Challenge SSTIC 2015

David BERARD
Thales Communication & Security
david.berard@thalesgroup.com

Résumé

Ce document présente la démarche choisie par l'auteur pour résoudre le challenge SSTIC 2015. Comme tous les ans, le but de ce challenge est de retrouver une adresse email cachée dans le fichier fourni comme énoncé. Cette année le challenge comporte 9 épreuves très variées.

Le challenge débute avec une archive contenant une image disque d'une carte microSD, cette image doit être analysée pour en extraire deux fichiers. Un de ces deux fichiers est un script pour la clé USB Rubber Ducky contenant une séquence de touche clavier. Une fois la séquence reconstituée un script Powershell est extrait, son contenu permet de continuer le challenge à l'étape suivante.

La seconde étape est dissimulée au milieu d'une carte additionnelle pour le jeu Quake 3 Arena, deux approches de résolution seront présentées dans ce document, la première sans jouer au jeu, depuis les fichiers du jeu, et l'autre en jouant au jeu. La résolution de cette épreuve permet d'obtenir un nouveau fichier constituant l'étape suivante.

La troisième étape est une trace USB contenant des mouvements de souris, la reconstruction des déplacements permet de reconstruire une image permettant d'accéder à l'étape suivante.

Le challenge continu avec une page HTML contenant du code JavaScript obfusqué. Une fois dé-obfusqué les indices présents dans ce code permettent de grandement réduire les possibilités pour mener une attaque par force brute efficace.

La cinquième étape est certainement la plus complexe, c'est une architecture de plusieurs microcontrôleurs en réseau (Transputers d'architecture ST20) dialoguant ensemble pour charger du code ou échanger des données. Un algorithme cryptographique est réparti sur l'ensemble des nœuds. Pour résoudre cette épreuve, un émulateur pour l'architecture a été développé. Ensuite le code de chacun des Transputers a été étudié pour mettre en place un simulateur. Ce simulateur permet l'attaque de l'algorithme en force brute, en exploitant une faiblesse de l'algorithme.

Les quatre dernières épreuves sont des étapes de stéganographie. La résolution de ces étapes permet de trouver l'adresse email tant recherchée.

Table des matières

1 Stage 1 - Forensics & USB-Rubber-Ducky script **4**
1.1 Découverte de l'épreuve . . . 4
1.2 USB-Rubber-Ducky script . . . 4

2 Stage 2 - Carte pour le jeu Quake3 **7**
2.1 Découverte de l'épreuve . . . 7
2.2 Carte supplémentaire pour Quake3 . . . 8
2.2.1 Résolution dans le jeu . . . 8
2.2.2 Résolution à partir du fichier sstic.bsp . . . 10
2.3 Déchiffrement des données . . . 13

3 Stage 3 - Souris USB et cryptographies exotiques **14**
3.1 Découverte de l'épreuve . . . 14
3.2 Analyse des échanges USB . . . 14
3.3 Reconstruction de l'image à partir de la capture . . . 16
3.4 Blake256 . . . 17
3.5 Serpent-1-CBC-With-CTS . . . 17

4 Stage 4 - Rétro Ingénierie de JavaScript obfusqué **19**
4.1 Découverte de l'épreuve . . . 19
4.2 Dé-obfuscation . . . 19
4.3 Attaque par force brute des UserAgents . . . 22

5 Stage 5 - Rétro ingénierie ST20 - Transputer **24**
5.1 Découverte de l'épreuve . . . 24
5.2 Transputer - architecture ST20 . . . 25
5.3 Implémentation d'un émulateur pour l'architecture ST20 . . . 26
5.4 Compréhension du système . . . 27
5.4.1 Boot de l'architecture complète . . . 27
5.4.2 Initialisation de l'algorithme . . . 27
5.5 Rétro-ingénierie de l'assembleur ST20 . . . 28
5.6 Faiblesse dans l'algorithme . . . 30
5.7 Simulation . . . 30
5.8 Attaque par force brute . . . 31

6 Stage 6 - Stéganographies diverses **34**
6.1 Découverte de l'épreuve . . . 34
6.2 Archive ajoutée à la fin de l'image . . . 34
6.3 Flux Zlib caché dans des Chunk PNG . . . 36
6.4 Données dissimulées dans les bits de poids faible . . . 37
6.5 Solution du challenge . . . 39

7 Conclusion **40**

8 Remerciements **40**

9 Annexes **41**
9.1 *Parseur* de fichier BSP . . . 41
9.2 Émulateur pour l'architecture ST20 . . . 46
9.3 Simulateur pour T0 . . . 59

Table des figures

1 Chargement de la carte dans OpenArena . . . 8
2 Activation du mode "passe murailles" . . . 9
3 Mur dans la salle secrète . . . 9
4 Exemple de deux tuiles similaires . . . 10
5 Suite de pictogrammes/couleurs reconstruite à partir du fichier `sstic.bsp` . . . 12
6 Textures restantes réelement affichées . . . 12
7 Ouverture du fichier `paint.cap`dans Wireshark . . . 14
8 Ouverture du fichier `paint.cap`dans Wireshark . . . 15
9 Image reconstruite à partir de la capture . . . 16
10 Contenu du fichier `schematic.pdf` . . . 24
11 Architecture complète à base d'émulateurs . . . 26
12 Code final de T4 dans IDA Pro . . . 28
13 Répartition des calculs . . . 29
14 Simulation de T4 et T5 . . . 30
15 Simulation de T1, T2 et T3 . . . 31
16 congratulations.jpg . . . 34
17 congratulations.png . . . 35
18 congratulations.tiff . . . 37
19 visualisation avec `stegsolve`des bits de poids faible des composantes rouge et verte . . . 37
20 Extraction des données avec `stegsolve`, visualisation de l'en-tête Bzip . . . 38
21 congratulations.gif . . . 38
22 Radom colour map sur congratulations.gif . . . 39

Listings

1 Analyse de l'image disque avec testdisk . . . 4
2 Montage de l'image . . . 4
3 Recherche de fichiers supprimés avec l'outil testdisk . . . 4
4 Contenu de `build.sh` . . . 4
5 Décodage de `input.bin`avec `ducky-decode` . . . 5
6 Script Powershell contenu dans les chaînes base64 . . . 5
7 Script Powershell assurant la vérification d'intégrité . . . 6
8 Reconstruction du fichier ZIP . . . 6
9 Contenu de `stage2.zip` . . . 7
10 Stage 2 : Contenu du fichier `memo.txt` . . . 7
11 Analyse du fichier `sstic.pk3` . . . 7
12 Conversion du fichier `maps/sstic.bsp`pour analyse avec l'éditeur de niveau `GtkRadiant` . 8
13 Extraction des textures réelement affichées, reconstruction du "mur" . . . 11
14 Test des 4 possibilités restantes . . . 12
15 Résultat du test des 4 possibilités . . . 12
16 Déchiffrement du fichier `encrypted`avec la clé retrouvée dans le jeu . . . 13
17 Vérification d'intégrité du résultat . . . 13
18 Type du fichier déchiffré . . . 13
19 Contenu de l'archive `stage3.zip` . . . 14
20 Stage 3 : Contenu du fichier `memo.txt` . . . 14
21 Extraction des mouvements de la souris . . . 15
22 Reconstruction de l'image à partir des mouvement de souris . . . 16
23 Calcul de la clé avec l'algorithme Blake . . . 17
24 Résultat du calcul de la clé . . . 17
25 Script de déchiffrement avec l'algorithme Serpent-1-CBC-With-CTS . . . 17
26 Résultat du calcul de la clé . . . 18
27 Extraction de l'archive `stage4.zip` . . . 19
28 Contenu de `stage4.html` . . . 19
29 Modification du Javascript pour extraire des données . . . 19
30 Exemple de code javascript après premier niveau de dé-obfuscation . . . 19
31 Script python assurant la dé-obfuscation Javascript . . . 20
32 Résultat de la dé-obfuscation . . . 21
33 Génération de l'IV et de la clé . . . 22

34 Bruteforce des UserAgents 22
35 Solution du stage4 23
36 Extraction de l'archive `stage5.zip` 24
37 Format du vecteur de test 27
38 Écritures et lectures sur les tubes de la clé 27
39 Sortie du vecteur de test (*link* 0 du *Transputer* 0) 27
40 pseudo code T4 28
41 Opération sur le premier bloc 30
42 Génération de la liste des clés 31
43 Second bloc de l'en-tête Bzip2 32
44 Creation du fichier de test 32
45 Attaque par force brute 33
46 Résultat de l'attaque par force brute 33
47 Contenu de l'archive `congratulations.tar.bz2` 34
48 Recherche de signatures avec l'outil `binwalk`dans `congratulations.jpg` 34
49 Extraction de l'archive contenue dans `congratulations.jpg` 35
50 Structure de l'image `congratulations.png` 36
51 Extraction des "chunks" avec le tag `sTic` 36
52 Extraction des "chunks" avec le tag `sTic` 36
53 Parser de fichier BSP 41
54 Émulateur pour l'architecture ST20 46
55 Simulateur de T0 59

1 Stage 1 - Forensics & USB-Rubber-Ducky script

1.1 Découverte de l'épreuve

Le fichier `challenge.zip` fourni comme énoncé du challenge, est un fichier ZIP contenant l'image d'une carte mémoire microSD. L'analyse de cette image révèle la présence un système de fichier FAT.

```
$ testdisk /list sdcard.img
TestDisk 6.14, Data Recovery Utility, July 2013
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org
Please wait...
Disk sdcard.img - 128 MB / 122 MiB - CHS 123 64 32
Sector size:512


Disk sdcard.img - 128 MB / 122 MiB - CHS 123 64 32
     Partition      Start        End    Size in sectors
   P FAT16                    0   0  1    122   4 16     250000 [NO NAME]
     FAT16, blocksize=2048
```

Listing 1: Analyse de l'image disque avec testdisk

Cette partition FAT peut être montée simplement :

```
# mount -o loop sdcard.img /mnt
# ls -lah /mnt/*
-rwxr-xr-x 1 root root 33M 26 mars  04:49 /mnt/inject.bin
```

Listing 2: Montage de l'image

L'image disque contient un fichier `inject.bin` d'un format non déterminé. Sans information supplémentaire il va être difficile de l'analyser.
Afin d'obtenir plus d'information, une analyse de l'image disque à la recherche de fichiers supprimés est réalisée avec l'outil `testdisk` :

```
TestDisk 6.14, Data Recovery Utility, July 2013
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org
   P FAT16                    0   0  1    122   4 16     250000 [NO NAME]
Directory /

>-rwxr-xr-x     0     0        45 26-Mar-2015 02:49 build.sh
 -rwxr-xr-x     0     0  34253730 26-Mar-2015 02:49 inject.bin
```

Listing 3: Recherche de fichiers supprimés avec l'outil testdisk

Le fichier supprimé `build.sh` est restauré pour analyse. Ce fichier semble être le script qui a permis de générer le fichier `inject.bin`, son contenu est le suivant :

```
java -jar encoder.jar -i /tmp/duckyscript.txt
```

Listing 4: Contenu de `build.sh`

Le but de cette première épreuve est de décoder le contenu du fichier `inject.bin`.

1.2 USB-Rubber-Ducky script

Une rapide recherche sur Internet avec le mot-clé "`duckyscript`" permet de trouver le format de ce fichier, c'est un script pour la clé USB Rubber Ducky.
La clé USB Rubber Ducky est une clé programmable se comportant comme un clavier, celle-ci va envoyer une séquence de touches programmée à sa connexion. Le script contenant la séquence de touches est stocké sur une carte microSD insérée dans la clé USB Rubber Ducky.
Le fichier `inject.bin` est donc un script contenant une séquence de touches clavier à envoyer au branchement de la clé.

Un décodeur en Perl est disponible sur le site du projet communautaire associé. En utilisant ce décodeur, la séquence de touche est obtenue :

```
$ ./ducky-decode.pl -f inject.bin
00ff 007d
GUI R

DELAY 500

ENTER

DELAY 1000
 c m d
ENTER

DELAY 50
 p o w e r s h e l l
SPACE
 - e n c
SPACE
 Z g B 1 A G 4 A Y w B 0 A G k A b w B u A C A A d w B y A G k A d A B l A F 8 A Z g B p
 A G w A Z Q B f A G I A e Q B 0 A G U A c w B 7 A H A A Y Q B y A G E A b Q A o A F s A
[...]
 C c A K Q A p A D s A f Q A = 00a0
[...]
```

Listing 5: Décodage de `input.bin` avec `ducky-decode`

La séquence de touches contenue dans `inject.bin` réalise les actions clavier suivantes :

- Ouverture de la console Windows (Super+R, "cmd" puis touche entrée)
- Exécution de multiples commandes Powershell du type : `powershell -enc CHAINE_BASE64`, la chaîne base64 en argument est différente à chaque commande.

Une fois décodées, les chaînes base64 passées en argument (à exception de la dernière) sont des scripts PowerShell (en UTF-16) de la forme suivante :

```
function write_file_bytes{
  param([Byte[]] $file_bytes, [string] $file_path = ".\stage2.zip");
  $f = [io.file]::OpenWrite($file_path);
  $f.Seek($f.Length,0);
  $f.Write($file_bytes,0,$file_bytes.Length);
  $f.Close();
}
function check_correct_environment{
  $e=[Environment]::CurrentDirectory.split('\');
  $e=$e[$e.Length-1]+[Environment]::UserName;
  $e -eq "challenge2015sstic";
}

if(check_correct_environment){
  write_file_bytes([Convert]::FromBase64String('YpeeGfN+MZKhDrRH0WNGhAIeMsJVTSLwjeV
      [...]
      ZAyBm/4Pd2Bik7xzqeGbvCpFyVBA=='));
}else{
  write_file_bytes([Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));
}
```

Listing 6: Script Powershell contenu dans les chaînes base64

La chaîne base64 du premier script powershell (en rouge dans l'exemple ci-dessous) est le début d'un fichier ZIP. Le contenu décodé de cette chaîne base64 est écrit à la fin du fichier `stage2.zip`. Les scripts Powershell suivants contiennent la suite du fichier ZIP.
La chaîne base64 `VAByAHkASABhAHIAZABlAHIA` correspond à la chaîne de caractères `"TryHarder"`.

Le dernier script Powershell réalise le contrôle d'intégrité (sha1) sur le fichier `stage2.zip`.

```
function hash_file{
  param([string] $filepath);
  $sha1 = New-Object -TypeName System.Security.Cryptography.SHA1CryptoServiceProvider;
  $h = [System.BitConverter]::ToString(
      $sha1.ComputeHash([System.IO.File]::ReadAllBytes($filepath))
  );
  $h
}
$h = hash_file(".\stage2.zip");
if($h -eq "EA-9B-8A-6F-5B-52-7E-72-65-20-19-31-3C-25-B5-6A-D2-7C-7E-C6"){
  echo "You WIN";
}else{
  echo "You LOSE";
}
```

Listing 7: Script Powershell assurant la vérification d'intégrité

Pour reconstruire le fichier `stage2.zip`, il faut donc décoder les chaînes base64 contenues dans les scripts Powershell, et concaténer les résultats dans le fichier `stage2.zip`. Après reconstruction le condensat sha1 du fichier `stage2.zip` doit être `EA9B8A6F5B527E72652019313C25B56AD27C7EC6`.

Le code Python ci-dessous permet de reconstruire le fichier `stage2.zip` à partir du fichier décodé par `ducky-decode` :

```
import unicodedata
import hashlib

powershell=open("decoded_ducky.txt","rb")
out=""

for line in powershell:
  if len(line) > 100:
    line = line.replace(" ", "")
    line = line.replace("00a0", "").replace("\n", "")
    line = unicodedata.normalize('NFKD',
        ↪ unicode(line.decode("base64"))).encode('ascii','ignore')
    ret = ""
    for char in line:
      if char != "\x00":
        ret += char
    if "FromBase64String" in ret:
      base64=ret.split("FromBase64String('")[1].split("'));}else")[0]
      out+=base64.decode("base64")

if hashlib.sha1(out).hexdigest() == "ea9b8a6f5b527e72652019313c25b56ad27c7ec6":
  print "sha1 ok"
  stage2=open("stage2.zip","wb")
  stage2.write(out)
  stage2.close()
else:
  print "bad sha1"
```

Listing 8: Reconstruction du fichier ZIP

2 Stage 2 - Carte pour le jeu Quake3

2.1 Découverte de l'épreuve

L'épreuve débute avec le fichier `stage2.zip` extrait à l'étape précédente, cette archive contient les fichiers suivants :

```
$ bsdtar -tf stage2.zip
encrypted
memo.txt
sstic.pk3
```

Listing 9: Contenu de `stage2.zip`

Le fichier `memo.txt` donne plusieurs informations sur l'épreuve :

- Le type de chiffrement utilisé pour le fichier `encrypted` : **AES-OFB**
- L'IV utilisé pour chiffrer le fichier `encrypted` : **0x5353544943323031352d537461676532**
- Un indice indiquant que la clé de chiffrement a été cachée dans un jeu vidéo.
- Les condensats SHA256 du fichier `encrypted` ainsi que le condensat du fichier déchiffré.

```
$ cat memo.txt
Cipher: AES-OFB
IV: 0x5353544943323031352d537461676532
Key: Damn... I ALWAYS forget it. Fortunately I found a way to hide it into my favorite
    ↪ game !

SHA256: 91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945 - encrypted
SHA256: 845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187 - decrypted
```

Listing 10: Stage 2 : Contenu du fichier `memo.txt`

Le but de cette épreuve est donc de retrouver la clé de chiffrement cachée dans un jeu vidéo. Le seul fichier pour lequel nous n'avons pas d'indication est `sstic.pk3`. Une rapide analyse permet d'avoir plus d'information :

```
$ file sstic.pk3
sstic.pk3: Zip archive data, at least v2.0 to extract
$ bsdtar xvf sstic.pk3
x AUTHORS
x levelshots/
x levelshots/sstic.tga
x maps/
x maps/sstic.bsp
x README
x scripts/
x scripts/sstic.arena
x sound/
x sound/world/
x sound/world/bj3.wav
x textures/
x textures/sstic/
x textures/sstic/01.tga
x textures/sstic/02.tga
x textures/sstic/103336131.tga
[...]
x textures/sstic/logo.tga

$ file maps/sstic.bsp
maps/sstic.bsp: Quake III Map file (BSP)
```

Listing 11: Analyse du fichier `sstic.pk3`

Le fichier `sstic.pk3` est une carte supplémentaire pour le jeu Quake3 et ses dérivés. Les cartes supplémentaires sont des fichiers ZIP contenant l'arborescence suivante :

- levelshots : image de la carte présentée lors de la sélection de niveaux.
- maps : dossier contenant le fichier `sstic.bsp`.
- sound : sons supplémentaires pour la carte.
- textures : ensemble des textures (images) à afficher dans le niveau.

