roll
				pop pop pop    I2

				% ----- Bloc A(2)
				0 index resetfile 61440 string readstring
				pop dup 3 index
				2
				2 getinterval dup exch dup length 2
				index length add string dup dup 4 2 roll
				copy length 4 -1 roll putinterval 0
				0 1 1
				{
					pop 2 index length
				} for
				exch 1 sub
				{
					3 copy exch length getinterval 2 index mark 3 1 roll
					0 1 3 -1 roll dup length 1 sub exch 4 1 roll
					{
						dup 3 2 roll dup 5 1 roll exch get 3 1 roll
						exch dup 5 1 roll exch get xor 3 1 roll
					} for
					pop pop ] dup length string
					0 3 -1 roll
					{
						3 -1 roll dup 4 1 roll exch 2 index exch put 1 add
					} forall
					pop 4 -1 roll dup 5 1 roll 3 1 roll dup
					4 1 roll putinterval exch pop
				} for
				0 1 1 {pop pop} for
				cvx exec
				% ----- Fin du bloc

				I3 resetfile I4

				% ----- Bloc A(0)
				0 index resetfile 61440 string readstring
				pop dup 3 index
				0
				2 getinterval dup exch dup length 2
				index length add string dup dup 4 2 roll
				copy length 4 -1 roll putinterval 0
				0 1 1
				{
					pop 2 index length
				} for
				exch 1 sub
				{
					3 copy exch length getinterval 2 index mark 3 1 roll
					0 1 3 -1 roll dup length 1 sub exch 4 1 roll
					{
						dup 3 2 roll dup 5 1 roll exch get 3 1 roll
						exch dup 5 1 roll exch get xor 3 1 roll
					} for
					pop pop ] dup length string
					0 3 -1 roll
					{
```

```
                                3 -1 roll dup 4 1 roll exch 2 index exch put 1 add
                        } forall
                        pop 4 -1 roll dup 5 1 roll 3 1 roll dup
                        4 1 roll putinterval exch pop
                } for
                0 1 1 {pop pop} for
                cvx exec
                % ----- Fin du bloc A(0)

                } if false
        }
        {
                (no key provided\n) error true
        } ifelse
}
{
        (missing '--' preceding script file\n) error true
} ifelse
{
        (usage: gs -- script.ps key\n) error flush
} if
} bind def

main clear quit
```

On voit que la clef passée en paramètre, convertie en hexadécimal (readhexstring) doit faire 16 octets.

J'aimerais bien mettre les 2 blocs A(x) en « sous-routine » pour plus de lisibilité mais, en Postscript, cela n'est pas simple car le paramètre passé sur la pile décale toutes les références aux autres objets…

Comme nous savons que la clef fait 16 octets (32 digits hexa), je tente une exécution du programme en passant une clef de la bonne longueur :

```
# gs – p1.ps 0102030405060708090A0B0C0D0E0F10
GPL Ghostscript 9.05 (2012-02-08)
Copyright (C) 2010 Artifex Software, Inc.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
```

Toujours rien… il va falloir comprendre ce que fait ce programme.

5.2.3 **Debugger Postscript**

Pour comprendre le fonctionnement du programme et suivre son cheminement, il me faudrait un debugger…

Quelques recherches web plus tard, il faut se rendre à l'évidence : je n'en ai pas trouvé de gratuit…

Je décide donc de suivre l'exécution du programme Postscript en affichant des messages de trace dans le code. Pour cela, je me crée les « fonctions » Postscript suivantes et je les ajoute à la fin du fichier, juste avant l'appel à la fonction « main »:

```
% Affichage d'un texte
/print { (%stdout)(w) file exch writestring } bind def

% Saut de ligne
/newline { (\n) print } bind def

% Concaténation de 2 chaînes de caractères
/concatstrings % (a) (b) -> (ab)
{ exch dup length
        2 index length add string
        dup dup 4 2 roll copy length
        4 -1 roll putinterval
```

```
} bind def

% Affichage d'un octet en hexadécimal
/chartohex { 16 10 string cvrs (00000) exch concatstrings dup length 2 sub 2 getinterval }
bind def

% Affichage d'un mot de 16 bits en hexa
/wordtohex { 16 10 string cvrs (00000) exch concatstrings dup length 4 sub 4 getinterval }
bind def

% Affichage des 2 premiers octets d'une chaîne en hexa
/s2tohex
{
        dup
        0 get chartohex
        1 index 1 get chartohex concatstrings
        exch pop
} bind def

% Affichage des 4 premiers octets d'une chaîne en hexa
/s4tohex
{
        dup
        0 get chartohex
        1 index 1 get chartohex concatstrings
        1 index 2 get chartohex concatstrings
        1 index 3 get chartohex concatstrings
        exch pop
} bind def


main clear quit
```

Ainsi, je pourrai ajouter des traces dans le code, comme :

```
(Je passe par ici !\n) print
123 chartohex print
1234 wordtohex print
(WXYZ) s4tohex print
quit
```

qui affichera lors de son exécution :

```
Je passe par ici !
7B
04D2
5758595A
```

En plaçant des traces de manière judicieuse dans le programme, j'espère comprendre son fonctionnement.

5.3 Du code Postscript dynamique

5.3.1 **Le bloc A(x)**

Je décide de m'intéresser au bloc A(x) identifié précédemment. En particulier, je souhaite comprendre à quoi correspond la valeur x. Ce paramètre semble être utilisé pour manipuler une chaîne de caractères. J'ajoute donc une trace dans le programme pour mieux comprendre :

```
% ----- Bloc A(2)
0 index resetfile 61440 string readstring
pop dup 3 index
2
2 getinterval
(-----) print s2tohex print (-----) print quit
% …
```

Et j'exécute le programme (en ajoutant l'option « –q » pour supprimer l'affichage de la bannière) :

```
# gs –q -- p1.ps 0102030405060708090A0B0C0D0E0F10
-----0304-----
```

C'est peut être un hasard, mais je retrouve une partie de ma clef (0304). Je fais donc la même manipulation en remplaçant x=2 par x=0 comme pour la 2ème utilisation du bloc A(x) ; cette fois-ci, les 2 premiers octets de ma clef sont affichés (0102).

Le paramètre du bloc A(x) indique donc quelle partie de clef utiliser ! A(x) utilise ensuite les 2 octets qui commencent à partir de cette position.

A la fin du bloc A(x), je remarque les instructions suivantes :

```
cvx exec
% ----- Fin du bloc A(2)
```

Ces 2 instructions convertissent la chaîne de caractères qui se trouve sur la pile en instructions Postscript et les exécute. Je comprends alors que le bloc A(x) est un générateur de code Postscript dynamique : le programme génère lui-même ses propres instructions.

Je décide donc de les afficher en ajoutant juste avant :

```
print quit
cvx exec
% ----- Fin du bloc A(2)
```

Le résultat est illisible, et la conversion en instruction ne peut donc pas fonctionner correctement. Les instructions sont donc certainement codées avec la partie de clef identifiée (les 3ème et 4ème octets pour le premier bloc « A(2) »).

5.3.2 **Le code dynamique est codé**

Je redirige la sortie vers un fichier pour pouvoir l'afficher en hexadécimal :

```
# gs –q -- p1.ps 0102030405060708090A0B0C0D0E0F10 >a1
# hexdump –n 32 –C a1
00000000  c6 9c d4 c8 9d cf 80 8c  96 c9 93 c5 9a 8c db f8  |................|
00000010  d4 f7 d4 94 d7 9f c6 9b  c7 9e c1 9e c6 9d c3 9c  |................|
```

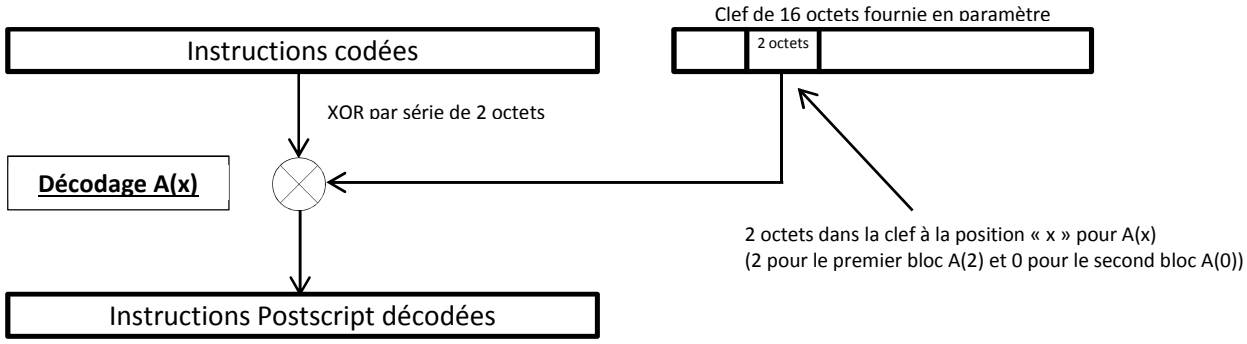
En utilisant 2 autres clefs complémentaires (0000 et FFFF), je m'aperçois que les résultats sont également complémentaires (a2 ^ a3 = 0xff), octet par octet.

```
# gs –q -- p1.ps 0000000000000000000000000000000 >a2
# hexdump -n 32 -C a2
00000000  c5 98 d7 cc 9e cb 83 88  95 cd 90 c1 99 88 d8 fc  |................|
00000010  d7 f3 d7 90 d4 9b c5 9f  c4 9a c2 9a c5 99 c0 98  |................|

# gs –q -- p1.ps 0000FFFF000000000000000000000000 >a3
# hexdump -n 32 -C a3
00000000  3a 67 28 33 61 34 7c 77  6a 32 6f 3e 66 77 27 03  |:g(3a4|wj2o>fw'.|
00000010  28 0c 28 6f 2b 64 3a 60  3b 65 3d 65 3a 66 3f 67  |(.(o+d:`;e=e:f?g|
```

Il s'agit donc certainement d'un codage par fonction XOR. Pour le vérifier, je décide de comparer les sorties a1 et a2 :

```
Sortie a2 (clef 00 00)     XOR   Sortie a1 (clef 03 04)
[c5 98 d7 cc 9e cb 83 88]  XOR   [c6 9c d4 c8 9d cf 80 8c]  =  [03 04 03 04 03 04 03 04]
```

Cette fois-ci, tout devient clair, la partie de clef utilisée sert à décoder des instructions Postscript. Cela se fait de la manière suivante :



Les instructions codées se trouvent vraisemblablement dans les éléments I2, I3 et/ou I4 mais il est inutile de s'attarder à comprendre leur fonctionnement. Les instructions codées peuvent être simplement retrouvées en utilisant une partie de clef égale à 0000 : de cette manière la transformation par XOR est sans effet, et la sortie de A(x) est égale à son entrée.

Les instructions codées se trouvent donc dans le fichier « a2 » que nous avons généré plus tôt.

5.4 Décodage du code dynamique

5.4.1 **Méthode utilisée**

Je suis assez confiant sur le décodage liée aux blocs A(x). En effet :

- La méthode de codage est connue (XOR avec 2 octets de la clef)
- Le codage par XOR est très simple
- La partie de clef utilisée n'est que sur 16 bits (2 octets)
- Nous savons ce que nous devons obtenir : des instructions Postscript

Nous pouvons donc envisager de trouver la partie de clef par « Brute Force » en cherchant des instructions Postscript courantes. Je choisis de chercher l'instruction « roll ».

5.4.2 **Programme de recherche de clef**

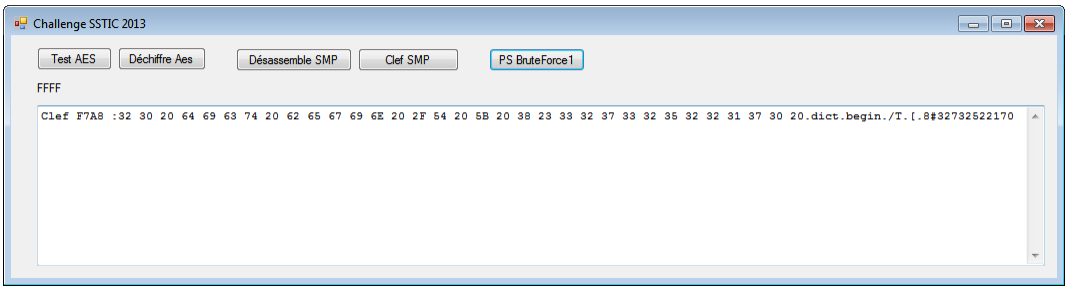
Le programme suivant est écrit pour tester toutes les parties de clefs possibles (2 octets) à la recherche du texte « roll » :

```
private void btnPS1_Click(object sender, EventArgs e)
{
    FileStream InFs = new FileStream("a2", FileMode.Open, FileAccess.Read);
    Byte[] Key = new Byte[2] { 0x00, 0x00 };

    // Lecture du fichier avec clef 0000
    Byte[] Data = new Byte[(int)InFs.Length];

    // Brute Force
    while (true)
    {
        // Affiche un message de progression
        labelInfos.Text = Key[0].ToString("X2") + Key[1].ToString("X2");
        labelInfos.Update();

        // On incrémente la clef
        Key[1]++; if (Key[1] == 0) { Key[0]++; if (Key[0] == 0) break; }
```

```
        // Lecture fichier
        InFs.Seek(0, SeekOrigin.Begin);
        int Count = InFs.Read(Data, 0, Data.Length);

        // Décodage avec la clef à tester
        for (int i = 0; i < Data.Length; i += 2)
        {
            Data[i] ^= Key[0];
            Data[i + 1] ^= Key[1];
        }

        // Clef trouvée ?
        string t = Encoding.ASCII.GetString(Data);
        if (t.IndexOf("roll") >= 0)
        {
            textResult.Text += "Clef " + Key[0].ToString("X2") + Key[1].ToString("X2") +
                               " :" + Dump(Data, 32, true) + "\r\n";
            textResult.Update();
        }
    }
    InFs.Close();
}
```

5.4.3 **Application au premier bloc : A(2)**

L'exécution du programme ci-dessus dure une dizaine de secondes et la clef est trouvée :



La partie de clef trouvée est F7A8. La clef globale utilisée en paramètre du fichier Postscript est donc de la forme « xxxxF7A8 xxxxxxxx xxxxxxxx xxxxxxxx ».

En utilisant la même trace Postscript que précédemment, on visualise les instructions décodées :

```
# gs –q -- p1.ps 0000F7A800000000000000000000000 >a2.decode
# head -c 128 a2.decode
20 dict begin /T [ 8#32732522170 8#35061733526 8#4410070333 8#30157347356 8#36537007657
8#10741743052 8#25014043023 8#3752151240
```

Pour mieux comprendre le fonctionnement du programme Postscript, les instructions finales du bloc A(2) sont remplacées de la manière suivante :

```
%cvx exec      % Cette ligne est commentée pour éviter l'exécution des instruction décodées
% ----- Fin du bloc A(2)
pop            % Cette ligne est ajoutée pour supprimer les instructions générées par A(2)
% ----- Le contenu du fichier a2.decode est inséré entre ces lignes
…
% ----- Fin d'insertion de a2.decode
```

On annule le décodage effectué par A(2) en supprimant la conversion et l'exécution des instructions décodées ; on nettoie la pile (en supprimant la chaîne qui contient les instructions) ; et on ajoute les instructions décodées.

De cette manière, on peut passer à la suite du programme : le bloc A(0)

5.4.4 **Application au second bloc : A(0)**

On sait déjà que le second bloc A(0) décode des instructions Postscript de la même manière que pour le premier bloc A(2).

On applique donc la même méthode en allant droit au but :

On commence par afficher le résultat du décodage à la fin du bloc A(0) :

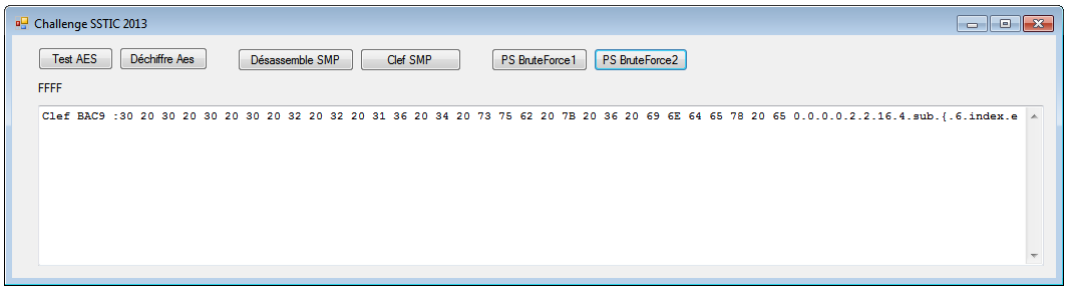
```
print quit
cvx exec
% ----- Fin du bloc A(0)
```

Et on exécute le programme pour récupérer les instructions codées dans le fichier a0 :

```
# gs –q -- p1.ps 0000F7A8000000000000000000000000 >a0
# hexdump –n 32 –C a0
00000000  8a e9 8a e9 8a e9 8a e9  88 e9 88 e9 8b ff 9a fd  |................|
00000010  9a ba cf ab 9a b2 9a ff  9a a0 d4 ad df b1 9a ac  |................|
```

Le programme précédent est recopié. La seule différence concerne le nom du fichier (a2 remplacé par a0). Un deuxième bouton est ajouté à l'interface graphique pour lancer cette 2ème recherche.

Je préfère utiliser 2 fonctions différentes avec 2 boutons dans l'interface utilisateur pour une meilleure traçabilité de mes manipulations. L'exécution de cette 2ème recherche donne le résultat suivant :



On peut vérifier que les instructions sont bien décodées avec :

```
# gs –q -- p1.ps BAC9F7A8000000000000000000000000 >a0.decode
# head -c 128 a0.decoded
0 0 0 0 2 2 16 4 sub { 6 index exch 4 getinterval 10240 { 0 0 1 3 { 3 -1 roll dup 4 1 roll
exch get exch 8 bitshift add } for ex
```

La partie de clef trouvée est BAC9. La clef globale utilisée en paramètre du fichier Postscript est donc de la forme « BAC9F7A8 xxxxxxxx xxxxxxxx xxxxxxxx ».

De la même manière que pour le premier bloc, on insère les instructions décodées dans le programme Postscript et on annule les effets du bloc A(0) :

```
%cvx exec      % Cette ligne est commentée pour éviter l'exécution des instruction décodées
% ----- Fin du bloc A(0)
pop            % Cette ligne est ajoutée pour supprimer les instructions générées par A(0)
% ----- Le contenu du fichier a0.decode est inséré entre ces lignes
…
% ----- Fin d'insertion de a0.decode
```

5.5 Fichier Output.bin

5.5.1 **Génération du fichier output.bin**

Nous avons maintenant un programme Postscript dans lequel les instructions générées dynamiquement par les 2 blocs A(2) et A(0) ont été insérées de manière statique (je décide de copier le fichier p1.ps en p2.ps pour conserver une trace de mes modifications).

Lorsqu'on lance l'exécution du programme avec les 4 octets de clef connus, on obtient maintenant :

```
# gs –q -- p2.ps BAC9F7A8000000000000000000000000
Key is invalid. Exiting ...
```

Mais le programme est très long à s'exécuter. Un examen rapide montre qu'une boucle est utilisée pour vider la pile juste avant l'affichage du message « Key is invalid » :

