SSTIC 2015 - Solution du challenge

Yoann Francou
<yoann dot francou at gmail dot com>

21 avril 2015

Résumé

Le SSTIC 2015 est une conférence francophone sur la sécurité de l'information. Chaque année, un challenge est proposé. Ce document présente une des solutions possibles pour résoudre le challenge SSTIC 2015.

Le challenge est composé de 6 étapes successives. Le point de départ est une carte MicroSD téléchargée depuis le site `https://www.sstic.org`. Le but final est de trouver l'adresse e-mail de validation de la forme '@challenge.sstic.org'.

Les épreuves sont successivement l'analyse d'une carte MicroSD, la recherche d'un code dans une partie de Quake3, l'analyse d'une capture USB, la désobfuscation d'un code JavaScript, la rétro-ingénierie d'un programme utilisé sur une architecture ST20 et la stéganalyse de 4 images.

Table des matières

1 Épreuve 1 : Analyse de la carte MicroSD

1.1 Découverte de l'épreuve

L'épreuve n° 1 commence avec le fichier sdcard.img.

```
~/SSTIC2015/stage1$ ll
128000000 sdcard.img
~/SSTIC2015/stage1$ file sdcard.img
sdcard.img: x86 boot sector
```

Il s'agit de l'image d'une carte MicroSD de 128Mo. Regardons ce qu'elle contient en la montant :

```
E50D-883B$ ll
34253730 inject.bin
E50D-883B$ file inject.bin
inject.bin: data
```

La carte MicroSD contient un fichier nommé `inject.bin` d'environ 34Mo. Aucun autre détail n'est visible. Avant de se lancer directement dans l'analyse de ce fichier, regardons si des indices supplémentaires ne se trouvent pas sur la carte MicroSD.

```
~/SSTIC2015/stage1$ fls sdcard.img
r/r * 4:        build.sh
r/r 6:  inject.bin
v/v 3992179:    $MBR
v/v 3992180:    $FAT1
v/v 3992181:    $FAT2
d/d 3992182:    $OrphanFiles

~/SSTIC2015/stage1$ icat sdcard.img 4
java -jar encoder.jar -i /tmp/duckyscript.txt
```

La commande `fls` nous permet d'identifier les fichiers présents dans l'image. Nous voyons apparaître le fichier `build.sh`.

La command `icat` nous permet ensuite de regarder le contenu du fichier. Celui-ci contient une commande *java -jar encoder.jar -i /tmp/duckyscript.txt*.

Une recherche sur Google de 'encode.jar duckyscript' nous donne rapidement des informations via le github `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript`.

Voici un extrait intéressant :

> Google - Ducky Script
> Ducky Script is the language of the USB Rubber Ducky. Writing scripts for can be done from any common ascii text editor such as Notepad, vi, emacs, nano, gedit, kedit, TextEdit, etc.
>
> Ducky Scripts are compiled into hex files ready to be named inject.bin and moved to the root of a microSD card for execution by the USB Rubber Ducky. This is done with the tool duckencoder.
> duckencoder is a cross-platform command-line Java program which converts the Ducky Script syntax into hex files.
>
> For example on a Linux system :
> java -jar duckencoder.jar -i exploit.txt -o /media/microsdcard/inject.bin

A partir de ces informations, nous pouvons imaginer un scénario plausible concernant la carte MicroSD.

* Le fichier `build.sh` contient la commande *java -jar encoder.jar -i /tmp/duckyscript.txt*

* L'utilisateur a exécuté ce script avec comme paramètre son Ducky Script nommé `duckyscript.txt`.
* le fichier `inject.bin` a été créé. Il s'agit du Ducky Script compilé.
* L'utilisateur a supprimé le fichier `build.sh` et le fichier `encoder.jar`

Nous devons maintenant analyser le fichier `inject.bin`. Ça n'a pas l'air d'une tâche facile et évidente. Hum ... `encoder.jar` ... `encoder.jar` ... Ça serait bien si il existait un `decoder.jar` quelque part.

En faisant quelques recherches sur Google, nous découvrons le script `ducky-decode.pl`.

Le script `ducky-decode.pl` prend comme argument un Ducky Script compilé du type `inject.bin` et ressort le Ducky Script original décodé (`duckyscript.txt`).

```
~/SSTIC2015/stage1$ ./ducky_decode.pl inject.bin > duckyscript.txt
~/SSTIC2015/stage1$ ll
    8736 ducky_decode.pl
34324954 duckyscript.txt
34253730 inject.bin
```

1.2 Analyse du Ducky Script

Nous commençons l'analyse avec le Ducky Script décodé au format txt. Le fichier fait un peu plus de 30Mo. Voyons ce qu'il contient.

```
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 B A E I A b A B B A E g A S Q B B A C c A K Q A p A D s A f Q A = 00a0
ENTER
 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 B A E I A b A B B A E g A S Q B B A C c A K Q A p A D s A f Q A = 00a0
ENTER
 p o w e r s h e l l
SPACE
 [...]
```

Le script fait 20352 lignes. Il contient une liste de commandes, avec à intervalle régulier des gros blocs de données d'environ 5000 caractères qui nous font penser à du Base64.

En se référant à la documentation sur le site `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript` :

* GUI R : Simule un Windows+R (Ouvre la fenêtre Exécuter)
* DELAY X : Pause le script pendant X millisecondes
* ENTER : Simule la touche ENTREE
* SPACE : Simule la touche ESPACE

Le script fonctionne sous Windows. Il ouvre la fenêtre 'Exécuter' en simulant un Windows+R. Il y écrit 'cmd' et fait ENTREE. Cela lance un cmd.exe. Il exécute ensuite la commande *powershell -enc ZgB1AG4AYwB0AGkAbwB...*

L'argument -enc de powershell permet de passer un argument encodé en base64.

Nous nous faisons un petit script Python qui va aller parser l'ensemble du script à la recherche des commandes powershell en Base64, et va les décoder.

Voici à quoi ressemble l'argument de la première commande powershell une fois décodé :

```
p o w e r s h e l l
SPACE
 - e n c
SPACE
function write_file_bytes{param([Byte[]] $file_bytes, [string]
$file_path = ".\stage2.zip"); $f = [io.file]::OpenWrite($file_p
ath); $f.Seek($f.Length,0); $f.Write($file_bytes,0,$file_bytes.L
ength); $f.Close();} function check_correct_environment{$e=[Envi
ronment]::CurrentDirectory.split("\"); $e=$e[$e.Length-1]+[Envi
ronment]::UserName; $e -eq "challenge2015sstic";} if(check_corre
ct_environment){write_file_bytes([Convert]::FromBase64String('
UEsDBAoDAAAAADaKeUbfS/XdEKUHABClBwAJAAAAZW5jcnlwdGVkfL/V3iijvV
ZtKEk8fNyka1PvkbrI0KkNNLxpo0np9mik2+VSO8DpYpziIGfM8vObQ6eCnQea
SrcOE9Lp9sRkxSyqYp/V1tjWu5V1NeJs4Vom37rm7Pdoym+SoGdEyrFay1pbK1
rzSNAfJiLMvtDBPZSwPcYnDg6Irh1U5U+o5MmS+LhV5LFRRTqcbi8INTNCfig/
l1xFdK+/wFGai22ni90Mkzxzz0rV+09nBCHnm1+h5CAr2foL9oe8e11GNNKZt1
mUmf4OB+/kw93C+Bya4HCBjJ3n7v+fBTwZq3D6BrlqRS1cnyZsEGllLopvLxJU
SMwUlL+uD2VBoaRodD/ZCYh2YEMRtTgNF+pN/2O2lQK79RsyO6EQL6RfejxdPY
sPxqvPfw7UsnfjLydxKoPLPMOSxORYo1tcdh81ozeR4Iv3ZaYFUJlTFYSWpCBQ
tZFEuQfhdUTsVTCf0LcVG7fFtMxG9mwr+kez7Wt7VyANSQsKzxiSEU+2x+xAG0
A1U5rDUqOOhdupHI1+TLggOHlbHP2KKbW0dYtRYKomNEWhApdZWhHSU+QF2KYh
FYDx8xGiEUpId8kVvif1tAbNpA+WzmvXyHlofmb3SMXqbngAC8KkHM28Nmjx8D
+6kpyhawnR8YRxJEbNdJOSLnfAGgCqRFQExQqHl4FfDpPQUGM2s1rw4mFDN6xi
wXFKoXtBVUH2EESo67u5TmWmpBUOqPA28iXC/4ZVdn1nqOmOCR2aHXYQBiQ9uJ
G0QOukor8XphyV4mfWHgCnytpVhF61W0kDsid7cullAe52+m0Ze5s28bF22xAj
Rdy0lY+0ALp1UKLUUqJbEYG46h++8VKzNEbKdHjkeSoaDtqFoen0e6C+W67qmM
WxGlZQC4nM1nkH9fkFiOWlPYsDHWHxaBXAJ0HeW3HjJ3Ec0SMNuW7KTM5mH2JU
5BpFshzbbKAh9fa+7GMT9F3ebLaW2KEGQuvkfx5XJQwnc+FPUAyTkf9yoL14vl
G1ieC547zY4kBenCy09IeWOxWCWTyKak+kYAmKsohUEYkF5dKArhFCnZZYmUCn
1kDXftQommJWrGuya69nR+ZvtuQAmSDlgorWgJl/Q89vA7OA+KXB8eTPtBO7Hk
U6xRbIg+It3fChNGOLM6LJvtC/V6FbiXqeXGcYUTiX4/sKlCqbWIllxKQi0jc4
DHZKmaniZGDhtJKxX1zfv1t6OcI0TU5oNdv2GcqWIn0WBATN/spOiFsY9etTfj
Hu7pHGq+khCnRB3REA4AvKyKYfPb+nXHdCvsY4S+s8AJFW2UwdXtOh6gUsBzWn
Xw==''));} else{write_file_bytes([Convert]::FromBase64String('VA
ByAHkASABhAHIAZABlAHIA'));}
ENTER
```