2.2 Carte supplémentaire pour Quake3

Cette épreuve peut être résolue de plusieurs façons. La première (celle explorée lors de la résolution de ce challenge) est de charger la carte supplémentaire dans le jeu (ici OpenArena), puis jouer pour trouver des indices et construire la solution (cf. 2.2.1).
La seconde est d'utiliser les données fournies pour trouver la solution sans avoir à lancer le jeu. Cette solution plus élégante a été développée après la résolution du challenge (cf. 2.2.2)

2.2.1 Résolution dans le jeu

```
$ q3map2 -convert -format map maps/sstic.bsp
2.5.17
Q3Map         - v1.0r (c) 1999 Id Software Inc.
Q3Map (ydnar) - v2.5.17
GtkRadiant    - v1.6.4 Jul  5 2014 10:56:57
We're still here

Loading maps/sstic.bsp
--- Convert BSP to MAP ---
writing maps/sstic_converted.map
```

Listing 12: Conversion du fichier `maps/sstic.bsp` pour analyse avec l'éditeur de niveau `GtkRadiant`

Pour analyser cette carte supplémentaire, elle est chargée dans le jeu OpenArena :

FIGURE 1: Chargement de la carte dans OpenArena

La première chose visible sur la carte est une tuile contenant 3 lignes de 4 octets en hexadécimal, de 3 couleurs différentes. Ces images sont réparties sur l'ensemble de la carte. Ces tuiles sont présentes dans le dossier `textures/sstic` sous forme d'image.

Le but est visiblement d'activer une suite de boutons, puis d'effectuer un saut de type *Rocket Jump* pour sauter au-dessus d'une mare de lave. Le saut demande une dextérité que n'a pas l'auteur, une autre méthode a donc été choisie.
Pour explorer plus rapidement la carte, il est possible d'activer le mode "passe murailles" avec la commande `\noclip`, après avoir chargé la carte avec la commande `\devmap sstic` dans la console du jeu :

FIGURE 2: Activation du mode "passe murailles"

Une fois cette commande activée le personnage peut se déplacer dans toutes les directions et traverser les murs. Cela permet de trouver une pièce cachée (accessible uniquement avec un téléporteur), avec des pictogrammes et des couleurs sur un mur :

FIGURE 3: Mur dans la salle secrète

Les pictogrammes sont les mêmes que ceux sur les images contenant les suites de caractères hexadécimaux répartis sur l'ensemble de la carte. Le but doit être de reconstruire la clé avec les morceaux d'hexadécimal en utilisant les pictogrammes et les couleurs comme index pour ordonner les bouts de clé.
La première tentative a été d'utiliser directement les textures présentes dans le fichier ZIP `sstic.pk3`, malheureusement le pictogramme est présent sur plusieurs images différentes :

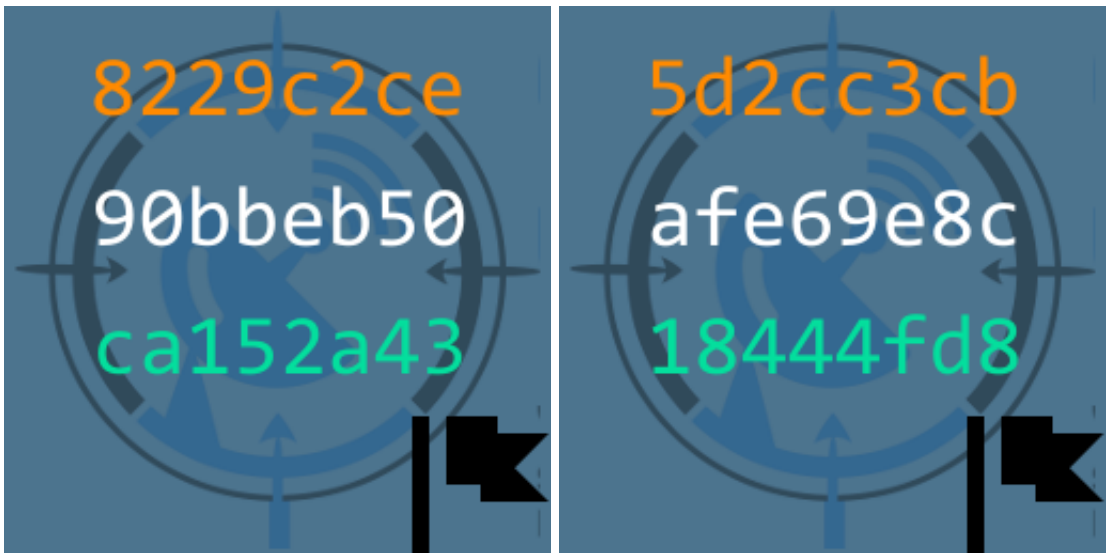


FIGURE 4: Exemple de deux tuiles similaires

Pour pouvoir reconstruire la clé, il faut donc identifier les bonnes tuiles, et en extraire les morceaux de clé en utilisant le couple pictogramme / couleur présent sur le mur de la salle secrète.
Pour cela la première idée est de parcourir le fichier de description de la carte (`sstic.bsp`) , malheureusement pour retrouver les tuiles réellement affichées dans le jeu l'analyse de ce fichier nécessite trop de temps (cette méthode a été explorée après la validation du challenge, cf. 2.2.2).

Une résolution par un parcours manuel de la carte dans le jeu est donc choisie. Dans le jeu on ne trouve qu'une seule tuile pour un pictogramme donné, les autres tuiles sont certainement cachées dans la carte pour ne pas être visibles du joueur (elles sont bien référencées dans le fichier de description).

Après quelques dizaines de minutes à parcourir la carte, les tuiles nécessaires à la reconstruction de la clé sont retrouvées, la clé peut être reconstruite manuellement : `9E2F31F7 8153296B 3D9B0BA6 7695DC7C B0DAF152 B54CDC34 FFE0D355 26609FAC`.

2.2.2 Résolution à partir du fichier sstic.bsp

Après extraction des données contenues dans `sstic.pk3`, les informations nécessaires à la résolution de cette épreuve sont présentes dans le fichier `sstic.bsp` ainsi que dans les textures présentes dans le dossier `textures/sstic`.

Le fichier `sstic.bsp` contient les données constituant la carte supplémentaire (positions des objets/-mur/textures, couleurs, etc.). Ce fichier peut donc être utilisé pour :
- Identifier les textures réellement affichées.
- Reconstruire des suites de textures en fonction de leurs positions dans la carte.

Ce fichier est un format propriétaire à l'éditeur "id Software", plusieurs documentations non officielles existent sur internet [1].

Le format est assez simple, la structure est la suivante :
- 4 octets de marqueur : chaîne "IBSP".
- 4 octets (int) pour marquer la version : 0x2e pour Quake 3.
- Dictionnaire de 17 entrées. Ces entrées sont composées d'un couple `offset` relatif au début du fichier (4 octets) et d'une taille (4 octets également).
- Tableau pointé par les entrées du dictionnaire, les tableaux ont des formats différents.

1. http://www.mralligator.com/q3/

Chaque entrée pointe sur un tableau d'objet, les deux tableaux intéressants pour la résolution de l'épreuve sont :

1. Textures : Liste des textures
13. Faces : Géometrie, position des textures, etc.

Les objets contenus dans le tableau `Faces` contiennent deux informations importantes :
— texture : le numéro de la texture (indice dans le tableau des textures).
— lm_origin : une position d'origine dans l'espace.

Un parseur pour les fichiers BSP a été développé, il est disponible en annexe 9.1.
Le script suivant récupère les textures réellement affichées (présentes dans le tableau `Faces`), puis reconstruit la suite de pictogrammes/couleur présente sur le même plan géométrique (y,z fixes) et enfin génère une image contenant les textures restantes.

```
#!/usr/bin/python2
from BSP import BSP
from PIL import Image
import sys

if len(sys.argv) != 3:
  print "%s [map.bsp] [base_dir]"
  sys.exit(0)

base=sys.argv[2]

bsp=BSP(sys.argv[1])
textures = bsp.getTextures()
faces = bsp.getFaces()
usefull_textures = []
for face in faces:
  name = textures[face["texture"]]["name"]
  if "sstic" in name:
    if name not in usefull_textures:
      usefull_textures.append(name)

key_images = []
wall_images = {}
for face in faces:
  texture_id = face["texture"]
  name = textures[texture_id]["name"]
  if name in usefull_textures:
    if face["lm_origin"][1] == 2401.75 and face["lm_origin"][2] == -352.5:
      wall_images[face["lm_origin"][0]] = name
    else:
      if name not in key_images:
        key_images.append(name)


im = Image.new('RGB', (202*8, 100))
count=0
for key in sorted(wall_images.keys()):
  print wall_images[key]
  wall_image = Image.open(base+"/"+wall_images[key]+".tga")
  im.paste(wall_image, (count*202,0))
  count+=1
im.save("wall.png","PNG")
im = Image.new('RGB', (260*5, 260*2))
count=0
for name in key_images:
  if name not in ("textures/sstic/logo","textures/sstic/01","textures/sstic/02"):
    image = Image.open(base+"/"+name+".tga")
    if image.size[0] != 256:
      continue
    im.paste(image, ((count/2)*260,(count%2)*260))
    count+=1
im.save("key.png","PNG")
```

Listing 13: Extraction des textures réelement affichées, reconstruction du "mur"

Le résultat obtenu est le suivant :

FIGURE 5: Suite de pictogrammes/couleurs reconstruite à partir du fichier `sstic.bsp`

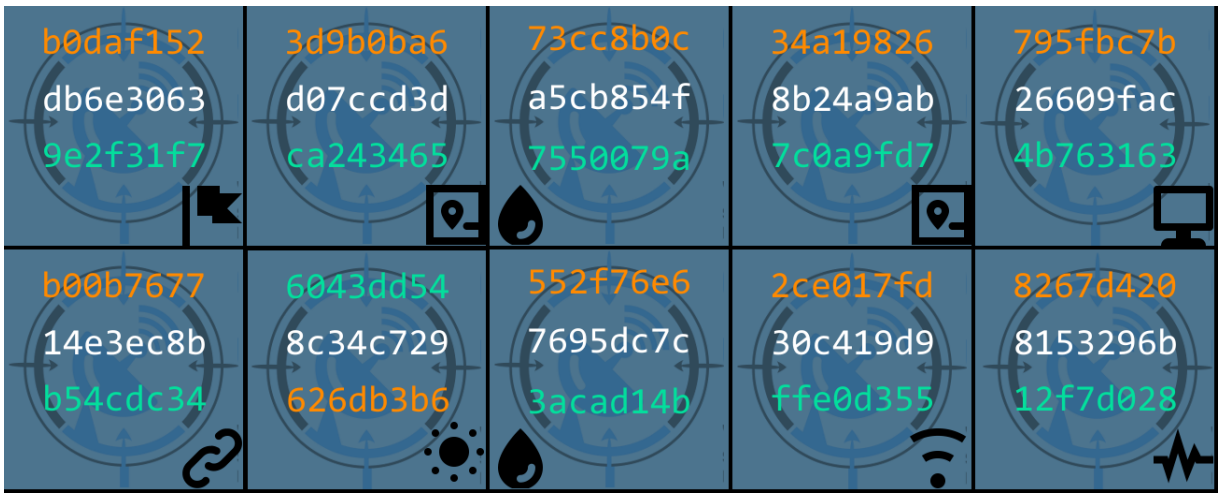


FIGURE 6: Textures restantes réelement affichées

Malheureusement il reste deux tuiles qui ont un doublon (pictogramme *goûte* et pictogramme *carré*), pour trouver la clé il faut donc tester les 4 combinaisons possibles.
Le script suivant réalise le test de ces 4 possibilités, le sha256 du fichier `decrypted` donnée dans l'énoncé de l'épreuve et utilisé pour identifier la clé correcte :

```
from Crypto.Cipher import AES
import hashlib

sha256_decrypted = "845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187"

key="9e2f31f78153296b%s%sb0daf152b54cdc34ffe0d35526609fac"
for carre_jaune in ("3d9b0ba6","34a19826"):
  for goute_blanche in ("a5cb853f","7695dc7c"):
    test_key = key % (carre_jaune,goute_blanche)

    data=open("encrypted","rb").read()
    iv="535354494332303135 2d537461676532".decode("hex")
    obj = AES.new(test_key.decode("hex"), AES.MODE_OFB, iv)
    dec = obj.decrypt(data)
    padlen = ord(dec[-1])
    dec = dec[:-padlen]
    sha = hashlib.sha256(dec).hexdigest()
    if sha == sha256_decrypted:
      print "KEY = "+test_key
```

Listing 14: Test des 4 possibilités restantes

```
$ ./decrypt-bf.py
KEY = 9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac
```

Listing 15: Résultat du test des 4 possibilités

2.3 Déchiffrement des données

Les informations présentes dans le fichier `memo.txt` ainsi que la clé reconstruite permettent de déchiffrer le fichier `encrypted` fourni au début de l'épreuve, le code python ci-dessous permet de réaliser ce déchiffrement :

```
from Crypto.Cipher import AES

data=open("encrypted","rb").read()
iv="535354494332303135 2d537461676532".decode("hex")
key="9E2F31F78153296B3D9B0BA67695DC7CB0DAF152B54CDC34FFE0D35526609FAC".decode("hex")
obj = AES.new(key, AES.MODE_OFB, iv)
dec = obj.decrypt(data)
padlen = ord(dec[-1])
dec = dec[:-padlen]
out=open("decrypted","wb")
out.write(dec)
```

Listing 16: Déchiffrement du fichier `encrypted` avec la clé retrouvée dans le jeu

Afin de valider le bon déchiffrement, le condensat sha256 présent dans `memo.txt` est vérifié :

```
$ shasum -a 256 decrypted \
    |grep -i 845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187
845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187  decrypted
```

Listing 17: Vérification d'intégrité du résultat

Le contrôle d'intégrité valide le bon déchiffrement, le fichier déchiffré est un fichier ZIP, c'est la prochaine épreuve de ce challenge :

```
$ file decrypted
decrypted: Zip archive data, at least v1.0 to extract
$ mv decrypted stage3.zip
```

Listing 18: Type du fichier déchiffré

3 Stage 3 - Souris USB et cryptographies exotiques

3.1 Découverte de l'épreuve

L'épreuve trois débute avec le fichier ZIP obtenu à l'étape précédente, ce fichier contient 3 fichiers :

```
$ bsdtar xvf stage3.zip
x encrypted
x memo.txt
x paint.cap
$ file encrypted memo.txt paint.cap
encrypted: data
memo.txt:  ASCII text
paint.cap: tcpdump capture file (little-endian) - version 2.4, capture length 262144)
```

Listing 19: Contenu de l'archive `stage3.zip`

Comme pour la précédente épreuve le but est de déchiffrer le fichier `encrypted`, le fichier `memo.txt` donne des informations sur l'épreuve :

- Le type de chiffrement utilisé pour le fichier `encrypted` : **Serpent-1-CBC-With-CTS**
- L'IV utilisé pour chiffrer le fichier `encrypted` : **0x535354494332303135 2d537461676533**
- Un indice indiquant que la clé de chiffrement a été stockée avec Paint.
- Les condensats SHA256 du fichier `encrypted` ainsi que le condensat du fichier déchiffré.

```
$ cat memo.txt
Cipher: Serpent-1-CBC-With-CTS
IV: 0x535354494332303135 2d537461676533
Key: Well, definitely can't remember it... So this time I securely stored it with Paint.

SHA256: 6b39ac2220e703a48b3de1e8365d9075297c0750e9e4302fc3492f98bdf3a0b0 - encrypted
SHA256: 7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2 - decrypted
```

Listing 20: Stage 3 : Contenu du fichier `memo.txt`

3.2 Analyse des échanges USB

Le fichier `paint.cap` est un fichier de capture *pcap*, il peut être ouvert dans Wireshark pour analyse :

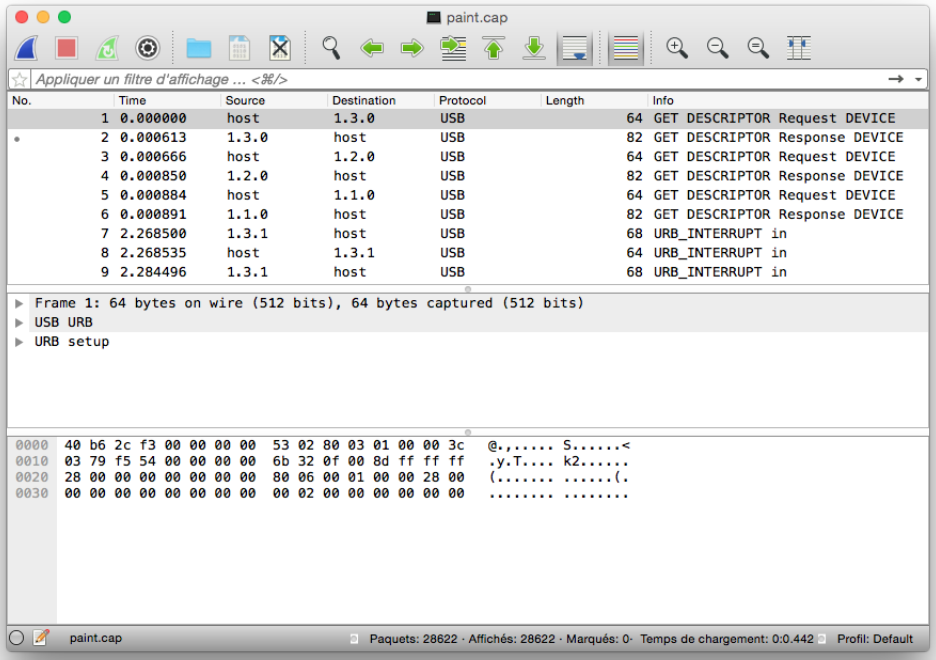


FIGURE 7: Ouverture du fichier `paint.cap` dans Wireshark

La réponse du périphérique à la première sollicitation de l'host (`GET DESCRIPTOR Request DEVICE`), montre qu'il s'agit d'une souris :

```
▶ USB URB
▼ DEVICE DESCRIPTOR
    bLength: 18
    bDescriptorType: 0x01 (DEVICE)
    bcdUSB: 0x0200
    bDeviceClass: Device (0x00)
    bDeviceSubClass: 0
    bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
    bMaxPacketSize0: 8
    idVendor: IBM Corp. (0x04b3)
    idProduct: Wheel Mouse (0x310c)
    bcdDevice: 0x0200
    iManufacturer: 0
    iProduct: 2
    iSerialNumber: 0
    bNumConfigurations: 1
```

FIGURE 8: Ouverture du fichier `paint.cap` dans Wireshark

Le fichier `memo.txt` parlait d'une clé dissimulée dans Paint, le but de l'épreuve est probablement de reconstruire les déplacements de la souris pour reconstruire un dessin.