```
0 1 1073741823 {pop} for
(Key is invalid. Exiting ...\n) print flush quit
```

Le nombre d'itérations de cette boucle est gigantesque (1073741823) et c'est la raison pour laquelle le programme est si long. Je décide donc de commenter ce code :

```
%0 1 1073741823 {pop} for
(Key is invalid. Exiting ...\n) print flush quit
```

Et cette fois-ci, le message d'erreur est immédiat !

Je remarque également l'instruction « quit » juste après l'affichage du message d'erreur. Juste pour voir ce qui se passe, je décide de commenter cette instruction :

```
%0 1 1073741823 {pop} for
%(Key is invalid. Exiting ...\n) print flush %quit
```

On obtient maintenant 6 fois le même message, et un fichier « output.bin » est généré :

```
# gs –q -- p2.ps BAC9F7A8000000000000000000000000
Key is invalid. Exiting ...
Key is invalid. Exiting ...
Key is invalid. Exiting ...
Key is invalid. Exiting ...
Key is invalid. Exiting ...
Key is invalid. Exiting ...
# hexdump -n 64 -C output.bin
00000000  76 1c 73 72 4f 17 8b c3  0e 8d ab 4b eb e6 2e 36  |v.srO......K...6|
00000010  03 18 e2 0a 0e 2a 05 1f  38 0d 44 25 e1 1a 67 31  |.....*..8.D%..g1|
00000020  3e d6 bd 4e 68 ec a2 52  4a 55 1e e6 8d 2e 67 70  |>..Nh..RJU....gp|
00000030  8f da 9c b7 a8 85 ab 50  c3 66 c2 03 39 3c 04 7e  |.......P.f..9<.~|
```

Un essai avec une clef différente génère un fichier « output.bin » complètement différent…

Le fichier « output.bin » est chiffré !

Après avoir essayé différentes clefs très proches les unes des autres, et avoir constaté que la sortie est complètement différente, je dois me rendre à l'évidence : le chiffrement de ce fichier est complexe !

5.5.2 **Fonctionnement du déchiffrement de output.bin**

La génération et le déchiffrement du fichier « output.bin » sont effectués par les instructions générées par le bloc A(0). Nous les avons précédemment isolées dans le fichier a0.decode et elles sont également incluses dans le programme « p2.ps ». Je décide de les analyser en détail. La tâche semble rude car le langage Postscript n'est pas facile à « réverser ».

Les premières et dernières instructions de cette partie sont les suivantes (les autres sont masquées) :

```
0 0 0 0
2 2 16 4 sub
{
        6 index exch 4 getinterval
        %
        % Suite des instructions suprrimées pour plus de clarté
        % …

             (Key is invalid. Exiting ...\n) print flush %quit
             } if
} for
```

Il s'agit d'une boucle 'for' que l'on pourrait écrire en C sous la forme :

```
for (i=2 ; i<=12 ; i+=2)
{
        memcpy((char *)SommetPile, (char *)PileElement6 + i, 4);
        //
        // Suite des instructions supprimées pour plus de clarté
        // …

        printf("Key is invalid. Exiting ...\n"); // exit(0);
}
```

Cette boucle fait 6 itérations en parcourant un tableau de 2 en 2, et en traitant à chaque fois 4 valeurs. Je me demande bien quelles sont ces valeurs…

Pour le savoir, j'insère une trace de la manière suivante :

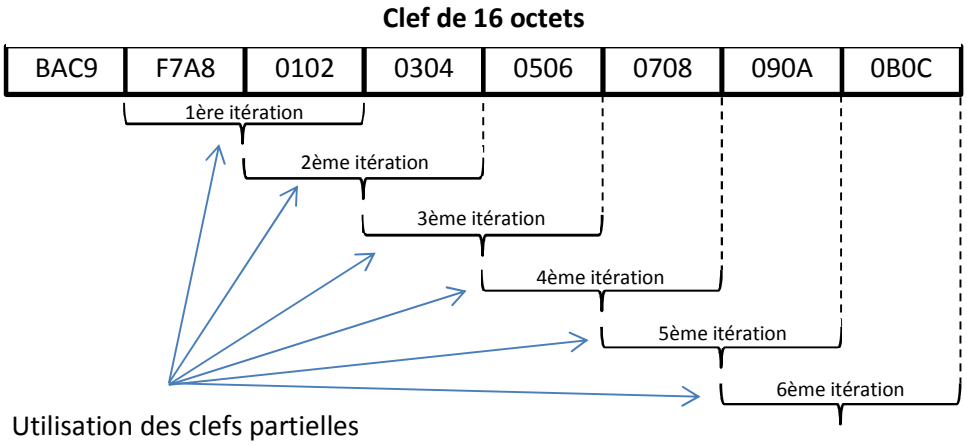
```
0 0 0 0
2 2 16 4 sub
{
        6 index exch 4 getinterval
        dup s4tohex print newline
        %
        % Suite des instructions suprrimées pour plus de clarté
        % …

             (Key is invalid. Exiting ...\n) print flush %quit
             } if
} for
```

Et l'exécution du programme donne :

```
# gs –q -- p2.ps BAC9F7A8010203040506070809 0A0B0C
F7A80102
Key is invalid. Exiting ...
01020304
Key is invalid. Exiting ...
03040506
Key is invalid. Exiting ...
05060708
Key is invalid. Exiting ...
0708090A
Key is invalid. Exiting ...
090A0B0C
Key is invalid. Exiting ...
```

On reconnait la clef passée en paramètre et découpée en morceaux…

Chaque itération correspond à une partie de la clef :



Utilisation des clefs partielles

Ce qui est intéressant, c'est qu'à chaque itération, le programme est capable de dire si la partie de clef est valide. C'est la raison pour laquelle nous obtenons 6 fois le message d'erreur. J'analyse donc le code de vérification :

```
0 3 2 roll { 3 copy putinterval length add } forall pop calc I3
16 string readstring pop
ne
{
	%0 1 1073741823 {pop} for
	(Key is invalid. Exiting ...\n) print flush %quit
} if
```

Je me rends compte qu'à la fin de chaque itération ce code compare les 16 premiers caractères du résultat d'un déchiffrement partiel à une chaîne prédéfinie, fournie par I3. Pour que le déchiffrement soit complet, il faut que ce test réussisse à chacune des 6 itérations. Cela confirme donc qu'à chaque utilisation de la clef, le programme de déchiffrement est capable de dire si la clef partielle est correcte ou non.

5.5.3 « Brute Force » progressif

On peut donc imaginer un algorithme de « brute force » qui teste progressivement chaque partie de 4 octets de la clef. Comme on a, à chaque itération, un recouvrement de 2 octets, et que les 2 premiers sont connus (F7A8), chaque étape de « brute force » n'aura que 16 bits à chercher !

Ainsi, en partant de la partie de clef connue (BAC9F7A8), notre programme va :

- Tester toutes les clefs possibles pour l'itération en cours (65536 possibilités) :
  BAC9F7A8.AAAA.0000.0000.0000.0000.0000
- Un fois la partie manquante trouvée (AAAA), il va passer à l'itération suivante :
  BAC9.F7A8.AAAA.BBBB.0000.0000.0000.0000
- Et ainsi de suite pour les 6 itérations, jusqu'à trouver la clef complète
  BAC9.F7A8.AAAA.BBBB.CCCC.DDDD.EEEE.FFFF

Le problème réside dans la vérification de la clef à chaque itération. Dans les précédentes étapes du Challenge, nous avons réécrit les algorithmes de déchiffrement en C# par souci de performance. Le *reverse engineering* de chacune des méthodes utilisées n'a pas posé trop de difficultés. Dans le cas

présent, je suis encore loin de comprendre la méthode de chiffrement utilisée et il est donc plus simple d'utiliser le programme Postscript existant pour effectuer le décryptage.

5.5.4 **Programme itératif sur la clef**

Pour mettre en application la méthode décrite ci-dessus, je modifie le programme Postscript « p2.ps » pour commenter l'appel à la fonction principale, et ajouter ma fonction « newmain » en remplacement.

Ce programme va effectuer les 6 itérations sur la clef, et pour chacune d'entre-elles, il va tester toutes les combinaisons possibles (de 0 à 65536). Pour vérifier que le mécanisme d'itération fonctionne correctement, le programme de déchiffrement n'est pas appelé pour le moment, et le premier essai de clef est considéré comme bon. Pour chaque partie de clef, la première valeur testée sera 0001 (et non pas 0000) afin de voir plus facilement l'effet sur la clef :

```
/newmain
{
	% On commence avec ce qu'on connait
	% C'est le préfixe de clef
	(BAC9F7A8)

	mark

	% 6 itérations sur les parties de clef
	1 1 6
	{
		% Indicateur de sous clef trouvée
		% La fonction main devra mettre true pour
		% indiquer que la clef partielle est correcte
		false

		% Itère les 65536 possibilités de clef
		% Pour le moment, on commence à 1
		1 1 65536
		{
			% Récupère le préfixe de la clef
			4 index

			% Utilise l'index de la boucle 'for'
			% pour générer les 2 octets suivants de la clef
			exch wordtohex concatstrings

			% Sauvegarde le préfixe en cours
			dup

			% Ajoute le suffixe pour compléter
			% la clef à 32 caractères
			(00000000000000000000000000) concatstrings
			dup 0 32 getinterval exch pop

			% Affiche un message de progression
			dup    (Test ) print print newline

			% Indique la profondeur de clef en cours à la function main
			3 index exch

			% Appel du programme de déchiffrement
			% On lui passe sur la pile : La profondeur en cours et la clef à tester
			% Lorsque la clef partielle est correcte
			% il doit modifier l'indicateur de clef trouvée
			% Pour le moment, l'appel est désactivé
			%main

			% Supprime l'indicateur de profondeur de clef
			pop

			% Supprime la clef de la pile
			pop
```

```
                        % Récupère l'indicateur de clef partielle trouvée
                        1 index
                        % Clef partielle trouvée ?
                        {
                                % Remplace le nouveau préfixe pour l'itération suivante
                                5 -1 roll pop 4 1 roll
                                % Passe à l'itération de clef suivante

                                exit
                        } if

                        % Supprime le préfixe sauvegardé
                        pop
                } for

                % Supprime l'indicateur de clef partielle trouvée
                pop
                % Supprime l'index de la boucle for
                pop
        } for
} bind def

newmain clear quit
```

Quand on lance ce programme, on constate bien les itérations sur la clef telles que nous les avons précédemment décrites. Comme nous avons simulé un succès à chaque étape, la première valeur pour chaque itération est considérée correcte, et le programme passe à la partie de clef suivante.

```
# gs -q -- p2.ps
Test BAC9F7A800010000000000000000000000
Test BAC9F7A800010001000000000000000000
Test BAC9F7A800010001000100000000000000
Test BAC9F7A800010001000100010000000000
Test BAC9F7A800010001000100010001000000
Test BAC9F7A800010001000100010001000010
Test BAC9F7A800010001000100010001000001
```

5.5.5 **Recherche de la clef**

Nous devons modifier légèrement le programme principal pour :

- Qu'il utilise la clef que lui passe « newmain » sur la pile et non plus celle passée en ligne de commande de Ghostscript
- Qu'il modifie l'indicateur de « clef partielle trouvée » de telle manière que « newmain » soit informée qu'une partie de clef a été trouvée ; pour cela nous allons modifier le comportement du programme lorsqu'il teste si une clef partielle est valide.

La première modification est simple : il suffit de désactiver le test sur les arguments passés en ligne de commande ; on commente les lignes suivantes et on ajoute la valeur « true » pour forcer les tests « if » :

```
/main
{
        %mark shellarguments
        true
        {
                % Vérifie qu'on a bien un argument
                %counttomark 1 eq
                true
                {
```

Le test d'une clef partielle est actuellement celui-ci :

```
% Résultat du test de clef partielle sur la pile
ne
{
	%0 1 1073741823 {pop} for
	(Key is invalid. Exiting ...\n) print flush %quit
} if
```

Nous le remplaçons par :

```
% Résultat du test de clef partielle sur la pile
ne
{
	%0 1 1073741823 {pop} for
	%(Key is invalid. Exiting ...\n) print flush %quit
	% Clef partielle invalide - on arrête
	exit
}
{
	% décrémente la profondeur de clef en cours
	7 -1 roll 1 sub 7 1 roll

	% On a atteint la profondeur cherchée ?
	6 index 0 eq
	{
		(Clef partielle Trouvee !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n) print
		%Indique au programme newmain que la sous-clef est correcte
		9 -1 roll pop true 9 1 roll
		% Passe à la sous-clef suivante
		exit
	} if
} ifelse
```

Cette modification fait en sorte que le programme « main » arrête immédiatement si une clef partielle est incorrecte. Si une clef partielle est correcte (jusqu'à la profondeur souhaitée), il informe « newmain » et arrête également.

Nous pouvons maintenant lancer ce programme :

```
# gs -q -- p2.ps
Test BAC9F7A80000000000000000000000000
Test BAC9F7A80001000000000000000000000
Test BAC9F7A80002000000000000000000000
Test BAC9F7A80003000000000000000000000
Test BAC9F7A80004000000000000000000000
Test BAC9F7A80005000000000000000000000
Test BAC9F7A80006000000000000000000000
Test BAC9F7A80007000000000000000000000
Test BAC9F7A80008000000000000000000000
Test BAC9F7A80009000000000000000000000
Test BAC9F7A8000A000000000000000000000
…
```

Cela va prendre un certain temps !

Nous allons mettre ce temps perdu à profit pour essayer de mieux comprendre la méthode de chiffrement utilisée…

5.6 Méthode de chiffrement utilisée

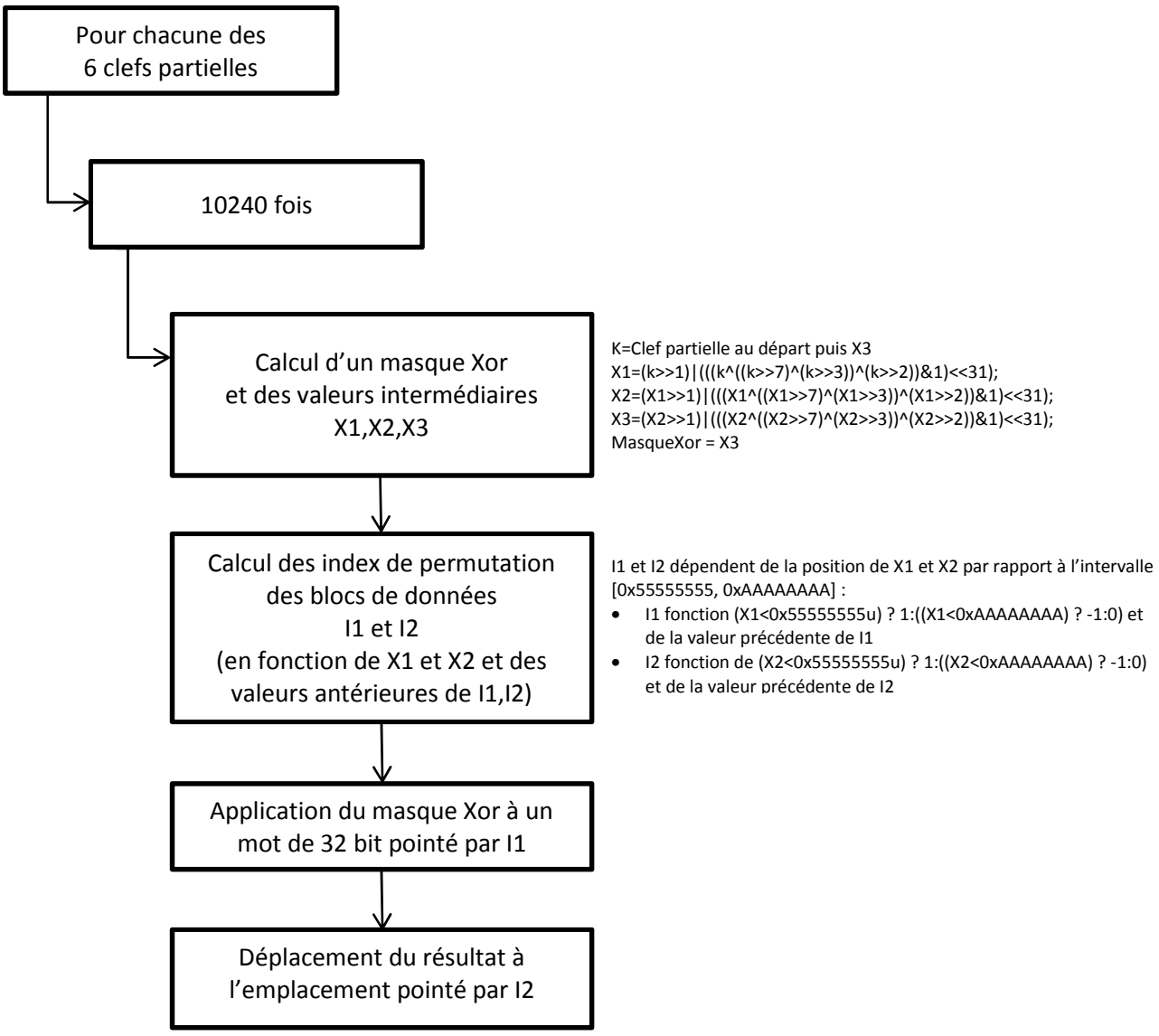
D'après la progression du programme « Brute Force » que j'ai lancé, j'estime à une douzaine d'heures le temps nécessaire pour trouver l'ensemble de la clef…

Pendant ce temps, je décide d'analyser en profondeur le programme de déchiffrement. Pour rappel, ce sont les instructions qui ont été générées par A(0) et que nous avons isolées dans « a0.decode ». Jusqu'à présent, nous avons simplement vu que ce code faisait 6 itérations sur des clefs partielles, et que pour chacune d'entre-elles, il était capable de dire si elle était correcte ou non.

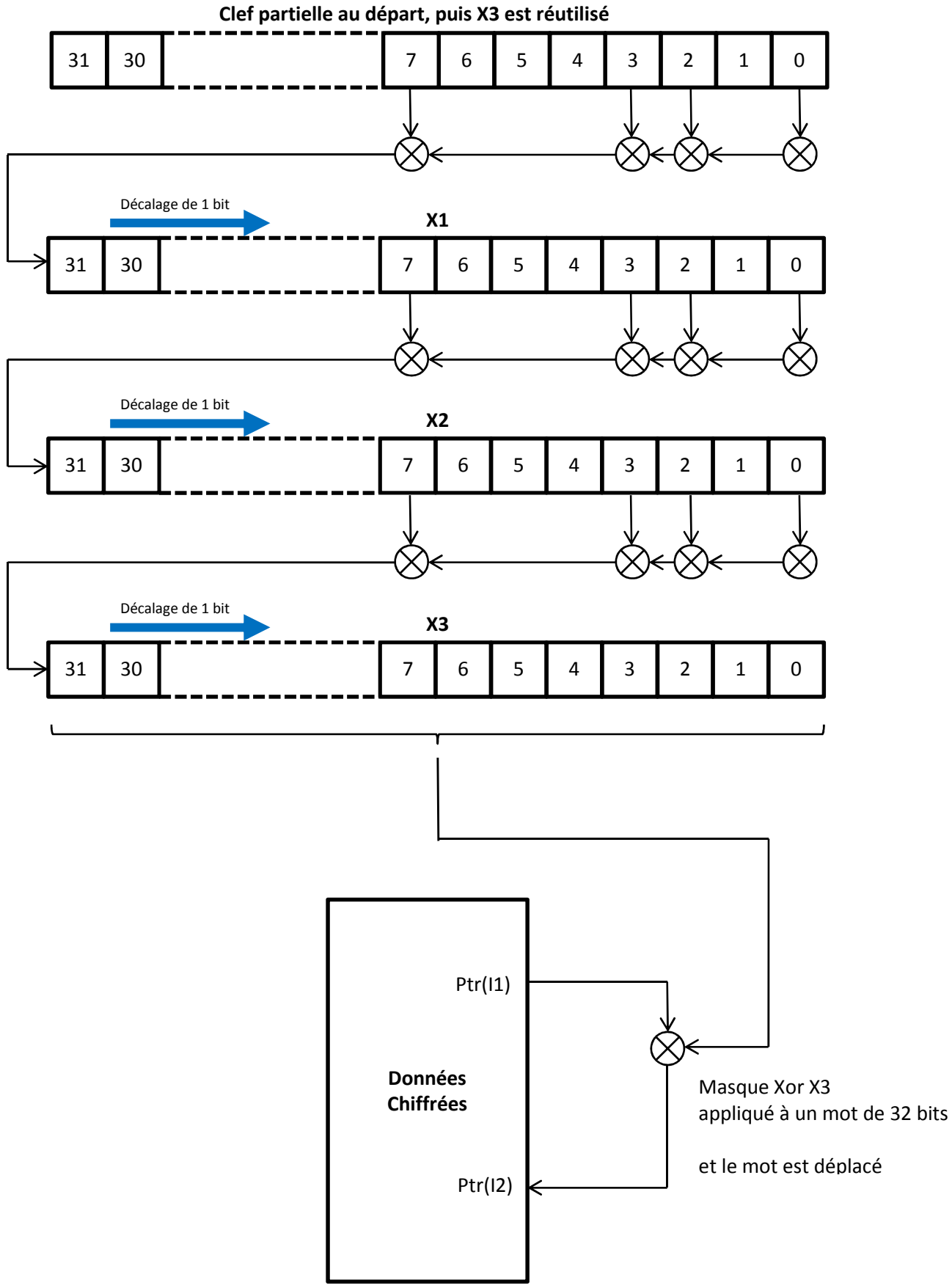
Nous allons maintenant nous intéresser à ce qui se passe au milieu… Pour cela, il faut faire un peu de « reverse ». Pour un programme Postscript, il faut être méthodique et suivre la progression de la pile.

Je décortique donc le programme Postscript en ajoutant des commentaires (en particulier sur le contenu de la pile), et des traces pour vérifier son fonctionnement. Il faut également faire très attention à certains détails comme la fonction Postscript 'mod' qui ne calcule pas un modulo (%), mais plutôt un reste de division, ce qui n'est pas pareil. Le listing complet figure en annexe.

Cela permet d'affiner la compréhension de l'algorithme :



Le calcul du masque Xor et le déplacement des données est représenté sur le schéma suivant :



Le même algorithme en C# ressemblerait à :