Aaah du code ! Voyons ce que cela donne une fois indenté.

```
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('UEsDBAoDAAAAADaKeUbfS/XdEKU
        HABClBwAJAAAAZW5jcnlwdGVkfL/V3iijvVZtKEk8fNyka1PvkbrI0Kk
        NNLxpo0np9mik2+VSO8DpYpziIGfM8vObQ6eCnQeaSrcOE9Lp9sRkxSy
        qYp/V1tjWu5V1NeJs4Vom37rm7Pdoym+SoGdEyrFay1pbK1rzSNAfJiL
        MvtDBPZSwPcYnDg6Irh1U5U+o5MmS+LhV5LFRRTqcbi8INTNCfig/l1x
        FdK+/wFGai22ni90Mkzxzz0rV+09nBCHnm1+h5CAr2foL9oe8e11GNNK
        Zt1mUmf4OB+/kw93C+Bya4HCBjJ3n7v+fBTwZq3D6BrlqRS1cnyZsEGl
        lLopvLxJUSMwUlL+uD2VBoaRodD/ZCYh2YEMRtTgNF+pN/2O2lQK79Rs
        yO6EQL6RfejxdPYsPxqvPfw7UsnfjLydxKoPLPMOSxORYo1tcdh81oze
        R4Iv3ZaYFUJlTFYSWpCBQtZFEuQfhdUTsVTCf0LcVG7fFtMxG9mwr+ke
        z7Wt7VyANSQsKzxiSEU+2x+xAG0A1U5rDUqOOhdupHI1+TLggOHlbHP2
        KKbW0dYtRYKomNEWhApdZWhHSU+QF2KYhFYDx8xGiEUpId8kVvif1tAb
        NpA+WzmvXyHlofmb3SMXqbngAC8KkHM28Nmjx8D+6kpyhawnR8YRxJEb
```

```
        NdJOSLnfAGgCqRFQExQqHl4FfDpPQUGM2s1rw4mFDN6xiwXFKoXtBVUH
        2EESo67u5TmWmpBUOqPA28iXC/4ZVdn1nqOmOCR2aHXYQBiQ9uJG0QOu
        kor8XphyV4mfWHgCnytpVhF61W0kDsid7cullAe52+m0Ze5s28bF22xA
        jRdy0lY+0ALp1UKLUUqJbEYG46h++8VKzNEbKdHjkeSoaDtqFoen0e6C
        +W67qmMWxGlZQC4nM1nkH9fkFiOWlPYsDHWHxaBXAJ0HeW3HjJ3Ec0SM
        NuW7KTM5mH2JU5BpFshzbbKAh9fa+7GMT9F3ebLaW2KEGQuvkfx5XJQw
        nc+FPUAyTkf9yoLl4vlG1ieC547zY4kBenCy09IeWOxWCWTyKak+kYAm
        KsohUEYkF5dKArhFCnZZYmUCn1kDXftQommJWrGuya69nR+ZvtuQAmSD
        lgorWgJl/Q89vA7OA+KXB8eTPtBO7HkU6xRbIg+It3fChNGOLM6LJvtC
        /V6FbiXqeXGcYUTiX4/sKlCqbWIllxKQi0jc4DHZKmaniZGDhtJKxX1z
        fv1t6OcI0TU5oNdv2GcqWIn0WBATN/spOiFsY9etTfjHu7pHGq+khCnR
        B3REA4AvKyKYfPb+nXHdCvsY4S+s8AJFW2UwdXtOh6gUsBzWnXw==')));
}
else{
    write_file_bytes(
        [Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));
}
```

La fonction `write_file_bytes` prend comme arguments une liste d'octets et un nom de fichier. Elle va ouvrir le fichier, se positionner à la fin, y écrire les données et le refermer. La fonction `check_correct_environment` renvoie True si la concaténation du nom du dossier courant et du nom d'utilisateur donne la chaîne de caractères 'challenge2015sstic'.

Ce code va donc vérifier le nom du dossier courant et le nom de l'utilisateur. Si ceux-ci sont valides, il écrit à la fin du fichier `stage2.zip` les données en base64 qu'il décode. Si la vérification échoue, il affiche la chaîne de caractères 'TryHarder'.

Le Ducky Script est composé d'une succession d'appels à la fonction `write_file_bytes`, avec à chaque fois des données en base64 différente. Une fonction à la fin du fichier est différente.

```
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)));
}
$h = hash_file(".\stage2.zip");
if($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";
}
```

Le but de cette fonction et de tester la validité du fichier `stage2.zip` créé en vérifiant le hash SHA-1 des données qu'il contient. Il doit être égal à "EA 9B 8A 6F 5B 52 7E 72 65 20 19 31 3C 25 B5 6A D2 7C 7E C6".

Nous ajustons notre script Python pour qu'il aille extraire les données en Base64, qu'il les décode et qu'il les mettent dans un fichier `stage2.zip`.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import re
import base64

f = open("duckyscript.txt")
data = f.read()
f.close()

data_out = ""

# Skip first intructions
blocks = re.split("ENTER", data)
blocks = blocks[2:]

for block in blocks:
```

```
    # Extract base64 data
    base_64 = re.split("SPACE", block)[-1]
    base64_without_space = base_64.replace(" ", "")
    txt = base64.b64decode(base64_without_space).replace("\x00", "")

    # Extract base64 data contained in first base64 data
    b1 = re.split(r"if\(check_correct_environment\)\{write_file_bytes\(
                \[Convert\]::FromBase64String\('", txt)[-1]
    b2 = re.split("'", b1)[0]
    data_out += base64.b64decode(b2)

f = open("stage2.zip", "w+")
f.write(data_out)
f.close()
```

```
~/SSTIC2015/stage1$ file stage2.zip
stage2.zip: Zip archive data, at least v1.0 to extract
~/SSTIC2015/stage1$ sha256sum stage2.zip
7f92b7d259be555d863f039f34af4b44752734617d3d891005ae2a8804a7b14a  stage2.zip
```

Nous venons de terminer l'épreuve n° 1, et nous passons maintenant à l'épreuve n° 2.

2 Épreuve 2 : Analyse de la carte de Quake3

2.1 Découverte de l'épreuve

L'épreuve n° 2 commence avec le fichier `stage2.zip` récupéré à l'issue de l'épreuve n° 1. Voyons ce qu'il contient.

```
~/SSTIC2015/stage2$ unzip stage2.zip
Archive:  stage2.zip
 extracting: encrypted
  inflating: memo.txt
  inflating: sstic.pk3
~/SSTIC2015/stage2$ file *
encrypted:   data
memo.txt:    ASCII text
sstic.pk3:   Zip archive data, at least v2.0 to extract
stage2.zip: Zip archive data, at least v1.0 to extract
~/SSTIC2015/stage2$ sha256sum *
91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945  encrypted
80773373402beaa3bcf7b2101514aba3df0975ad6abecbb15a98a64a3275061c  memo.txt
74bdf023f9e09f33b807b9a21047787bfbaf85cf0fb7ab72d82d4dce51003a64  sstic.pk3
7f92b7d259be555d863f039f34af4b44752734617d3d891005ae2a8804a7b14a  stage2.zip
```

L'archive ZIP `stage2.zip` contient 3 fichiers `memo.txt`, `encrypted` et `sstic.pk3`.

Le fichier `memo.txt` contient des instructions relatives à l'étape n° 2.

```
~/SSTIC2015/stage2$ 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
```

Les informations utiles contenues dans ce mémo sont les suivantes :

* Le fichier `encrypted` a été chiffré
* L'algorithme utilisé est l'AES-OFB
* Le vecteur d'initialisation utilisé est 0x5353544943323031352d537461676532, soit "SSTIC2015-Stage2"
* Nous devons trouver une clé de chiffrement.

La clé de chiffrement se trouve probablement dans le 3ème fichier à notre disposition, `sstic.pk3`.

2.2 Analyse du fichier sstic.pk3

Les fichiers PK3 sont des archives utilisées pour les cartes de Quake III Engine. Elles contiennent des informations sur les modèles, les textures, les armes, les sons, les scripts, etc ...

Avant de suivre notre instinct premier et de lancer QuakeIII, nous allons commencer par analyser le contenu du fichier `sstic.pk3` pour voir ce qu'il contient.