Les mouvements de la souris sont dans les échanges de type "`URB_INTERRUPT in`", le filtre "`usb.request_in usb.transfer_type == URB_INTERRUPT`" permet de filtrer les échanges contenant les données de déplacement. Les interruptions contenant les mouvements de la souris sont dans le champ `usb.capdata`, les données peuvent être extraites simplement avec `tshark` :

```
$ tshark -r paint.cap -Y 'usb.request_in && usb.transfer_type == URB_INTERRUPT' -Tfields
    ↪ -e usb.capdata
00:ff:00:00
00:fe:00:00
00:ff:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
[...]
00:00:00:00
```

Listing 21: Extraction des mouvements de la souris

Le format de ces données est le suivant :

- octet 0 : positions des boutons
- octet 1 : delta en X (complément à 2)
- octet 2 : delta en Y (complément à 2)
- octet 3 : delta de la molette

3.3 Reconstruction de l'image à partir de la capture

Pour reconstruire l'image les 3 premiers octets sont utilisés, le vecteur delta_x,delta_y est utilisé pour déplacer un pointeur, et lorsque le bouton est appuyé une ligne est tracée entre l'ancienne et la nouvelle position du pointeur. Le script suivant permet de reconstruire l'image en utilisant les données extraites par `tshark` (stockées dans `usb_capdata`) :

```
import struct
from PIL import Image, ImageDraw

img_width=1000
img_height=800

pos_y=100
pos_x=50

im = Image.new('RGBA', (img_width, img_height), (255, 255, 255, 0))
draw = ImageDraw.Draw(im)

data=open("usb_capdata","rb")
for capdata_line in data:
  capdata=capdata_line.split(":")
  bouton=capdata[0]
  rel_x=struct.unpack('<b',capdata[1].decode("hex"))[0]
  rel_y=struct.unpack('<b',capdata[2].decode("hex"))[0]
  wheel=capdata[3]
  old_pos_x=pos_x
  old_pos_y=pos_y
  pos_x+=rel_x
  pos_y+=rel_y
  if bouton != '00':
    draw.line((old_pos_x,old_pos_y, pos_x,pos_y), fill=255)
im.show()
```

Listing 22: Reconstruction de l'image à partir des mouvement de souris

Le résultat obtenu est le suivant :

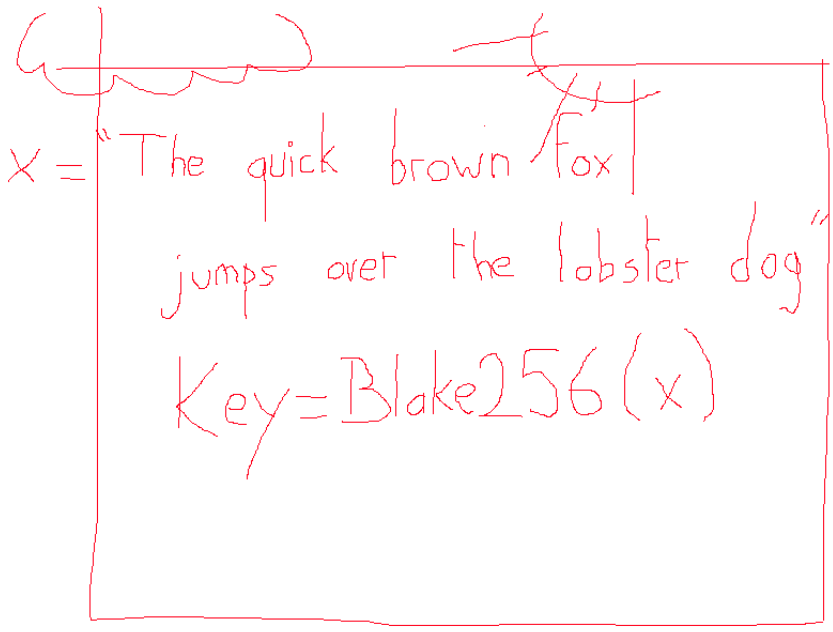


FIGURE 9: Image reconstruite à partir de la capture

L'image retrouvée contient la clé à utiliser pour déchiffrer les données : `key=Blake256("The quick brown fox jumps over the lobster dog")`. La suite de cette épreuve est d'utiliser des algorithmes cryptographiques peu courant : Blake256 pour obtenir la clé en générant le condensat du texte obtenu dans l'image et Serpent-1-CBC-With-CTS pour déchiffrer les données.

3.4 Blake256

BLAKE est un des finalistes dans la compétition pour le SHA-3, plusieurs implémentations existent dont une en python [2] qui a été utilisée ici.
Le script python suivant génère la clé comme indiqué dans l'image à l'étape précédente :

```
from blake import BLAKE

blake = BLAKE(256)
blake.update("The quick brown fox jumps over the lobster dog")
print blake.hexdigest()
```

Listing 23: Calcul de la clé avec l'algorithme Blake

```
$ ./blake-key.py
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d
```

Listing 24: Résultat du calcul de la clé

3.5 Serpent-1-CBC-With-CTS

Serpent-1 était un finaliste pour le standard AES [3].
Comme pour Blake plusieurs implémentations existent, c'est une implémentation en python qui a été utilisée [4]. En revanche cette implémentation ne fournit pas le mode *CipherText Stealing* (CTS) nécessaire au déchiffrement pour cette épreuve.
Ce mode a donc dû être implémenté, le script suivant implémente le mode CTS et réalise le déchiffrement :

```
import serpent
import Crypto.Cipher.XOR
import sys


def one_round(k,iv,ct):
    s = serpent.Serpent()
    s.set_key(k)
    x = Crypto.Cipher.XOR.new(iv)
    pt = x.decrypt(s.decrypt(ct))
    return pt

if __name__ == "__main__":
    k = "66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d".decode("hex")
    iv = 'SSTIC2015-Stage3'
    fi = open("encrypted", "rb").read()
    fo = open("decrypted", "wb")

    lc = []
    for i in xrange(0, len(fi)+1, 16):
        lc += [fi[i:i+16]]
    olc = len(lc)
    dn = one_round(k,"\x00"*16, lc[-2])
    last = lc[-1]
    if len(last) == 16:
        raise RuntimeError()
    last2 = last + dn[len(last):17]
    lc = lc[:-2] + [last2, lc[-2]]
    if len(lc) != olc:
        raise RuntimeError()
    for cyphtext in lc:
        pt = one_round(k,iv,cyphtext)
        iv = cyphtext
        fo.write(pt)
    fo.close()
```

Listing 25: Script de déchiffrement avec l'algorithme Serpent-1-CBC-With-CTS

2. http://www.seanet.com/ bugbee/crypto/blake/blake.py
3. http://csrc.nist.gov/archive/ipsec/papers/aes-draft.00.txt
4. http://www.bjrn.se/code/serpentpy.txt

```
$ ./serpent-decrypt.py
$ file decrypted
decrypted.zip: Zip archive data, at least v2.0 to extract
$ mv decrypted stage4.zip
```

Listing 26: Résultat du calcul de la clé

4 Stage 4 - Rétro Ingénierie de JavaScript obfusqué

4.1 Découverte de l'épreuve

L'épreuve débute avec l'archive ZIP obtenue à l'étape précédente, cette archive contient un seul fichier HTML :

```
$ bsdtar xvf stage4.zip
x stage4.html
```

Listing 27: Extraction de l'archive `stage4.zip`

Le fichier HTML contient un gros script Javascript obfusqué, une des variables est un gros bloc de données en hexadécimale d'environ 258 Ko, une autre est un condensat de 20 octets.

```
<html>
<head>
<style>
  * { font-family: Lucida Grande,Lucida Sans Unicode,Lucida
      ↪ Sans,Geneva,Verdana,sans-serif; text-align:center; }
  #status { font-size: 16px; margin: 20px; }
  #status a { color: green; }
  #status b { color: red; }
</style>
</head>
<body>
  <script>
    var data = "2b1f25cf8db5d243f59b065da6b56753b72e28f5[...]"
    var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
    $=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$:({}+"")[...]
  </script>
</body>
</html>
```

Listing 28: Contenu de `stage4.html`

Le but de l'épreuve est certainement de déchiffrer le contenu de la variable "data", pour comprendre le fonctionnement du script, l'obfuscation javascript doit être inversée.

4.2 Dé-obfuscation

La première approche a été de remplacer certains appels de fonctions par la fonction `console.log` pour tracer les paramètres :

```
[...]
$.$=($.___)[$.$_][$.$_];
console.log($.$($.$$+"\""+"__="+$.$$_~[...]
```

Listing 29: Modification du Javascript pour extraire des données

Cette méthode permet de trouver un code JavaScript moins obfusqué, uniquement les noms variables sont masqués :

```
[...]
___$__ = {
    name: 'SHA-1'
};
____$_ = data;
_____$ = hash;
_$_____ = Blob;
[...]
function _____(______) {
    _ = [];
    for (___________ = _______; ___________ < ______[____________]; ++___________)
        ↪ _[____________________](______[___________________](___________));
    return new _______________________(_);
}
[...]
```

Listing 30: Exemple de code javascript après premier niveau de dé-obfuscation

Après quelques recherches l'obfuscateur est identifié, il s'agit de `jjencode`, plusieurs outils permettant la dé-obfuscation du premier niveau ont été trouvés, en revanche aucun outil n'a été trouvé pour le second niveau d'obfuscation (dissimulation du nom des variables). La dé-obfuscation de ce second niveau est relativement simple, c'est une simple substitution du nom des variables, le script suivant effectue la dé-obfuscation de premier niveau avec `python-jjdecoder`, puis la dé-obfuscation de second niveau :

```
#!/usr/bin/python2
import re
# https://github.com/crackinglandia/python-jjdecoder/
from jjdecode import JJDecoder
# https://github.com/beautify-web/js-beautify
import jsbeautifier

if __name__ == '__main__':
  html=open("stage4.html","rb").read()
  js=html[html.index('$=~[];'):html.index('</script>')]

  # first decode using jjdecoder
  jjdecoded=JJDecoder(js).decode().replace(";",";\n")

  # get used variables
  used_var = re.findall('[$_]+',jjdecoded)
  variables={}
  for i,var in enumerate(used_var):
    variables[var]="var%03d" % i

  code=""

  # Get variables values, eval arithmetic values
  for line in jjdecoded.split("\n"):
    affectation = re.compile("^[$_]+=.*;")
    if affectation.match(line):
      this_var = line.split("=")
      key=this_var[0]
      value=this_var[1][:-1]
      invar=re.findall('[$_]+',value)
      invar.sort(key=len, reverse=True)
      for var in invar:
        value=value.replace(var,variables[var])
      op = re.compile("^[0-9+*/]+$")
      if op.match(value):
        value=str(eval(value))
      variables[key]=value
    else:
      code+=line+"\n"

  # Replace variables in code
  varlist = variables.keys()
  varlist.sort(key=len,reverse=True)
  for var in varlist:
    code=code.replace(var,variables[var])

  # Code beautifier
  code=re.sub(r"\['([a-zA-Z0-9]+)'\]",r".\1",code)
  code=code.replace("\n","")
  nice=jsbeautifier.beautify(code)
  nice=nice.replace("' + '","")
  nice=nice.replace("' + ","")
  nice=nice.replace(" + '","")
  nice=nice.replace("'++'","=")
  print nice
```

Listing 31: Script python assurant la dé-obfuscation Javascript

Le résultat obtenu est le suivant (après une légère simplification manuelle, pour améliorer la lisibilité du code) :

```
var228 = "raw";
document.write('<h1>Download manager</h1>');
document.write('<div id="status"><i>loading...</i></div>');
document.write('<div style="display:none"><a target="blank"
    ↪ href="chrome://browser/content/preferences/preferences.xul">Back to
    ↪ preferences</a></div>');

function var206(var194) {
    var171 = [];
    for (var184 = 0; var184 < var194.length; ++var184)
        ↪ var171.push(var194.charCodeAt(var184));
    return new Uint8Array(var171);
}

function var238(var194) {
    var171 = [];
    for (var184 = 0; var184 < var194.length / 2; ++var184)
        ↪ var171.push(parseInt(var194.substr(var184 * 2, 2), 16));
    return new Uint8Array(var171);
}

function var252(var183) {
    var194 = '';
    for (var184 = 0; var184 < var183.byteLength; ++var184) {
        'write' = var183[var184]['toString'](16);
        if ('write'.length < 2) var194 += 0;
        var194 += 'write';
    }
    return var194;
}

function var291() {
    userAgent = window.navigator['userAgent']
    key = var206(userAgent.substr(userAgent.indexOf(')') - 16, 16))
    iv = var206(userAgent.substr(userAgent.indexOf('(') + 1, 16));
    algo={}
    algo.name = 'AES-CBC';
    algo.iv = iv;
    algo.length = key.length * 8;
    window.crypto.subtle['importKey'](var228, key, algo, false,
        ↪ .decrypt).then(function(var237) {
        window.crypto.subtle.decrypt({}, var237, var238(data)).then(function(var244) {
            var261 = new Uint8Array(var244);
            window.crypto.subtle.digest({
                name: 'SHA-1'
            }, var261).then(function(var254) {
                if (new Blob([var261], var262) == var252(new Uint8Array(var254))) {
                    var262 = {};
                    var262.type = 'application/octet-stream';
                    var272 = URL.createObjectURL(new Blob([var261], var262));
                    document.getElementById('status').innerHTML = '<a href="var272"
                        ↪ download="stage5.zip">download stage5</a>';
                } else {
                    document.getElementById('status').innerHTML = '<b>Failed to load
                        ↪ stage5</b>';
                }
            });
        }).catch(function() {
            document.getElementById('status').innerHTML = '<b>Failed to load stage5</b>';
        });
    }).catch(function() {
        document.getElementById('status').innerHTML = '<b>Failed to load stage5</b>';
    });
}
window.setTimeout(var291, 1000);
```

Listing 32: Résultat de la dé-obfuscation

Maintenant que le code est dé-obfusqué, son analyse est très simple. Le code en question réalise les opérations suivantes :

- Création d'une clé et d'un vecteur d'initialisation (voir plus bas).
- Déchiffrement (AES-CBC-128) de la variable "`data`" (après conversion de l'hexadécimale) avec la librairie SubtleCrypto.
- Calcul du condensat SHA-1 des données déchiffrées
- Si le condensat est identique au condensat stocké dans la variable "`hash`" :
  - Ajout d'un lien sur la page pour télécharger les données déchiffrées sous le nom `stage5.zip`.
- Sinon :
  - Affichage du message d'erreur "Failed to load stage5"

4.3 Attaque par force brute des UserAgents

La clé ainsi que le vecteur d'initialisation sont les seules données d'entrée. L'IV et la clé sont obtenus par tronquage de la chaîne UserAgent (avec des offsets relatifs aux parenthèses) :

```
userAgent = window.navigator['userAgent']
key = var206(userAgent.substr(userAgent.indexOf(')') - 16, 16))
iv = var206(userAgent.substr(userAgent.indexOf('(') + 1, 16));
```