```
// Pour chacune des clefs partielles
for (int i=0; i<6; i++)
{
    // Lecture de la clef partielle
    int ki=i*2+2; UInt32 k=GetDword(ref Key, ki);

    // Parcours de données
    for (int j = 0; j < 10240; j++)
    {
        // Calcul du Crc et des index de déplacement
        X1 = (k >> 1) | (((k ^ ((k >> 7) ^ (k >> 3)) ^ (k >> 2)) & 1) << 31);

        int iTest1=(X1<0x55555555u) ? 1:((X1<0xAAAAAAAA) ? -1:0);
        UInt32 x1 = (UInt32)(I2+iTest1+Length1);
        UInt32 x2 = (UInt32)(Length1);
        I2 = (UInt32)(x1 - x2 * (x1 / x2)); if (I2 >= x2) I2 = 0;

        X2 = (X1 >> 1) | (((X1 ^ ((X1 >> 7) ^ (X1 >> 3)) ^ (X1 >> 2)) & 1) << 31);

        int iTest2 = (X2 < 0x55555555u) ? 1 : ((X2 < 0xAAAAAAAA) ? -1 : 0);
        x1 = (UInt32)((Length2 / 4) + I1 + iTest2);
        x2 = (UInt32)(Length2 / 4);
        I1 = (UInt32)(x1 - x2 * (x1 / x2)); if (I1 >= x2) I1 = 0;

        X3 = (X2 >> 1) | (((X2 ^ ((X2 >> 7) ^ (X2 >> 3)) ^ (X2 >> 2)) & 1) << 31);

        UInt32 v1 = GetDword(ref A1, (int)(I1 * 4 + I2 * Length2));
        UInt32 v2 = GetDword(ref A1, (int)(I3 * 4 + I4 * Length2));

        SetDword(ref A1, (int)(I1 * 4 + I2 * 128), v2);
        SetDword(ref A1, (int)(I3 * 4 + I4 * 128), v1);

        v2 = GetDword(ref A1, (int)(I3 * 4 + I4 * 128));
        Xor = (UInt32)(v2 ^ X3);
        SetDword(ref A1, (int)(I1 * 4 + I2 * 128), Xor);

        k = X3;
        I3 = I1;
        I4 = I2;
    }
}
```

Cette méthode de chiffrement utilise un triple « registre à décalage à rétroaction linéaire » ou LFSR (Linear Feedback Shift Register). Ce type de chiffrement est par exemple utilisé dans les communications GSM.

Dans notre cas, une permutation des blocs a été ajoutée et rend la méthode de chiffrement beaucoup plus efficace.

5.7 Déchiffrement du fichier « output.bin »

La méthode de recherche itérative des clefs partielles a bien fonctionné : après quelques heures, le programme a trouvé la clef complète :

```
Test BAC9F7A8721FAD3C9FCF271EED9ABBC4
Test BAC9F7A8721FAD3C9FCF271EED9ABBC5
Test BAC9F7A8721FAD3C9FCF271EED9ABBC6
Test BAC9F7A8721FAD3C9FCF271EED9ABBC7
Test BAC9F7A8721FAD3C9FCF271EED9ABBC8
Clef partielle Trouvee !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

La clef complète est : « BAC9F7A8721FAD3C9FCF271EED9ABBC8 » et le fichier « output.bin » a bien été déchiffré :

```
# file output.bin
output.bin: vCard visiting card

# head -n 5 output.bin
BEGIN:VCARD
VERSION:2.1
FN:Angela J. Matthews
N:Angela;J.;Matthews
ADR;WORK;PREF;QUOTED-PRINTABLE:;4139 Ryan Road;Sioux Falls, SD 57102
```

Il s'agit d'un carnet d'adresses au format VCard !

6. Le carnet d'adresses

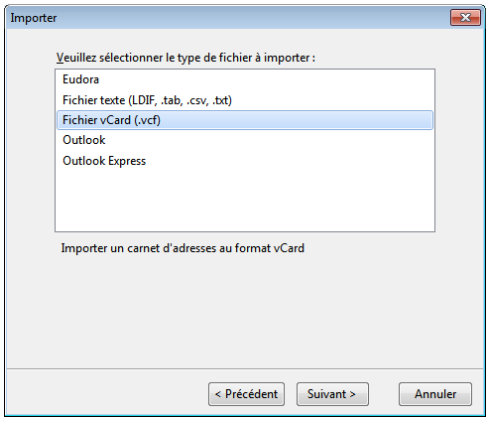
6.1 Ouverture du carnet d'adresses obtenu

Je commence par renommer le carnet d'adresses pour qu'il ait un nom plus parlant :

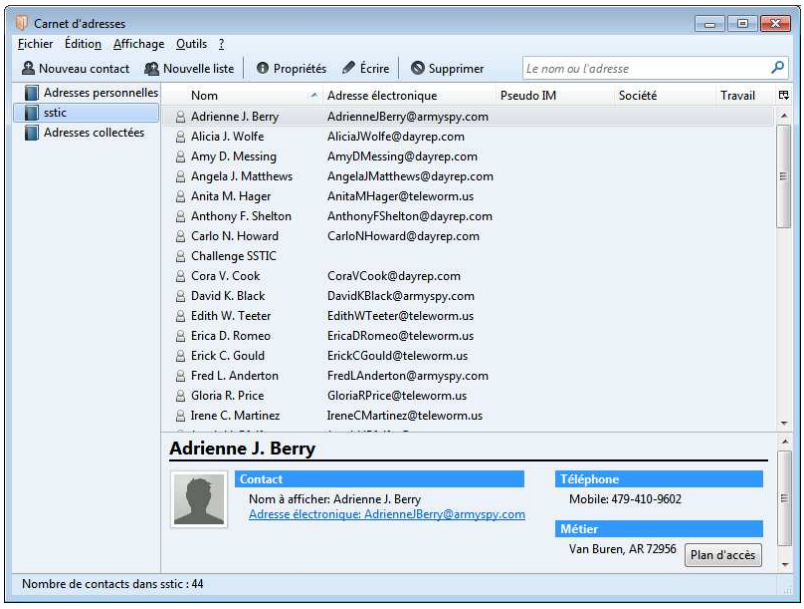
```
mv output.bin sstic.vcf
```

Le format VCard étant assez universel, il peut être ouvert avec de nombreux logiciels de gestion de carnet d'adresses. Thunderbird, que j'utilise déjà, fera l'affaire…

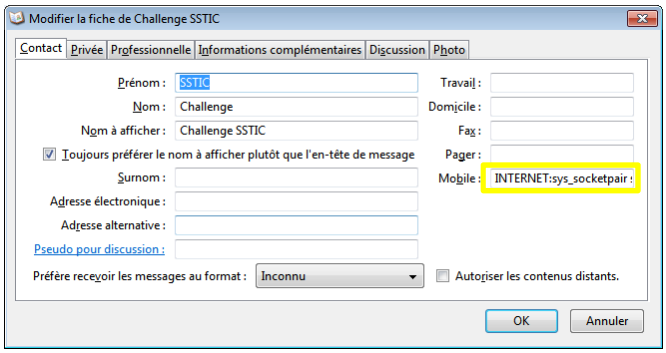
Je commence par importer le fichier « sstic.vcf » :



Le carnet d'adresses peut ensuite être affiché :



Nous recherchons une adresse au format « @challenge.sstic.org ». La seule entrée du carnet d'adresses qui concerne le SSTIC ne comportent malheureusement pas d'adresse mail…



Cependant, cette entrée comporte un champ « Mobile » qui attire mon attention :

```
INTERNET:sys_socketpair stub_fork sys_socketpair sys_getsockopt sys_socketpair sys_ptrace
sys_shutdown sys_ptrace sys_getsockopt sys_bind sys_getuid sys_bind sys_ptrace sys_getsockname
sys_ptrace stub_fork stub_fork sys_getpeername sys_setsockopt sys_getrusage sys_sysinfo
sys_getsockname sys_shutdown sys_getsockopt sys_getuid sys_sysinfo sys_getsockopt
sys_getrlimit sys_setsockopt sys_shutdown stub_clone sys_times sys_shutdown sys_getrusage
sys_socketpair sys_setsockopt stub_clone sys_getpeername sys_socketpair stub_clone sys_semget
sys_sysinfo sys_getgid sys_getrlimit sys_getegid sys_getegid sys_ptrace sys_getppid sys_syslog
sys_ptrace sys_sendmsg sys_getgroups sys_getgroups sys_setgroups sys_setuid sys_sysinfo
sys_sendmsg sys_getpgrp sys_setregid sys_syslog
```

Il s'agit vraisemblablement d'un texte codé. Je pense à un code du type César où chaque mot représente un caractère (en fait, la *vraie* méthode César associe une lettre à une autre).

6.2 Décodage de l'adresse mail

S'il s'agit d'un codage type César, 2 mots identiques doivent donner le même caractère décodé.

Comme l'adresse attendue est de la forme « @challenge.sstic.org », on devrait retrouver 2 mots identiques aux emplacements des caractères « e », « l », « s » et « . ».

Vérifions-le en analysant les derniers mots du texte codé :

Mot dans la phrase codé	Caractère attendu
sys_getegid	l
sys_getegid	l
sys_ptrace	e
sys_getppid	n
sys_syslog	g
sys_ptrace	e
sys_sendmsg	.
sys_getgroups	s
sys_getgroups	s
sys_setgroups	t
sys_setuid	i
sys_sysinfo	c
sys_sendmsg	.
sys_getpgrp	o
sys_setregid	r
sys_syslog	g

On constate que c'est bien le cas : le même mot code les mêmes lettres.

L'ensemble des mots utilisés concerne des appels système linux. Une recherche sur internet aboutit rapidement à la table des vecteurs des appels système (par exemple sur http://stuff.mit.edu:8001/afs/sipb.mit.edu/contrib/linux/arch/x86/syscalls/syscall_64.tbl) :

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
…
46      64      sendmsg                 sys_sendmsg
…
101     64      ptrace                  sys_ptrace
…
```

L'appel système « sys_ptrace » qui code la lettre « e » a pour numéro de vecteur « 101 » qui correspond justement au code Ascii de cette lettre. Pour l'appel système « sys_sendmsg » qui code le caractère « . » on fait le même constat : le numéro de l'appel système (46) correspond au code Ascii de la lettre codée.

Pour décoder plus facilement ce texte, je décide de l'importer dans un tableur, en plaçant un mot par ligne. J'importe aussi le tableau des appels systèmes :

	A	B	C	D	E	F
1						
2						
3	**Contenu du fichier VCF**				**System call**	**Entry vector**
4	sys_socketpair				sys_read	0
5	stub_fork				sys_write	1
6	sys_socketpair				sys_open	2
7	sys_getsockopt				sys_close	3
8	sys_socketpair				sys_newstat	4
9	sys_ptrace				sys_newfstat	5
10	sys_shutdown				sys_newlstat	6
11	sys_ptrace				sys_poll	7
12	sys_getsockopt				sys_lseek	8
13	sys_bind				sys_mmap	9
14	sys_getuid				sys_mprotect	10
15	sys_bind				sys_munmap	11
16	sys_ptrace				sys_brk	12
17	sys_getsockname				sys_rt_sigaction	13
18	sys_ptrace				sys_rt_sigprocmas	14
19	stub_fork				stub_rt_sigreturn	15
20	stub_fork				sys_ioctl	16
21	sys_getpeername				sys_pread64	17

Et je rajoute une colonne qui fait l'association Mot/Code Ascii en ajoutant la formule suivante dans la colonne C :

```
=CAR(SI(RECHERCHEV($A4;$E:$F;2;FAUX)=0;"";RECHERCHEV($A4;$E:$F;2;FAUX)))
```

On voit que cela fonctionne parfaitement pour la fin de l'adresse mail :

	A	B		C	D	E	F
43	stub_clone		**8**			sys_getpid	39
44	sys_semget		**@**			sys_sendfile64	40
45	sys_sysinfo		**c**			sys_socket	41
46	sys_getgid		**h**			sys_connect	42
47	sys_getrlimit		**a**			sys_accept	43
48	sys_getegid		**l**			sys_sendto	44
49	sys_getegid		**l**			sys_recvfrom	45
50	sys_ptrace		**e**			sys_sendmsg	46
51	sys_getppid		**n**			sys_recvmsg	47
52	sys_syslog		**g**			sys_shutdown	48
53	sys_ptrace		**e**			sys_bind	49
54	sys_sendmsg		**.**			sys_listen	50
55	sys_getgroups		**s**			sys_getsockname	51
56	sys_getgroups		**s**			sys_getpeername	52
57	sys_setgroups		**t**			sys_socketpair	53
58	sys_setuid		**i**			sys_setsockopt	54
59	sys_sysinfo		**c**			sys_getsockopt	55
60	sys_sendmsg		**.**			stub_clone	56
61	sys_getpgrp		**o**			stub_fork	57
62	sys_setregid		**r**			stub_vfork	58
63	sys_syslog		**g**			stub_execve	59

Pour obtenir une adresse facile à lire, je rajoute la formule suivante qui concatène l'ensemble des caractères trouvés :