```
~/SSTIC2015/stage2$ unzip sstic.pk3
Archive:  sstic.pk3
  inflating: AUTHORS
   creating: levelshots/
  inflating: levelshots/sstic.tga
   creating: maps/
  inflating: maps/sstic.bsp
  inflating: README
   creating: scripts/
  inflating: scripts/sstic.arena
   creating: sound/
   creating: sound/world/
  inflating: sound/world/bj3.wav
   creating: textures/
```

```
   creating: textures/sstic/
  inflating: textures/sstic/01.tga
  inflating: textures/sstic/02.tga
  inflating: textures/sstic/103336131.tga
  inflating: textures/sstic/1036082074.tga
  inflating: textures/sstic/1039223422.tga
  inflating: textures/sstic/1042015984.tga
  inflating: textures/sstic/1065969731.tga
  inflating: textures/sstic/1068346222.tga
  inflating: textures/sstic/110183801.tga
  inflating: textures/sstic/1111555576.tga
  inflating: textures/sstic/1111730476.tga
  [...]
  inflating: textures/sstic/86520831.tga
  inflating: textures/sstic/883915618.tga
  inflating: textures/sstic/902610813.tga
  inflating: textures/sstic/905737444.tga
  inflating: textures/sstic/90736223.tga
  inflating: textures/sstic/928614786.tga
  inflating: textures/sstic/968350285.tga
  inflating: textures/sstic/976571476.tga
  inflating: textures/sstic/994048089.tga
  inflating: textures/sstic/logo.tga
```

A première vue, les dossiers `levelshots`, `sound` et `scripts` n'ont pas l'air super intéressant. En revanche, nous remarquons 2 choses :

* le dossier `map` qui contient la carte QuakeIII `sstic.bsp`
* le dossier `textures` qui contient pas loin de 108 images au format TGA.

Premières conclusions :

* On va essayer de trouver un map editor pour QuakeIII et essayer d'avoir du code source.
* Le concepteur du challenge n'a pas dû créer 108 images diffèrentes juste par plaisir. Il faut jouer.

2.2.1 Indices dans la carte

C'est un peu la solution de facilité que de rechercher un map editor .. mais allons, c'est un challenge de rapidité après tout.

En fouillant un peu sur le net, nous trouvons pleins d'exe qui sont censés faire le boulot (certains inspirent plus confiance que d'autres - toujours avoir une VM Windows sur soi).

Manque de chance ou volonté du concepteur, à chaque fois que nous essayons d'ouvrir la carte `sstic.bsp` dans un des éditeurs trouvés en ligne, nous avons une erreur du type `InvalidID`.

Après plusieurs essais, et voyant que la tâche devient chronophage, nous faisons un `strings` par dépit sur la carte avant d'aller jouer.

```
~/SSTIC2015/stage2/maps$ strings sstic.bsp | grep message
"message" "SSTIC Challenge !  Are you ready ?!"
"message" "Outdoors"
"message" "Outdoors (Near Grenade)"
"message" "Sewer Entrance"
"message" "Generator Room"
"message" "Storage Area"
"message" "Depot Entrance"
"message" "Storage Area"
"message" "Storage Entrance"
"message" "Outdoors (Ceilings)"
"message" "Outdoors (Ceilings)"
"message" "Outdoors (Ceilings)"
"message" "Outdoors (Near Red Armor)"
"message" "Outdoors (Near Rocket)"
"message" "Sewers (Teleporter area)"
"message" "Sewers (Near Railgun)"
"message" "Sewers"
"message" "Entrance"
"message" "Generator Room"
"message" "Depot Area"
"message" "15 seconds ..."
"message" "Welcome n00b !"
```

```
"message" "The secret area \n is now open during \n30 seconds !"
"message" "Yes!\n You found my key !"
"message" "OooOps! \n You failed!"
"message" "Time to Rocket Jump ?!"
```

Nous nous doutons que les messages contenus dans la carte sont des messages qui apparaissent au cours de la partie. Il y aurait une zone secrète à découvrir dans laquelle nous devrions trouver une clé. Nous devrions également devoir faire du `RocketJump` qui est une technique de saut particulière dans Quake III. Elle permet de sauter plus haut et plus loin avec l'aide d'un lance-roquette (en tirant à ses pieds).

2.2.2 Indices dans les textures

Visiblement, le concepteur veut que nous jouions à Quake III. Et pour cela nous l'en remercions. Malheureusement, la fibre n'est pas encore partout, et il nous reste 5 minutes de download pour OpenArena. Profitons-en pour faire un tour dans les textures.

Nous observons que 80 images sur les 108 ont une forme générique. Elles se composent chacune de 3 chaines de 4 octets (chacune avec une couleur diffèrente) et d'un symbole. Il y a 8 symboles diffèrents. Chaque symbole est présent sur 10 images.

Remarquons que les couleurs du symbole du soleil sont inversées par rapport aux autres images. Le symbole de la goutte d'eau se trouve également sur la gauche, et non pas sur la droite de l'image.

Nous observons ensuite que 25 des 108 images ont encore une forme générique. Il s'agit d'un rectangle rouge sur lequel se trouve un des 8 symboles précédents associé à un carré de la couleur des chaines hexadécimales.

Les 3 dernières images sont les suivantes :

A ce stade, nous nous doutons que la clé de chiffrement est inscrite sur les images contenant des chaines héxadécimales. Les images à fond rouge indiquent certainement la couleur de la chaine héxadécimale à prendre suivant le symbole.

N'ayant pas plus d'indices et l'installation d'OpenArena étant terminée, allons maintenant jouer !

2.2.3 Jouons !

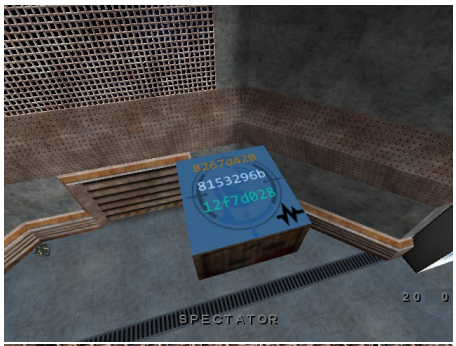
La première étape va être de charger la carte `sstic.pk3` dans OpenArena. Rien de bien compliqué, il suffit de la copier-coller dans le dossier `maps` de l'installation. Nous lançons le jeu.

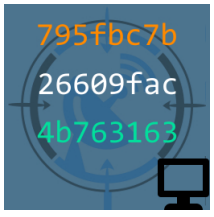
Le mode Single Player fournit une sorte de scénario de jeu, dans lequel il faut gagner une partie sur une carte avant de pouvoir accéder à la suivante. Nous mettons ce mode de côté, pour quand nous aurons fini le challenge SSTIC. Ce qui nous intéresse ici, c'est de lancer la carte SSTIC dès maintenant. Pour cela, nous lançons le mode multi-joueur et nous créons une partie avec la carte SSTIC sans aucun autre joueur ni bot. Cela va nous permettre d'explorer la carte sans se faire tirer dessus.

Nous arrivons sur la carte. Elle n'est pas très grande. Nous reconnaissons rapidement des panneaux accrochés aux murs qui nous semblent familiers. Ces panneaux sont plus ou moins cachés sur la carte.

Astuce : Nous nous mettons en mode spectateur. Ce mode ne permet pas de pouvoir interagir avec la carte, mais permet en revanche de pouvoir s'y balader sans tenir compte de la gravité, ce qui est relativement utile dans notre cas.

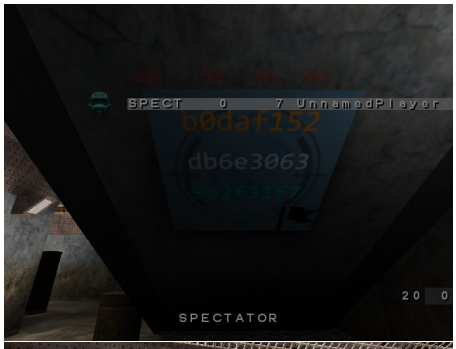
Voila les 8 panneaux présents sur la carte :






SPECT 0 17 UnnamedPlayer
6043dd54
8c34c729
626db3b6
SPECTATOR


6043dd54
8c34c729
626db3b6


SPECT 0 7 UnnamedPlayer
b0daf152
db6e3063
SPECTATOR

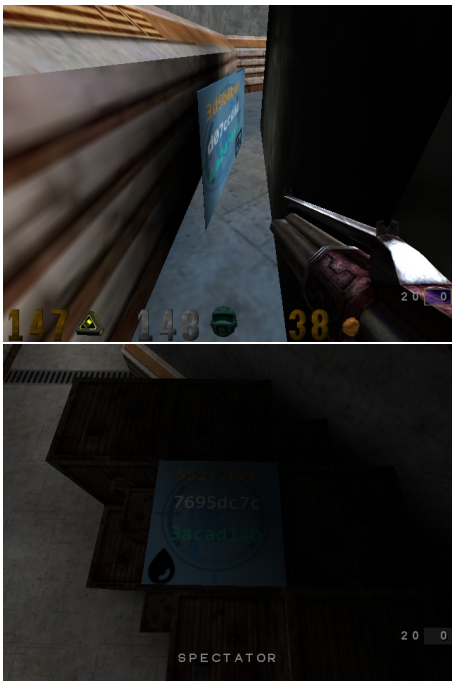

b0daf152
db6e3063
9e2f31f7


b00b7677
14e3ec8b
b54cdc34


b00b7677
14e3ec8b
b54cdc34


SPECT 0 9 UnnamedPlayer
SPECTATOR


2ce017fd
30c419d9
ffe0d355



En continuant notre balade, nous remarquons 2 boutons-poussoirs sur les murs, chacun associé à l'image d'un petit diable. Chacun des boutons influe sur la carte d'une certaine manière. Les boutons s'actionnent soit en tirant dessus, soit en s'en rapprochant

Le bouton n° 1 actionne un mécanisme qui dévoile l'un des panneaux pendant 15 secondes.

Mécanisme actionné :

Le bouton n° 2 actionne un mécanisme qui, d'après le message, ouvre la zone secrète pendant 30 secondes.

Mécanisme actionné (le panneau se lève et dévoile le bouton) :

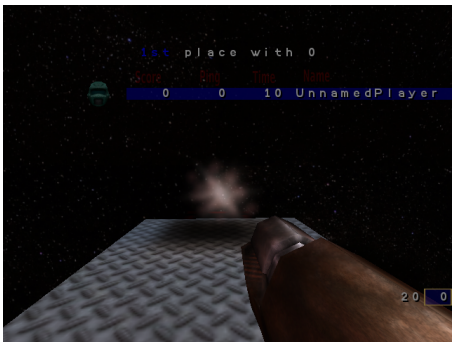
Le mécanisme actionné par le bouton n° 2 dévoile un 3ème bouton qui, en tirant dessus, va nous téléporter dans la zone secrète.

Il s'agit d'une zone de type couloir. Quatre obstacles nous barrent le passage.

Les 2 premiers sont des rivières de lave, facile à franchir avec un saut classique. Le 3ème obstacle est aussi une rivière de lave, mais bien plus large. Un saut classique ne nous fait pas atterrir assez loin, et nous mourons dans d'atroces souffrances fictives.

Pour franchir cet obstacle, nous nous servons d'un `RocketJump`, un saut propulsé par lance-roquette. Ça tombe bien, un stand de lance-roquette est justement positionné au début de la rivière. Le principe est simple, nous courrons en marche arrière en prenant de l'élan, nous sautons et nous tirons sur le sol à nos pieds. En échange de quelques points de vie, nous sautons beaucoup plus loin. Le marché est équitable.

Une fois la 3ème rivière de lave franchie, nous arrivons devant un téléporteur.



Cool ! Nous sommes enfin arrivés au bout ! Sautons dedans à pieds-joints !

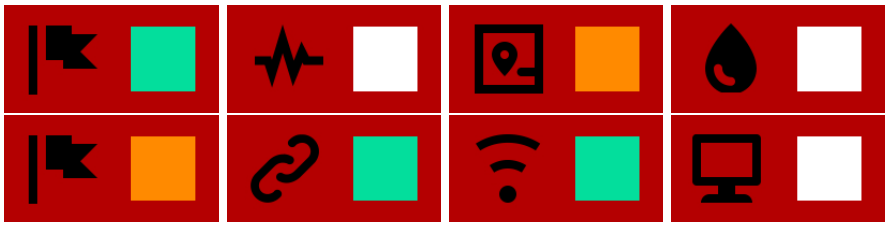
Nous sommes téléportés au début de la carte avec le message 'OooOps You failed !' qui apparait. Nous venons de nous faire avoir.

Le téléporteur est en réalité le 4ème obstacle, il faut donc le contourner. Les côtés n'étant pas assez larges, il nous faut sauter par-dessus avec un `RocketJump`.

Une fois l'obstacle n° 4 franchi, nous arrivons à la fin du couloir, un grand mur noir en face de nous. Nous remarquons un bouton à son sommet.

Le bouton actionne un téléporteur qui nous emmène dans la salle secrète. Le message 'Yes ! You found my key !' apparait. Nous voyons la clé sur le mur.

L'image étant assez sombre, voici la liste des images formant la clé :



Nous remarquons que le symbole `soleil`, bien que présent sur l'un des panneaux de la carte, n'est pas utilisé pour la clé de chiffrement. En revanche, le symbole `drapeau` est utilisé 2 fois. Reprenons les panneaux trouvés précédemment sur la carte et trouvons notre clé de chiffrement

En prenant les bons symboles et les bonne couleurs dans le bon ordre, nous retrouvons la clé de chiffrement :

`9e2f31f7-8153296b-3d9b0ba6-7695dc7c-b0daf152-b54cdc34-ffe0d355-26609fac`

2.3 Déchiffrement du fichier chiffré

Maintenant que nous avons la clé, nous devons déchiffrer le fichier `encrypted`. Faisons ça en Python :