Listing 33: Génération de l'IV et de la clé

Par exemple pour le UserAgent "`Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.101 Safari/537.36`" la clé est "`11; Linux x86_64`" et l'IV est "`X11; Linux x86_6`".
Pour retrouver le bon IV et la bonne clé, il faut donc retrouver le bon UserAgent. Pour cela un indice est présent dans le code limitant les possibilités, l'URL chrome://browser/content/preferences/preferences.xul correspond à l'URL d'accès aux préférences de Firefox, il faut donc se concentrer sur les UserAgents Firefox.
La documentation de Mozilla "Gecko user agent string reference" permet de comprendre la structure des UserAgents Firefox. Il est ensuite simple de générer la liste des UserAgents possible.
Le script Python suivant génère la liste des UserAgent, extrait la clé et l'IV de ceux-ci et tente un déchiffrement, si le condensat est vérifié, c'est que la solution est trouvée.

```
#!/usr/bin/python
from Crypto.Cipher import AES
import hashlib
data = "2b1f25cf8db5d243[...]".decode("hex")
sha1 = "08c3be636f7dffd91971f65be4cec3c6d162cb1c"

windows_vers=[
  "6.3", # windows 8.1
  "6.2", # windows 8
  "6.1", # windows 7
  "6.0", # windows Vista
  "5.2", # windows 2003 server , XP 64
  "5.1", # windows XP
]

mac_vers=[ "10.4","10.5","10.6","10.7","10.8","10.9","10.10"]

rev_min = 10
rev_max = 37

UAs=[]

# Windows NT
for ver in windows_vers:
  for rev in range(rev_min,rev_max+1):
    UAs.append("(Windows NT %s; rv:%d.0)" % (ver,rev))
    UAs.append("(Windows NT %s; Win64; x64; rv:%d.0)" % (ver,rev))
    UAs.append("(Windows NT %s; WOW64; rv:%d.0)" % (ver,rev))
# Macintosh
for ver in mac_vers:
  for rev in range(rev_min,rev_max+1):
    UAs.append("(Macintosh; Intel Mac OS X %s; rv:%d.0)" % (ver,rev))
    UAs.append("(Macintosh; PPC Mac OS X %s; rv:%d.0)" % (ver,rev))
# Linux
for rev in range(rev_min,rev_max+1):
```

```
  UAs.append("(X11; Linux i686; rv:%d.0)" % (rev))
  UAs.append("(X11; Linux x86_64; rv:%d.0)" % (rev))
  UAs.append("(X11; Linux i686 on x86_64; rv:%d.0)" % (rev))

for userAgent in UAs:
  iv=userAgent[userAgent.index('(')+1:userAgent.index('(')+1+16]
  key=userAgent[userAgent.index(')')-16:userAgent.index(')')]
  obj = AES.new(key, AES.MODE_CBC, iv)
  dec = obj.decrypt(data)
  padlen = ord(dec[-1])
  dec = dec[:-padlen]
  sha1_dec = hashlib.sha1(dec).hexdigest()
  if sha1_dec == sha1:
    print "UA  : '"+userAgent+"'"
    print "Key : '"+key+"'"
    print "IV  : '"+iv+"'"
    stage5 = open("stage5.zip","wb")
    stage5.write(dec)
    stage5.close()
    break
```

Listing 34: Bruteforce des UserAgents

Grâce à ce script, la solution est trouvée en quelques secondes :

```
$ ./bf_ua.py
UA  : '(Macintosh; Intel Mac OS X 10.6; rv:35.0)'
Key : ' X 10.6; rv:35.0'
IV  : 'Macintosh; Intel'

$ file stage5.zip
stage5.zip: Zip archive data, at least v2.0 to extract
```

Listing 35: Solution du stage4

L'archive `stage5.zip` est donc l'épreuve suivante de ce challenge.

5 Stage 5 - Rétro ingénierie ST20 - Transputer

5.1 Découverte de l'épreuve

Le stage 5 débute avec l'archive `stage5.zip` déchiffrée à la fin de la précédente épreuve :

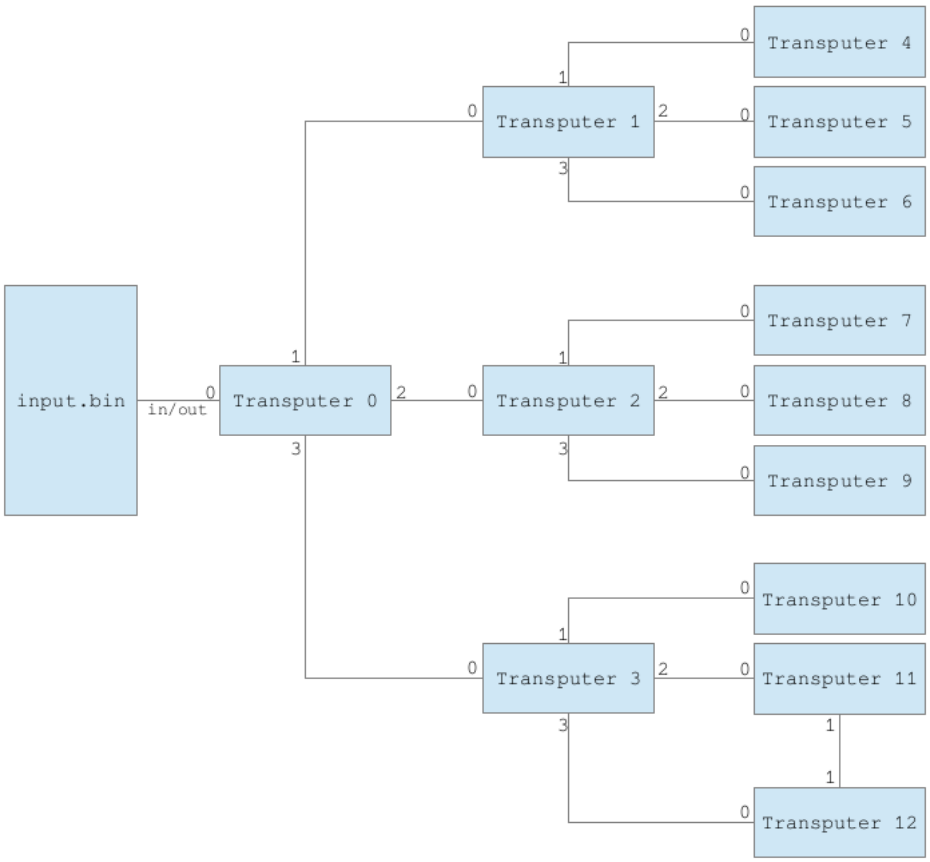
```
$ bsdtar xvf stage5.zip
x input.bin
x schematic.pdf

$ file input.bin schematic.pdf
input.bin:     data
schematic.pdf: PDF document, version 1.4
```

Listing 36: Extraction de l'archive `stage5.zip`

Dans l'archive deux fichiers sont présents, le premier (`input.bin`) est d'un format non identifié, il contient entre autres les chaînes de caractères suivantes :

- `Boot ok`
- `Code ok`
- `Decrypt`
- `KEY:`
- `congratulations.tar.bz2`

Le second (`schematic.pdf`) est le fichier PDF suivant :




```
SHA256:
    a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81 - encrypted
    9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321 - decrypted

Test vector:
    key = "*SSTIC-2015*"
    data = "1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e".decode("hex")
    decrypt(key, data) == "I love ST20 architecture"
```

FIGURE 10: Contenu du fichier `schematic.pdf`

Ce fichier contient plusieurs informations :

— Les condensats SHA256 des fichiers `encrypted` et `decrypted` (non fournis dans l'archive).
— Les données d'un vecteur de test (Clé, données chiffrées, et résultat déchiffré).

Le résultat déchiffré du vecteur de test donne un indice pour l'épreuve : "I love ST20 architecture".

5.2 Transputer - architecture ST20

Cette épreuve va donc se concentrer sur les *Transputers*, dans l'architecture ST20. Les *Transputers* sont des microcontrôleurs qui peuvent être mis en réseau via des liens série (appelés *link* dans la suite de ce document). Le réseau dans cette épreuve est composé de 12 *Transputers*. Chaque *Transputeur* peut avoir jusqu'à 4 *links* half-duplex.
Le ST20 est une architecture 32 bits, son jeu d'instruction est décrit dans la documentation "ST20C2/C4 Core Instruction Set Reference Manual" [5].
Le schéma dans le PDF fourni comme énoncé de l'épreuve montre que le fichier .bin est "connecté" au link 0 du Transputer 0. La documentation du ST20-GP1 [6] décrit de boot depuis un link :

> When booting from a link, the ST20-GP1 will wait for the first bootstrap message to arrive on the link. The first byte received down the link is the control byte. If the control byte is greater than 1 (i.e. 2 to 255), it is taken as the length in bytes of the boot code to be loaded down the link. The bytes following the control byte are then placed in internal memory starting at location MemStart. Following reception of the last byte the ST20-GP1 will start executing code at MemStart. The memory space immediately above the loaded code is used as work space. A byte arriving on the bootstrapping link after the last bootstrap byte, is retained and no acknowledge is sent until a process inputs from the link.

Le premier octet du fichier `input.bin` correspond donc à la taille du code de boot, les octets suivants correspondent au code du boot.
Les *links* 0 des autres *Transputers* sont raccordés à un *Transputer* en amont. Le code de boot des *Transputers* 1 à 12 est donc certainement envoyé par le *Transputer* en amont.
Deux stratégies sont possibles à ce point de l'analyse :

— Effectuer la rétro-ingénierie du code présent dans le fichier `input.bin` et ce pour chacun des *Transputers* (après avoir identifié le code transitant sur les différents links).
— Mettre en place un émulateur de ST20, relier plusieurs instances de cet émulateur et les connecter ensemble (simulation des *links*). Ensuite si besoin une rétro-ingénierie du code de chacun des *Transputers* (post boot) sera effectuée.

La seconde stratégie a été prise, elle permet de s'assurer du bon fonctionnement de l'ensemble (avec le vecteur de test), puis d'effectuer uniquement la rétro-ingénierie du code utile (hors boot). De plus cette stratégie permet d'intégrer ensuite une simulation pour chacun des *Transputers* en s'assurant du fonctionnement à chaque étape. En revanche cette stratégie consomme certainement plus de temps qu'une rétro-ingénierie directe.

5. http://www.transputer.net/iset/pdf/st20core.pdf
6. http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/4942.pdf

5.3 Implémentation d'un émulateur pour l'architecture ST20

Plusieurs émulateurs d'architecture ST20 existent [7], cependant aucun ne semble fonctionner correctement avec le fichier `input.bin`. L'architecture ST20 n'étant pas très complexe, un émulateur complet a été développé. Son code étant trop volumineux il est disponible en annexe (cf. 9.2).
Les *links* entre les différentes instances de l'émulateur sont réalisés avec des tubes nommés (un pour chaque direction).
L'architecture complète construite avec des émulateurs est la suivante :

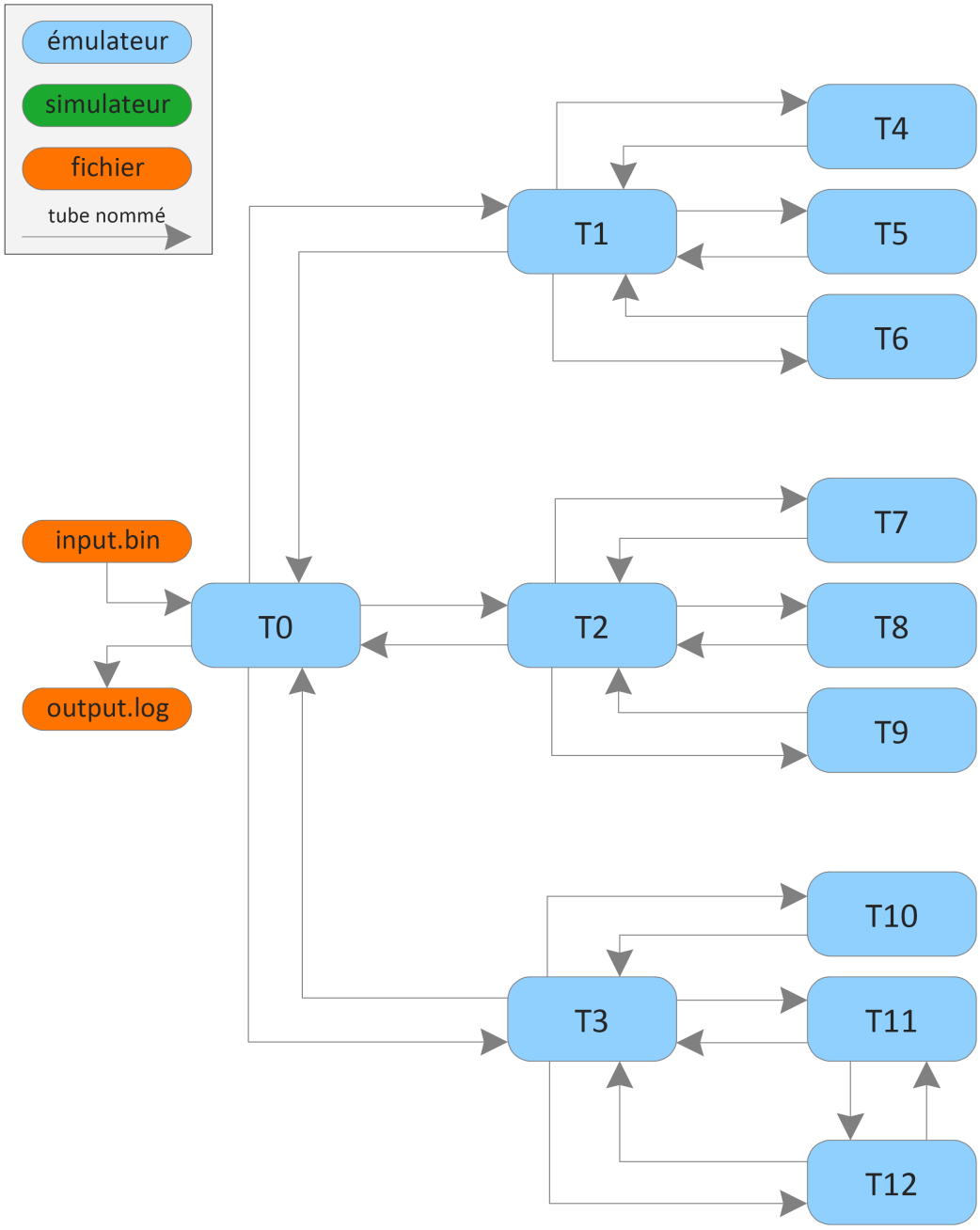


FIGURE 11: Architecture complète à base d'émulateurs

7. https://sites.google.com/site/transputeremulator/

5.4 Compréhension du système

5.4.1 Boot de l'architecture complète

L'architecture émulée permet de mieux comprendre le système. Le *Transputer* 0 boot en premier sur fichier `input.bin`. Le code du boot écrit la chaîne "Boot OK" sur le *link* 0 (raccordé sur le fichier `output.log` dans l'émulateur).
Le code de Boot lit ensuite le code de boot des *Transputers* 1,2 et 3 dans le fichier `input.bin` puis les envoie sur les *links* correspondants.
Les *Transputers* 1, 2 et 3 démarrent donc sur le code reçu sur leur *link* 0.
La même étape est réalisée pour démarrer les *Transputers* du dernier niveau (T0 envoie le code à T1 qui envoie le code à T4).