```
=C4&C5&C6&C7&C8&C9&C10&C11&C12&C13&C14&C15&C16&C17&C18&C19&C20&C21&C22&C23&C24&C25&C26&C27&C28
&C29&C30&C31&C32&C33&C34&C35&C36&C37&C38&C39&C40&C41&C42&C43&C44&C45&C46&C47&C48&C49&C50&C51&C
52&C53&C54&C55&C56&C57&C58&C59&C60&C61&C62&C63
```

Le résultat final est le suivant :

	A	B	C	D	E	F
1	@Email =	**59575e0e71f1e3e9946bc307fc7a608d0b568458@challenge.sstic.org**				
2						
3	**Contenu du fichier VCF**		**Résultat**		**System call**	**Entry vector**
4	sys_socketpair		**5**		sys_read	0
5	stub_fork		**9**		sys_write	1
6	sys_socketpair		**5**		sys_open	2
7	sys_getsockopt		**7**		sys_close	3
8	sys_socketpair		**5**		sys_newstat	4

6.3 C'est fini ?

L'adresse cherchée est :

59575e0e71f1e3e9946bc307fc7a608d0b568458@challenge.sstic.org

J'envoie un mail à cette adresse en pensant qu'un automate va me renvoyer une confirmation dans les minutes qui suivent, mais ce n'est pas le cas… La première partie de cette adresse est de l'hexadécimal (59 57 5E 0e …) : y-aurait-il un codage supplémentaire ???

Il est 6 heures du matin… La confirmation du SSTIC arrivera quelques heures plus tard.

7. Conclusion

La résolution de ce Challenge était vraiment passionnante. Dès le départ, j'ai été captivé, et il m'a été impossible de m'en détacher avant qu'il ne soit résolu.

Chacune des étapes a fait appel à des techniques différentes (crypto, reverse hard, reverse soft, …)

Merci aux concepteurs de ce Challenge !

8. Annexes

8.1 Listing complet du programme développé

8.1.1 Fichier SSTIC2013.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.IO;

namespace WindowsFormsApplicationForSSTIC
{
    public partial class SSTIC2013 : Form
    {
        public SSTIC2013()
        {
            InitializeComponent();
        }

        //
        // Etape AES
        //
        public static byte[] AesDechiffre(byte[] DataToDecrypt, byte[] Key, byte[] IV)
        {
            AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
            Provider.Key = Key;
            Provider.Mode = CipherMode.CBC;
            Provider.Padding = PaddingMode.None;
            Provider.IV = IV;

            MemoryStream ms = new MemoryStream(DataToDecrypt);
            CryptoStream cs = new CryptoStream(ms, Provider.CreateDecryptor(Provider.Key, Provider.IV),
                                                                        CryptoStreamMode.Read);
            byte[] DecryptedData = new byte[DataToDecrypt.Length];
            var byteCount = cs.Read(DecryptedData, 0, DataToDecrypt.Length);
            Array.Resize(ref DecryptedData, byteCount);
            return DecryptedData;
        }

        private void TestPermutationTTL(ref Byte[] Key, Byte iPermut)
        {
            Byte[][] Swaps = new Byte[][]
            {
                new Byte[4] {0,1,2,3},
                new Byte[4] {0,1,3,2},
                new Byte[4] {0,2,1,3},
                new Byte[4] {0,2,3,1},
                new Byte[4] {0,3,1,2},
                new Byte[4] {0,3,2,1},

                new Byte[4] {1,0,2,3},
                new Byte[4] {1,0,3,2},
                new Byte[4] {1,2,0,3},
                new Byte[4] {1,2,3,0},
                new Byte[4] {1,3,0,2},
                new Byte[4] {1,3,2,0},

                new Byte[4] {2,0,1,3},
                new Byte[4] {2,0,3,1},
                new Byte[4] {2,1,0,3},
                new Byte[4] {2,1,3,0},
                new Byte[4] {2,3,1,0},
                new Byte[4] {2,3,0,1},

                new Byte[4] {3,0,1,2},
                new Byte[4] {3,0,2,1},
                new Byte[4] {3,1,0,2},
                new Byte[4] {3,1,2,0},
                new Byte[4] {3,2,1,0},
                new Byte[4] {3,2,0,1}
            };
            // Pour tous les octets de la clef
            for (int i = 0; i < Key.Length; i++)
            {
                // Pour chacun des quartets
                Key[i] = (Byte)((Key[i] & ~0x03) | Swaps[iPermut][Key[i] & 0x03]);
                Key[i] = (Byte)((Key[i] & ~0x30) | (Swaps[iPermut][(Key[i] >> 4) & 0x03] << 4));
            }
        }

        private void TestPolarite(ref Byte[] Key, Byte XorMask)
        {
            // Applique le mask Xor sur chacun des quartets de la clef
            for (int i = 0; i < Key.Length; i++)
                Key[i] = (Byte)(Key[i] ^ XorMask);
        }

        private void TestPermutationBits(ref Byte[] Key, Byte iPermut)
        {
            UInt16[] Swaps = new UInt16[24]
            {
```

```
            0x3210, // Pas de permutation
            0x3201,
            0x3120,
            0x3102,
            0x3021, // Bit2->Bit0, Bit1->Bit2, Bit0->Bit1
            0x3012,
            0x2310,
            0x2301,
            0x2130,
            0x2103,
            0x2021,
            0x2012,
            0x1320,
            0x1302,
            0x1230,
            0x1203,
            0x1032,
            0x1023,
            0x0321,
            0x0312,
            0x0231,
            0x0213,
            0x0132,
            0x0123 // Bit3->Bit0, Bit2->Bit1, Bit1->Bit2, Bit0->Bit3
        };
        // Permute les bits de chacun des quartets de la clef
        UInt16 sw = Swaps[iPermut];
        for (int i = 0; i < Key.Length; i++)
        {
            Byte c = 0;
            for (Byte iSrcBit = 0; iSrcBit < 4; iSrcBit++)
            {
                Byte iDestBit = (Byte)((sw >> (iSrcBit * 4)) & 0xf);
                // Quartet de poids faible
                if ((Key[i] & (1 << iSrcBit)) != 0) c |= (Byte)(1 << iDestBit);
                // Quartet de poids fort
                if ((Key[i] & (1 << (iSrcBit + 4))) != 0) c |= (Byte)(1 << (iDestBit + 4));
            }
            Key[i] = c;
        }
    }

    string Dump(Byte[] Data, int Len, Boolean bAscii)
    {
        string s = "";
        for (int i = 0; i < Len; i++) s += Data[i].ToString("X2") + " ";
        if (bAscii)
            for (int i = 0; i < Len; i++)
                s += (Data[i] <= 0x20 || Data[i] > 0x7f) ? "." : Encoding.UTF8.GetString(new Byte[] { Data[i] });
        return s;
    }

    byte[] IV = new byte[16] { 0x76,0xC1,0x28,0xD4,0x6A,0x6C,0x4B,0x15,0xB4,0x30,0x16,0x90,0x4B,0xE1,0x76,0xAC };
    private void BtnTestAes_Click(object sender, EventArgs e)
    {
        byte[] Key = new byte[] { 0xBB,0xCF,0xA5,0xB3,0x5E,0x68,0xDD,0xA9,0x21,0x7E,0x1D,0xFB,0x4E,0x7D,0x68,0xEC,
                                  0x1E,0xB8,0x19,0xF7,0xC2,0x4E,0xFB,0x4B,0x39,0x6A,0xDD,0xBC,0x6D,0x61,0xDE,0x87 };

        byte[] EncryptedData = { // 256 premiers octets du fichier sstic.tar.gz-chiffre
            0x96,0xc7,0x1d,0x86,0x0f,0x62,0xeb,0x7d,0xf8,0x01,0x53,0x43,0xdf,0x4c,0x2e,0xca,
            0xd1,0xd1,0xb3,0x9e,0x1f,0x7c,0x75,0xb7,0xfb,0xe3,0xf6,0x47,0x28,0xac,0x88,0x5e,
            0x53,0xa4,0x5e,0x93,0x8a,0xb5,0x38,0xde,0x53,0x4a,0x8c,0x28,0x4e,0xaf,0x40,0x2b,
            0xc2,0xd2,0x94,0xc3,0x1e,0xc9,0x29,0x16,0x74,0xc2,0x94,0x73,0xad,0x16,0xe3,0x50,
            0xd9,0x73,0xa4,0x8e,0x56,0x44,0x71,0x01,0xf2,0x5f,0x20,0x27,0x1b,0xba,0x9e,0x85,
            0x40,0x61,0xb5,0xc2,0x3c,0x80,0x42,0xb4,0x93,0xe5,0x72,0xf5,0x0b,0x71,0x92,0xc2,
            0x68,0x5d,0x20,0x94,0x07,0xf7,0x74,0x4a,0x54,0xc6,0xc4,0xd0,0xdf,0xe5,0x11,0x53,
            0x2e,0x12,0x3e,0x3e,0xe6,0x5c,0xbd,0xa2,0x45,0x33,0xf0,0xca,0x1e,0xc2,0x4e,0x57,
            0x54,0xe1,0xf7,0x4b,0xde,0x1d,0x98,0x77,0x2f,0xfc,0x33,0xe9,0x52,0x9c,0xe7,0x7c,
            0xc6,0x02,0xf8,0x7d,0xaa,0x5c,0x2e,0xfc,0x98,0x93,0xb9,0x36,0xc8,0x53,0x7f,0xdd,
            0x2f,0xb1,0x6e,0x04,0x9b,0xbe,0x48,0x56,0x3d,0x3b,0x5d,0x86,0x67,0x64,0x01,0x9f,
            0xf9,0x9c,0xa7,0x49,0x65,0xd2,0x06,0xd1,0xa5,0xc4,0xa2,0xd9,0xcf,0x3a,0xb2,0x18,
            0x33,0xf8,0x26,0xd0,0xec,0xdb,0xd1,0xe1,0x44,0x0f,0xe8,0xda,0x03,0xa7,0x09,0xe3,
            0x14,0x06,0x0c,0x34,0x98,0x54,0x56,0x16,0x8e,0xf0,0x80,0x88,0xd4,0x72,0x82,0xbf,
            0x11,0xea,0x56,0x91,0x33,0x70,0xa6,0x15,0xe5,0xc2,0x56,0xb1,0x20,0xaa,0xee,0x66,
            0xf5,0x87,0x46,0xf1,0x5c,0x87,0x3e,0x03,0x69,0xd9,0xae,0x6c,0x65,0xe7,0x69,0xde};

        Byte[] TestKey = new Byte[32];
        for (Byte iPermutTTL = 0; iPermutTTL < 24; iPermutTTL++)
        {
            for (Byte iPermutBitDelai = 0; iPermutBitDelai < 2; iPermutBitDelai++)
            {
                for (Byte iPermutBitTOS = 0; iPermutBitTOS < 2; iPermutBitTOS++)
                {
                    for (Byte iPermutBits = 0; iPermutBits < 24; iPermutBits++)
                    {
                        labelInfos.Text = "iPermutTTL=" + iPermutTTL.ToString() +
                                          " iPermutDelai=" + iPermutBitDelai.ToString() +
                                          " iPermutTOS=" + iPermutBitTOS.ToString() +
                                          " iPermutBits=" + iPermutBits.ToString();
                        labelInfos.Update();

                        // Préparation de la clef à tester
                        for (int i = 0; i < TestKey.Length; i++) TestKey[i] = Key[i];

                        // Permutation des symboles du TTL
                        TestPermutationTTL(ref TestKey, iPermutTTL);

                        // Polarité des bits Délai et TOS
                        Byte XorMask = (Byte)(((iPermutBitDelai == 1) ? 0x88:0x00) | ((iPermutBitTOS==1) ? 0x44:0x00));
                        TestPolarite(ref TestKey, (Byte)XorMask);

                        // Permutation des bits du quartet
                        TestPermutationBits(ref TestKey, iPermutBits);
```

```
                        // Tentative de décryptage
                        AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
                        Provider.Key = TestKey;
                        Provider.Mode = CipherMode.CBC;
                        Provider.Padding = PaddingMode.None;
                        Provider.IV = IV;
                        MemoryStream ms = new MemoryStream(EncryptedData);
                        CryptoStream cs = new CryptoStream(ms, Provider.CreateDecryptor(Provider.Key, Provider.IV),
                                                                        CryptoStreamMode.Read);
                        byte[] DecryptedData = new byte[EncryptedData.Length];
                        var byteCount = cs.Read(DecryptedData, 0, EncryptedData.Length);
                        Array.Resize(ref DecryptedData, byteCount);

                        // En-tête GZIP trouvée ?
                        if (DecryptedData[0] == 0x1f && DecryptedData[1] == 0x8b)
                        {
                            // Affichage de la clef
                            textResult.Text += labelInfos.Text + " / Key=" + Dump(TestKey, 32, false);
                            textResult.Text += "\r\nRésultat=" + Dump(DecryptedData, 16, true) + "\r\n";
                        }
                    }
                }
            }
        }
    }

    private void btnDechiffreAes_Click(object sender, EventArgs e)
    {
        Byte[] Key = new Byte[32] { 0xDD,0x8C,0xF2,0xD5,0x2E,0x69,0xAA,0xFB,0x73,0x4E,0x3A,0xCD,0x0E,0x4A,0x69,0xE8,
                                    0x3E,0xD9,0x3B,0xC4,0x87,0x0E,0xCD,0x0D,0x5B,0x6F,0xAA,0xD8,0x6A,0x63,0xAE,0x94 };

        AesCryptoServiceProvider Provider = new AesCryptoServiceProvider();
        Provider.Key = Key;
        Provider.Mode = CipherMode.CBC;
        Provider.Padding = PaddingMode.ISO10126;
        Provider.IV = IV;

        FileStream InFs = new FileStream("sstic.tar.gz-chiffre.unbase64", FileMode.Open, FileAccess.Read);
        CryptoStream cs = new CryptoStream(InFs, Provider.CreateDecryptor(Provider.Key, Provider.IV),
                                                                        CryptoStreamMode.Read);
        Byte[] DecryptedData = new Byte[InFs.Length];
        var byteCount = cs.Read(DecryptedData, 0, DecryptedData.Length);
        InFs.Close();

        FileStream OutFs = new FileStream("sstic.tar.gz-chiffre.dechiffre", FileMode.Create, FileAccess.Write);
        OutFs.Write(DecryptedData, 0, byteCount);
        OutFs.Close();
    }

    //
    // Etape SMP
    //

    // Programme SMP
    Byte[] smp = new Byte[231]
    {
        0x00,0xb0,0x10,0xd0,0xb7,0xa8,0xa0,0xb6,0x0e,0xb5,0x71,0xb4,0x00,0xbc,0x01,0xb6,
        0x16,0xb5,0x71,0xb4,0x00,0xbc,0x52,0xb6,0xaf,0xbe,0x11,0xb7,0xa8,0xb6,0x24,0xb5,
        0x71,0xb4,0x00,0xbc,0xaf,0xb1,0xa9,0xd0,0xb6,0x0f,0x88,0xd0,0x96,0xb6,0xa9,0xd0,
        0xb7,0x0f,0x88,0xd0,0x8f,0xb7,0xaf,0xa0,0x8e,0xb7,0x40,0xb6,0x53,0xb5,0x00,0xbd,
        0xaf,0xc1,0x01,0xb6,0xa8,0xb7,0x4c,0xb5,0x71,0xb4,0x00,0xbc,0xaf,0xb0,0x02,0xb6,
        0x00,0xbe,0xc8,0x01,0xb5,0x00,0xb4,0x6d,0xb3,0xad,0xbb,0x66,0xb3,0xac,0xd8,0xb4,
        0xad,0x8f,0xbb,0x01,0x94,0xb4,0xad,0xd8,0xb5,0x57,0xb3,0x00,0xbb,0xac,0xb7,0x00,
        0xbe,0x00,0xb1,0xb3,0x01,0xb2,0x63,0xe0,0xb4,0xaa,0xbc,0x2c,0xe0,0xb4,0xaf,0x8a,
        0xbc,0x16,0xe0,0xb4,0xae,0x8a,0xbc,0x0f,0xe0,0xb4,0xa9,0xbc,0xab,0x92,0xb3,0xaa,
        0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x22,0xe0,0xb4,0xa9,0xbc,0xaa,0xb1,0x59,0xe0,0xb4,
        0x00,0xbc,0xab,0x92,0xb3,0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x48,0xe0,0xb4,0xae,
        0x8a,0xbc,0x3e,0xe0,0xb4,0xa9,0xbc,0xaa,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0xaa,0x93,
        0xb3,0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x57,0xe0,0xb4,0xa9,0xbc,0xaa,0x93,0xb3,
        0x00,0xb1,0x59,0xe0,0xb4,0x00,0xbc,0x00,0xb1,0xaa,0xd8,0xb2,0xa9,0xd8,0xb1,0x76,
        0xb4,0x00,0xbc,0xab,0xb7,0x00,0xbd
    };

    private void btnDesassemble_Click(object sender, EventArgs e)
    {
        // Parcours toutes les instructions du programme SMP
        for (Byte ip = 0; ip < smp.Length; ip++)
        {
            // Lecture de l'instruction
            Byte Instruction = smp[ip];
            // Isole le n° de registre
            Byte Register = (Byte)(Instruction & 0x7);
            // Affiche l'adresse
            string s = ip.ToString("X2") + ": ";
            // Display instruction
            if ((Instruction >= 0x00) && (Instruction <= 0x7F))
                s += "A=0x" + Instruction.ToString("X2");
            else if ((Instruction >= 0x88) && (Instruction <= 0x8F))
                s += "A=A&R" + Register.ToString();
            else if ((Instruction >= 0x90) && (Instruction <= 0x97))
                s += "A=A|R" + Register.ToString();
            else if ((Instruction >= 0xA0) && (Instruction <= 0xA7))
                s += "A=~A";
            else if ((Instruction >= 0xA8) && (Instruction <= 0xAF))
                s += "A=R" + Register.ToString();
            else if ((Instruction >= 0xB0) && (Instruction <= 0xB7))
                s += "R" + Register.ToString() + "=A";
            else if ((Instruction >= 0xB8) && (Instruction <= 0xBF))
                s += "IF A==0 GOTO R" + Register.ToString();
            else if ((Instruction >= 0xC0) && (Instruction <= 0xC7))
                s += "Mem[R" + Register.ToString() + "]=A";
            else if (Instruction == 0xC8)
```

```
                s += "HALT";
            else if (Instruction == 0xD0)
                s += "A=Mem[A]";
            else if ((Instruction >= 0xD8) && (Instruction <= 0xDF))
                s += "LSHIFT A";
            else if ((Instruction >= 0xE0) && (Instruction <= 0xE7))
                s += "A=A|0x80";
            else
                s += "Illegal OpCode !";
            // Affiche l'instruction
            textResult.Text += s + "\r\n";
        }
    }

    Byte[] Reg = new Byte[8]; Byte RegA;
    void SimulateFpga(Byte[] Mem)
    {
        Byte ip = 0;
        RegA = 0;
        for (int i = 0; i < 8; i++) Reg[i] = 0;

        while (true)
        {
            labelInfos.Text = ip.ToString("X2");
            labelInfos.Update();
            Byte Instruction = smp[ip];
            Byte Register = (Byte)(Instruction & 0x7);

            if ((Instruction >= 0x00) && (Instruction <= 0x7F))
                RegA = Instruction;
            else if ((Instruction >= 0x88) && (Instruction <= 0x8F))
                RegA = (Byte)(RegA & Reg[Register]);
            else if ((Instruction >= 0x90) && (Instruction <= 0x97))
                RegA = (Byte)(RegA | Reg[Register]);
            else if ((Instruction >= 0xA0) && (Instruction <= 0xA7))
                RegA = (Byte)~RegA;
            else if ((Instruction >= 0xA8) && (Instruction <= 0xAF))
                RegA = Reg[Register];
            else if ((Instruction >= 0xB0) && (Instruction <= 0xB7))
                Reg[Register] = RegA;
            else if ((Instruction >= 0xB8) && (Instruction <= 0xBF))
            { if (RegA == 0) { ip = (Byte)Reg[Register]; continue; } }
            else if ((Instruction >= 0xC0) && (Instruction <= 0xC7))
                Mem[Reg[Register]] = RegA;
            else if (Instruction == 0xC8)
                return;
            else if (Instruction == 0xD0)
                RegA = Mem[RegA];
            else if ((Instruction >= 0xD8) && (Instruction <= 0xDF))
                RegA <<= 1;
            else if ((Instruction >= 0xE0) && (Instruction <= 0xE7))
                RegA |= 0x80;
            ip++;
        }
    }
    private void SmpDechiffre(ref Byte[] Key, ref Byte[] Data)
    {
        for (int i = 0; i < Data.Length; i++)
        {
            // Xor avec la clef
            Byte v1 = (Byte)(Data[i] ^ Key[i & 0x0f]), v2 = 0;
            // Inversion des bits
            for (int j = 0; j < 8; j++)
            {
                v2 <<= 1;
                v2 |= (Byte)(v1 & 1);
                v1 >>= 1;
            }
            Data[i] = v2;
        }
    }
    private void SmpChiffre(ref Byte[] Key, ref Byte[] Data)
    {
        for (int i = 0; i < Data.Length; i++)
        {
            Byte v1 = Data[i], v2 = 0;
            // Inversion des bits
            for (int j = 0; j < 8; j++)
            {
                v2 <<= 1;
                v2 |= (Byte)(v1 & 1);
                v1 >>= 1;
            }
            // Xor avec la clef
            Data[i] = (Byte)(v2 ^ Key[i & 0x0f]);
        }
    }
    private void btnSmpKey_Click(object sender, EventArgs e)
    {
        string sOut;
        FileStream InFs = new FileStream("data", FileMode.Open, FileAccess.Read);
        FileStream OutFs = new FileStream(sOut = "data.dechiffre", FileMode.Create, FileAccess.Write);

        // Lecture du fichier
        int Max = (int)InFs.Length;
        Byte[] Data = new Byte[Max];

        for (int Base64RecordLength = 16; Base64RecordLength < 1025; Base64RecordLength++)
        {
            // Lecture du fichier
            InFs.Seek(0, SeekOrigin.Begin);
            int Count = InFs.Read(Data, 0, Max);
```

```
            // Composition d'une clef à la recherche des séparateurs
            Byte[] WhatWeExpect = new Byte[16] { 0x0a,0x0a,0x0a,0x0a,0x0a,0x0a,0x0a,0x0a,
                                                 0x0a,0x0a,0x0a,0x0a,0x0a0x0a0x0a,0x0a,0x0a,0x0a };
            Byte[] Tmp = new Byte[16]; for (int i = 0; i < 16; i++) Tmp[i] = Data[(i + 1) * Base64RecordLength - 1];
            // Encodage de ce qu'on recherche
            SmpChiffre (ref Tmp, ref WhatWeExpect);

            // On s'en sert de nouvelle clef
            SmpDechiffre (ref WhatWeExpect, ref Data);

            // Base64 trouvé ?
            Boolean bFound = true;
            for (int i = 0; i < 16; i++)
            {
                if ((Data[i] >= 'A' && Data[i] <= 'Z') ||
                    (Data[i] >= 'a' && Data[i] <= 'z') ||
                    (Data[i] >= '0' && Data[i] <= '9') ||
                    (Data[i] == '+') || (Data[i] == '/'))
                    continue;
                bFound = false; break;
            }
            if (bFound)
            {
                // Clef trouvée !
                textResult.Text += "Trouvé ! " + Dump(WhatWeExpect, 16, true) + " !\r\n";
                OutFs.Write(Data, 0, Count);
                OutFs.Close();
                InFs.Close();

                // On vérifie le MD5
                string sExpectedMD5 = "6c0708b3cf6e32cbae4236bdea062979", sComputedMD5 = "";
                MD5 md5 = MD5.Create();
                InFs = new FileStream(sOut, FileMode.Open, FileAccess.Read);
                sComputedMD5 = BitConverter.ToString(md5.ComputeHash(InFs)).Replace("-", "").ToLower();
                InFs.Close();
                if (sComputedMD5 == sExpectedMD5) textResult.Text += "MD5 Ok !\r\n";

                break;
            }
        }
    }

    //
    // Etape Postscript
    //

    private void btnPS1_Click(object sender, EventArgs e)
    {
        FileStream InFs = new FileStream("a2", FileMode.Open, FileAccess.Read);
        Byte[] Key = new Byte[2] { 0x00, 0x00 };

        // Lecture du fichier avec clef 0000
        Byte[] Data = new Byte[(int)InFs.Length];

        // Brute Force
        while (true)
        {
            // Affiche un message de progression
            labelInfos.Text = Key[0].ToString("X2") + Key[1].ToString("X2");
            labelInfos.Update();

            // On incrémente la clef
            Key[1]++; if (Key[1] == 0) { Key[0]++; if (Key[0] == 0) break; }

            // Lecture fichier
            InFs.Seek(0, SeekOrigin.Begin);
            int Count = InFs.Read(Data, 0, Data.Length);

            // Décodage avec la clef à tester
            for (int i = 0; i < Data.Length; i += 2)
            {
                Data[i] ^= Key[0];
                Data[i + 1] ^= Key[1];
            }

            // Clef trouvée ?
            string t = Encoding.ASCII.GetString(Data);
            if (t.IndexOf("roll") >= 0)
            {
                textResult.Text+="Clef "+Key[0].ToString("X2")+Key[1].ToString("X2")+" :"+Dump(Data, 32, true)+"\r\n";
                textResult.Update();
            }
        }
        InFs.Close();
    }

    private void btnPS2_Click(object sender, EventArgs e)
    {
        FileStream InFs = new FileStream("a0", FileMode.Open, FileAccess.Read);
        Byte[] Key = new Byte[2] { 0x00, 0x00 };

        // Lecture du fichier avec clef 0000
        Byte[] Data = new Byte[(int)InFs.Length];

        // Brute Force
        while (true)
        {
            // Affiche un message de progression
            labelInfos.Text = Key[0].ToString("X2") + Key[1].ToString("X2");
            labelInfos.Update();

            // On incrémente la clef
            Key[1]++; if (Key[1] == 0) { Key[0]++; if (Key[0] == 0) break; }
```

```
                // Lecture fichier
                InFs.Seek(0, SeekOrigin.Begin);
                int Count = InFs.Read(Data, 0, Data.Length);

                // Décodage avec la clef à tester
                for (int i = 0; i < Data.Length; i += 2)
                {
                    Data[i] ^= Key[0];
                    Data[i + 1] ^= Key[1];
                }

                // Clef trouvée ?
                string t = Encoding.ASCII.GetString(Data);
                if (t.IndexOf("roll") >= 0)
                {
                    textResult.Text+="Clef "+Key[0].ToString("X2")+Key[1].ToString("X2")+" :"+Dump(Data, 32, true)+"\r\n";
                    textResult.Update();
                }
            }
            InFs.Close();
        }
    }
}
```