```
from Crypto.Cipher import AES  # pip install pycrypto

iv = "53535449433230313 52d537461676532".decode("hex")
key = "9e2f31f78153296b3d9b0ba67695dc7c" \
      "b0daf152b54cdc34ffe0d35526609fac".decode("hex")
```

```
obj = AES.new(key, AES.MODE_OFB, iv)

m = obj.decrypt(open("encrypted").read())
open("stage3.zip", "w+").write(m)
```

Ce script Python minimaliste déchiffre le fichier chiffré `encrypted` et met les données déchiffrées dans le fichier `decrypted`. Nous vérifions le SHA256 du fichier `stage3.zip` pour voir si il correspond bien au SHA256 dans le fichier `memo.txt`

```
~/SSTIC2015/stage2$ 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

~/SSTIC2015/stage2$ sha256sum stage3.zip
f9ca4432afe87cbb1fca914e35ce69708c6bfa360b82bff21503b6723d1cfbf0  stage3.zip

~/SSTIC2015/stage2$ file stage3.zip
stage3.zip: Zip archive data, at least v1.0 to extract

~/SSTIC2015/stage2$ unzip stage3.zip
Archive:  stage3.zip
 extracting: encrypted
  inflating: memo.txt
  inflating: paint.cap
```

Le hash SHA256 du fichier ne correspond pas à ce que le mémo a annoncé. Cependant, notre fichier déchiffré est bien une archive ZIP contenant les fichiers de l'étape n° 3.

Nous considérons donc l'étape n° 2 comme terminée, et nous passons à l'épreuve n° 3

3 Épreuve 3 : Analyse de la capture USB

3.1 Découverte de l'épreuve

L'épreuve n° 3 commence avec le fichier `stage3.zip` récupéré à l'issue de l'épreuve n° 2. Voyons ce qu'il contient.

```
~/SSTIC2015/stage3$ unzip stage3.zip
Archive:  stage3.zip
 extracting: encrypted
  inflating: memo.txt
  inflating: paint.cap

~/SSTIC2015/stage3$ file *
encrypted:   data
memo.txt:    ASCII text
paint.cap:   tcpdump capture file (little-endian) - version 2.4
             (Memory-mapped Linux USB, capture length 262144)
stage3.zip: Zip archive data, at least v1.0 to extract

~/SSTIC2015/stage3$ sha256sum *
6b39ac2220e703a48b3de1e8365d9075297c0750e9e4302fc3492f98bdf3a0b0  encrypted
04919395e10a25087022209b9c2f423077d280679a17f22e5ed989a5d01708c6  memo.txt
14e3418827ff7b0c5b80baa59f49d71d09c517ed9dcbfd8550a1e909b28a86a1  paint.cap
f9ca4432afe87cbb1fca914e35ce69708c6bfa360b82bff21503b6723d1cfbf0  stage3.zip
```

L'archive ZIP `stage3.zip` contient 3 fichiers `memo.txt`, `encrypted` et `paint.cap`.

Le fichier `memo.txt` contient des instructions relatives à l'étape n° 3.

```
~/SSTIC2015/stage3$ cat memo.txt
Cipher: Serpent-1-CBC-With-CTS
IV: 0x5353544943323031352d537461676533
Key: Well, definitely can't remember it...
     So this time I securely stored it with Paint.

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

Les informations utiles contenues dans ce mémo sont les suivantes :

* Le fichier `encrypted` a été chiffré
* L'algorithme utilisé est Serpent-1 en mode CBC avec du CTS
* Le vecteur d'initialisation utilisé est 0x5353544943323031352d537461676533, soit "SSTIC2015-Stage3"
* Nous devons trouver une clé de chiffrement cachée avec le logiciel de dessin Paint.

La clé de chiffrement se trouve probablement dans le 3ème fichier à notre disposition, `paint.cap`.

Le fichier `paint.cap` est une capture de trames USB.

Voyons avec Wireshark ce que cela donne.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	host	3.0	USB	64	GET DESCRIPTOR Request DEVICE
2	0.000613	3.0	host	USB	82	GET DESCRIPTOR Response DEVICE
3	0.000666	host	2.0	USB	64	GET DESCRIPTOR Request DEVICE
4	0.000850	2.0	host	USB	82	GET DESCRIPTOR Response DEVICE
5	0.000884	host	1.0	USB	64	GET DESCRIPTOR Request DEVICE
6	0.000891	1.0	host	USB	82	GET DESCRIPTOR Response DEVICE
7	2.268500	3.1	host	USB	68	URB_INTERRUPT in
8	2.268535	host	3.1	USB	64	URB_INTERRUPT in
9	2.284496	3.1	host	USB	68	URB_INTERRUPT in
10	2.284535	host	3.1	USB	64	URB_INTERRUPT in
11	2.324491	3.1	host	USB	68	URB_INTERRUPT in
12	2.324532	host	3.1	USB	64	URB_INTERRUPT in
13	2.332455	3.1	host	USB	68	URB_INTERRUPT in
14	2.332492	host	3.1	USB	64	URB_INTERRUPT in
15	2.340494	3.1	host	USB	68	URB_INTERRUPT in
16	2.340526	host	3.1	USB	64	URB_INTERRUPT in
17	2.348451	3.1	host	USB	68	URB_INTERRUPT in
18	2.348481	host	3.1	USB	64	URB_INTERRUPT in
19	2.356490	3.1	host	USB	68	URB_INTERRUPT in

La capture est composée de 6 requêtes/réponses de type `GET_DESCRIPTOR` et 28616 requêtes/réponses de type `URB_INTERRUPT`.

3.2 Analyse des trames USB

Les requêtes `URL_INTERRUPT` contiennent les données USB transmises. Elles sont de type `URB_SUBMIT` quand les données vont de l'hôte vers le périphérique, et de type `URL_COMPLETE` quand les données vont du périphérique vers l'hôte.

D'après l'énoncé, nous devons trouver une clé de chiffrement cachée avec Paint. Nous pouvons supposer que l'utilisateur a utilisé sa souris USB pour dessiner son mot de passe.

Les données envoyées par la souris contiennent des informations relative au positionnement du curseur. Nous devons donc extraire ces coordonnées et les afficher sur une image pour retracer le comportement de l'utilisateur.