5.4.2 Initialisation de l'algorithme

L'étude des échanges permet de comprendre les champs dans le fichier `input.bin`, cela permet de construire le vecteur de test (après le code des différents *Transputers* dans le fichier) :

```
           0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
[...]
00000980  00 00 00 00 00 4b 45 59  3a 2a 53 53 54 49 43 2d  |.....KEY:*SSTIC-|
00000990  32 30 31 35 2a 17 74 65  73 74 76 65 63 74 6f 72  |2015*.testvector|
000009a0  2d 6f 75 74 70 75 74 2e  62 69 6e 61 72 1d 87 c4  |-output.binar...|
000009b0  c4 e0 ee 40 38 3c 59 44  7f 23 79 8d 9f ef e7 4f  |...@8<YD.#y....O|
000009c0  b8 24 80 76 6e                                    |.$.vn|
```

Listing 37: Format du vecteur de test

La clé de 12 octets est présente à l'offset 0x989, le contenu de `encrypted` est présent à l'offset 0x9ad après une chaîne contenant le nom de fichier (non utilisé).
L'ensemble des *Transputers* est finalement amorcé sur le code final, chaque bloc réalise ensuite des opérations différentes. La clé est ensuite partagée à l'ensemble des *Transputers* :

```
<<<< read (vectest.bin) ask:12 (sofar 2453) bytes data ['*SSTIC-2015*']
>>>> write (pipes/pipe_t0_1_to_t1_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t0_2_to_t2_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t0_3_to_t3_0) '*SSTIC-2015*'
<<<< read (pipes/pipe_t0_1_to_t1_0) ask:12 (sofar 620) bytes data ['*SSTIC-2015*']
>>>> write (pipes/pipe_t1_1_to_t4_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t1_2_to_t5_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t1_3_to_t6_0) '*SSTIC-2015*'
<<<< read (pipes/pipe_t0_2_to_t2_0) ask:12 (sofar 660) bytes data ['*SSTIC-2015*']
>>>> write (pipes/pipe_t2_1_to_t7_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t2_2_to_t8_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t2_3_to_t9_0) '*SSTIC-2015*'
<<<< read (pipes/pipe_t0_3_to_t3_0) ask:12 (sofar 716) bytes data ['*SSTIC-2015*']
>>>> write (pipes/pipe_t3_1_to_t10_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t3_2_to_t11_0) '*SSTIC-2015*'
>>>> write (pipes/pipe_t3_3_to_t12_0) '*SSTIC-2015*'
<<<< read (pipes/pipe_t1_1_to_t4_0) ask:12 (sofar 129) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t1_2_to_t5_0) ask:12 (sofar 129) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t3_1_to_t10_0) ask:12 (sofar 201) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t2_2_to_t8_0) ask:12 (sofar 205) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t2_1_to_t7_0) ask:12 (sofar 149) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t2_3_to_t9_0) ask:12 (sofar 133) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t1_3_to_t6_0) ask:12 (sofar 189) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t3_2_to_t11_0) ask:12 (sofar 161) bytes data ['*SSTIC-2015*']
<<<< read (pipes/pipe_t3_3_to_t12_0) ask:12 (sofar 181) bytes data ['*SSTIC-2015*']
```

Listing 38: Écritures et lectures sur les tubes de la clé

S'en suit un grand nombre d'échanges entre les *Transputers*. Le résultat des opérations est ensuite écrit sur le *link* 0 du *Transputer* 0 (`output.log`). À l'issue des opérations le vecteur de test est validé :

```
$ hexdump -C output.log
00000000  42 6f 6f 74 20 6f 6b 00  43 6f 64 65 20 4f 6b 00  |Boot ok.Code Ok.|
00000010  44 65 63 72 79 70 74 84  49 20 6c 6f 76 65 20 53  |Decrypt.I love S|
00000020  54 32 30 20 61 72 63 68  69 74 65 63 74 75 72 65  |T20 architecture|
```

Listing 39: Sortie du vecteur de test (*link* 0 du *Transputer* 0)

5.5 Rétro-ingénierie de l'assembleur ST20

L'émulation de l'architecture complète permet d'extraire le code final de chaque *Transputer*.
Le code final de chacun des nœuds est analysé individuellement, pour chaque un pseudo code est écrit (pour implémentation dans le futur simulateur).
Le code est chargé dans IDA Pro (type de processeur SGS-Thomson ST20/C2-C4) pour analyse.
Le code de T4 est un bon exemple, c'est une fonction "ADD" avec accumulateur.

```
entry_point:
        var1 = 0
        var1[0] = 0
mainloop:
    while () {
        var0 = 0xC (12)
        read(var6, link0in, &var2)
        var0 = 0
loc_1F:
    while(var0 <= 0xC) {
        var1[0] = (var1+var2[var0]) & 0xFF
        var0+=1
    }
loc_36:
        var0 = 1
        write(var6, MostNeg(link0out), &var1)
}
```

Listing 40: pseudo code T4

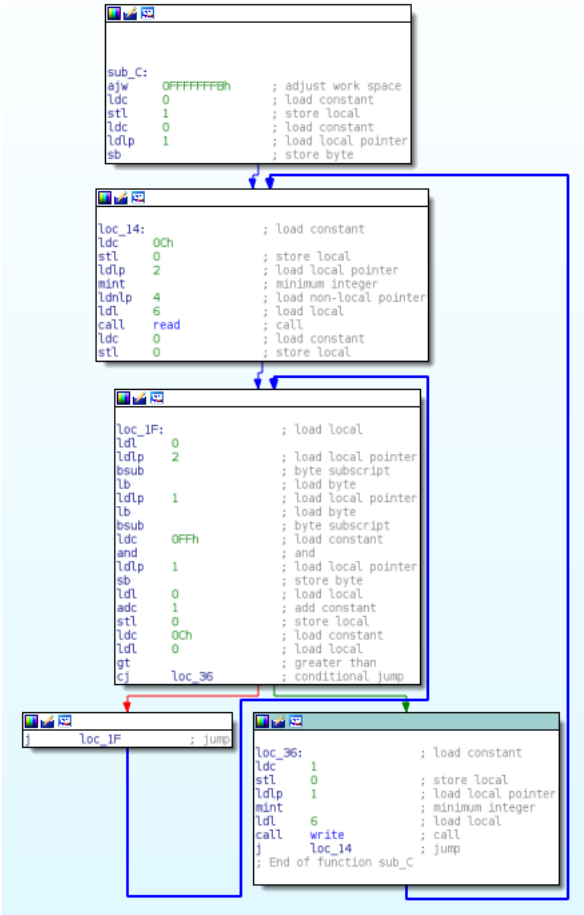


FIGURE 12: Code final de T4 dans IDA Pro

L'analyse de ces nœuds permet de les classer en 4 catégories :

- T0 : Assure le déchiffrement, envoie et reçoit des données depuis T1, T2 et T3.
- T4-T10 : Calculs simples (exemple : XOR avec accumulateur).
- T11-T12 : Calculs partagés sur deux nœuds.
- T1-3 : Agrégation des résultats de calculs simples.

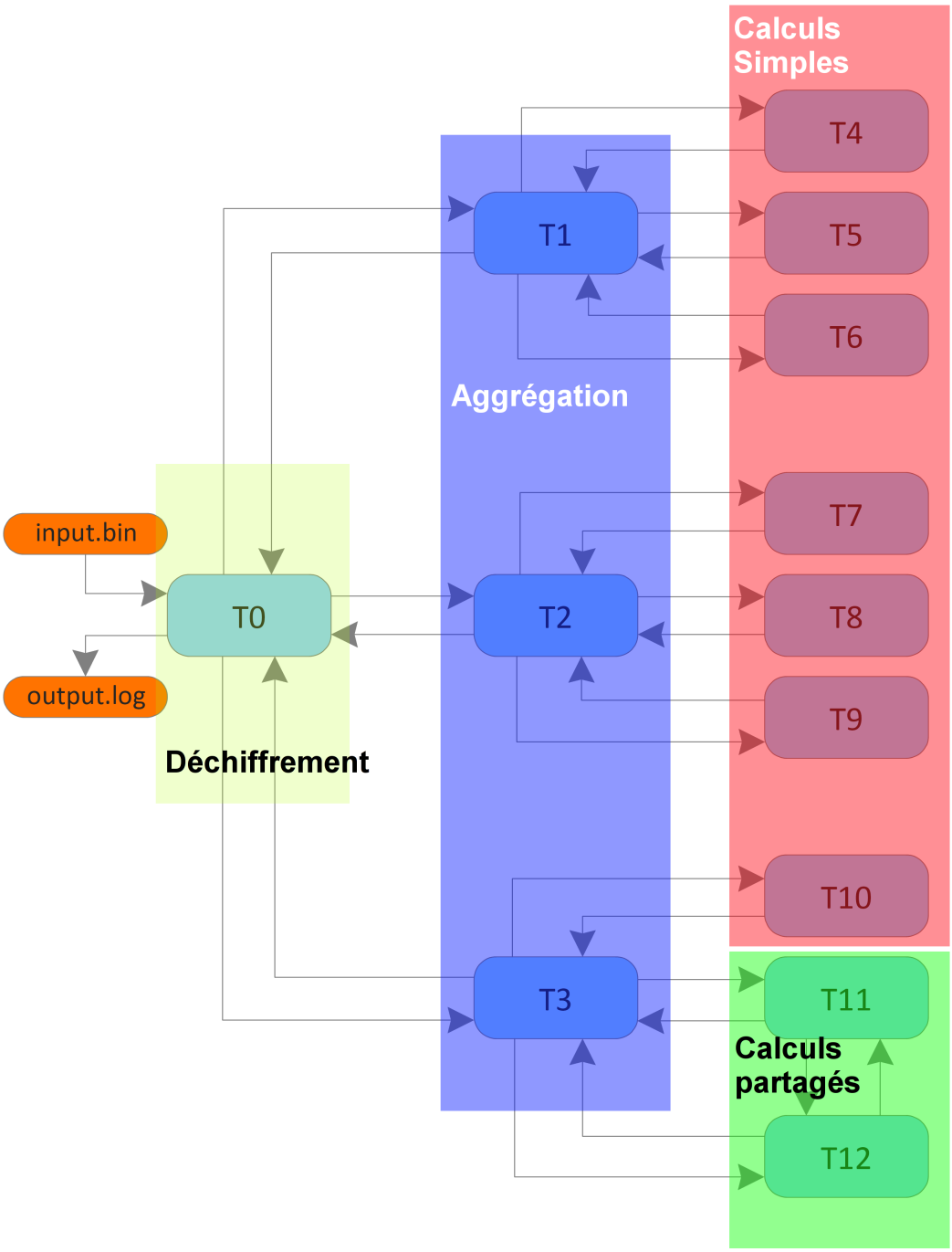


FIGURE 13: Répartition des calculs

5.6 Faiblesse dans l'algorithme

La rétro-ingénierie montre qu'une seule opération est réalisée sur le premier bloc de 12 octets, c'est un simple XOR (valeurs initiales des accumulateurs connues) :

```
clear_byte = (encrypted_byte ^ (offset + 2 * KEY[offset])) & 0xFF
```

Listing 41: Opération sur le premier bloc

Si la valeur déchiffrée du premier bloc est connue, il est possible de générer la liste des clés produisant ce premier bloc (4096 possibilités, car deux valeurs possibles pour chaque octet).

5.7 Simulation

L'émulateur ST20 développé à l'occasion de ce challenge est très peu performant, avec le fichier `input.bin` il faut 6 minutes 30 sur un Core i7 pour avoir le résultat. Pour pouvoir mener une attaque par force brute il faut un simulateur de plus performant.
La rétro-ingénierie de chaque nœud permet d'écrire un simulateur réalisant les mêmes opérations que le code ST20 dans l'émulateur. L'architecture décomposée en blocs connectés par des tubes nommés permet de remplacer des émulateurs par des simulateurs bloc par bloc. À chaque remplacement d'un émulateur par un simulateur le vecteur de test est rejoué pour valider le fonctionnement du simulateur.

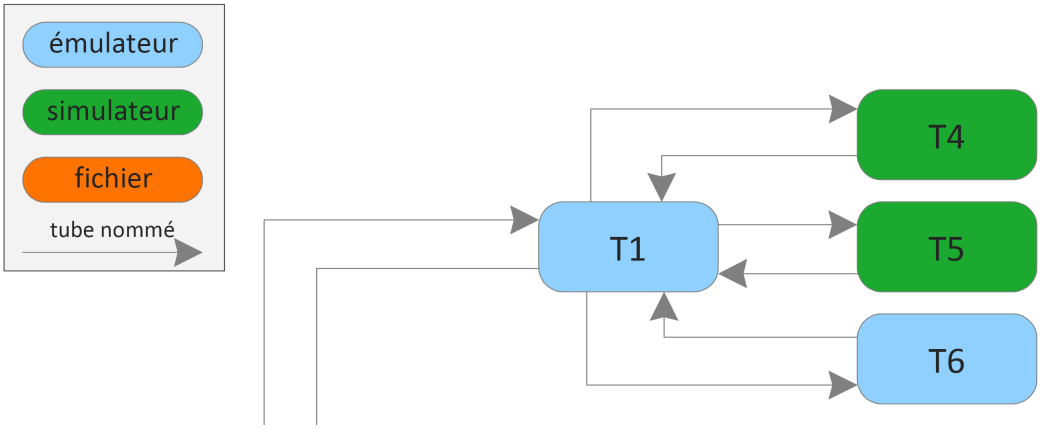


FIGURE 14: Simulation de T4 et T5

Lorsque tous les simulateurs d'un niveau sont fonctionnels, ils sont intégrés dans le niveau supérieur, la encore à chaque intégration le vecteur de test est rejoué pour vérifier que le simulateur est correct.

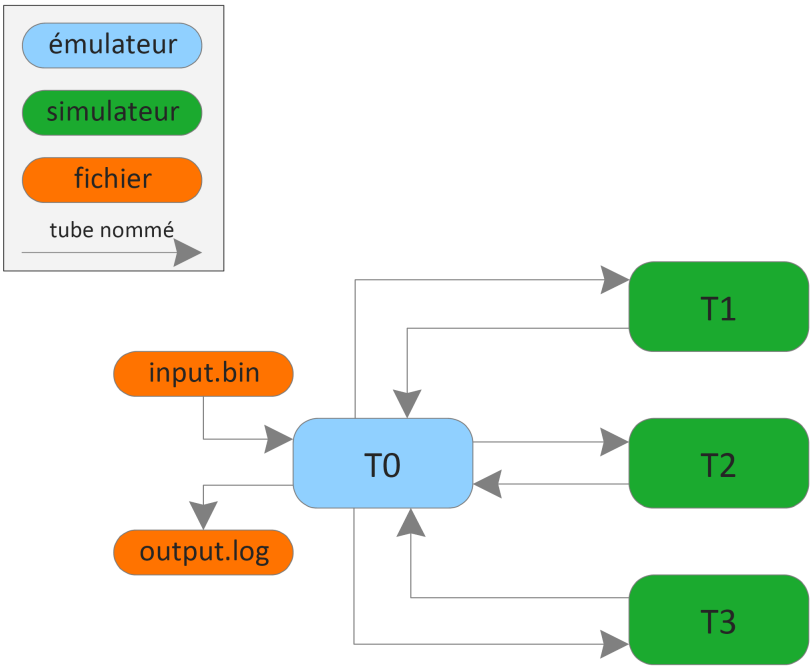


FIGURE 15: Simulation de T1, T2 et T3

Pour finir les simulateurs T1, T2 et T3 sont intégrés dans un simulateur pour T0, le code Python de ce simulateur est disponible en annexe 9.3.
Le simulateur complet est beaucoup plus performant que les émulateurs interconnectés. En émulation le fichier `input.bin` nécessite 6 minutes 30 sur un Core i7 alors que la simulation ne nécessite que 0,6 seconde. Même si le simulateur prend beaucoup moins de temps, c'est encore trop pour faire une attaque par force brute efficace. Il faut donc trouver un raccourci.

5.8 Attaque par force brute

Le nom du fichier de sortie est présent dans le fichier `input.bin` : "`congratulations.tar.bz2`".
La faiblesse de l'algorithme permet de réduire les possibilités de clé générant une en-tête Bzip2 à 2^{26}, c'est encore trop de possibilités pour faire une attaque en force brute efficace.
La liste des clés générant une en-entête Bzip2 peut être calculée de la manière suivante (10 octets avec deux possibilités, puis 2 octets sans condition) :

```
# encrypted[:12]
dd="fef350dc81bc972789ac7228cb50a409d31817".decode("hex")[:12]
search="BZh91AY&SY" # Bzip2 header
keyl=[0]*12
for i,c in enumerate(search):
  for j in range(0x100):
    if ord(dd[i])^((i+2*j))&0xFF == ord(c):
      keyl[i]=j
      break

for k in range(2**10):
  key = list(keyl)
  for pos in range(10):
    if (k >> pos) & 1:
      key[pos] |= 0x80

  for i in range(0x100):
    for j in range(0x100):
      key[10]=i
      key[11]=j
      print "".join(map(chr,key)).encode("hex")
```

Listing 42: Génération de la liste des clés

Le fichier complet ne peut pas être utilisé pour l'attaque par force brute, car le simulateur prend trop de temps à le traiter. Les derniers octets de l'en-tête Bzip2 ne peuvent pas être identifiés, il faut donc

trouver une autre condition d'arrêt. Une analyse rapide d'en-tête Bzip2 permet de trouver une condition d'arrêt, celui-ci contient dans la plupart des cas des octets à 0xFF :

```
$ dd if=/dev/urandom bs=1 count=$RANDOM 2>/dev/null|bzip2 |hexdump -C |head -n2
00000000  42 5a 68 39 31 41 59 26  53 59 a2 4f 69 77 00 05  |BZh91AY&SY.Oiw..|
00000010  ce 7f ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
$ dd if=/dev/urandom bs=1 count=$RANDOM 2>/dev/null|bzip2 |hexdump -C |head -n2
00000000  42 5a 68 39 31 41 59 26  53 59 e1 1d f5 75 00 08  |BZh91AY&SY...u..|
00000010  80 7f ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
$ dd if=/dev/urandom bs=1 count=$RANDOM 2>/dev/null|bzip2 |hexdump -C |head -n2
00000000  42 5a 68 39 31 41 59 26  53 59 36 59 f1 d5 00 00  |BZh91AY&SY6Y....|
00000010  d2 ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
$ dd if=/dev/urandom bs=1 count=$RANDOM 2>/dev/null|bzip2 |hexdump -C |head -n2
00000000  42 5a 68 39 31 41 59 26  53 59 d0 1c 69 33 00 0b  |BZh91AY&SY..i3..|
00000010  62 ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |b...............|
```

Listing 43: Second bloc de l'en-tête Bzip2

La condition d'arrêt retenue est de trouver au minimum 5 octets à 0xFF dans les deux premiers blocs. Les 24 premiers octets du fichier `encrypted` sont écrits dans le fichier de test :

```
$ dd if=encrypted of=first-two-blocks.bin bs=1 count=24
```

Listing 44: Creation du fichier de test

Le script suivant réalise l'attaque par force brute :

```
import sys
# simulateur T0, voir annexe
from t0 import Streamgen

def mainloop(initkey, fin):
    ks = map(ord,initkey)
    s = Streamgen()
    it = 0
    doit = True
    ff=0
    while doit:
        b = fin.read(1)
        if len(b) == 0:
            doit = False
        else:
            x = s.getbyte(map(chr,ks))
            clear = (ord(b) ^ (it+2*ks[it])) & 0xff
            ks[it] = x
            it += 1
            it %= 12
            if clear == 255:
                ff+=1
            # fout.write(chr(clear))
    return ff

if __name__ == "__main__":
    dd="fef350dc81bc972789ac7228cb50a409d31817".decode("hex")[:12]
    search="BZh91AY&SY"
    keyl=[0]*12
    for i,c in enumerate(search):
        for j in range(0x100):
            if ord(dd[i])^((i+2*j))&0xFF == ord(c):
                keyl[i]=j
                break

    file_2blocks=open("first-two-blocks.bin","rb")
    start = (int(sys.argv[1])*(2**10)) / 100

    for k in range(start, 2**10):
        key = list(keyl)
        for pos in range(10):
            if (k >> pos) & 1:
                key[pos] |= 0x80

        for i in range(0x100):
            for j in range(0x100):
                key[10]=i
                key[11]=j
                chr_key=map(chr,key)
                ff=mainloop(chr_key, file_2blocks)
                if ff > 4:
                    print "".join(map(chr,key)).encode("hex")
                    sys.exit(0)
                file_2blocks.seek(0)
```

Listing 45: Attaque par force brute

tL'attaque est lancé sur 10 processus (sur un Core i7-4700HQ), le résultat est trouvé en 3 minutes.

```
$ pypy brute.py 30
5ed49b7156fce47de976dac5
```

Listing 46: Résultat de l'attaque par force brute

La clé trouvée par force brute permet de déchiffrer le fichier encrypted dans sont intégralité, le sha256 du fichier `decrypted` est identique à celui présent dans l'énoncé (PDF).