8.1.2 **Fichier SSTIC2013.designer.cs**

```
namespace WindowsFormsApplicationForSSTIC
{
    partial class SSTIC2013
    {
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Code généré par le Concepteur Windows Form
        private void InitializeComponent()
        {
            this.BtnTestAes = new System.Windows.Forms.Button();
            this.textResult = new System.Windows.Forms.TextBox();
            this.labelInfos = new System.Windows.Forms.Label();
            this.btnDecryptAes = new System.Windows.Forms.Button();
            this.btnDecompile = new System.Windows.Forms.Button();
            this.btnSmpKey = new System.Windows.Forms.Button();
            this.btnPS1 = new System.Windows.Forms.Button();
            this.btnPS2 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // BtnTestAes
            //
            this.BtnTestAes.Location = new System.Drawing.Point(26, 12);
            this.BtnTestAes.Name = "BtnTestAes";
            this.BtnTestAes.Size = new System.Drawing.Size(75, 23);
            this.BtnTestAes.TabIndex = 0;
            this.BtnTestAes.Text = "Test AES";
            this.BtnTestAes.UseVisualStyleBackColor = true;
            this.BtnTestAes.Click += new System.EventHandler(this.BtnTestAes_Click);
            //
            // textResult
            //
            this.textResult.Anchor = ((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
                                      System.Windows.Forms.AnchorStyles.Bottom)
            | System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.textResult.Font = new System.Drawing.Font("Courier New", 8.25F, System.Drawing.FontStyle.Regular,
                                      System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.textResult.Location = new System.Drawing.Point(26, 70);
            this.textResult.Multiline = true;
            this.textResult.Name = "textResult";
            this.textResult.ScrollBars = System.Windows.Forms.ScrollBars.Both;
            this.textResult.Size = new System.Drawing.Size(1436, 425);
            this.textResult.TabIndex = 1;
            //
            // labelInfos
            //
            this.labelInfos.AutoSize = true;
            this.labelInfos.Location = new System.Drawing.Point(23, 46);
            this.labelInfos.Name = "labelInfos";
            this.labelInfos.Size = new System.Drawing.Size(25, 13);
            this.labelInfos.TabIndex = 5;
            this.labelInfos.Text = "Info";
            //
            // btnDechiffreAes
            //
            this.btnDechiffreAes.Location = new System.Drawing.Point(107, 12);
            this.btnDechiffreAes.Name = "btnDechiffreAes";
            this.btnDechiffreAes.Size = new System.Drawing.Size(75, 23);
            this.btnDechiffreAes.TabIndex = 6;
            this.btnDechiffreAes.Text = "Déchiffre Aes";
            this.btnDechiffreAes.UseVisualStyleBackColor = true;
            this.btnDechiffreAes.Click += new System.EventHandler(this.btnDechiffreAes_Click);
            //
            // btnDecompile
            //
            this.btnDecompile.Location = new System.Drawing.Point(225, 13);
            this.btnDecompile.Name = "btnDecompile";
            this.btnDecompile.Size = new System.Drawing.Size(116, 23);
            this.btnDecompile.TabIndex = 8;
            this.btnDecompile.Text = "Désassemble SMP";
            this.btnDecompile.UseVisualStyleBackColor = true;
            this.btnDecompile.Click += new System.EventHandler(this.btnDesassemble_Click);
            //
            // btnSmpKey
            //
            this.btnSmpKey.Location = new System.Drawing.Point(347, 13);
            this.btnSmpKey.Name = "btnSmpKey";
            this.btnSmpKey.Size = new System.Drawing.Size(101, 23);
            this.btnSmpKey.TabIndex = 13;
            this.btnSmpKey.Text = "Clef SMP";
            this.btnSmpKey.UseVisualStyleBackColor = true;
            this.btnSmpKey.Click += new System.EventHandler(this.btnSmpKey_Click);
            //
            // btnPS1
            //
            this.btnPS1.Location = new System.Drawing.Point(478, 13);
            this.btnPS1.Name = "btnPS1";
            this.btnPS1.Size = new System.Drawing.Size(96, 23);
            this.btnPS1.TabIndex = 15;
            this.btnPS1.Text = "PS BruteForce1";
            this.btnPS1.UseVisualStyleBackColor = true;
            this.btnPS1.Click += new System.EventHandler(this.btnPS1_Click);
```

```
            //
            // btnPS2
            //
            this.btnPS2.Location = new System.Drawing.Point(580, 13);
            this.btnPS2.Name = "btnPS2";
            this.btnPS2.Size = new System.Drawing.Size(94, 23);
            this.btnPS2.TabIndex = 16;
            this.btnPS2.Text = "PS BruteForce2";
            this.btnPS2.UseVisualStyleBackColor = true;
            this.btnPS2.Click += new System.EventHandler(this.btnPS2_Click);
            //
            // SSTIC2013
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(1474, 507);
            this.Controls.Add(this.btnPS2);
            this.Controls.Add(this.btnPS1);
            this.Controls.Add(this.btnSmpKey);
            this.Controls.Add(this.btnDecompile);
            this.Controls.Add(this.btnDechiffreAes);
            this.Controls.Add(this.labelInfos);
            this.Controls.Add(this.textResult);
            this.Controls.Add(this.BtnTestAes);
            this.Name = "SSTIC2013";
            this.Text = "Challenge SSTIC 2013";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Button BtnTestAes;
        private System.Windows.Forms.TextBox textResult;
        private System.Windows.Forms.Label labelInfos;
        private System.Windows.Forms.Button btnDecryptAes;
        private System.Windows.Forms.Button btnDecompile;
        private System.Windows.Forms.Button btnSmpKey;
        private System.Windows.Forms.Button btnPS1;
        private System.Windows.Forms.Button btnPS2;
    }
}
```