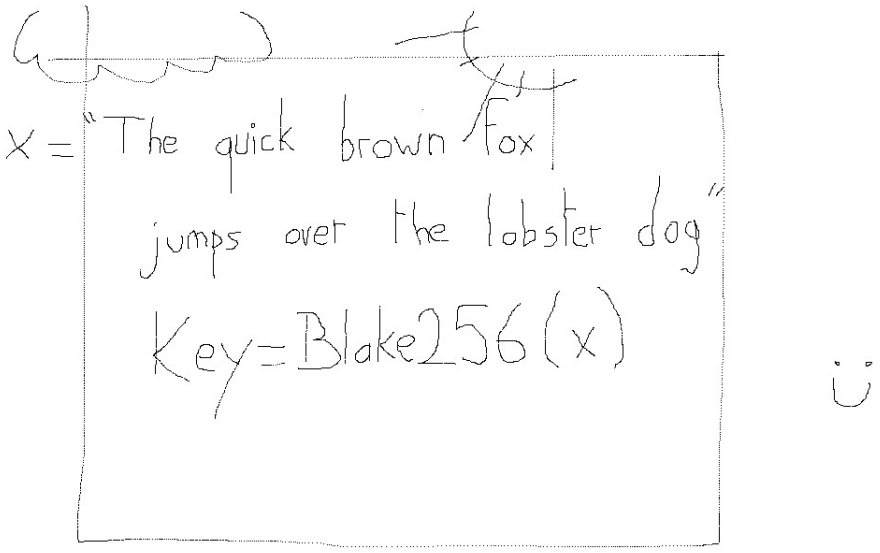
Les trames qui nous intéressent sont celles provenant du périphérique (type `URL_COMPLETE`). Le dernier bloc de 4 octets contient les coordonnées transmises.
Le 1er octet est le statut. Celui-ci définit l'état du périphérique (bouton cliqué, etc ..). Le 2ème octet représente la distance horizontale parcourue depuis la dernière capture. Le 3ème octet représente la distance verticale parcourue depuis la dernière capture.

En faisant quelque recherches sur comment extraire les coordonnées envoyées par la souris à l'hôte, nous tombons sur le site `http://wiremask.eu/boston-key-party-2015-riverside/`. Ce site propose un script (dans la section Mapping) faisant exactement ce dont on a besoin. Quelques modifications mineures sont nécessaires lors de la création de l'image.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import dpkt  # pip install dpkt
import struct
from PIL import Image  # pip install pillow

def extract_mouse_coordinates():
    # Load the pcap file
    pcap_data = dpkt.pcap.Reader(open('paint.cap', 'rb'))
```

```
    # Init a new image
    picture = Image.new('RGB', (1050, 700), 'white')
    pixels = picture.load()

    pixel_x, pixel_y = 0, 100

    for ts, buf in pcap_data:

        # Select only device with ID 3
        if struct.unpack('b', buf[0x0B])[0] != 0x3:
            continue

        # Select only requests with type "COMPLETE" (CODE:0x43)
        if struct.unpack('b', buf[0x08])[0] != 0x43:
            continue

        # Extract data about mouse coordinates
        (status, move_x, move_y, osef) = struct.unpack('bbbb', buf[-4:])
        pixel_x += move_x
        pixel_y += move_y

        # Select only coordinates when mouse is clicked (STATUS:1)
        if status != 1:
            continue

        pixels[pixel_x, pixel_y] = (0, 0, 0, 0)  # Draw point in black

    # Save the generated PNG image
    picture.save('key.png', 'PNG')

extract_mouse_coordinates()
```

Voici à quoi ressemble le comportement de la souris de l'utilisateur lors de la capture.



La clé de chiffrement est donc égale au hash Blake256 de la chaîne de caractères "The quick brown fox jumps over the lobster dog"

Calculons tout de suite ce hash. Nous allons utiliser pour cela un script en Python fait pour calculer les hash Blake256. Ce script est disponible à l'URL `https://raw.githubusercontent.com/grimd34th/PyCash/master/hash/blake.py`.

```
~/SSTIC2015/stage3$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> from blake import *
>>> blake256 = BLAKE(256)
>>> blake256.update("The quick brown fox jumps over the lobster dog")
>>> print " ".join([hex(ord(c))[2:] for c in blake256.final()])
66 c1 ba 5e 8c a2 9a 8a b6 c1 5 a9 be 9e 75 fe b a0 79 97 a8 39 ff ea e9 70 b 0 b7 26 9c 8d
```

Voici notre clé de chiffrement :

`66c1ba5e8ca29a8ab6c15a9be9e75feba07997a839ffeae970b0b7269c8d`

3.3 Déchiffrement du fichier chiffré

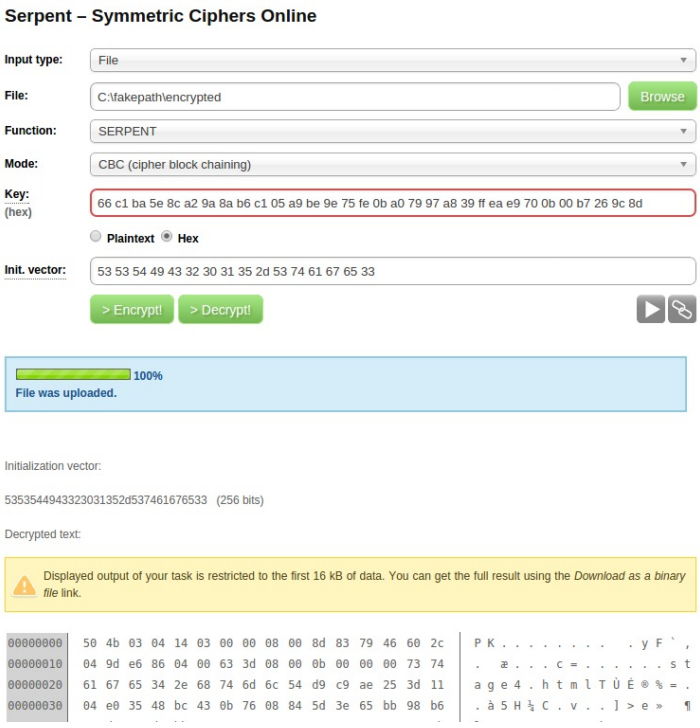
Nous disposons à présent de l'algorithme de chiffrement, de la clé et du vecteur d'initialisation.

L'algorithme est Serpent-1 en mode CBC avec du CTS.

Le terme CTS est une abréviation pour "Ciphertext stealing". Il s'agit d'une méthode d'utilisation des diffèrents modes de chiffrement (ECB, OCB, CBC ..) qui permet de traiter les messages dont la taille n'est pas un multiple de la longueur des blocs.

Wikipédia nous donne la procédure à suivre pour le chiffrement et le déchiffrement utilisant le CTS (`http://en.wikipedia.org/wiki/Ciphertext_stealing#CBC_implementation_notes`).

Ne trouvant pas rapidement de module en Python pour déchiffrer des données avec l'algorithme Serpent-1, nous recherchons un outils pouvant le faire. C'est le cas de l'outil en ligne `http://serpent.online-domain-tools.com/`. Il nous permet de déchiffrer un fichier de données en lui précisant le mode, la clé et le vecteur d'initialisation. Il ne tient pas compte du CTS.



Il semblerait, d'après l'aperçu hexadécimal du contenu du fichier déchiffré, qu'il s'agisse d'un fichier ZIP.

Nous téléchargeons le fichier déchiffré et le sauvegardons sous le nom `decrypted1`.

```
~/SSTIC2015/stage3$ sha256sum decrypted1
d29d61de3cde4e727dedcda2e4745b86a40e70b15edd999548c12557c828a682  decrypted1

~/SSTIC2015/stage3$ file decrypted1
decrypted1: Zip archive data, at least v2.0 to extract

~/SSTIC2015/stage3$ unzip decrypted1
Archive:  decrypted1
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
unzip:  cannot find zipfile directory in one of decrypted1 or
        decrypted1.zip, and cannot find decrypted1.ZIP, period.
```

Le fichier semble bien être une archive ZIP, mais lorsque nous essayons de la décompressé, une erreur se produit. Cela est dû au fait que nous n'avons pas encore pris en compte le CTS.

Nous allons appliquer le CTS manuellement.

Les instructions pour chiffrer les données sont :

* Padder les données en clair avec des 0 de façon à ce que la longueur soit un multiple de la taille de bloc
* Chiffrer les données
* Permuter les 2 derniers blocs
* Tronquer les données chiffrées pour qu'elles soient de la même longueur que les données en clair

Les instructions pour déchiffer les données sont :

* Si la longueur des données chiffrées n'est pas un multiple de la taille des blocs, padder avec les n derniers octets du dernier bloc entièrement déchiffré.
* Permuter les 2 derniers blocs des données chiffrées
* Déchiffrer les données chiffrées
* Tronquer les données déchiffrées à la taille des données chiffrées initiales

Plutôt que de déchiffrer l'avant-dernier bloc pour trouver les 2 octets de padding, nous allons ajouter 2 octets 0x00.

Mettre des 0x00 comme padding va permettre, une fois les données déchiffrées, de connaître la valeur du vrai padding à appliquer. En effet, il s'agira des 2 derniers octets des données déchiffrées.

Cela est dû au fait que le padding a été fait en rajoutant 2 octets 0x00 aux données en clair avant le chiffrement. Il est donc normal que les données déchiffrées possèdent également 2 octets 0x00 à la fin avant tronquage.

Cela étant dit, il nous est facile d'ajuster la valeur du padding suivant le résultat du déchiffrement.