6 Stage 6 - Stéganographies diverses

6.1 Découverte de l'épreuve

L'épreuve débute avec l'archive `congratulations.tar.bz2` trouvée à l'épreuve précédente.

```
$ bsdtar xvf congratulations.tar.bz2
x congratulations.jpg
```

Listing 47: Contenu de l'archive `congratulations.tar.bz2`

Le l'archive contient le l'image suivante au format `jpeg` :

FIGURE 16: congratulations.jpg

6.2 Archive ajoutée à la fin de l'image

Le texte "... un dernier petit effort ?", montre que le challenge n'est pas fini, une nouvelle phase de reconnaissance débute. Un moyen simple de dissimuler du contenu dans une image est d'ajouter des données après celle-ci (si le format le permet, c'est le cas de JPEG). La première idée est donc de parcourir les données à la recherche d'une autre signature de fichier.
L'outil `binwalk` permet d'effectuer cette recherche :

```
$ binwalk congratulations.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
55248         0xD7D0          bzip2 compressed data, block size = 900k
```

Listing 48: Recherche de signatures avec l'outil `binwalk` dans `congratulations.jpg`

Une signature d'une archive Bzip2 est identifiée à l'offset 55248 du fichier, cette archive peut être extraite :

```
$ dd if=congratulations.jpg of=step6.2.tar.bz2 bs=1 skip=55248
$ bsdtar xvf step6.2.tar.bz2
x congratulations.png
```

Listing 49: Extraction de l'archive contenue dans `congratulations.jpg`

L'image suivante est présente dans l'archive extraite :

FIGURE 17: congratulations.png

6.3 Flux Zlib caché dans des Chunk PNG

Ici encore le texte "... deux derniers petits efforts ?", montre que le challenge n'est pas encore terminé. L'analyse du fichier `congratulations.png` avec la méthode précédente ne donne aucun résultat. Pour comprendre plus en détail, l'image est analysée avec l'outil `pngcheck`. Cet outil permet de décoder le format de fichier PNG, la structure du fichier peut être affiché avec la commande suivante :

```
$ pngcheck -vtf congratulations.png
File: congratulations.png (197557 bytes)
  chunk IHDR at offset 0x0000c, length 13
    636 x 474 image, 32-bit RGB+alpha, non-interlaced
  chunk bKGD at offset 0x00025, length 6
    red = 0x00ff, green = 0x00ff, blue = 0x00ff
  chunk pHYs at offset 0x00037, length 9: 3543x3543 pixels/meter (90 dpi)
  chunk tIME at offset 0x0004c, length 7: 27 Feb 2015 13:40:19 UTC
  chunk sTic at offset 0x0005f, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x013a2, length 4919:  illegal reserved-bit-set chunk
  [...]
  chunk sTic at offset 0x1f52d, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x20870, length 38:  illegal reserved-bit-set chunk
  chunk IDAT at offset 0x208a2, length 8192
    zlib: deflated, 32K window, maximum compression
  chunk IDAT at offset 0x228ae, length 8192
  chunk IDAT at offset 0x248ba, length 8192
  chunk IDAT at offset 0x268c6, length 8192
  chunk IDAT at offset 0x288d2, length 8192
  chunk IDAT at offset 0x2a8de, length 8192
  chunk IDAT at offset 0x2c8ea, length 8192
  chunk IDAT at offset 0x2e8f6, length 6827
  chunk IEND at offset 0x303ad, length 0
ERRORS DETECTED in congratulations.png
```

Listing 50: Structure de l'image `congratulations.png`

L'analyse du format du fichier permet de découvrir des "chunks" avec un tag invalide : `sTic`. C'est certainement dans ces parties du fichier que se cache la solution. Le premier "chunk" avec le tag `sTic` commence avec un en-tête ZLIB.
Le script suivant permet de concaténer les "chunks" avec le tag `sTic`, pour cela la librairie hachoir-parser est utilisé, le résultat est ensuite décompressé avec la libraire `zlib` :

```
from hachoir_core.cmd_line import unicodeFilename
from hachoir_parser import createParser
import zlib

filename = "congratulations.png"
filename, realname = unicodeFilename(filename), filename
parser = createParser(filename)

out=""
for field in parser:
    if "chunk" in field.name:
      if field.getField("tag").value == "sTic":
        out+=field[2].value

open("stage6.3.bin","wb").write(zlib.decompress(out))
```

Listing 51: Extraction des "chunks" avec le tag `sTic`

Le fichier obtenu est de nouveau une archive Bzip2 contenant une nouvelle image (au format TIFF cette fois-ci) :

```
$ file stage6.3.bin
stage6.3.bin: bzip2 compressed data, block size = 900
$ mv stage6.3.bin stage6.3.tar.bz2
$ bsdtar xvf stage6.3.tar.bz2
x congratulations.tiff
```

Listing 52: Extraction des "chunks" avec le tag `sTic`

L'image suivante est présente dans l'archive extraite :

FIGURE 18: congratulations.tiff

6.4 Données dissimulées dans les bits de poids faible

Ici encore le texte "... trois derniers petits efforts ?", annonce que le challenge n'est pas encore terminé. Les méthodes précédentes ne révèlent pas la présence de données cachées. Pour résoudre des énigmes de stéganographie, l'outil `stegsolve` est particulièrement efficace lorsque les données sont dissimulées dans les données utiles de l'image.
Cet outil permet de naviguer dans les couches de couleur, pour chacune des couches de couleur il permet de faire ressortir les bits de la composante, sur cette image le bit de poids faible des couches rouge et vert semble contenir des données cachées :

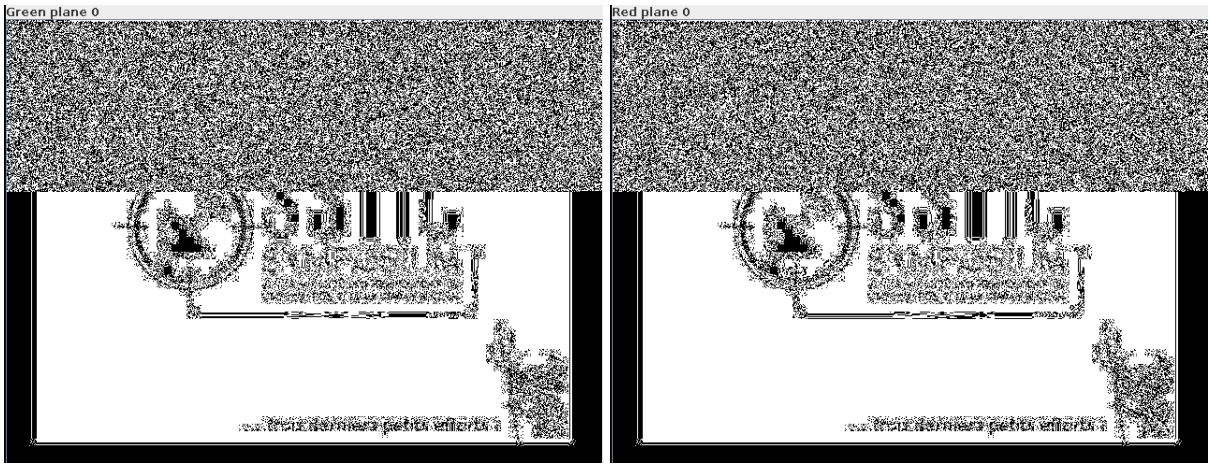


FIGURE 19: visualisation avec `stegsolve` des bits de poids faible des composantes rouge et verte

En combinant les bits de poids faible des couches rouge et vert (toujours avec `stegsolve`), on obtient une nouvelle archive Bzip2 (sauvegardée dans stage6.4.tar.bz2) :

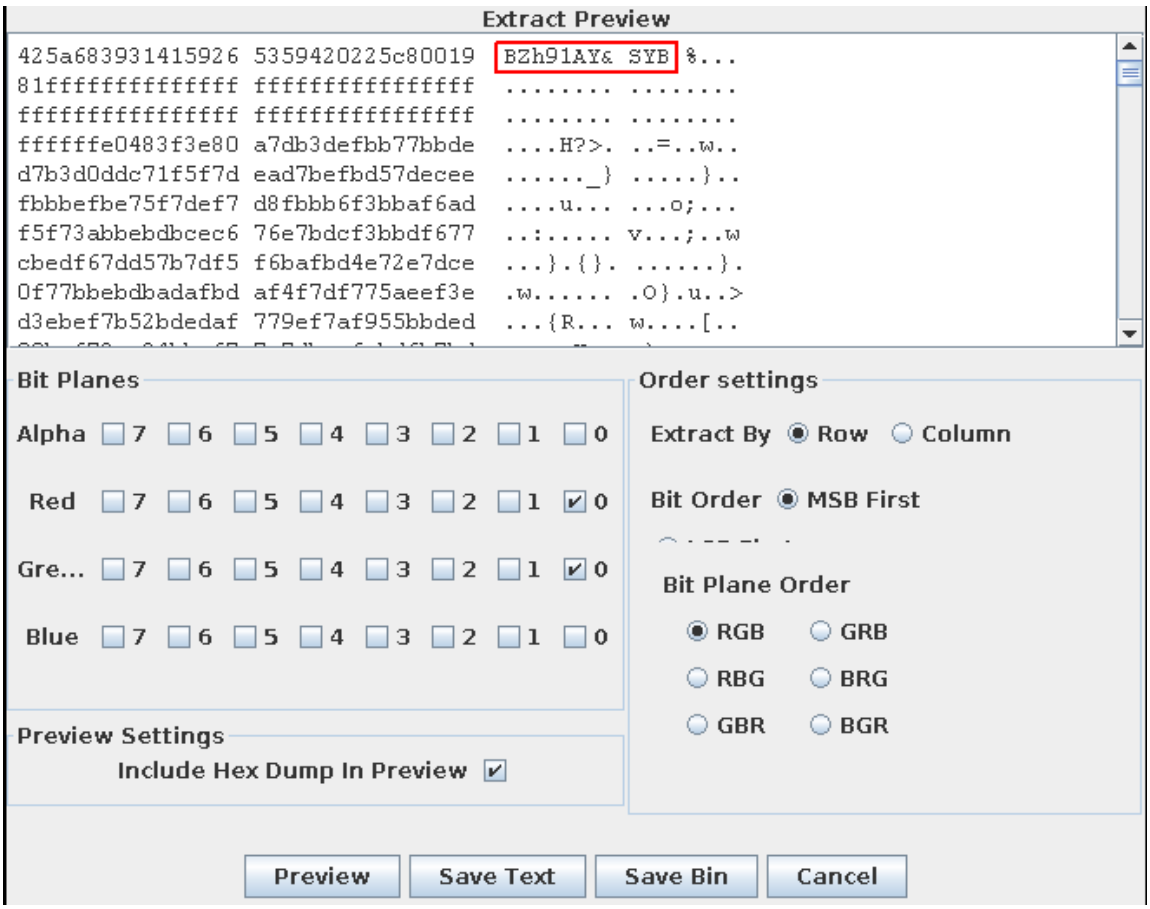


FIGURE 20: Extraction des données avec `stegsolve`, visualisation de l'en-tête Bzip

Cette archive contient une nouvelle image : `congratulations.gif` :

FIGURE 21: congratulations.gif

6.5 Solution du challenge

La même méthode que l'image précédente est utilisée sur cette nouvelle image, elle est ouverte dans l'outil `stegsolve`, les différentes couches et filtres disponibles sont parcourus à la recherche de données cachées. La solution du challenge (l'adresse email) est trouvée avec le filtre "Radom colour map", dans le cadre en bas de l'image :

Figure 22: Radom colour map sur congratulations.gif

7 Conclusion

Le challenge SSTIC 2015 était très intéressant par la diversité de ses épreuves. Il m'a permis de découvrir une architecture qui m'était inconnue (ST20).

8 Remerciements

Je tiens à remercier mes collègues Jean-Yves Burlett et Vincent Fargues pour leurs coups d'oeil avisés et leurs relectures. De manière plus générale, je remercie le CESTI Thales de m'avoir soutenu et m'avoir laissé du temps sur mes horaires de travail pour résoudre ce challenge.
Merci également à Guillaume BERARD et Sébastien DELCROIX pour leurs relectures.
Enfin, merci à l'équipe de la DGA-MI pour ce challenge original et aux organisateurs du SSTIC.