8.2 Listing du programme Postscript

Fichier p2.ps (avec brute force) :


```
/I1
currentfile 0 (cafebabe)
/SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode filter
cf760bc77db1f282e881ede9a10122b220887466b973b854218b85c230d6733ab459fda9a879973664130312d5ff3e1a8e2f25dc6d3ada1daf6e481438a3a7fe8e36a77e6be0ab3192b6a18
3e02f84721563b9586d2977335dcdba80530994c097cd054baf0d690ab58e6576c10a6e4d6fd20005621a6ec492bd0d0b3f2fe3709c9e51c9d69464819f111812a14ff0a5ce8c7cf0953d4a
ca103da4cbe544962226c914dd0d505658963213a757ce8a6063755de62eecd85ce1358c3031d77bee049e6087b671d71eb6239f40be8ceafdbd63a61ba4a87c3770887f72db4711eba54a4
f257009729db6cb7369b312cf0781cad85ee7d022c8bc3436cf2f9b2d1b422fd2c07e137af5017b0ae74893e0df3a5cb665cd86715d53215308a34387f28aa4613474d9f8cdc668d3fc1922
f3388aee42eb0769a3d4a61dc6ee9314f6bd3c936c38d2bb91bda6032504f3cf46bb7805ebb1a71393601226b2c67705f2d54f281f80c5c60dd0e29bc4e044d5e3261920a29d37ae14b8de
9103b28612a93f5ff780be4ed4d5bf2c26e84ad0840870ddb3682b45cfa0126ef3285e343e36d1af40a41917be20dbd2b38fd5ab35913bd06a2e6ec72e61c420d587ebbbd42df622ab863ea
45455818bb35ddc3a152b8a8050f6d5885eddd4447d87f60448d86d451369209138a9705ccd1f095cd1893fb23282dd48f9dc3fc4867d3a1832e45e084e54f76ddc3226052af767382df13c
0e111805eeb14c6f1a598aff52ac3b1fad9f8bfebd06577e917f5c9096e19222e940129e657b5853e29c1642180a6f937b71a5e5573a387c31899f439d68bd4d26225f2e068ef23660acefa
e52b9137fee23991daf5758005e45fafae30ec9130adce7be170336cfc1b14bae0a3c28e527b4ef37ad06168ce7efc68d404bc8e27463596e167af4ce4ebcc6a3d6bab1f9b32069fdb5eee7
3c8f06ebc10ce5120477f3cb04acc7598b0041d59f81792d040d08a0a196c630d1db8f769c144890839394f6456fbb08940c18f7199b23a275bda7592687572448edb0b9c8db2013374849c
f5b7381fed20a29b3b00fde7e9847bec41e4f9907482c3198872f077f226ea376fd4e2dd1b1d95a84395d1ddd3cc5d9d2a264c0924803b0299368ed59b2ea92aa515c5b50dd73f950aa1b7d
1f637629d721a8d136a5124fc2ba261e252f19de262b292db44705eb66ce43cdbb6bbb0de1b6264e01b58b0faf4c64db7de80d94e57c0c80c61f8df6f1942e560f0cb7100cd07fb42f10855
cb3b042f9bb7d0a3ff997ddf70d019d1613e31d697dd65862a144f34a6e413489cdaffc69510dfea38b947af17e15db638c16c2b6ce27716027b8cc3e989404c9005861a45dc85b4597ad26
3c899830180cab2ec6a43974a56f39e2a705697703851e95fd5c610a588940a901914e5499f7c3fa2969ce7ed98eb7c13741162f322ffa9ae91e21109508f32987e442ffc929a4399b179e9
23662d365ca26122b1efbe2a5c9b6b5694a5a0af2a859be91aec65abea4f60a6d058f19ef64069152ce31bde73d55bdd4f8d649238c323845e0e3f662b2780ab14304db9e64880830514b72
6bb788849781d6f151c53c89804f893b2892382f40f1947483d6be6c161de401dc2549de7a4f2b5a6a87f4b787ec1e31c635dc6890cb24b9b8aa3e997375174e686b6e518b23c6850427691
242cfad117a60d5eed0801db9de81e6c9b9b4fa02d4ba9a827443c4f77ca715ccd91fc838eeab2324559cc07b12527e4a61cfadc482c6e235c169619b2e98e6e199c7f1ab81d7efc9b91bef
810c80892b03e0c42e48ac8e1a60d8cbca21754a8f92c3a3f599210115284f00e222a8fb4e4c10f5cf40f7c26b2337e8428161673225d659adf671ea0e1db3027cf95ecd29559c2850782a9
2891fde271b06c38c1839ce4ffd70923d426b491765e150500b3a8fabfad44b3998c3527d608438cc0c13ac43327c35059cb4e5d042e88ec8a488c2bdae05d84c9e2eb80dea865d05101a93
1f72acf49e7a7b9dc7cf3efe6c262b947940fbde21416db0402bb86a3f542e2149322f6175f9b54d329ab051b933c6bf753dea1ccad06dc8fa1f01bf8e7253aca0baf5741ff107f495bfe9e
0e46affef5f97f349c447e7ab2c8ba4d22133c99898095f0c75fba756d92aa619b91925463a8b308b6281879c71bde1c24455adaa576a891a2a87b28a6dd99699afeaadeb5e9e8d801bfda8
17042fbf85ec1a058296c9222967c24d41778f0e2f4d0fbd359e8f799ef75c95765eaf2765d5664b2ea2e8196f1755bc8485237cdd88b81b5f6bff547f352753617618d92a5858cec02f61c
34b7de2e7cec018fe86dcd83e6c6e8be1071bc493d98d6edcc9c30feeabbf88ae0e418a0efef015c57a21a0c78df0f89b64103f19fe1118c7dfcefc4775f2c0acaf8a67ccb29a9c8058234e
5fdfcb532840a51e464497cdb94c38d24331b0cea8d1dd0d4e8ca141db21cd74114d1200e27909da6c17ae81bcd127be2d78b64f15af6c7d6fdff2a97218368fae0b988c288419ba853dec5
b020de9efdd0d60bd1ce8061ab2e4531e1df7259a183bde817804600989f875bece9bdf321624adb63dc787e6c8137e99bcda29699810936bd893c8f2ecdcfbe31bf76c6deaf4dd8e8d24c3
d9dcb0ed8965f238c715924a0843db4cfbbaedcb7527368b4e90749a5afc4c9d63dba9fd55f49a3cb4c6a4883cdc9d7fcb3395740ddc80f03a4dc54313d5538df08cfbfa525323fb484cd77
d6d441fc857bc23078d137f11a93b4e806da2af6aaf3c2ecde387fee39a03d028185f56ca69fca9b1e23bc66e2c55ee888c82f84b291c79c6cc9b05f2dabc2fd0de32ba8d445264c62933e7
74bc6970faab310cdbe6d1571f0e2775854a1675763c567a9f17d17149af64b7dd705da1ce7e04a20a7a9bab3b3998804d1bef16c029fd2f580336870f28526dc643fbdeaaa2a28bbea7ca3
29c691732943b2307d697a2717ca2ad4373549480dda3d377b21382c9d7553a96608ef1d5efd8d05c777c54f28abdecad7a8eefe67754fe1600c746e3617f235eefe6f26e849cd718e6f04f
4fe5bc9e609e7168d9dbfae16482ad6ae63faaf4e5040a81f1e45717c3684e9b677738e149eb72ff58c445264389e466d997a808a9f19cd506fa1f60a7ffa66070f8041bfe689fdc69b313d
55b5e2898175628e73c79beeebc484f77dd18230caa89d6c43691127bc98b6c832d462d8d4e8725074b771bc98e8584a8296e836cb097a226d1e5554f0d6759625f7ec6a707e8d3712f9e1d
c013cbacafece08108deaeae4b772a47b7ff6b672957efaf8f5f1c72a610c7c3deb4c13b2e6962bcd579f7a8f1c964ecf16573eeec1125472621a46e1f540628874ea5548d44c38165fd32f
98cdce18e43f1413d6c63e109bba1061e5d39ad820d1b6d539467d0e228742de9aa339ba54a303e482747da9c92bde45567d43873fd79ed624ebd4ef1fa80bc1f88937d58f1bc2f70498bd1
8abdcc533780076d35699295344da829342a90f6816a3ae8a5b74dc2b598afc3803b387220277731401236c40c16e0327a42bb08c09ee7cb60c9f8ce11f939ea99f51f01adf124798a885ab
5015aa1c026abe47c16c25e6e352f176961e768470a0116aa36cf1d3eddfdbc87015759b9852fc935ddc9f36ed7f2a998a6c558e8cd4c43b0f517dac08ae3a72811450b57126acd3dd6609d
646928eee71aac89ab020db293e217d6427399f5a0b380d4425e11a6731805df5a4c81a5f39d64bc0e21f1681a58d2e67708fda9cb7639958317d2169d12428105d48b18950a8b291ec4bf9
74a7a2a06b9b397357d01b959740a6a5dd3805f2d87a6c243d0bda52dcdb1089b80a8c9c303c75a59aa752ff17b27577c5a275664c12c4ca90f898a0a9460e8dab4b73f10b240318e20a0e2
a051ff076d5c83f2274875bfa1b936340b7a61fe76386ef75a97f498513579e47330d31b615b6b57c9fd595a52ad130b86da8a961536dbdb15320d56193ac514389ffb3f16b035a3a410cfd
4cf84a654045ad3b4826812760a2df1f41fcfe0755faa0f1c503eef176e48a409393c2d86b3f75f42b067a28ff6f3214dc062af78a9c7438c5ef3699fad521d9522e7f08285991a520b5ab7
bf38070ad08c8a2ed0dfd8d5b540940d835d0c47258608babea264829cc916f5d2dbf0cc81e2c22400f1fe67ffa8404923de450724c02f33a5366b64a0e09ad5e4652b80661a8c1ee59c6b3
259a70ff8ca6c22ec0bb18ec090cc75414340b5878919fc935a1581f6fa1d31aa720cb4185fa098e35bf096609122eb2f91f2ca139775243a16ba937fdc482e06437fadbdb000a87ba6a630
f6eadd5f3b2cb4616be497ad9b50673d271539d995c21b41b556b1f238ebf41a40df403632bd289e459d64c84438121c2b80bb87ae521aa0adf5ea24832825e814aadde6b4d34cbff9e58a4
b331e85466db0338bb521c3a4f65d891edfc4cc9c86f68537d77b22dd71e77a7f02938328585c5a2e039b37d5d5e3c2f06357a67754854d7316efca09564476ed33ea8196ca493fce197af1
abb0f4719d2e503aebed308bdd7034d7c4c6028034cabd7d519aae15bbd12e59509137f99a299e67745e606c7431d0d84de97a1adeb90340fcedfaa07058aed618beab2693925a3bcd17910
c0db2eb6c2f4c479cdf058a5d1385a14315294d1d99b9569f9b061110412358a513fa9ca8baa8ee0685db2fa04a4ba2b388d667a1ebc28a77158b6939ef2784796318eef806a027fbda7521
de99e42c8a91049a71241068ad349aa45e8372104ffbe165009576faab37f5c244747bb3a9e1766d32612c8b65c393de7ec1cb14cc3927aa41fe5f11ab39405991aafc2fa99c6c3701ee96a
b19f79b98d8c18d6ece042e7e5e65ef4b09858d68bf2f42ab32e91413b02ee7a7fc08d2e927fa55b63eb28caf4ce2ed52f0f0a39a989c60d7de5e98fda42865f4335e8ddfe7cbbbb48cdd28
7e670953230249d5fb460041e251208540c3175844b0d36ecf4112a9bd88ed6a9313f184e2b2629af9d478434f2b1fac47e93906c9c0ac832352e38e4dff1ec5e5cfbaab538a3ff4a43fb5d
812ccfffc09f9fa841cca5b9c91e6cb8b9bc9886194bc4c93ec679cc8abefd6e9de72483e0ed2f8f3105d270b6923ec3bd910859afa56d87c2d5b4556fd9621682aaf2aeea206ae4a814406
17b5a023ea424b6549c08337edf64d3ab6f7ee7169d83df7feeeaee00c69ab0804ca79c928014c2946c10a0dafb6b3e3d4494f064015fb306f5532730133e596e3189d5bbdef11d29eb2f82
68558658497957a47560e247c0a81717653918094382750ba79c6e2bae666daa648ba8cf191a9d431c7f129fbb6fa231bcdb671d08b137107339cc6ebcad15fe6f30f09348eb9ccc3970302
e8e1f2701a3cb4cf9e5246c48dc509d7020ce462a61580ac2ff82b3a62ff6e5d64ddc4f85cb2c7c79d41939043c66e2864c3e733b718bd83b187aa8cfe490f6c8234f9977a1a6edf3056bb9
646591f1ce75337069f827b1dbe69d842306210e8d9a9edcb971d88c31672485a4045e3f357947129d83d59cbbefe9049dd9ce45c1f9f42bb6ba49afb207f6a66078a267c498b3da273640c
3ad0cf112cea4729edfb0e2554c91387a44fd5f8a7bec6a0faeb9bf795c0ba7bbc749572edcb17192889f481b2b8f0c7ee2df42eca20b487de6413e35c618d3290a95bd147654300d14d790
2df801e1156963c3ad71429f122256ecb788ec3b345fed6498bdc3d73e46fb89f5db7ac0c774a0c1c293c81ebff545f15ed50a9df5fe63a012f80c741008c44911977fd73ff5a2d61c0a99a
75206fb1ac52f6a7f5e946d6dda4b642dfbb3e8894209e6877fbaef84d500bffad9a294615aba3b86e9d3b0efd09940e85b9712339bf043cd3bdf609e68a9eb157b1088eacc8adee3f179c1
fd66309cf5b98d6710c812793fd526c6ae8ab9a87b31bf8e8c9815c280feb0f4fc864f41ee44345ee8b8f081d88d4bd14123e11c0335962be1265e4c5f8a5e215e73172534f78219ed3b67a
205b78f21444866081e9b1c29f1fd4bd496967f239d622ba4e24e1259e52325f95d1f45df36b5193cd519d9d2aa4beae0cc2655ac56d0be0106351513862eeb043de82eedc8ceec5e400b0d
c729bc916123446c9d94fe8a4e9fa43af7104e5e1919f08dbc349fed7c1a041324fe2e0f2fc4ba0009532dd9efe5894a7f643181d3afeedff5073cc21d7726c2eda750348debfae879855b3
9eea610a26fe7f34b0140fd69ace2ee6a9898c28d3a36d8ffd076a29be92f180ab1f81d75c40470f80ff2db31d26d16fb50aefbebd7a74dbc0afe60fe9c2258804324007c6deee863cb843f
49e871a7823c581a56246497fe6df06f467d6b2bac465439d5fabb8c18c3e900e12d97f103c34d60e7ca7b679e3dcd70287bb16d884bb351f92f081f9199e6620bb7a63ad985372c0940af8
7e70301387de96c8c7e2bc6292c248fbec5f736d92f8c31fd92aac224d0bba00a12bfad0c9813d1b76482641aa2f13d7668b36d13f60979c3ffe0c9303b45fbd1b850f2e3da454c796b5dc1
b387ccd9b43e868d3800da44f00d0557106f8d8c5ddf0af8b1df122f96ee26b9e860c3c993a84ae477fa3e131c272eb42d19acef3c5cbc28b5e26109a06be420422836c17d69b0e64f5722b
cfe9d89d5fb100c8318b46f861834e8bb51ce51261a32439dde684b8ff305e5b5f7e1d152d4311e26e9a59b44c7f009b1941851c9ba3ab8424020e520959ed5ec1eea67d4bb510c916a5922
31cf99f16111b95870fd616522d609fa67053762591840fded0480a7cffc819224aa7eff4a5b3d8eadb28732e63a4c76b9d53296294707d49c3f9edcc275a18caf6fe0eac7d92733901f680
0cd41bdda04a87cb03caf5d72bd762ff66c331d158ce25b8b25ddbdd1e9aceef8d18576ff215cc32f0a951b479de8223140582708991fba7afc7d3e21637ad133a028048537b2977a47c38a
60c6e96176262eafc044b5a97543de0122ac679e2e2e03794bebe2006c660190e93466b62f5d53f57dd5d5528193d5e57fff8de11b265da9a45d432b03af9d6aeca3585d0d073b9ad68f791
a8a51f2258decb4e125bacfdece6e4195ca5be9cc434838191009ba6342b06b19120215d7e6609720e5fd9c355ffd98ea552f6f8336fb8f494d7d79a935f2cc871b4dacc8a3173e865cc33a
5ae3c1235775f25ffe2a03f17a81eca9c824e6261e99a7cb856911217c620a35ee042678f5f512177e7673ebd3630ddd380d891912d892f77d59e132b3cd42f200f8ad664ef5d2386f58b63
89f8a90df7de6149195993e7b26eee2f42b8ec8aa6f0dc8cc7daad72c0d5a323ffb2195e81f549187b9f13775485250aa26589cf9dbca10a4b5525fafd3446c0478b751c65bf187f263c2f6
5f0c84f3d73966568ba248ecdaec2c8f94e4a214cafb46adad6f6f0c3501463217dd833fab5589493e80526d521bd71e6ac5449032c56330457e8cde470e49d08348e5484b05ef45378f2c7
1100bd8a45baea94e5fb2110e975544e57bd086662903d13fb8361634fc885beda70396b0f2ae99cd72e44db87eff89938c41f5d31786cb0910b5eb108f87e029b1fd3f2eddff44069cb65a
f631dc8efd882f726e85a3a465256d68216b1fcd8880f963cc51c5a7ee010c2bdaa2240abaf6adf35f6e351dd9620657c31f17340e92864707efeaccfdf5be205e016157ec32fab122934bc
36d56dec77a22b9b5be4b8a80acb74708c9aa2467e7a1173696942d877566a88972026f9fb9766a195b4c9b703fa1d53e196cf8aec9402a1f35d1c9a4a8d8a7d18a16227dbd19bb2acde335
78f432939c14f3b3ea9e10f9be1ef35ebdcc961bb73aba30b05e7bdf3e2100e063b7ec16cbe34e2b57dceeac7f2299da8398a860c3b6ddd38ebe43ead208ecc30e30176355333419463 2b90
5130decbfbe2481e16e3691483d8999dd295a3b96d23fd4ce97238570bbf1899355a2926f09ce159302019f9300d47fc6b545eb64db673ae61a6f5917da55e4edff628d5595f4827ab183ba
f00c38abfe508b7f0679c5e1227b179cc94f1c71c9bec11aad0533d1465c563996c7db2022e66baafffa4007b89cfc62523d2f6be3398aceffe4570830eabf91e4677718387fecc3f290b38
7bd13ee8fc3bc4f000a5a0cf7e96dee14fa08af051dc2c4506a8a7d14bde5cb78e20518461d5abc91a848b5942f5f057cb9447514f4dbb0b562f2d1f283a62d0ac968c9e08f4e01912bb52c
10ed1cc38c93cc835a63a5152f08ee8efa4b72592433662faf5386c699d33a43329a7975deadf48efc1d840dc9709811bedd46666709374350dc21d0391d7ddbb1dfbf3cd17528b8b613974
d8c969f127a0ef687f8665290c4ba8785fddfd8ce510b797430e70dcd385d8eab03c893cbfedf4e6992f4d3493fb8f0c9f247e7b114e979af1f866a47bed199e2231e05f3cfe2c5d3410f3d
313a0c85c7b3ddca38e29d0f9a9ce3fc9819175e76baf10bdac131d4a36ecadfdd8b63add542a21149721b141cfff3c649bd3eb342ff795e5df075808bf66181d9109be15eaf784538e4fba
98c0b455eac07a2e8dad57eefed86ba1255f5ac4f7562ebfe2e98b3c15fcb94fa24a337718af0b478d3445aa2e24842fdfe3d2d9ffced0df6ce5da358f1744df29fe202a9b3f4acd7266043
ee3460324a176efa2ac3d0d7f76879b105e5a0b77c068d431792cb238fc437d16e2a84452fcab4a5cf606ad0c4fcd76b65e38d01a409afe59e1117a1110ee01e7d3dae7de6e0d49d0c920ac
cfe40d061e61a912da1eaeb457ffcd6cf3152addd29163c82a2a72baea0956430e07e055ad6c97e8d82af183358f58fde77e3e33ef0b119482106e4563924b7d9283c9d9911eb9b5d9dce86
d3afe0869077ea96d6c60fd11d37a3f7da1483f80afcbabef636e0820a09371c7556a16eb525922bafcfa95720d6866620ff20b2975b6ee19d782394cca7335b65e714ce30833cea5eb9f21
2ad2b4c94905335ebda60dfe6e611d2decce9b4f7f034e760ea8d987b708f000f39bf089f44218fc8b2a7ef6575beba14e62d2c8663a9cb932c62ad3df02cab55d3b2c8b4bf3507771a2e4d
229a37e4fed4d78f4e23eeef2497da664c7665469f1b682c933e0d322a830bcbf239ca58a2fa41715e92c3c7083e8b56d23c2cdf3d96d126dff16538b2720cee4b2e5177c6e8c49808edfae
722dfc2ea0a7285e7e7dee4fa342e919dd05204ceeed920bfee6b062f4990d614d84929416480c9d7749760bfd3939796ff68beab815c1d4e7b5676b31b195ace2ed5aacb8050a52b7403e3
b31a2e6b94144cde7d9f32faff47f043922b46cd49505e010dd1489161b2195ded57f9b2d75e30a6003ff65e15d5286b9dc1c3f60ca0c8417b8892dd73af19cb1efa2313272a4257a1e2542
0310963661c01374788466916ee2bdfb417633d3fa84665513e19351d15a116f2027c5044e1a5129031d9aa6030803ec482c3c1702b378572c92fde1cce2cf0b3f6b18f14c8b7d91f628b30
191047f9f75f1e2dd90bd16ab0590fd0ab82267b30f2d81ae8449ed727cf288ed27e51304abc75a569422f6bfb019318e319a05d7839e5f3ec313d7e356330a4c0b90f287a57500a3417508
1d0361adb6801e2bacc98f74e30bf465b5f05f24e442b10cd175b785437810f8f7e6d437f2d7a1bdfbda91d4562a5679452d412bcddb27e7851e7e681e0a718a60e31aa5933ce4cdb48124b
6a98cd71c562cd727e2accafb254740c2c4e91e80f06bc107798cb09b1f9403c22489e82bb4b46bd73ac2d6c0456a0f2a8db57f500ead3ef0301aba8141764ff4f5cfe3d0e4d3ba7bb9d640
c3cc343752794c79cbd388bc51d6c53d01a68455f202831b887711f36ed2936119f8a873bcce6576c59e9e901523432b5aee3761e85b59725b07cbe90a4213a6c634f08d06bf3e5a2f5e0d0
ce75f8266af9836549c4bcaf08dad516564b1614b5a01afef394819f3a2fdb16956a1ee2b20813cdbeb1f91a008d3251efcdfa78f2ac15305a16cbfd68bed6bcd3a79b76f96870ef613da69
68b8d04bf93a8fb243b0da28819e9a2580064d0438dab2c73e35532990223d6c5aed7a3c1657f878981069e1a8a8636028d4c59565deac564713a6902bf7044f26c861da74fa94ed19853af
391440cc7622685aa98f32892e2f307ce841f7bc4b9dc6e7e5c794898957397c5aafcb5baf97b58ece75bac9a9e407e68efc908c467a852200316ceb6431e4a0cdcd3f02d74fc6edc8930a9
9fdd0fb5b2a1312f47abfafe8a5a46afc485417502d1da903f301e6497aa92fc7cf03e2fea078583c5d9d5bd01db78d5c60855f1f9dbe8eec2a3bf05a19e1133e82930fb16388c07155777a
b5b22eb240891b24e4e5691aacf6b195ac53a24f6fda565ca91b86b6261d1d8a5951de311d619954f5a81aad2646f4ce421f395c15a14424c1aa014496377a57be14cb23796b89772ad9cbb
246555c11282341f18bc3ebf1de5eb87985242ee77afc40254f1eca0d940d4ea68c1b7c9186057abdf8c530e4373c2deded9ca8a9700115773cb75df63f92562e456c15003cc6135e798d02
2544afa2e3457b4d5d7cb22496b31d6b57da7abda0c0d493be7a0a41e54f56bb4e35082b9f628878c1c6efa1dd9383e2ddf3c98f1663a515f8ced717f72101382ebf68d89feb28964998dc6
22d74d7c83fe882ad99143538f864d694ec250948d9e64eeddf10703d7440c8db989f0c70a70a03f1d79821d834cc2aaf138f21b52fe6b75ac643f27205eea9d5707e2f282431f8920d49c6
d9ec4c9c89fe016b93c5635d662e59a3a7b339be93df0432f7dd0ef4aaed64b4da9b68305f6d30e91bc1764d6d18243908dec1cf5393fe670425f52d9673b1765acaba2e80c026d6491f409
d555e054b0326f420a5a382250f6ae8c07b34054afe922502469c7a992cea70a1bf451f5ccb00056a79ba02aafea02e41c4cba78fbd113902d23f02e015fdef8cd7a61fed8582ac4847be2c
933162508e78dfdddf702dcd52122dd836e1a503e22091a9b95f14af4804c300ff8c408cd1b043e3651b9f33489503dbab0c18e3b6cba35431308a19fbcc45723cfd6d8d908870413814fca
496d665b42ea121ae68d31a5efdee5a8945053ddba81ba40e03114628823a21d73b5ca3d6049a451ddd74257b66a482f1ba0080913cabb9744b9824c775b6e5b2a7ac1690e461863edaae8f
f417cf307f13018a80a1675d6ad18ff3f209facdc53ceb9e3792556be99ecae0bbae3cc3961858c8a24ba87508ce4ad8f92a5bf45f8278fe4af1ba4100c11e55c64d95220c1f6bf857d9e1b
f6ee24c368725aad49ea0c08e30bae076fea2b005c9c156beb04f1cda304ff137a5619cab8a0e6db389ffc4d6b7f7909069956492e6631ad5746f385a60f6de4c432e64e11b90f5eafdbf60
3db7a0f3aaa1d7f85e64e6679d0d90cc0335b9b17d5e8aefc6ceb33d3d4b65688c8c19efb793528b7079e3cd7cedaa947bdf683d778962acfba0a78c71034ca60309dfc3e13d8575c836a1c
99e3ded7a83ee405b7bca8f0ea650f7b3bd1eea1ede8b0f3ae1fefccf94ec811685b807fcf1488f592d089aa06f8e4e76c1fc944c662c88bd5199e77dc4660c2957e6e738049c21e2c7af95
ba122c6deb90c6a7006116f8c9cacc1f0fefdeb782b40436fcd00daf589509076dde465b72e3b2311382781b598cbd51ac9f59c33ac731e6842dac85d50b496b7dd456d1ccbbc10c54c6b21
```

```
c52f34181875417aa69a00cf1154d88698524ab742be1bcd85f2bd8a75fd2c11c35e4ec427e3e283518cb3471407deff97aef8f63cdf7290b8a984875a4015124edb69a29559f45074dd776
82e49dbd41d46334dd1c742a2110af744f06b84247dc6e5eb2b0d392157a10fc8652f8a0986f7be7efc43ee49b906baa6bbbb37b9ed4547c52e40c4a54a68baa706c0d3f1c8bb98b9056abb
2c215ca4129c912a21976fedb11d05807e743140556b7d08b4c39fb9c64b9200f71391f0089e0643e1dedb7f086956d684d3a34202b185f8db789408d35340079605b89a2a0a86b23477d6d
f3e3e0382bae92a8c65cb75fb29d27416b3ccb9b0553527daf8dfde2cf91c079c9a98561e9ab7df319a15362032b654a60b86a4262cd1f331ff6e99b459ebf3babd86f3e67669a6510c2d7a
4579a59e33b8a70fad65f1f0ea52fd1a92a88dc5719d3ccd385bc1dfa50d881c42073d6be35ec288e0a184a7ed2bbf7985ed577fa17db00fcfdb5c515c259b4ddf755cb1a8b4c93a2ed3411
692420aa5ac985336f180a1ecf343f3381b2893ee10134a0f9dac32f003c78baf71c5ecbea32621dc151b6deb709e12a484363c9d3bb586d769c9e3dad47cc8e4d1bc0946df227db84845a5
18663239509396d678dbb58981704c119ff286c32a8136037f4e561f2b09fe0ee47cba1e267fc28a9f2d1774e47cb0e924cd8f002aaa094687a0f9cb6f45a998f7038a63b518c01483ee17a
c95f319c4d49ccc2bf3a01d5d4b6e706c3cfca37b2ced83985bbd82901cbfe54dd407c7f95ecf0e4e5fd563c2d8a68d159c7c71e394225763395 77c657c9f2e91af65d227509226fee6cb08
aa0878da60c29e510ad63ad9d0d48b92821ba70b5be8108a648cf18bc3dc4b18725aed7dbed392be88eb3d8d889de71b450ac5e8ccaf893424c929dce5515486a27d836aac2e8965a3687d4
1ca7bb709702fb711bfd1a136bf9f4615ff97b74469520aebd1195b3277d8ebee180178987a2d7be5d577fc854f657d6df8432f32b4273fe2f370d198320f7a8574585304c8d9d67fa8140b
46312649e99aad1ea730e2b116cc1950eba5612921fcd48d2e0d237eb6752082c3f1a4b11cd6c414c3523226cf16393c2089c64b6170c241b08d45b0ed83c262858788561ae97d86b35db5d
4199e615b4b81b57436ca041629ea77fd6c7ffe58599b2aebcd3198c5410c3789426f4aacccde46b1ed689205292f58eb8c9912e52e78d8202ee290878f0bcc3679a88551f50f36f1d5f212
bbee3564cadd338dbc4dcbcdb489114061b9129863c9c121dbea51d560d80c1bef2b8b1ce31b7b7a28c1a87502327435969ae95031077669b0b9ee7d6902ab76a776b0c95d60e78d6b467d9