```
Donnees chiffrees:
f824 7d83 880c 9aee 9c07 f655 2166 5117
348d 6035 bc98 4931 bce2 b36a 27ad

Ajout du padding 0x0000 aux donnees chiffrees
f824 7d83 880c 9aee 9c07 f655 2166 5117
348d 6035 bc98 4931 bce2 b36a 27ad 0000

Dechiffrement des donnees chiffrees
8549 4c14 47de 0805 016d 0cb5 2002 75b7
0100 0100 3900 0000 0f87 0400 0000 62d0

Resultats attendu: 0000, et non pas 62D0
Ajustement du padding
8549 4c14 47de 0805 016d 0cb5 2002 75b7
0100 0100 3900 0000 0f87 0400 0000 62d0
Donn es chiffr es:
```

```
f824 7d83 880c 9aee 9c07 f655 2166 5117
348d 6035 bc98 4931 bce2 b36a 27ad

Ajout du padding 0x62d0 aux donnees chiffrees
f824 7d83 880c 9aee 9c07 f655 2166 5117
348d 6035 bc98 4931 bce2 b36a 27ad 62d0

Dechiffrement des donnees chiffrees
6765 342e 6874 6d6c 504b 0506 0000 0000
0100 0100 3900 0000 0f87 0400 0000 0000
```

Il aura donc fallu que nous rajoutions les octets 62 et D0 aux données initiales, puis que nous permutions les 2 derniers blocs pour appliquer le CTS

```
~/SSTIC2015/stage3$ file stage4.zip
stage4.zip: Zip archive data, at least v2.0 to extract

~/SSTIC2015/stage3$ sha256sum stage4.zip
92a9acd154a73ed80ae9a5dd65853325690fec870c966fbffc0de1f1d6616b48  stage4.zip

~/SSTIC2015/stage3$ unzip stage4.zip
Archive:  stage4.zip
  inflating: stage4.html
```

Nous venons de terminer l'épreuve n° 3, nous passons maintenant à l'épreuve n° 4.

4 Épreuve 4 : Analyse du JavaScript

4.1 Découverte de l'épreuve

L'épreuve n° 4 commence avec le fichier `stage4.zip` récupéré à l'issue de l'épreuve n° 3. Voyons ce qu'il contient.

```
~/SSTIC2015/stage4$ unzip stage4.zip
Archive:  stage4.zip
  inflating: stage4.html

~/SSTIC2015/stage4$ file *
stage4.html: HTML document, ASCII text, with very long lines,
             with CRLF line terminators
stage4.zip:  Zip archive data, at least v2.0 to extract

~/SSTIC2015/stage2$ sha256sum *
8d8725995b5d43223f2708eb52f51d8f6bf3e4d72cc8d55422d5d901ec863a87 stage4.html
7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2 stage4.zip
```

L'archive ZIP `stage4.zip` contient 1 fichier `stage4.html`.

Le fichier `stage4.html` contient principalement du code JavaScript. Ce code est composé de données chiffrées contenues dans une variable `data` (528032 caractères), du hash SHA-1 "08c3be636f7dffd91971f65be4cec3c6d162cb1c" et d'une partie obfusquée.

Aperçu de la partie obfusquée :

```
$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$:
({}+"")[$],$$_$:($[$]+"")[$],_$$:++$,$$$_:(!""+"")[$],$__:++$,$_$:++$,$$
__:({}+"")[$],$$_:++$,$$$:++$,$___:++$,$__$:++$};$.$_=($.$_=$+"")[$.$_$]
+($._$=$.$_[$.__$])+($.$$=($.$+"")[$.__$])+((!$)+"")[$._$$]+($.__=$.$_[$
.$$_])+($.$=(!""+"")[$._$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$.__+$._$
+$.$;$.$$=$.$+(!""+"")[$._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[$.$_][$.$_];
$.$($.$($.$$+"\""+"__="+$.$$_$+$._$+$.$$__+$._+"\\"+$.__$+$.$_$+$.$_$+$.
$$$_+"\\"+$.__$+$.$_$+$.$$_+$.__+";$$$='\\"+$.__$+$.$$_+$._$$+$.__+$.$_$
_+"\\"+$.__$+$.$__+$.$$$+$.$$$_+$.$_$+" ';$$_$='"+(![]+"")[$._$_]+$._$+$.
$_$_+$.$$_$+" ';$_$$='\\"+$.$__+$.___+" ';_$$$$$='"+$._+"\\"+$.__$+$.$$_+$

[...]

_____(__________[__________](__________[__________________](______$_)
+_____________,_________________));_$__=_____(__________[__________]
(__________[__________________](_______$)-_________________,___________
_____));_$={};_$[_$_____]=__$_____;_$[___$____]=$_;_$[___________]=_$_
_+"()\{__[____](___$)[__$___]=$;\});}$$[$________](_____________________
_____,$_$);\\"+$.__$+$._$_+"\\"+$.__$+$.__$+"\\"+$.__$+$.__$+"\\"+$.__$+$.__$+"\"")())();
```

D'après l'analyse rapide de ce fichier, nous pouvons formuler les hypothèses suivantes :

* Les données contenues dans la variable `data` sont des données chiffrées correspondant aux données de l'épreuve n° 5.
* Le hash SHA-1 "08c3be636f7dffd91971f65be4cec3c6d162cb1c" est celui des données une fois déchiffrées.
* Il va nous falloir désobfusquer le code JavaScript pour savoir comment les déchiffrer et avec quelle clé.

4.2 Désobfuscation du code JavaScript

Nous commençons par indenter le code. Cela nous permet de repérer la position des variables et des fonctions. Nous affichons la valeur des variables pour voir ce qu'elles contiennent avec la fonction JavaScript `console.log(..)`.

```
<script>
var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
$=~[];
$={                             // | Deobfuscated values:
    ___   : ++$,                // | ___   : 0
```

```
    $$$$ : (![]+"")[$],     // | $$$$ : "f"
    __$  : ++$,             // | __$  : 1
    $_$_ : (![]+"")[$],     // | $_$_ : "a"
    _$_  : ++$,             // | _$_  : 2
    $_$$ : ({}+"")[$],      // | $_$$ : "b"
    $$_$ : ($[$]+"")[$],    // | $$_$ : "d"
    _$$  : ++$,             // | _$$  : 3
    $$$_ : (!""+"")[$],     // | $$$_ : "e"
    $__  : ++$,             // | $__  : 4
    $_$  :++$,              // | $_$  : 5
    $$__ : ({}+"")[$],      // | $$__ : "c"
    $$_  : ++$,             // | $$_  : 6
    $$$  : ++$,             // | $$$  : 7
    $___ : ++$,             // | $___ : 8
    $__$ : ++$              // | $__$ : 9
};

$.$_ = ($.$_=$+"")[$.$_$] + ($._$=$.$_[$.__$]) +        // | $_   : "constructor"
       ($.$$=($.$+"")[$.__$]) + ((!$)+"")[$._$$] +
       ($.__=$.$_[$.$$_]) + ($.$=(!""+"")[$.__$]) +
       ($._=(!""+"")[$._$_]) + $.$_[$.$_$] + $.__  +
       $._$  +  $.$;
$.$$ = $.$ + (!""+"")[$._$$] + $.__ +                   // | $$   : "return"
       $._ + $.$ + $.$$;
$.$  = ($.___)[$.$_][$.$_];         // | $    : function Function() { [native code] }

console.log($);

$.$(    // <-- Function()
        $.$(    // <-- Function()
                $.$$ + "\"" + "__=" + $.$$_$ + $._$ + $.$$__+$._+"\\"+$.__$+$.$_$+$.$_$+
                $.$$$_+"\\"+$.__$+$.$_$+$.$$_+$.__+";$$$='\\"+$.__$+$.$$_+$._$$+$.__+$.$
                _$_+"\\"+$.__$+$.$__+$.$$$+$.$$$_+$.$_$+" ';$$_$='"+(![]+"")[$._$_]+$._$+
                $.$_$_+$.$$_$+" ';$_$$='\\"+$.$__+$.___+" ';_$$$$='"+$._+"\\"+$.__$+$.$$_
                +$._$$+$.$$$_+"\\"+$.__$+$.$$_+$._$_+" ';_$$$='"+$.$$_$+"\\"+$.__$+$.$_$+
                [...]
                .$$__+"\\"+$.__$+$.$_$+$.___+"("+$.$$$$+$._+"\\"+$.__$+$.$_$+$.$$_+$.$$_
                _+$.__+"\\"+$.__$+$.$_$+$.__$+$._$+"\\"+$.__$+$.$_$+$.$$_+"(){__[____](_
                __$)[__$___]=$;});}$$[$________]( ____________________________ ,$_$);\\"+$.
                __$+$._$_+"\\"+$.__$+$.__$+"\\"+$.__$+$.__$+"\""
        )()
)();
</script>
```

Nous repérons un dictionnaire et une création de fonction. Le dictionnaire `$` contient les constantes utilisées pour obfusquer le code. Il y a deux appels à la fonction `Function` imbriqués. Cette fonction crée une nouvelle fonction à partir du code passé en argument sous la forme d'une chaîne de caractères. Regardons ce que contient cet argument.

```
<script>
//$.$(  // <-- Function()
//    $.$(      // <-- Function()
                console.log(
                $.$$ + "\"" + "__=" + $.$$_$ + $._$ + $.$$__+$._+"\\"+$.__$+$.$_$+$.$_$+
                $.$$$_+"\\"+$.__$+$.$_$+$.$$_+$.__+";$$$='\\"+$.__$+$.$$_+$._$$+$.__+$.$
                _$_+"\\"+$.__$+$.$__+$.$$$+$.$$$_+$.$_$+" ';$$_$='"+(![]+"")[$._$_]+$._$+
                $.$_$_+$.$$_$+" ';$_$$='\\"+$.$__+$.___+" ';_$$$$='"+$._+"\\"+$.__$+$.$$_
                +$._$$+$.$$$_+"\\"+$.__$+$.$$_+$._$_+" ';_$$$='"+$.$$_$+"\\"+$.__$+$.$_$+
                [...]
                .$$__+"\\"+$.__$+$.$_$+$.___+"("+$.$$$$+$._+"\\"+$.__$+$.$_$+$.$$_+$.$$_
                _+$.__+"\\"+$.__$+$.$_$+$.__$+$._$+"\\"+$.__$+$.$_$+$.$$_+"(){__[____](_
                __$)[__$___]=$;});}$$[$________]( ____________________________ ,$_$);\\"+$.
                __$+$._$_+"\\"+$.__$+$.__$+"\\"+$.__$+$.__$+"\""
                );
//    )()
//)();
</script>
```

Résultats (tronqués) :

```
<script>
return"__=docu\155e\156t;$$$='\163ta\147e5 ';$$_$='load ';$_$$='\40';_$$$$$='u\163e\162';
[...]
{__[____](___$)[__$___]=$;});}$$[$________]( ____________________________ ,$_$);\12\11\11"
</script>
```

Le résultat ci-dessus est l'argument désobfusqué de l'appel n° 2 à la fonction `Function`. En supprimant le `return`, nous pouvons faire afficher l'argument désobfusqué de l'appel n° 1 à la fonction `Function`.