9 Annexes

9.1 *Parseur* de fichier BSP

```
#!/usr/bin/python
# http://www.mralligator.com/q3/
import struct


class BSP:
  Lump_Names = {
    "Entities":0,
    "Textures":1,
    "Planes":2,
    "Nodes":3,
    "Leafs":4,
    "Leaffaces":5,
    "Leafbrushes":6,
    "Models":7,
    "Brushes":8,
    "Brushsides":9,
    "Vertexes":10,
    "Meshverts":11,
    "Effects":12,
    "Faces":13,
    "Lightmaps":14,
    "Lightvols":15,
    "Visdata":16,
  }
  def __init__(self,filename):
    self.filename = filename
    self.bsp = open(filename,"rb").read()
    self.direntries = [''']*17
    self.parseHeader()
    self.parseEntities()
    self.parseTextures()
    self.parsePlanes()
    self.parseNodes()
    self.parseLeafs()
    self.parseLeaffaces()
    self.parseLeafbrushes()
    self.parseModels()
    self.parseBrushes()
    self.parseBrushsides()
    self.parseVertexes()
    self.parseMeshverts()
    self.parseEffects()
    self.parseFaces()
    self.parseLightmaps()
    self.parseLightvols()
    self.parseVisdata()
  def readInts(self,pos,num):
    return struct.unpack("<"+num*"i",self.bsp[pos:pos+(4*num)])[:num]
  def readFloats(self,pos,num):
    return struct.unpack("<"+num*"f",self.bsp[pos:pos+(4*num)])[:num]
  def parseHeader(self):
    # print 'magic : '+self.bsp[:4]
    version = self.readInts(4,1)[0]
    # print 'version :'+hex(version)
    direntries = self.readInts(8,17*2)
    for i in range(0,len(direntries),2):
      self.direntries[i/2]={}
      self.direntries[i/2]["offset"]=direntries[i]
      self.direntries[i/2]["len"]=direntries[i+1]
  def getDirentry(self,name):
    return self.direntries[self.Lump_Names[name]]

  def parseEntities(self):
    offset = self.getDirentry("Entities")["offset"]
    size   = self.getDirentry("Entities")["len"]
    self.entities = self.bsp[offset:offset+size]

  def parseTextures(self):
    offset = self.getDirentry("Textures")["offset"]
    size   = self.getDirentry("Textures")["len"]
```

```
    texture_size = 64+2*4
    self.textures = [''']*(size/texture_size)

    count=0
    for pos in range(offset,offset+size,texture_size):
      self.textures[count]={}
      name = self.bsp[pos:pos+64]
      name = name[:name.index('\0')]
      data = self.readInts(pos+64,2)
      self.textures[count]["name"]=name
      self.textures[count]["flags"]=data[0]
      self.textures[count]["contents"]=data[1]
      count+=1

  def parsePlanes(self):
    offset = self.getDirentry("Planes")["offset"]
    size   = self.getDirentry("Planes")["len"]

    # float[3] normal, float dist
    plane_size = 3 * 4 + 4
    self.planes = [''']*(size/plane_size)

    count=0
    for pos in range(offset,offset+size,plane_size):
      values = self.readFloats(pos,4)
      self.planes[count]={}
      self.planes[count]["normal"]=(values[0],values[1],values[2])
      self.planes[count]["dist"]=values[3]
      count+=1

  def parseNodes(self):
    offset = self.getDirentry("Nodes")["offset"]
    size   = self.getDirentry("Nodes")["len"]
    node_size = 4 + 4*2 + 4*3 + 4*3
    self.nodes = [''']*(size/node_size)

    count=0
    for pos in range(offset,offset+size,node_size):
      values = self.readInts(pos,9)
      self.nodes[count]={}
      self.nodes[count]["plane"]=values[0]
      self.nodes[count]["children"]=(values[1],values[2])
      self.nodes[count]["mins"]=(values[3],values[4],values[5])
      self.nodes[count]["maxs"]=(values[6],values[7],values[8])
      count+=1

  def parseLeafs(self):
    offset = self.getDirentry("Leafs")["offset"]
    size   = self.getDirentry("Leafs")["len"]

    leaf_size = 4 + 4 + 3*4 + 3*4 + 4 +4 +4 +4
    self.leafs = [''']*(size/leaf_size)

    count=0
    for pos in range(offset,offset+size,leaf_size):
      values = self.readInts(pos,12)
      self.leafs[count]={}
      self.leafs[count]["cluster"]=values[0]
      self.leafs[count]["area"]=values[1]
      self.leafs[count]["mins"]=(values[2],values[3],values[4])
      self.leafs[count]["maxs"]=(values[5],values[6],values[7])
      self.leafs[count]["leafface"]=values[8]
      self.leafs[count]["n_leaffaces"]=values[9]
      self.leafs[count]["leafbrush"]=values[10]
      self.leafs[count]["n_leafbrushes"]=values[11]
      count+=1

  def parseLeaffaces(self):
    offset = self.getDirentry("Leaffaces")["offset"]
    size   = self.getDirentry("Leaffaces")["len"]
    leafface_size = 4
    self.leaffaces = [''']*(size/leafface_size)

    count=0
    for pos in range(offset,offset+size,leafface_size):
      values = self.readInts(pos,1)
```

```
      self.leaffaces[count]={}
      self.leaffaces[count]["face"]=values[0]
      count+=1

  def parseLeafbrushes(self):
    offset = self.getDirentry("Leafbrushes")["offset"]
    size   = self.getDirentry("Leafbrushes")["len"]
    leafbrush_size = 4
    self.leafbrushs = [''']*(size/leafbrush_size)

    count=0
    for pos in range(offset,offset+size,leafbrush_size):
      values = self.readInts(pos,1)
      self.leafbrushs[count]={}
      self.leafbrushs[count]["brush"]=values[0]
      count+=1

  def parseModels(self):
    offset = self.getDirentry("Models")["offset"]
    size   = self.getDirentry("Models")["len"]
    model_size = 4*3 + 4*3 + 4 + 4 + 4 + 4
    self.models = [''']*(size/model_size)

    count=0
    for pos in range(offset,offset+size,model_size):
      values_float = self.readFloats(pos,6)
      values = self.readInts(pos+6*4,4)
      self.models[count]={}
      self.models[count]["mins"]=(values_float[0],values_float[1],values_float[2])
      self.models[count]["maxs"]=(values_float[3],values_float[4],values_float[5])
      self.models[count]["face"]=values[0]
      self.models[count]["n_faces"]=values[1]
      self.models[count]["brush"]=values[2]
      self.models[count]["n_brushes"]=values[3]
      count+=1


  def parseBrushes(self):
    offset = self.getDirentry("Brushes")["offset"]
    size   = self.getDirentry("Brushes")["len"]
    brush_size = 4 + 4 + 4
    self.brushes = [''']*(size/brush_size)

    count=0
    for pos in range(offset,offset+size,brush_size):
      values = self.readInts(pos,3)
      self.brushes[count]={}
      self.brushes[count]["brushside"]=values[0]
      self.brushes[count]["n_brushsides"]=values[1]
      self.brushes[count]["texture"]=values[2]
      count+=1

  def parseBrushsides(self):
    offset = self.getDirentry("Brushsides")["offset"]
    size   = self.getDirentry("Brushsides")["len"]
    brushside_size = 4 + 4
    self.brushsidesides = [''']*(size/brushside_size)

    count=0
    for pos in range(offset,offset+size,brushside_size):
      values = self.readInts(pos,2)
      self.brushsidesides[count]={}
      self.brushsidesides[count]["plane"]=values[0]
      self.brushsidesides[count]["texture"]=values[1]
      count+=1

  def parseVertexes(self):
    offset = self.getDirentry("Vertexes")["offset"]
    size   = self.getDirentry("Vertexes")["len"]
    vertexe_size = 4*3 + 2*2*4 + 3*4 + 4
    self.vertexes = [''']*(size/vertexe_size)

    count=0
    for pos in range(offset,offset+size,vertexe_size):
      values = self.readFloats(pos,10)
      ubyte = self.bsp[pos+10*4:pos+10*4+4].encode("hex")
```

```
      self.vertexes[count]={}
      self.vertexes[count]["position"]=(values[0],values[1],values[2])
      self.vertexes[count]["texcoord"]=((values[3],values[4]),(values[5],values[6]))
      self.vertexes[count]["normal"]=(values[7],values[8],values[9])
      self.vertexes[count]["color"] = ubyte
      count+=1

  def parseMeshverts(self):
    offset = self.getDirentry("Meshverts")["offset"]
    size   = self.getDirentry("Meshverts")["len"]
    meshvert_size = 4
    self.meshverts = ['']*(size/meshvert_size)

    count=0
    for pos in range(offset,offset+size,meshvert_size):
      values = self.readInts(pos,1)
      self.meshverts[count]={}
      self.meshverts[count]["offset"]=values[0]
      count+=1

  def parseEffects(self):
    offset = self.getDirentry("Effects")["offset"]
    size   = self.getDirentry("Effects")["len"]

    # string(64) + 2 INT
    effect_size = 64+2*4
    self.effects = ['']*(size/effect_size)

    count=0
    for pos in range(offset,offset+size,effect_size):
      self.effects[count]={}
      name = self.bsp[pos:pos+64]
      name = name[:name.index('\0')]
      data = self.readInts(pos+64,2)
      self.effects[count]["name"]=name
      self.effects[count]["brush"]=data[0]
      self.effects[count]["unknown"]=data[1]
      count+=1

  def parseFaces(self):
    offset = self.getDirentry("Faces")["offset"]
    size   = self.getDirentry("Faces")["len"]
    face_size = 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 2*4 + 2*4
    face_size += 3*4 + 2*3*4 + 3*4
    face_size += 2*4
    self.faces = ['']*(size/face_size)

    count=0
    for pos in range(offset,offset+size,face_size):
      values = self.readInts(pos,12)
      values_float = self.readFloats(pos+12*4,12)
      values2 = self.readInts(pos+24*4,2)
      self.faces[count]={}
      self.faces[count]["texture"]=values[0]
      self.faces[count]["effect"]=values[1]
      self.faces[count]["type"]=values[2]
      self.faces[count]["vertex"]=values[3]
      self.faces[count]["n_vertexes"]=values[4]
      self.faces[count]["meshvert"]=values[5]
      self.faces[count]["n_meshverts"]=values[6]
      self.faces[count]["lm_index"]=values[7]
      self.faces[count]["lm_start"]=(values[8],values[9])
      self.faces[count]["lm_size"]=(values[10],values[11])

      self.faces[count]["lm_origin"]=(values_float[0],values_float[1],values_float[2])
      self.faces[count]["lm_vecs"]=( (values_float[3],values_float[4],values_float[5]) ,
          ↪ (values_float[6],values_float[7],values_float[8]))
      self.faces[count]["normal"]=(values_float[9],values_float[10],values_float[11])

      self.faces[count]["size"]=(values2[0],values2[1])

      count+=1

  def parseLightmaps(self):
    offset = self.getDirentry("Lightmaps")["offset"]
    size   = self.getDirentry("Lightmaps")["len"]
```

```
    # todo
    self.lightmap = self.bsp[offset:offset+size]

  def parseLightvols(self):
    offset = self.getDirentry("Lightvols")["offset"]
    size   = self.getDirentry("Lightvols")["len"]
    # todo

  def parseVisdata(self):
    offset = self.getDirentry("Visdata")["offset"]
    size   = self.getDirentry("Visdata")["len"]
    # todo

  def getFaces(self):
    return self.faces

  def getTextures(self):
    return self.textures
```

Listing 53: Parser de fichier BSP

9.2 Émulateur pour l'architecture ST20

```
#!/usr/bin/python
import time
import sys
import socket
import os
import pipes

def debug(message,level=1):
  print_level = 0
  if level > print_level:
    print message

class Memory():
  def __init__(self,start,size,cpuno):
    self.start = start
    self.cpuno = cpuno
    self.size = size
    self.contents = [0]*size
    self.initMem(start,"\x00"*size)
  def getByteMem(self,addr):
    if addr - self.start >  self.size or addr - self.start < 0:
      print "#%02d# [memory] request invalid addr %08x" % (self.cpuno,addr)
      return 0
    return self.contents[(addr - self.start)]&0xFF
  def setByteMem(self,addr,value):
    old=self.getByteMem(addr)
    self.contents[(addr - self.start)]=(value&0xFF)
    debug("#%02d# [memory] byte write : addr:%08x old:%02x new:%02x" %
        ↪ (self.cpuno,addr,old,value),level=1)
  def getMem(self,addr):
    m1=self.contents[addr - self.start]
    m2=self.contents[(addr - self.start)+1]
    m3=self.contents[(addr - self.start)+2]
    m4=self.contents[(addr - self.start)+3]
    val = ( (m1 & 0xFF) | ((m2&0xFF) << 8) | ((m3&0xFF) << 16) | ((m4&0xFF) << 24))
    debug("#%02d# [memory] getmem %08x -> %08x" % (self.cpuno, addr, val),level=1)
    return val
  def setMem(self,addr,value):
    old=self.getMem(addr)
    self.contents[addr - self.start] = (value&0xFF)
    self.contents[(addr - self.start)+1] = (value>>8)&0xFF
    self.contents[(addr - self.start)+2] = (value>>16)&0xFF
    self.contents[(addr - self.start)+3] = (value>>24)&0xFF
    debug("#%02d# [memory] write : addr:%08x old:%08x new:%08x" % (self.cpuno,
        ↪ addr,old,value),level=1)

  def initMem(self,start_pos,content):
    for i,byte in enumerate(content):
      self.contents[(start_pos-self.start)+i]=ord(byte)




class CPU(object):
  IPtr = 0
  WPtr=0
  AReg=0
  BReg=0
  CReg = 0
  OReg=0
  temp=0
  temp2=0
  temp3=0

  ClockReg0=0
  ClockReg1=0
  TNextReg0=0
  TNextReg1=0
  FPtrReg0=0
  BPtrReg0=0
  FPtrReg1=0
  BPtrReg1=0
  HiTimer=0
```

```
  LoTimer=0
  TPtrLoc0 = 0
  TPtrLoc1 = 0
  ErrorFlag = False
  ErrorFlagInt=0
  Interrupt = False
  HaltOnErrorFlag=0

  instr_count=0


  Link0In = 0x80000010



  def __init__(self,WPtr_init,cpuno,links=dict()):
    self.MemStart = 0x80000048
    self.IPtr=self.MemStart
    self.memory=Memory(self.MemStart,2*1024*1024,cpuno)
    self.CReg = self.Link0In
    self.WPtr = self.MemStart + WPtr_init
    self.links = links
    self.cpuno = cpuno
    while ((self.WPtr & 0x00000003) != 0x00000000):
      self.WPtr+=1
    self.icodes = {
      0x00: self.j,
      0x10: self.ldlp,
      0x20: self.pfix,
      0x30: self.ldnl,
      0x40: self.ldc,
      0x50: self.ldnlp,
      0x60: self.nfix,
      0x70: self.ldl,
      0x80: self.adc,
      0x90: self.call,
      0xA0: self.cj,
      0xB0: self.ajw,
      0xC0: self.eqc,
      0xD0: self.stl,
      0xE0: self.stnl,
      0xF0: self.opr,
    }
    self.opr_dispatcher = {
      0x00: self.rev,
      0x01: self.lb,
      0x02: self.bsub,
      0x03: self.endp,
      0x04: self.diff,
      0x05: self.add,
      0x06: self.gcall,
      0x07: self.IN,
      0x08: self.prod,
      0x09: self.gt,
      0x0A: self.wsub,
      0x0B: self.out,
      0x0C: self.sub,
      0x0D: self.startp,
      0x0E: self.outbyte,
      0x0F: self.outword,
      0x10: self.seterr,
      0x12: self.resetch,
      0x13: self.csub0,
      0x15: self.stopp,
      0x16: self.ladd,
      0x17: self.stlb,
      0x18: self.sthf,
      0x19: self.norm,
      0x1a: self.ldiv,
      0x1b: self.ldpi,
      0x1c: self.stlf,
      0x1d: self.xdble,
      0x1e: self.ldpri,
      0x1f: self.rem,
      0x20: self.ret,
      0x21: self.lend,
```

```
      0x22: self.ldtimer,
      0x23: self.testlds,
      0x24: self.testlde,
      0x25: self.testldd,
      0x26: self.teststs,
      0x27: self.testste,
      0x28: self.teststd,
      0x29: self.testerr,
      0x2a: self.testpranal,
      0x2b: self.tin,
      0x2c: self.div,
      0x2d: self.testhardchan,
      0x2e: self.dist,
      0x2f: self.disc,
      0x30: self.diss,
      0x31: self.lmul,
      0x32: self.NOT,
      0x33: self.xor,
      0x34: self.bcnt,
      0x35: self.lshr,
      0x36: self.lshl,
      0x37: self.lsum,
      0x38: self.lsub,
      0x39: self.runp,
      0x3a: self.xword,
      0x3b: self.sb,
      0x3c: self.gajw,
      0x3d: self.savel,
      0x3e: self.saveh,
      0x3f: self.wcnt,
      0x40: self.shr,
      0x41: self.shl,
      0x42: self.mint,
      0x43: self.alt,
      0x44: self.altwt,
      0x45: self.altend,
      0x46: self.AND,
      0x47: self.enbt,
      0x48: self.enbc,
      0x49: self.enbs,
      0x4a: self.move,
      0x4b: self.OR,
      0x4c: self.csngl,
      0x4d: self.ccnt1,
      0x4e: self.talt,
      0x4f: self.ldiff,
      0x50: self.sthb,
      0x51: self.taltwt,
      0x52: self.sum,
      0x53: self.mul,
      0x54: self.sttimer,
      0x55: self.stoperr,
      0x56: self.cword,
      0x57: self.clrhalterr,
      0x58: self.sethalterr,
      0x59: self.testhalterr,
      0x5a: self.dup,
      0xc1: self.ssub,
      }


  def index(self,a,b):
    return a+4*b

  def stackPush(self):
    self.CReg = self.BReg
    self.BReg = self.AReg

  def instructionEnding(self):
    self.IPtr += 1
    self.OReg = 0

  def loadBootstrap(self,data):
    debug("#%02d# loadBootstrap (%05d bytes) = [%s]" % (self.cpuno, len(data),
        ↪ repr(data)),level=2)
    self.memory.initMem(self.MemStart,data)
```

```
  def reset(self):
    self.Iptr = self.MemStart
    self.Wptr = 0
    self.AReg = 0
    self.BReg = 0
    self.CReg = 0
    self.OReg = 0
    self.ErrorFlag = False
    self.HaltOnErrorFlag = False

  def nextInst(self):
    return self.Iptr + 1

  def step(self):
    instr = self.memory.getByteMem(self.IPtr)
    Icode = instr & 0xf0;
    Idata = instr & 0x0f;
    self.OReg  = self.OReg | Idata;
    instr_name=self.icodes[Icode].func_name
    if instr_name == "opr" and self.OReg in self.opr_dispatcher:
      instr_name+="("+self.opr_dispatcher[self.OReg].func_name+")"

    debug("#%02d# instr_count=%6d IPtr=%8X, Instruction=%2X %10s, OReg=%8X, AReg=%8X,
        ↪ BReg=%8X, CReg=%8X, WPtr=%8X. WPtr[0]=%8X" % \
        (self.cpuno, self.instr_count, self.IPtr, instr, instr_name, self.OReg,
            ↪ self.AReg, self.BReg, self.CReg,
            ↪ self.WPtr,self.memory.getMem(self.WPtr)),level=4)
    if Icode not in self.icodes:
      print "#%02d# Bad opcode %x" % (self.cpuno, Icode)
      sys.exit(1)
    self.icodes[Icode](Idata)
    self.instr_count += 1

  # instructions functions
  def j(self,Idata):
    self.IPtr += 1
    self.IPtr += self.OReg
    self.OReg = 0

  def ldlp(self,Idata):
    self.CReg = self.BReg
    self.BReg = self.AReg
    self.AReg = self.index(self.WPtr&0xfffffffc, self.OReg)
    self.instructionEnding()

  def pfix(self,Idata):
    self.IPtr+=1
      self.OReg <<= 4

  def ldnl(self,Idata):
    # self.stackPush()
      self.AReg = self.memory.getMem(self.index(self.AReg, self.OReg))
    self.instructionEnding()

  def ldc(self,Idata):
    self.stackPush()
    self.AReg = self.OReg
    self.instructionEnding()

  def ldnlp(self,Idata):
      self.AReg = self.index(self.AReg, self.OReg)
    self.instructionEnding()

  def nfix(self,Idata):
    self.IPtr+=1
      self.OReg = ~self.OReg
      self.OReg <<= 4
        # XXX TODO passer en neg

  def ldl(self,Idata):
    self.stackPush()
    self.AReg = self.memory.getMem(self.index(self.WPtr&0xfffffffc, self.OReg))
    self.instructionEnding()

  def adc(self,Idata):
    # TODO check overflow
```

```
    self.AReg += self.OReg
    self.instructionEnding()

  def call(self,Idata):
    self.IPtr+=1
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, -1), self.CReg)
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, -2), self.BReg)
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, -3), self.AReg)
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, -4), self.IPtr)
    self.WPtr = self.index(self.WPtr&0xfffffffc, -4)
    self.AReg = self.IPtr
    self.BReg = self.CReg
    self.IPtr+= self.OReg
    self.OReg=0

  def cj(self,Idata):
    self.IPtr+=1
    if self.AReg != 0:
      self.AReg = self.BReg
      self.BReg = self.CReg
    else:
      self.IPtr+=self.OReg
    self.OReg=0

  def ajw(self,Idata):
    self.WPtr = self.index(self.WPtr&0xfffffffc,self.OReg)
    self.instructionEnding()


  def eqc(self,Idata):
    if self.AReg == self.OReg:
      self.AReg = 1
    else:
      self.AReg = 0
    self.IPtr+=1
    self.OReg = 0
  def stl(self,Idata):
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, self.OReg), self.AReg)
    self.AReg = self.BReg
    self.BReg = self.CReg
    self.instructionEnding()
  def stnl(self,Idata):
    self.memory.setMem(self.index(self.AReg, self.OReg), self.BReg)
    self.AReg = self.BReg = self.CReg
    self.instructionEnding()
  def opr(self,Idata):
    if self.OReg not in self.opr_dispatcher:
      print "#%02d# Bad opcode OPR %x" % (self.cpuno, self.OReg)
      sys.exit(1)
    debug("#%02d# >>> %s" % (self.cpuno,
        ↪ self.opr_dispatcher[self.OReg].func_name),level=1)
    self.opr_dispatcher[self.OReg](Idata)
    self.OReg = 0




  def rev(self,Idata):
    t = self.AReg
    self.AReg = self.BReg
    self.BReg = t
    self.instructionEnding()

  def lb(self,Idata):
    self.AReg = self.memory.getByteMem(self.AReg)
    self.instructionEnding()

  def bsub(self,Idata):
    self.AReg = (self.BReg + self.AReg) & 0xFFFFFFFF
    self.BReg = self.CReg
    self.instructionEnding()

  def endp(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