08062a9e0e1b1dd1535331c78e4feb8a01666814494c2ab6e80d4d3adb21b79aff0618c17ba36b48a2a35655378639f922099ca8dfd375200649239757 0b5e56298502ce7b4442881a76a0
50b27d2d642a33812e1140ce756d34c856f175439f4a1a39dba22da36326c0aae9176582b2968f565680577216b9b6c9de9a4f72843120760c68cc9b7aa42ce705ec3b3e6da2d0eebf1ac98
3af9669880b9c9f9acee34760b738081572fc9ca8905da4c8e904fe8cc79fd9472a61506e638873ff5eafb91e60a4f8309291d5d49f45d5c7eee02749f88abc5188415eb1cfc68242a08ed8
a325c6f461a12eb13753cdca0810e5500b0b9b49cb2d0a7f754f69b62da6ed3f56bcdcf8a68b7a616cf512cabbf1935cf858818858266fecbf422863d3565f9c5bca95390c55807a6881a44
54808dc30491e614d9254078830d00cfd006638e6ae143164b388ff8073a5537fcc58360e383447ca9df67d58d14f8f74949ce31dbb0ddf14120d84f8c4f567750fb9fabce73ca07d5d6b1e
7d1d9e7ddd516de6052d2ea3ff07a7a785e2f60e446cea06c34bc5619ff26a3bd58b1a3830b038926920c24b9ee83fc3be7faead0a46b88a6d76e5e342fb87eeac7b11421a77524f131f9ac
b92db510606129deca41026862c520bbc1f82fece681d6955477ac15428c16d5097a9121359b85c58f
cafebabe
def

/I2
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode filter
c598d7cc5b5354440b0613490c45483d4e7b0f6c0368120f10010705010302021209140315081502020305040505130d110214091304030103030410 0b1000100916040305020406021 70b1
605160d1004060504010507170e1606100f170007030701020613081600150812060504020402071308110017 0d110202040005050012 0c1300150917010400000302011 40b1002110b1103
000401070400130e1604170f1404010202010004150e150311091105000401010603130e1407150b170505020400040310 0c1701170d17050702040002040100e1101140e100406020200030
2110e12071109160401010150401144081702160e13040204022000012 0d170110081103010303050103110d1304160a1400030706060415091102150916060005060300061 3e1601120e
120004070102050410091007120913030000020212091100130816030006060203071 30a1302140f150703000303060715 0f1204110f130705010707021 60f1705100b12060603020204061
30f1105160c150604040003020716 0e1306100d110704040005010412 0b160715091402020703040703140c1306170c1400000606040002110f1401140f100202070200050111 0b1003110d
120005010700050616 0a1201170f1405030403030401140d1300100f100603000006030 13a1300150d170607030403060410 0c1701170d1402050603010307140812061 40c16050102000
605021509130511 0b150604070101000517 0e1602160e160500020003020415 0d1702170e160500050306010316091505100c150307030506011 70b1305150a160307000607050013 0f1401
120c1400040707010606100a1301140a1305020600046 0c13061309150602040605060410081402140e150503070201030140f1706170d130700030303071 60b1403100c16000000106020
400130917061 60c120104060000302061109130510 0b120305040604070017 0e1105150b170705010503040511 0e1704170c170700060404000310081400140d1107020006010606140f1301
110f130100030601030616 0a1101170f1400000005020002120914071 00d170300000206020416 0f1100150f160305000203010017091204170e120006000200050613 0b170214 0b1302060
701040000110c1105110e130301070004040010 0e1702160f110700000506060517 0a1300170d120402020006030211 0b18150012 0e160207070070066114 0a1400150b170200060507070411 6b
1255453b004b23467b0f3d0038001f004c580b520f1a4f5d410c440f054e4b580b520f054f09001f0018004d580c520f1c4f5d410a440f024e4b580c520f054f09001f0018004d580c520f1
c4f5d580b520f054f09001900544e0b540f0c4f5b4f10004c4f5d580c520f054f2f0019453b004b37467b0f6a036c00000003110c030c1205100a121108110014 0a15110c11041008141208
1305140917120c1207100a121308110240a15130c110210088141408130314091714 0c1201100a121508110414 0a15150c110010088141608130114091716 0c1203100a121708110614 0a151
70c1102100d030c130610091213081204140915150c120414181b110614081612 0c130010091215081206140915170c1204100a171208110014081614 0c130210091217081200140915110c
1202100a171408110214081616 0c130410091211081202140915130c1200100a171608110614081712 0c130010091014081206140a13150c1107100a141708110310 0c030c1306100910 0a1206
81200140a13130c1105100a141508110114081716 0c130513181b1202140a13110c1103100a141308110314081714 0c130210091016081204140a13170c1101100a141108110100000054 14
0911130c120510091417081205140a16120c110310081516081307140911110c120710091415081203120d030c110110081514081301140911170c120110091413081201140a16160c11071
008151208130314091115 0c120310091411081203140a1614691251453b004b3246110f641111425a45530a0012074859461354104402120 4b2446770f3a0078411f440f1c441f00524d1a
00524d0846184c1a0345081603004b1146770f0f0066174f14114a144f004c164a16114b115f46184c1a03450d01174118521604 1f004b2646100f27455600545575000613115b161c495b4
5570a48451c060e004b1746720f1b0035451d00030012004b04494200020060f570f0d000700084e57001301084c52344c434474430006000020f3f0f1a0038581848451c060e001945004f
0f410a0352154c2f002f0022001d0008006b1449424440530a001207480e030000021d1143451945220 04e5861520f55185242444c530a0012074846011154 0f26452000280033451d00264
523003a0066004812114 25a46530a0012074858460755530 05c140c440f1c4e0d465d0f0d13500f1655431152415d55004e0c0d545c0a0012074846030c4347450a435b1152415d5500525c
44534650005c170b545c0a0012074846090604d4f33004d4424460f394502552d0008101e47180d4e2454094427004e27444557204845 1c060e00141012491e01111b181509520b410a4f0
f00574f43007200000030555300085506550b00555503555300075507581748451f061c00480111460f07131e000607585b151e5006501404 5d4503174845 4a430b4f42594308150d000413
0c0013030c5d501c12184c1a03451153034f0f00314f5d4d23005b4d114e100006181c00011f1644420d0b02000a0a1200154f5b4d1a00544d5300184f425943171f0b00484f1e54105b4c1
5441f001c441f004c165818016138080000010485f4200190b1600505247180106130800000010485f4200054f1b46184c1a0345115307491344420d0b020019451d00524d1a00544d0044
1f005052471077065760000000066272822001c4e06491344420d0b020019450f46184c1a03454310044c4c431a4c51105b445b5354440b0613490c4548084e180c50140f1100060503050
203115055570040746180c51150f150060503050203155254570040646180c5113081604060503050203165752054 0e0007450b41430f0e4145570048451c060e0006450b41420f0f41455701
484511852160 41f004b0a1449481c520c1f1048451c060e004b0a154c4a0153031707471c5205021d47180d4e03110e004b16464f5c1f4501535b574e41535600575044495 25d5649525d5
64d43580009001c004b061c454153091d0d48451c060e00171114491d13520d0b010017455600 1c5259471c52191207491e01111b1815095205410353f4501535d534d1d4116534d440003
55074c164e5354451d555a001c060e4501535f544e415d5500524e064947530a00120748191300000355074c164e535b451d5555001c060e45015341504e424441530a001207485b5341005
35b5100111b10000355074c164e535a451d5554001c060e4501534154 5d004f5847530a00120748 5b534100535b5100111b100040554241410535a514e535a514e534356000855065 50b0055
5515585248454e5048415257001a4417001e581048451f061c001a45124f0f00575a520d0015100d4a0048001e581b48454e435a00444f5e4c525c4c17000b0019494c530a0012074806145
45d4f00450d4513115015140000d55124f0f00074f100002006000e5444444d1941440f0941455707484 5c060e0007450b41430f0e4145570048451c060e0006450b41420f0f41455701
48451c060e0005450b41410f0c4145570248451c060e001945004f0f0057595234544491d1352324e3c003900020006003e001f006e000000575 50b00494d5042444c530a001207481
44606450d4513115018412 4f0f410a03523c4c5c006c000115115b110600184f4359430b0509005755410041 4f5d4c5208091800555041005 35b5100111b1000001a04000d4f164f0f0016
000200150114001 84e06491344420d0b022a41
cafebabe
def

/I3
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode filter
338f25667eb4ec47763dab51c3fa41cba329e18536b83159b3a690a0265ec519aae94f0e715376c4f087bcccdd0be3b4a114f8be746142c44978faa76dae62cf197d7bce4eb38dd68c8ce5f
69f326e1effceae3f72f8eaa38e019a59b1dc0997
cafebabe
def

/I4
currentfile 0 (cafebabe) /SubFileDecode filter /ASCIIHexDecode filter /ReusableStreamDecode filter
8ae98ae900000000020002000316121411455556000855400124e5245114e011d1b48454c430f4540490911111b1815095247515e1401104914000040b00010003004a0000005611352425
d4c520b191c0050554100464f5d4c520a140f48451f061c00021d17484540430a494c530a0012074808021000 1c44024f0f000317114845080c1800141a000049475042444653 0a00120748
0c1e1748451c161800494650424447530a0012074858461d4e5545114e4952584244435 0a0012074811090600 1d171148451c161800505541004 64f5d4c5217031e000000000494f134e5
500525f44425a45530a001207480c1e1748455552484244455 0a0012074806145414 4f015446491d13520c160448454b43451113100d4a131043144359430a081a48454a565d00535b5100
111b1000001a04005d575042444c530a0012071549001b0f0016000200141a000058404515090000000000b1559501e5b5d1800001119550b00581411415d000000005f410d041e5b41545d5
b560100001256184c1a034511530c46184c1a0345585348111452425d4c520e08080054440d4e5145114e08001647180d4e06100c0054440d4e5145114e08001647180d4e0a1b0c005e4f55
00414f5d4c525f4c5c00010003004a0000005611352425d4c520b191c0050554100464f5d4c520a140f48451f061c00021d17484540430a494c530a0012074808021000 1c44024f0f00031
7114845080c1800141a000049475042444653 0a001207480c1e1748451c161800494650424447530a0012074858461d4e5545114e4952584244435 0a0012074811090600 1d171148451c16
180050554100464f5d4c5217031e000000000494f134e5500525f44425a45530a0012071549001b0f0016000200464f5d4c520a140f48451f061c00021d17484555524842444553 0a0012074806145414 4f015446491d13520c160448454b4345111
3100d4a131043144359430a081a48454a565d00535b5100111b1000001a04005d575042444c530a0012071549001b0f0016000200141a000058404515090000000000b1559501e5b5d180000
1119550b00581411415d000000005f410d041e5b41545d5b561000001256184c1a0345115 0c46184c1a03455f534811352425d4c520e08080054440d4e5145114e54451f4544000b001a4
7180d4e535401445d5649050d120054440d4e5145114e54451f4544000b001a47180d4e535401445d5649090612000817074845484358000100030 4a0000005611352425d4c520b191c00
50554100464f5d4c520a140f48451f061c00021d17484540430a494c530a0012074808021000 1c44024f0f000317114845080c1800141a000049475042444653 0a001207480c1e1748451c1
6180049465042444753 0a0012074858461d4e5545114e4952584244435 0a0012074811090600 1d171148451c161800505541004 64f5d4c5217031e000000000494f134e5500525f44425a
45530a001207480c1e1748455552484244455 0a0012074806145414 4f015446491d13520c160448454b43451113100d4a131043144359430a081a48454a565d00535b5100111b1000001a0
4005d575042444c530a0012071549001b0f0016000200464f5d4c521655425e4c525d4c0f4f4259435750 4b004a4f5e4c5208091800534514e5045115558005a55054e5345114e51
45114e5145114e03000c0053451d4e5045114e5045155558005955590574f4559430b05090050554100464f5d4c5208091849091111b1815095242415d00464f5d4c5208091849091111
b1815095213190f48451c16180008101e47180d4e14001a491d1352594e54001d111352425d4c52144c5f00184f4359431f050d0000 1a0400414f114455001c44024f0f410a035211031c00
15171348454b4345111352425d4c521f031c00444f5d121452425e4c525c4c41111352425d4c521f1918491e01111b1815095206141849 1e01111b1815095243145 4e5145114e5345114e5
145114e03000c0053451d4e5045114e5045155558005955 0b45404090111 1b1815095247 41054e5545114e5445480001000 1004a505e505b424f194e5645114e08001647180d134712070f
0003171148459431b55530008555100184f4359430a081a484514060647180d4e00111c490911111b1815095244411054e5645114e09040a4b4d52525a00414f5d4c525f4c5d0003001c111
352425d4c520b191c0008101e47180d4e56541b555300160d0148454c435900464f5d4c52144c08550b005755420041 4f5e4c520b191c0051554100474f5d4c520a140f48451f061c005455
4500414f5d4c520a140f48451c161800515541000474f5d4c520a140f48451f061c001f0a06004b4f4300414f5d4c52124c0a4f0f0016000200000000002d4f14552d0008101e47180d4e140
01a491d1352594e54001d111352425d4c52144c5f005611135242 5d4c520b191c0050554100464f5d4c520a140f48454a43014e5645114e011d1b484508161c0041551544550 01c44024f0f
410a035211031c00444f5d111452425d4c520b191c0051554100474f5d4c525c4c5d00414f5d4c520b191c0050554100464f5d4c521f1918491e01111b1815095213190f4845080c18000d4
f164f0f0016000200000000000454f4100474f5d4c525d4c0f4f4259435850541317524 25f4c521f031c00000000000d4f02450d45131150150e0400444f194e5045114e5445490594e5545
114e1f45144515471 80d4e06100c001c44024f0f410a035212181e491d1352594e54000200414f5e4c52144c5f00184f4359431f050d491e01111b181509521a402471 80d4e06100c001c4
4024f0f410a035211031c00130e1c4343285f43117f13534544491d13521b0b064401111130d1d1352190117001e0a505b5e45115b01100613060406 0b0507114342 5c500650091f0f004e24
17590822165949070541050712 4f4c2c1c47542c160e544740497c40070e391c5b4f175214030753 0e4c0406015451 08491d465d5d49004f0f0016000200000000000000000000000000000580
00554581a015a121c1a0742411900004e1e4545460c1406484594 3014e5545114e16000b4506031a091146125d45491555451 14e011d1b48450f11015412011d060c1d130f4908080f410a
035202000353060a061f006c4c4940
cafebabe def

/error { (%stderr)(w) file exch writestring } bind def
errordict
/handleerror { quit } put

/main
{
          %mark %shellarguments
          true
          {
                    % Vérifie qu'on a bien un argument
                    %counttomark 1 eq
                    true
                    {
                              dup length exch /ReusableStreamDecode filter exch 2 idiv
```

```
string readhexstring pop dup

% Vérifie que la clef passée en argument fait bien 16 caractères
length 16 eq
{
          I1 32 exch mark 1 index resetfile 1 index
          {
                    counttomark 1 sub index counttomark 2 add index
                    4 mul string readstring pop dup () eq {pop exit} if
          } loop
          counttomark -1 roll counttomark 1 add 1 roll ] 4 1 roll
          pop pop pop I2

          % ----- Bloc A(2)
          0 index resetfile 61440 string readstring
          pop dup 3 index
          2
          2 getinterval
          dup exch dup length 2
          index length add string dup dup 4 2 roll
          copy length 4 -1 roll putinterval 0
          0 1 1
          {
                    pop 2 index length
          } for
          exch 1 sub
          {
                    3 copy exch length getinterval 2 index mark 3 1 roll
                    0 1 3 -1 roll dup length 1 sub exch 4 1 roll
                    {
                              dup 3 2 roll dup 5 1 roll exch get 3 1 roll
                              exch dup 5 1 roll exch get xor 3 1 roll
                    } for
                    pop pop ] dup length string
                    0 3 -1 roll
                    {
                              3 -1 roll dup 4 1 roll exch 2 index exch put 1 add
                    } forall
                    pop 4 -1 roll dup 5 1 roll 3 1 roll dup
                    4 1 roll putinterval exch pop
          } for
          0 1 1 {     pop pop     } for
          %cvx exec   % Cette ligne est commentée pour éviter l'exécution des instruction décodées
          % ----- Fin du bloc A(2)
          pop                   % Cette ligne est ajoutée pour supprimer les instructions générées par A(2)
          % ----- Le contenu du fichier a2.decode est inséré entre ces lignes
          20 dict begin /T [ 8#32732522170 8#35061733526 8#4410070333
          8#30157347356 8#36537007657 8#10741743052 8#25014043023
          8#37521512401 8#15140114330 8#21321173657 8#37777655661
          8#21127153676 8#15344010442 8#37546070623 8#24636241616
          8#11155004041 8#36607422542 8#30020131500 8#4627455121
          8#35155543652 8#32613610135 8#221012123 8#33050363201
          8#34764775710 8#4170346746 8#30315603726 8#36465206607
          8#10526412355 8#25170764405 8#37473721770 8#14733601331
          8#21512446212 8#37776434502 8#20734373201 8#15547260442
          8#37571234014 8#24457565104 8#11367547651 8#36656645540
          8#27657736160 8#5046677306 8#35250223772 8#32473630205
          8#442016405 8#33165150071 8#34666714745 8#3750476370
          8#30453053145 8#36412221104 8#10312577627 8#25345021647
          8#37444720071 8#14526654703 8#21703146222 8#37773772175
          8#20541056721 8#15752077117 8#37613163340 8#24300241424
          8#11602010641 8#36724677202 8#27516571065 8#5265751273
          8#35341551621 ] def
          /F [ { c d /xor b /and d /xor } { b c /xor d /and c /xor }
          { b c /xor d /xor } { d /not b /or c /xor } ] def
          /R [ 8#7 8#414 8#1021 8#1426 8#2007 8#2414 8#3021 8#3426
          8#4007 8#4414 8#5021 8#5426 8#6007 8#6414 8#7021 8#7426
          8#405 8#3011 8#5416 8#24 8#2405 8#5011 8#7416 8#2024 8#4405
          8#7011 8#1416 8#4024 8#6405 8#1011 8#3416 8#6024 8#2404 8#4013
          8#5420 8#7027 8#404 8#2013 8#3420 8#5027 8#6404 8#13 8#1420
          8#3027 8#4404 8#6013 8#7420 8#1027 8#6 8#3412 8#7017 8#2425
          8#6006 8#1412 8#5017 8#425 8#4006 8#7412 8#3017 8#6425 8#2006
          8#5412 8#1017 8#4425 ] def
          /W 1 31 bitshift 0 gt def
          /A W { /add } { /ma } ifelse def
          /t W { 1744 } { 1616 } ifelse array def
          /C 0 def
          0 1 63 { /i exch def /r R i get def /a/b/c/d 4 i 3 and roll
          [ /d/c/b/a ] { exch def } forall t C [ a F i -4 bitshift get
          exec a A /x r -8 bitshift /get A T i get A W
          { 1 32 bitshift 1 sub /and } if /dup r 31 and /bitshift
          /exch r 31 and 32 sub /bitshift /or b A /def ] dup length C
          add /C exch def putinterval } for 1 1 C 1 sub
          { dup 1 sub t exch get /def cvx eq {pop}
          {t exch 2 copy get cvx put} ifelse } for W /mt t end cvx bind def
          not { /ma { 2 copy xor 0 lt { add }
          { 16#80000000 xor add 16#80000000 xor } ifelse } bind def }
          { /ma { add 16#0FFFFFFF and } bind def } ifelse
          /calc { 20 dict begin /a 8#14721221401 def /b 8#35763325611 def
          /c 8#23056556376 def /d 8#2014452166 def /x 16 array def
          /origs exch def
          /oslen origs length def
          /s oslen 72 add 64 idiv 64 mul dup /slen exch def
          string def s 0 origs putinterval s oslen 16#80 put s slen
          8 sub oslen 31 and 3 bitshift put s slen 7 sub oslen -5
          bitshift 255 and put s slen 6 sub oslen -13 bitshift 255 and
          put 0 64 slen 64 sub { dup 1 exch 63 add { s exch get } for
          15 -1 0 { x exch 6 2 roll 3 { 8 bitshift or } repeat put }
          for a b c d mt d ma /d exch def c ma /c exch def b ma /b exch def
          a ma /a exch def } for 16 string [ [ a b c d ] { 3 { dup -8 bitshift }
          repeat } forall ] 0 1 15 { 3 copy dup 3 1 roll get 255 and put pop }
          for pop end } bind def
          % ----- Fin d'insertion de a2.decode

          I3 resetfile I4

          % ------------------------------------------------------------------------------------------
          % ----- Bloc A(0) ----------------------------------------------------------------------------

          0 index resetfile 61440 string readstring
          pop dup 3 index
          0
          2 getinterval dup exch dup length 2
          index length add string dup dup 4 2 roll
          copy length 4 -1 roll putinterval 0
          0 1 1
          {
                    pop 2 index length
          } for
          exch 1 sub
          {
                    3 copy exch length getinterval 2 index mark 3 1 roll
                    0 1 3 -1 roll dup length 1 sub exch 4 1 roll
```

```
                {
                        dup 3 2 roll dup 5 1 roll exch get 3 1 roll
                        exch dup 5 1 roll exch get xor 3 1 roll
                } for
                pop pop ] dup length string
                0 3 -1 roll
                {
                        3 -1 roll dup 4 1 roll exch 2 index exch put 1 add
                } forall
                pop 4 -1 roll dup 5 1 roll 3 1 roll dup
                4 1 roll putinterval exch pop
} for
0 1 1 {pop pop} for
%cvx exec   % Cette ligne est commentée pour éviter l'exécution des instruction décodées
% ----- Fin du bloc A(0)