```
<script>
console.log("__=docu\155e\156t;$$$='\163ta\147e5';$$_$='load';$_$$='\40';_$$$$$='u\163e\162';
[...]
{__[____](___$)[__$___]=$;});}$$[$________](__________________________,$_$);\12\11\11");
</script>
```

Résultats (tronqués) :

```
<script>
__=document;$$$='stage5';$$_$='load';$_$$=' ';_$$$$$='user';_$$$='div';$$_$$$='navigator';
[...]
(___$)[__$___]=$;}});}).catch(function(){__[____](___$)[__$___]=$;});}).catch(function()
{__[____](___$)[__$___]=$;});}$$[$________](__________________________,$_$);
</script>
```

Le dernier résultat montre du code JavaScript plus lisible, mais encore obfusqué. Le code non tronqué et indenté est disponible en annexes A1.

Notre travail consiste à présent à renommer l'ensemble des variables et des fonctions avec des noms corrects, puis à simplifier le code jusqu'à le rendre compréhensible.

Voici le résultat de ce travail :

```
<script>
var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";

message = '<b>Failed to load stage5</b>';
user_agent = window['navigator']['userAgent'];

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 str_to_int_array(data){
    l = [];
    for(i = 0 ; i < data.length ; ++i)
        l.push(data.charCodeAt(i));
    return new Uint8Array(l);
}

function hex_to_int_array(data){
    results = [];
    for(i = 0 ; i < data.length / 2 ; ++i)
        results.push( parseInt( data.substr(i*2, 2), 16));
    return new Uint8Array(results);
}

function int_array_to_hex(arg1){
    v2 = '';
    for(i = 0; i < arg1.byteLength; ++i){
        v1 = arg1[i].toString(16);
        if(v1.length < 2)
            v2 += 0;
        v2 += v1;
    }
    return v2;
}

function main() {
    key = str_to_int_array(user_agent.substr(user_agent.indexOf(')') - 16, 16));
    context = {
        'name': 'AES-CBC',
        'iv': str_to_int_array(user_agent.substr(user_agent.indexOf('(') + 1, 16)),
        'length':  key.length * 8
    };

    window.crypto.subtle.importKey("raw", key, context, false, ['decrypt']).then(
        function(arg_key){
            window.crypto.subtle.decrypt(context, arg_key, hex_to_int_array(data)).then(
                function(plain_data){
```

```
                    window.crypto.subtle.digest({'name':'SHA-1'},
                                                new Uint8Array(plain_data)).then(
                        function(plain_data_sha1){
                            if( hash == int_array_to_hex(new Uint8Array(plain_data_sha1)) ){
                                v1 = {};
                                v1.type = 'application/octet-stream';
                                hash = new Blob([v2], v1);
                                url  =  URL.createObjectURL(hash);
                                document.getElementById('status').innerHTML =
                                    '<a href = "' + url +
                                    '" download="stage5.zip">download stage5</a>';
                            }
                            else{
                                document.getElementById('status').innerHTML = message;
                            }
                        }
                    );
                }
            ).catch(
                function(arg1){
                    document.getElementById('status').innerHTML = message;
                });
        }
    ).catch(
        function(){
            document.getElementById('status').innerHTML = message;
        });
}
window.setTimeout(main, 1000);
</script>
```

Le code JavaScript contient à présent 4 fonctions :

* str_to_int_array : convertit une chaîne de caractères en une liste d'entiers
* hex_to_int_array : convertit une chaîne de caractères héxadécimale en une liste d'entiers
* int_array_to_hex : convertit une liste d'entiers en une chaîne de caractères héxadécimale
* main : fonction principale

La fonction principale a pour rôle de déchiffrer les données contenues dans la variable `data`. Le chiffrement utilisé est l'AES en mode CBC. La clé de chiffrement et le vecteur d'initialisation sont extraits à partir de l'entête HTTP `User-Agent` fournie par notre navigateur. La clé se trouve dans les données entre parenthèses (les 16 derniers octets). Le vecteur se trouve dans les données entre parenthèses (les 16 premiers octets).

Le fichier HTML a fini de nous donner ses principaux indices, et les seuls que nous avons à propos de la clé de chiffrement, c'est qu'il faut la rentrer en tant que User-Agent de notre navigateur.

Que faire ?

4.3 Recherche de la clé de chiffrement

Nous avons fini l'analyse du fichier HTML avec les indices suivants :

* La variable `data` contient des données chiffrées
* L'algorithme utilisé est l'AES-CBC
* La clé et le vecteur d'initialisation doivent être dans le User-Agent de notre navigateur.

Après avoir refait l'épreuve 2-3 fois pour voir si il n'y avait pas d'autres indices, le seul indice que l'on trouve est une `div` cachée :

```
<div><a target="blank"
        href="chrome://browser/content/preferences/preferences.xul">
    Back to preferences</a>
</div>
```

Il s'agit d'un lien vers un fichier du navigateur Mozilla Firefox utilisé pour gérer les préférences.

En combinant tous ces indices, nous savons que la clé de chiffrement et le vecteur d'initialisation doivent être mis dans le User-Agent de notre navigateur, et que ce navigateur devrait être Mozilla Firefox.

Nous sommes bien avancés.

Par dépit, nous allons tester l'ensemble des User-Agent pouvant être utilisés par défaut par Mozilla Firefox, bien que dans l'absolu n'importe qui peut mettre n'importe quoi dans son User-Agent.

Renseignons nous sur le format standard des User-Agent dans Mozilla Firefox avec la documentation officielle : `https://developer.mozilla.org/en-US/docs/Web/HTTP/Gecko_user_agent_string_reference`

Nous apprenons que le User-Agent de Firefox est divisé en 4 composants :

```
Mozilla/5.0 (platform; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
```

Celui qui nous intéresse est le 2ème composant, celui entre parenthèses.

Il est composé du nom de la plateforme de l'utilisateur et de la version de Gecko, le moteur de rendu utilisé par Firefox. Pour les mobiles et les tablettes, les mot-clefs `Mobile` ou `Tablet` peuvent être ajoutés.

Exemple :

```
Mozilla/5.0 (platform; Mobile; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
Mozilla/5.0 (platform; Tablet; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
Mozilla/5.0 (Mobile; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
Mozilla/5.0 (Tablet; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
```

Nous allons donc maintenant coder un brute-forcer qui va tester toutes les plateformes et toutes les versions de Gecko possibles, jusqu'à ce que les données chiffrées puisse être déchiffrées.

- Comment savoir quand est-ce que les données déchiffrées le seront ?

Le nom du fichier créé par le code JavaScript étant `stage5.zip`, nous supposons que le fichier déchiffré sera une archive ZIP.

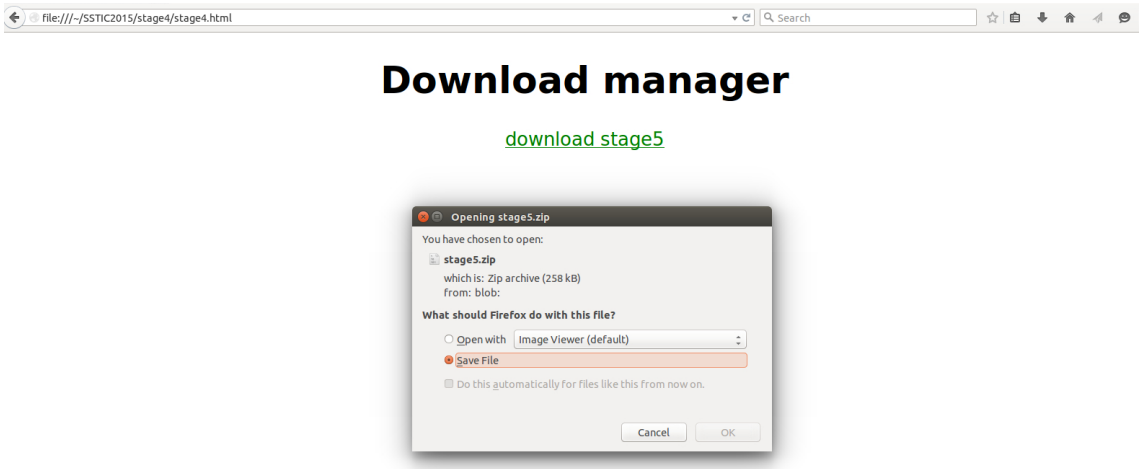
Dans cette optique, nous vérifions plusieurs informations contenues dans l'entête ZIP. Ces informations sont le MAGIC, la tailles des données décompressées, la longueur du nom du fichier et son orthographe.

Le code du bruteforcer est disponible en annexe A.2.

Le bruteforcer crée un fichier `stage5.zip` avec les données déchiffrées si la clé est valide.