```

```
  def diff(self,Idata):
    self.AReg = self.BReg - self.AReg
    self.BReg = self.CReg
    self.instructionEnding()

  def add(self,Idata):
    self.AReg += self.BReg # XXX TODO check overflow
    self.BReg = self.CReg
    self.instructionEnding()

  def gcall(self,Idata):
    self.IPtr += 1
    tmp = self.AReg
    self.AReg = self.IPtr
    self.IPtr = tmp

  def IN(self,Idata):
    debug("#%02d# read msg (len=%d) from channel %x" % (self.cpuno, self.AReg,
        ↪ self.BReg),level=5)
    if self.BReg in set([0x80000010,0x80000014,0x80000018,0x8000001c]):
        msg = self.links[(self.BReg-0x80000010) >>2][0].read(self.AReg)
        debug("#%02d# got message channel %x -> (len=%d) [%s]" % (self.cpuno, self.BReg,
            ↪ len(msg), repr(msg)),level=5)
        for i,c in enumerate(msg):
            self.memory.setByteMem(self.CReg+i, ord(c))
    else:
            raise RuntimeError("in: unknown channel")
    self.instructionEnding()

  def prod(self,Idata):
    self.AReg = self.AReg * self.BReg
    self.BReg = self.CReg
    self.instructionEnding()

  def gt(self,Idata):
    if self.BReg > self.AReg:
      self.AReg = 1
    else:
      self.AReg = 0
    self.BReg = self.CReg
    self.instructionEnding()

  def wsub(self,Idata):
    self.AReg = self.index(self.AReg, self.BReg)
    self.BReg = self.CReg
    self.instructionEnding()

  def out(self,Idata):
    length = self.AReg
    msg = ""
    for p in xrange(length):
        msg += chr(self.memory.getByteMem(self.CReg+p))
    debug("#%02d# output msg (len=%d) on channel %x [%s]" % (self.cpuno, self.AReg,
        ↪ self.BReg, repr(msg)),level=5)
    if self.BReg in set([0x80000000,0x80000004,0x80000008,0x8000000c]):
        self.links[(self.BReg-0x80000000) >>2][1].write(msg)
    else:
        raise RuntimeError("out: unknown channel")
    self.instructionEnding()

  def sub(self,Idata):
    self.AReg = self.BReg - self.AReg # XXX TODO check overflow
    self.BReg = self.CReg
    self.instructionEnding()

  def startp(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def outbyte(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def outword(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)
```

```
  def seterr(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def resetch(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def csub0(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def stopp(self,Idata):
    self.IPtr +=1
    self.memory.setMem(self.index(self.WPtr&0xfffffffc, -1),self.IPtr)

  def ladd(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def stlb(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def sthf(self,Idata):
    self.FPtrReg0 = self.AReg
    self.AReg = self.BReg
    self.BReg = self.CReg
    self.IPtr += 1

  def norm(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def ldiv(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def ldpi(self,Idata):
    self.IPtr += 1
    self.AReg = self.IPtr + self.AReg

  def stlf(self,Idata):
    self.FPtrReg1 = self.AReg
    self.AReg = self.BReg
    self.BReg = self.CReg
    self.IPtr += 1

  def xdble(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def ldpri(self,Idata):
    self.CReg = self.BReg
    self.BReg = self.AReg
    self.AReg = 1
    self.instructionEnding()

  def rem(self,Idata):
    if self.AReg == 0:
      print "Overflow in REM"
      sys.exit(0)
    elif self.AReg == -1 and self.BReg == 0x80000000:
      self.AReg = 0
      # self.CReg = self.AReg
    else:
      self.AReg = self.BReg % self.AReg
      # self.CReg = abs(self.AReg)
    self.BReg = self.CReg
    self.instructionEnding()

  def ret(self,Idata):
    self.IPtr = self.memory.getMem(self.WPtr)
    self.WPtr = self.index(self.WPtr&0xfffffffc, 4)

```

```
  def lend(self,Idata):
    temp = self.memory.getMem(self.index(self.BReg,1))
    self.IPtr += 1
    self.memory.setMem(self.index(self.BReg,1),(temp-1))
    if temp > 1:
      temp = self.memory.getMem(self.index(self.BReg,0))
      self.memory.setMem(self.index(self.BReg, 0), (temp+1))
      self.IPtr = self.IPtr - self.AReg
      # self.CReg = self.memory.getMem(self.BReg)
    else:
      # self.CReg = 0
      pass

  def ldtimer(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def testlds(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def testlde(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def testldd(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def teststs(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def testste(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def teststd(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def testerr(self,Idata):
    self.Areg = 0 # TODO Areg == error flag
    # TODO test error
    self.instructionEnding()


  def testpranal(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def tin(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def div(self,Idata):
    if self.AReg == 0 or (self.AReg == -1 and self.BReg == 0x80000000):
      print "Error"
      sys.exit(0)
    else:
      A = abs(self.AReg)
      B = abs(self.BReg)
      self.AReg = self.BReg / self.AReg
      self.BReg = self.CReg
      self.CReg = B - (abs(self.AReg)|1)*A
    self.instructionEnding()

  def testhardchan(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def dist(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def disc(self,Idata):
```

```
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def diss(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def lmul(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def NOT(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def xor(self,Idata):
    self.AReg = self.BReg ^ self.AReg
    BReg = self.CReg
    self.instructionEnding()

  def bcnt(self,Idata):
    self.AReg = self.AReg * 4
    self.instructionEnding()

  def lshr(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def lshl(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def lsum(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def lsub(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def runp(self,Idata):
    self.WPtr = self.AReg & 0xfffffffe
    self.IPtr = self.memory.getMem(self.index(self.WPtr & 0xfffffffc, -1))

  def xword(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def sb(self,Idata):
    self.memory.setByteMem(self.AReg, self.BReg)
    #self.BReg = self.AReg
    self.AReg = self.CReg
    self.CReg = 0
    self.instructionEnding()

  def gajw(self,Idata):
        temp = self.WPtr
        self.WPtr = self.AReg
        self.AReg = temp
        self.instructionEnding()

  def savel(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def saveh(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def wcnt(self,Idata):
    self.CReg = self.BReg
    self.BReg = self.AReg & 0x3
    self.AReg >>= 2
    self.instructionEnding()

  def shr(self,Idata):
```

```
    if self.AReg < 32:
      self.AReg = self.BReg >> self.AReg
    else:
      self.AReg = 0
    self.BReg = self.CReg
    self.instructionEnding()

  def shl(self,Idata):
    if self.AReg < 32:
      self.AReg = self.BReg << self.AReg
    else:
      self.AReg = 0
    self.BReg = self.CReg
    self.instructionEnding()

  def mint(self,Idata):
      self.stackPush()
      self.AReg = 0x80000000
      self.instructionEnding()

  def alt(self,Idata):
    self.memory.setMem(self.index(self.WPtr & 0xfffffffc, -3),0x80000001)
    self.instructionEnding()

  def altwt(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)


  def altend(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def AND(self,Idata):
    self.AReg = self.BReg & self.AReg
    self.BReg = self.CReg
    self.instructionEnding()

  def enbt(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def enbc(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def enbs(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def move(self,Idata):
    for i in range(self.AReg):
      temp = self.memory.getByteMem(self.CReg + i)
      self.memory.setByteMem(self.BReg+i, temp)
    self.CReg = self.CReg + (self.AReg * 4)
    self.instructionEnding()

  def OR(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def csngl(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def ccnt1(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def talt(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def ldiff(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)
```

```
  def sthb(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def taltwt(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def sum(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def mul(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def sttimer(self,Idata):
    self.ClockReg0 = self.AReg
    self.ClockReg1 = self.AReg
    self.AReg = self.BReg
    self.BReg = self.CReg
    self.instructionEnding()

  def stoperr(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def cword(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def clrhalterr(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

  def sethalterr(self,Idata):
    self.instructionEnding()

  def dup(self,Idata):
    self.stackPush()
    self.instructionEnding()

  def ssub(self,Idata):
    self.AReg = self.AReg + 2*self.BReg
    self.BReg = self.CReg
    self.instructionEnding()

  def testhalterr(self,Idata):
    print "todo : "+sys._getframe().f_code.co_name
    sys.exit(0)

class lazyproxy(object):
    def __init__(self, fname, mode):
        self.realfile = None
        self.rfname = fname
        self.rmode = mode
        self.cursor = 0
        # print "new lazyproxy "+self.rfname
        if self.rmode == "wb+":
            self.openrealfile()

    def __del__(self):
        self.close()

    def openrealfile(self):
      # print "open "+self.rfname
        self.realfile = open(self.rfname, self.rmode, 0)
        # print "open OK"

    def read(self, length):
      # print "in read "
        if self.realfile is None:
            self.openrealfile()
        try:
            remaining = length
```

```
824             outp = ""
825             while remaining > 0:
826                 buff=self.realfile.read(length)
827                 if len(buff) == 0:
828                     print "READ ATTEMPT FAILED %s %d" % (self.rfname, length)
829                     raise RuntimeError()
830                 self.cursor += len(buff)
831                 outp += buff
832                 remaining -= len(buff)
833             debug("read (%s) ask:%d (sofar %d) bytes data [%s]" %
                    ↪ (self.rfname,length,self.cursor,repr(buff)),level=5)
834             return buff
835
836         except:
837             # print "lazyproxy.read %s %d" % (self.rfname, length)
838             raise
839
840     def write(self, pkt):
841       debug("write ("+self.rfname+") "+repr(pkt),level=5)
842         if self.realfile is None:
843             self.openrealfile()
844         tmp = self.realfile.write(pkt)
845         self.realfile.flush()
846         return tmp
847
848     def close(self):
849         if self.realfile is not None:
850             self.realfile.close()
851
852
853 if __name__ == "__main__":
854   fn = "/home/david/share-vm/emu/hello.btl"
855   try:
856       fn = sys.argv[1]
857   except IndexError:
858       pass
859   cpuno=0
860   if len(sys.argv) > 9:
861     cpuno=sys.argv[9]
862   print "Transputer #%s" % (cpuno)
863
864   link = dict()
865   link[0] = ['']*2
866   link[1] = ['']*2
867   link[2] = ['']*2
868   link[3] = ['']*2
869
870
871
872
873   if len(sys.argv) > 2:
874     link[0][1]=lazyproxy(sys.argv[2], "wb+")
875   if len(sys.argv) > 4:
876     link[1][1]=lazyproxy(sys.argv[4], "wb+")
877   if len(sys.argv) > 6:
878     link[2][1]=lazyproxy(sys.argv[6], "wb+")
879   if len(sys.argv) > 8:
880     link[3][1]=lazyproxy(sys.argv[8], "wb+")
881
882   if len(sys.argv) > 1:
883     link[0][0] = lazyproxy(sys.argv[1], "rb")
884   if len(sys.argv) > 3:
885     link[1][0] = lazyproxy(sys.argv[3], "rb")
886   if len(sys.argv) > 5:
887     link[2][0] = lazyproxy(sys.argv[5], "rb")
888   if len(sys.argv) > 7:
889     link[3][0] = lazyproxy(sys.argv[7], "rb")
890
891
892   print "Links open #%s ..." % (cpuno)
893
894   plen = ord(link[0][0].read(1))
895   cpu=CPU(plen, int(cpuno), links=link)
896   cpu.loadBootstrap(link[0][0].read(plen))
897   print "\033[31;1mGO #%s >>>>\033[0m" % (cpuno)
898   #for i in xrange(6000):
```

```
while True:
  cpu.step()
print "\033[31;1m #%s END OF EXECUTION\033[0m" % (cpuno)
```

Listing 54: Émulateur pour l'architecture ST20

9.3 Simulateur pour T0

```
import sys

class Streamgen(object):
    def __init__(self):
        # t1
        self.t4 = 0
        self.t5 = 0
        self.t6 = 0
        self.t6v3 = 0

        # t2
        self.t8= 0
        self.t8var3=0
        self.t8var5 = dict()
        self.t8var4=0
        for t8var2 in range(4):
            for t8var0 in range(12):
                self.t8var5[12*t8var2+t8var0] = 0


        # t3
        self.t10=0
        self.t11=0
        self.t12=0

        self.t10v3 = 0
        self.t10v2 = 0
        self.t10v4 = dict()

        for t10v0 in range(4):
            for t10v1 in range(12):
                self.t10v4[t10v0*12+t10v1] = 0

        self.t11v2=0

        self.t12v2=0
        self.t12v3 = dict()
        for i in range(12):
            self.t12v3[i] = 0



    def getbyte(self, keystate):
        m = keystate
        # t1

        for i,c in enumerate(m):
            self.t4 += ord(c)
            self.t4 &= 0xff

        for i,c in enumerate(m):
            self.t5 ^= ord(c)
            self.t5 &= 0xff

        if self.t6v3 == 0:
            for i,c in enumerate(m):
                self.t6 += ord(c)
                self.t6 &= 0xffff
            self.t6v3 = 1
        t = (self.t6&0x8000) >> 0xf
        t ^= (self.t6&0x4000) >> 0xe
        t &= 0xffff
        self.t6 = t ^ ((self.t6 << 1) & 0xffff)
        self.t6 &= 0xffff


        t1 = self.t4 & 0xff
        t1 ^= self.t5 & 0xff
        t1 ^= self.t6 & 0xff

        # t2
        ## t7
        t7v1=0
```

```
        t7v2=0
        for i in xrange(6):
            t7v1 += ord(m[i])
            t7v1 &= 0xff
            t7v2 += ord(m[i+6])
            t7v2 &= 0xff

        t7 = t7v1 ^ t7v2
        t7 &= 0xff

        # t8
        for i,c in enumerate(m):
            self.t8var5[self.t8var4*12+i] = ord(c)
        self.t8var4 += 1
        if self.t8var4 >=4:
            self.t8var4 = 0
        self.t8var3 = 0
        for t8var2 in range(4):
            t8var1 = 0
            for t8var0 in range(12):
                t8var1 = (self.t8var5[t8var2*12+t8var0]+t8var1) & 0xff
            self.t8var3 = (self.t8var3 ^ t8var1) & 0xff

        t8 = self.t8var3

        #t9
        t9=0
        for i,c in enumerate(m):
            t9 ^= ord(c) << (i&7)
            t9 &= 0xff

        t2 = t7 & 0xff
        t2 ^= t8 & 0xff
        t2 ^= t9 & 0xff

        # t3
        # t10
        for i in range(12):
            self.t10v4[self.t10v2*12+i] = ord(m[i])
        self.t10v2 += 1
        if self.t10v2 >= 4:
            self.t10v2 = 0

        self.t10v1 = 0
        for i in range(4):
            self.t10v1 = (self.t10v1 + self.t10v4[i*12]) & 0xff
        self.t10v3 = self.t10v4[(self.t10v1 & 3)*12 + ((self.t10v1 >> 4) % 0xC) & 0xff]
        t10 = self.t10v3




        t11v1 = ord(m[0]) ^ ord(m[3]) ^ ord(m[7])
        t11v1 &= 0xff

        t12v1 = self.t12v3[1] ^ self.t12v3[5] ^ self.t12v3[9]
        t12v1 &= 0xff
        for i,c in enumerate(map(ord,m)):
            self.t12v3[i] = c
        self.t12v2 = t11v1

        t11v1 = t12v1

        self.t12v2 %= 0xC
        self.t12v2 &= 0xff
        t12v1 = self.t12v3[self.t12v2]

        t12 = t12v1

        t11v1 %= 0xC
        t11v1 &= 0xff
        self.t11v2 = ord(m[t11v1])

        t11 = self.t11v2


```

```
150         t3 = t10 & 0xff
151         t3 ^= t11 & 0xff
152         t3 ^= t12 & 0xff
153 
154         magic = (t1 ^ t2 ^ t3) & 0xff
155         return magic
156 
157 
158 def mainloop(initkey, fin, fout):
159     ks = map(ord,initkey)
160     s = Streamgen()
161     it = 0
162     doit = True
163     while doit:
164         b = fin.read(1)
165         if len(b) == 0:
166             doit = False
167         else:
168             x = s.getbyte(map(chr,ks))
169             clear = (ord(b) ^ (it+2*ks[it])) & 0xff
170             ks[it] = x
171             it += 1
172             it %= 12
173             fout.write(chr(clear))
174 
175 
176 
177 if __name__ == "__main__":
178     key = list("*SSTIC-2015*")
179     file1=open("testvector","rb") # only data
180     file2=open("out","wb")
181     mainloop(key, file1, file2)
```

Listing 55: Simulateur de T0