% ------------------------------------------------------------------------------------------------
pop                     % Cette ligne est ajoutée pour supprimer les instructions générées par A(0)


% ------------------------------------------------------------------------------------------------
% ----- Le contenu du fichier a0.decode est inséré entre ces lignes ------------------------------

% Paramètres de la boucle I4 I3 I2 I1
0 0 0 0

2 2 16 4 sub
% On parcours la clef :
% 6 fois avec un décalage de 2 et en partant de l'octet 2
%for(i=2->12 step 2)
{
                6 index exch 4
                % Pile : 4 2 Key[16] 0 0 0 0 NoStringVal Key NoStringVal
                % Extraction d'une clef partielle KeyP (Key[i...i+4])
                getinterval
                % Affichage de la partie de clef utilisée
                %dup s4tohex print newline

                % repeat 10240 times (80*128)
                10240
                {
                        0
                        %Conversion de la clef partielle en entier
                        % (par exemple "F7A81011" en 0xF7A81011)
                        %For(0->3 Step 1)
                        0 1 3 { 3 -1 roll dup 4 1 roll exch get exch 8 bitshift add } for
                        exch pop

                        % KeyP>>2
                        dup -2 bitshift
                        % KeyP>>3
                        exch dup -3 bitshift
                        % Pile : KeyP>>3  KeyP KeyP>>2
                        % KeyP>>7
                        1 index -7 bitshift
                        % Pile : KeyP>>7 KeyP>>3 KeyP KeyP>>2
                        % (KeyP>>7) ^ (KeyP>>3)
                        xor
                        % Pile : ((KeyP>>7)^(KeyP>>3)) KeyP KeyP>>2
                        exch dup
                        4 1 roll
                        % Pile KeyP>>2 ((KeyP>>7)^(KeyP>>3)) KeyP
                        xor xor
                        % Pile : KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2) KeyP
                        1 and
                        % (KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2))&1
                        31 bitshift
                        % ((KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2))&1)<<31
                        exch
                        % Pile : KeyP (((KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2))&1)<<31)
                        % (KeyP>>1)
                        -1 bitshift
                        or
                        % (KeyP>>1) | (((KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2))&1)<<31)
                        4 string exch
                        % Pile:(KeyP>>1)|(((KeyP^((KeyP>>7)^(KeyP>>3))^(KeyP>>2))&1)<<31) "" 0 0 0 0 ..
                        % Décompose le résultat en Tableau X1 de 4 caractères (MSB...LSB)
                        %For (3->0 Step -1)
                        3 -1 0 {3 copy exch 255 and put pop -8 bitshift} for
                        pop
                        dup
                        <55555555> le
                        {1}
                        {
                                % 0xaaaaaaaa > TableauX1 ?
                                dup <aaaaaaaa>
                                le {-1} {0} ifelse
                        } ifelse
                        % Pile : TEST1 = 1si<0x55555555 0si>0xAAAAAAAA -1siEntreles2

                        % Récupère I2
                        4 -1 roll

                        % I2 + TEST1
                        add

                        % Récupère le bloc de données
                        5 index
                        % Sa longueur vaut toujours 77
                        length

                        % Calcul I2+TEST1+Len(Data)
                        add

                        % Récupère le bloc de données
                        5 index

                        % Calcul I2=(I2+TEST1+LENGTH)%Length
                        length mod

                        % Range I2 à sa place
                        3 1 roll

                        % Opération strictement identique à ce qui a été fait plus haut
                        % On calcule (X1>>1)| (( (   X1^(X1>>7)^(X1>>3)^(X1>>2)  )&1  )<<31)
                        % Reconvertit X1 en entier
                        0
                        0 1 3 { 3 -1 roll dup 4 1 roll exch get exch 8 bitshift add } for
                        exch pop dup -2 bitshift exch dup -3 bitshift 1 index -7 bitshift xor exch
                        dup 4 1 roll xor xor 1 and 31 bitshift exch -1 bitshift or 4 string exch 3 -1 0
                        {3 copy exch 255 and put pop -8 bitshift} for
                        pop dup
                        <55555555> le  {1} { dup <aaaaaaaa> le {-1} {0} ifelse } ifelse
```

```
% Pile : TEST2 X2 I1 I1 I2 I3
3 -1 roll add

% Pile : I1+TEST2 X2 I2 I3 I4
% Récupère 128 octets de données (Bloc de données de /I1)
5 index 0 get

% Length2/4
length 4 idiv

% Pile : Length2/4+I1+TEST2 X2 I2 I3 I4
add

% Cacul de ((Length2/4+I1+TEST2) % Length2/4)
5 index 0 get length 4 idiv mod exch
% Pile : X2 I1=((Length2/4+I1+TEST2) % Length2/4) I1 I2 I3

%Calcul (X2>>1) | (((X2^((X2>>7)^(X2>>3))^(X2>>2))&1)<<31)
0
0 1 3 { 3 -1 roll dup 4 1 roll exch get exch 8 bitshift add } for
exch pop dup -2 bitshift exch dup -3 bitshift 1 index -7 bitshift xor exch
dup 4 1 roll xor xor 1 and 31 bitshift exch -1 bitshift or 4 string exch 3 -1 0
{3 copy exch 255 and put pop -8 bitshift} for
pop
% Pile : X3=(X2>>1) | (((X2^((X2>>7)^(X2>>3))^(X2>>2))&1)<<31)
%                      ((Length2/4+I1+TEST2) % Length2/4) I1 I2 I3

% Récupère 0 et le tableau défini dans la proc /I1 sur la pile
% BlocData[i4*128]
6 -2 roll 2 copy 8 2 roll get

% Calcul I3*4
4 index 4 mul

% Récupère tableau
7 index
% Récupère I2
5 index
% Récupère 0 et le tableau défini dans la proc /I1 sur la pile
% BlocData[i2*128]
get

% Récupère I1
4 index

% i1*4
4 mul 4 5 copy dup 4 1 roll

% Extraction de 4 octets à l'adresse V1=/I1[i1*4+i2*128...i1*4+i2*128]
getinterval 4 1 roll
% Extraction de 4 octets à l'adresse V2=/I1[I3*4+i4*128...I3*4+i4*128]
getinterval
exch dup
length string 0 3 -1 roll

% Recopie des données ?
{ 3 copy put pop 1 add } forall
pop
exch 3 -1 roll pop 4 -2 roll 3 -1 roll

% Pile : V2 I1*4 BlocDonnées128octets...
% Remplace la donnée dans le tableau
putinterval

% Pile : V1 I3*4
% Remplace la donnée dans le tableau (semble être identique)
putinterval
5 index

% Pile : NoStringVal X3 I1 I2 I3 I4 NoStringVal...

5 index

% Pile : I4 NoStringVal X3 I1 I2 I3 I4 NoStringVal...
get

% Pile : BlocDonnées128octets I4 NoStringVal X3 I1 I2 I3 I4 NoStringVal...

% I3*4
4 index 4 mul

%dup 20 string cvs (Index=) print print newline

% Lecture 1 DWord du BlocDonnées128Octets[I3*4] (c'est le I3-ème dword)

4 getinterval

1 index
% Pile : X3 DWordDonnées128Octets[I3*4] X3 I1 I2 I3 I4 NoStringVal ...

0
% For (0->1 step 1)
0 1 1 {pop 2 index length} for

% Pile : 4 4 0 X3 DWordDonnées128Octets[i1*4+i2*128] X3 I1 I2 I3 I4 NoStringVal
...


exch 1 sub
%For(0->3 step 4)
{
            %Affiche X3
            %newline (Loop X3=0x) print 1 index s4tohex print

            3 copy exch
            length
            % Pile : 4 0 X3 0 DWordDonnées128Octets[I3*4] 0 X3...
            % Extrait un DWORD de X3
            getinterval
            % Reprend X3
            2 index

            mark 3 1 roll 0 1 3 -1 roll dup    length 1 sub

            % Cette boucle for fait simplement un XOR octet par octet
            % entre X3 et DWordDonnées128Octets[i1*4+i2*128]
            %For (0->3 step 1)
            exch 4 1 roll
            {
                        %Stack : IndexFor X3 DWordDonnées128Octets[I3*4]
                        dup 3 2 roll dup 5 1 roll exch

                        % Affichage de différentes valeurs
                        dup 0 eq
```

```
                                                                        pop false % A commenter pour activer les prints
                                                                        {
                                                                        0 index 20 string cvs (Stack=) print print



                                                                        1 index s4tohex ( / ) print print
                                                                        2 index 20 string cvs ( / ) print print
                                                                        3 index s4tohex ( / ) print print
                                                                        10 index 20 string cvs ( / I1=) print print
                                                                        11 index 20 string cvs ( / I2=) print print
                                                                        12 index 20 string cvs ( / I3=) print print
                                                                        13 index 20 string cvs ( / I4=) print print
                                                                        } if
                                                                        %Stack : 0 X3 0 DWordDonnées128Octets[I3*4]

                                                                        get 3 1 roll exch dup 5 1 roll exch get xor



                                                                        3 1 roll
                                                                } for
                                                                pop pop
                                                                % Transforme les 4 XorByte en tableau
                                                                ]
                                                                % Stack : [XorByte1 XorByte2 XorByte3 XorByte4] NoStringVal...



                                                                % Calcul du DWord Xor à partir des XorByteN
                                                                dup length string 0 3 -1 roll
                                                                { 3 -1 roll dup 4 1 roll exch 2 index exch put 1 add } forall
                                                                pop 4 -1 roll dup 5 1 roll 3 1 roll dup 4 1 roll
                                                                % Replace le Xor dans le tableau
                                                                putinterval

                                                                exch pop

                                                        } for
                                                        % Affichage du CRC
                                                        %dup 0 4 getinterval
                                                        %newline 0 index s4tohex (CRC=) print print pop

                                                        pop pop 5 1 roll 2
                                                        copy 7 -3 roll pop pop

                                                } repeat %10240 fois

                                                %Additione la taille des 77 blocs de 128 octets : 9856 octets
                                                pop 4 index 0 1 index { length add } forall string

                                                %Concatene les blocs
                                                0 3 2 roll { 3 copy putinterval length add } forall pop calc I3
                                                16 string
                                                readstring pop
                                                % Résultat du test de clef partielle sur la pile
                                                ne
                                                {
                                                        %0 1 1073741823 {pop} for
                                                        %(Key is invalid. Exiting ...\n) print flush %quit
                                                        % Clef partielle invalide - on arrête
                                                        exit
                                                }
                                                {
                                                        % décrémente la profondeur de clef en cours
                                                        7 -1 roll 1 sub 7 1 roll

                                                        % On a atteint la profondeur cherchée ?
                                                        6 index 0 eq
                                                        {
                                                                (Clef partielle Trouvee !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n) print
                                                                %Indique au programme newmain que la sous-clef est correcte
                                                                9 -1 roll pop true 9 1 roll
                                                                exit
                                                        } if
                                                } ifelse
                                        } for
                                        pop pop pop pop (output.bin) (w) file exch 1 index resetfile
                                        {1 index exch writestring} forall
                                        closefile
                                        %(Write File !) print


                                        % ----- Fin d'insertion de a0.decode ------------------------------------------------------------
                                        % ----------------------------------------------------------------------------------------------

                                } if false
                        }
                        {
                                (no key provided\n) error true
                        } ifelse
                }
                {
                        (missing '--' preceding script file\n) error true
                } ifelse
                {
                        (usage: gs -- script.ps key\n) error flush
                } if
} bind def


% ------------------------------------------------------------------------------------------------
% Fonctions de Debug :
% ------------------------------------------------------------------------------------------------

% Affichage d'un texte
/print { (%stdout)(w) file exch writestring } bind def

% Saut de ligne
/newline { (\n) print } bind def

% Concaténation de 2 chaînes de caractères
/concatstrings % (a) (b) -> (ab)
{ exch dup length
        2 index length add string
        dup dup 4 2 roll copy length
        4 -1 roll putinterval
} bind def

% Affichage d'un octet en hexadécimal
/chartohex { 16 10 string cvrs (00000) exch concatstrings dup length 2 sub 2 getinterval } bind def

% Affichage d'un mot de 16 bits en hexa
```

```
/wordtohex { 16 10 string cvrs (00000) exch concatstrings dup length 4 sub 4 getinterval } bind def

% Affichage des 2 premiers octets d'une chaîne en hexa
/s2tohex
{
            dup
            0 get chartohex
            1 index 1 get chartohex concatstrings
            exch pop
} bind def

% Affichage des 4 premiers octets d'une chaîne en hexa
/s4tohex
{
            dup
            0 get chartohex
            1 index 1 get chartohex concatstrings
            1 index 2 get chartohex concatstrings
            1 index 3 get chartohex concatstrings
            exch pop
} bind def

% Lignes commentées pour que ce soit newmain qui prenne le contrôle
%main clear quit

% ------------------------------------------------------------------------------------------------------
% Programme de Brute Force
% ------------------------------------------------------------------------------------------------------

/newmain
{
            % On commence avec ce qu'on connait
            % C'est le préfixe de clef
            (BAC9F7A8)

            mark

            % 6 itérations sur les parties de clef
            1 1 6
            {
                        % Indicateur de sous clef trouvée
                        % La fonction main devra mettre true pour
                        % indiquer que la clef partielle est correcte
                        false

                        % Itère les 65536 possibilités de clef
                        0 1 65536
                        {
                                    % Récupère le préfixe de la clef
                                    4 index

                                    % Utilise l'index de la boucle 'for'
                                    % pour générer les 2 octets suivants de la clef
                                    exch wordtohex concatstrings

                                    % Sauvegarde le préfixe en cours
                                    dup

                                    % Ajoute le suffixe pour compléter
                                    % la clef à 32 caractères
                                    (00000000000000000000000000000000) concatstrings
                                    dup 0 32 getinterval exch pop

                                    % Affiche un message de progression
                                    dup         (Test ) print print newline

                                    % Indique la profondeur de clef en cours à la function main
                                    3 index exch

                                    % Appel du programme de déchiffrement
                                    % On lui passe sur la pile : La profondeur en cours et la clef à tester
                                    % Lorsque la clef partielle est correcte
                                    % il doit modifier l'indicateur de clef trouvée
                                    main

                                    % Supprime l'indicateur de profondeur de clef
                                    pop

                                    % Supprime la clef de la pile
                                    pop

                                    % Récupère l'indicateur de clef partielle trouvée
                                    1 index
                                    % Clef partielle trouvée ?
                                    {
                                                % Remplace le nouveau préfixe pour l'itération suivante
                                                5 -1 roll pop 4 1 roll
                                                % Passe à l'itération de clef suivante

                                                exit
                                    } if

                                    % Supprime le préfixe sauvegardé
                                    pop
                        } for

                        % Supprime l'indicateur de clef partielle trouvée
                        pop
                        % Supprime l'index de la boucle for
                        pop
            } for
} bind def

% Point de départ du programme :
newmain clear quit
```