```
~/SSTIC2015/stage4/$ python bf.py
UserAgent valide trouve: Macintosh; Intel Mac OS X 10.6; rv:35.0

~/SSTIC2015/stage4/$ sha1sum stage5.zip
56e6d6fee04536467841be1888275cffcdbdb220  stage5.zip

~/SSTIC2015/stage4/$ unzip stage5.zip
Archive:  stage5.zip
  inflating: input.bin
  inflating: schematic.pdf

~/SSTIC2015/stage4/$ sha256sum *
1ab4e1649b4bec560801def3507eb718289ba33092203f3363e4cdaa9a6110f1  input.bin
a32b192d52c1df63149ed5177d0cbcb12b8705f533c3e2efd59224392e308126  schematic.pdf
35f2728dd9b6d4f9811a80d72bb3d7388751913d1420c9d205c118f4d7048303  stage5.zip
```

L'épreuve n° 4 se termine maintenant avec les fichiers de l'épreuves n° 5.



5 Épreuve 5 : Analyse du binaire d'une architecture ST20

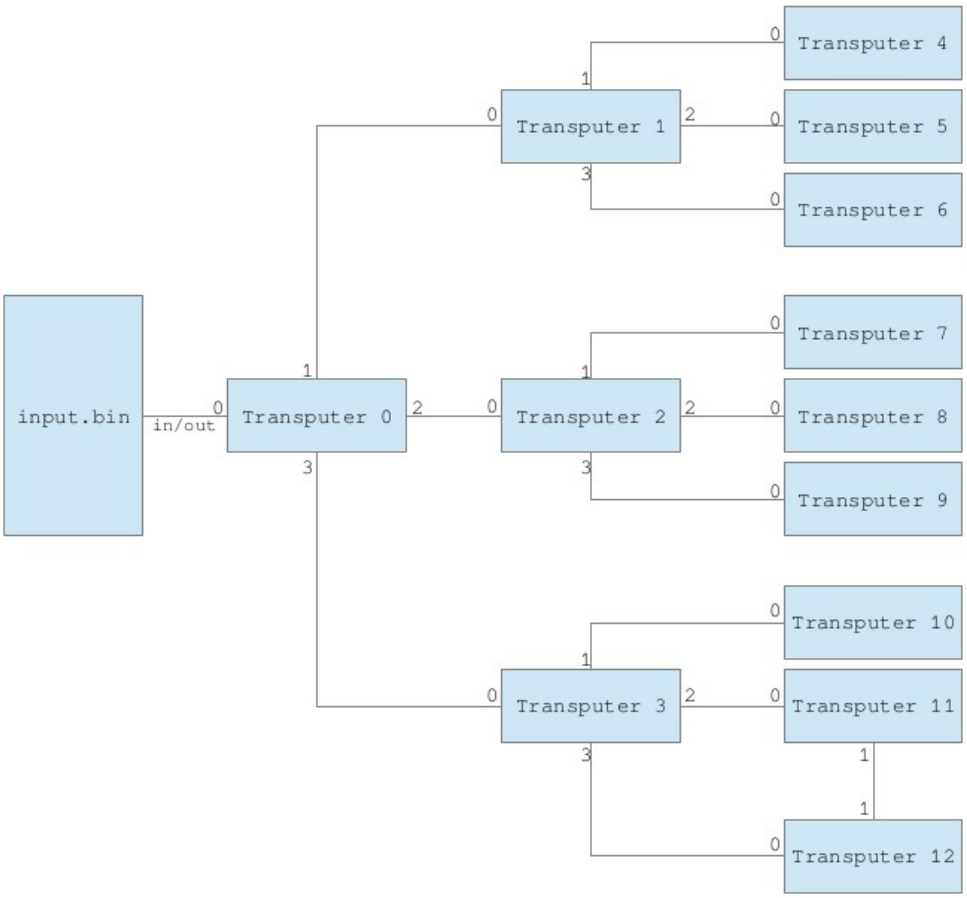
5.1 Découverte de l'épreuve

L'épreuve n° 5 commence avec le fichier `stage5.zip` récupéré à l'issue de l'épreuve n° 4. Voyons ce qu'il contient.

```
~/SSTIC2015/stage5/$ unzip stage5.zip
Archive:  stage5.zip
  inflating: input.bin
  inflating: schematic.pdf

~/SSTIC2015/stage5$ file *
input.bin:     data
schematic.pdf: PDF document, version 1.4
stage5.zip:    Zip archive data, at least v2.0 to extract

~/SSTIC2015/stage5/$ sha256sum *
1ab4e1649b4bec560801def3507eb718289ba33092203f3363e4cdaa9a6110f1  input.bin
a32b192d52c1df63149ed5177d0cbcb12b8705f533c3e2efd59224392e308126  schematic.pdf
35f2728dd9b6d4f9811a80d72bb3d7388751913d1420c9d205c118f4d7048303  stage5.zip
```

L'archive ZIP `stage5.zip` contient 2 fichiers `schematic.pdf` et `input.bin`.

Le fichier `schematic.pdf` contient un plan d'architecture ST20 et des instructions relatives à l'étape n° 5.



SHA256 :

```
        a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81 - encrypted
        9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321 - decrypted
Test vector:
        key = "*SSTIC-2015*"
        data = "1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e".decode("hex")
        decrypt(key, data) == "I love ST20 architecture"
```

Les informations utiles contenues dans ce fichier PDF sont les suivantes :

* Le schéma de l'architecture utilisé pour ce système ST20.
* Les composants de l'architecture sont des `Transputers`.
* Le hash SHA-256 des données chiffrées et des données déchiffrées.
* Des données pour tester un algorithme de chiffrement dont nous devons être l'auteur.

Faisons quelques recherchers sur les architectures ST20 et les transputers.

D'après Wikipédia (`http://en.wikipedia.org/wiki/Transputer`) :

* Le Transputer est une architecture développée [...] dans les années 1980 pour réaliser des **machines parallèles**.
* Le principe repose sur l'utilisation d'une **pile de registres** plutôt qu'un jeu de registres directement adressables.
* Chaque processeur est relié au réseau constitué par l'ensemble des processeurs via des **liens série** rapides.
* Cette structure a été implémenté dans de nombreux produits, dont les **microcontrôleurs ST20** [...].

Voici quelques documents intéressants à lire pour comprendre le fonctionnement des transputers :

`theory.cs.uni-bonn.de/info5/system/parlab/transbook/trans-chap1.ps`
`theory.cs.uni-bonn.de/info5/system/parlab/transbook/trans-chap2.ps`
`theory.cs.uni-bonn.de/info5/system/parlab/transbook/trans-chap4.ps`
`theory.cs.uni-bonn.de/info5/system/parlab/transbook/trans-chap5.ps`
`theory.cs.uni-bonn.de/info5/system/parlab/transbook/trans-chap6.ps`

Nous pouvons extraire, d'après les documents ci-dessus, certaines informations utiles :

> Each transputer has four bidirectional one-bit wide serial ports called "links." A link on one transputer can be connected to a link on any other transputer. Inmos developed a general technique for one process to communicate to another process. Each such communicating process writes to and reads from memory addresses called "channels." Channels can be used for communication between processes inside a single transputer (in that case, the channels can reside almost anywhere in memory), or channels can be used to communicate over links to a process on another transputer (in which case certain fixed memory addresses in on-chip RAM must be used).

Synthèse : Chaque transputer possède 4 ports série bidirectionnels permettant de communiquer avec 4 composants voisins. Ces liaisons sont appellées des liens.

> Link Address (hexadecimal)
>
> Link 3 input 0x8000001C
> Link 2 input 0x80000018
> Link 1 input 0x80000014
> Link 0 input 0x80000010
> Link 3 output 0x8000000C
> Link 2 output 0x80000008
> Link 1 output 0x80000004
>
> Link 0 output 0x80000000 (base address of memory)

Synthèse : Chaque lien est bidirectionnel et possède 2 adresses/identifiants pour son entrée et sa sortie. (Exemple : Lien0 -> IN=0x80000010, OUT=0x80000000)

> Assuming reset (and not analysis), there are two modes that the transputer can be powered up in: boot from ROM or boot from a link. The boot mode is controlled by a pin on the transputer package.

Synthèse : Un transputer peut être démarrer de 2 façons diffèrentes, depuis la ROM ou depuis un lien.

Booting from a Link

If the BootFromROM pin is held low, the transputer will "listen" to its links and try to receive a message from the first link to become active. The message should consist of a small boot program. There are three actions the transputer can take, depending on the value of the first byte of the received message.

If the value of the first byte of the received message is zero, the transputer expects to receive two more words. The first word is an address and the second word is data to write to that address. The transputer writes the data to the address and then returns to its previous state of listening to its links. These messages can be used to initialize memory.

If the value of the first byte of the received message is one, the transputer expects to receive one more word that contains an address. After receiving that word (containing the address), the transputer reads the data at that address and sends it out the output channel of the same link the message came in on. The transputer then returns to its previous state of listening to its links. These messages can be used to query the state of a transputer's memory.

If the value of the first byte of the received message is two or greater, the transputer inputs that number of bytes (2 or greater, whatever the value of the first byte was) into its memory starting at MemStart (the beginning of user memory in on-chip RAM). Then, after receiving the entire message, it transfers execution to MemStart, that is begins running the program that was sent in the message. Note that since the entire message length must be represented in one byte, the maximum size of a boot program is 255 bytes (since the largest number representable in one byte is 255). Such a boot program may in fact be only the first stage of a larger boot program, since the initial boot program may simply be designed to receive a much larger program over a link.

Synthèse : Quand un transputer veut démarrer depuis un lien, il se met en écoute sur ses liens et attend de recevoir un programme de démarrage de moins de 256 octets. Le premier octet envoyé définit l'un des 3 comportements que doit avoir le transputer vis-à-vis des données suivantes.

* Si le premier octet est 0, les 2 DWORD suivants représentent une adresse et une valeur. Le transputer écrit la valeur à l'adresse.
* Si le premier octet est 1, le DWORD suivant désigne une adresse que le transputer doit lire puis renvoyer la valeur.
* Pour tous les autres cas, le premier octet correspond au nombre d'octets que le transputer doit lire sur le lien entrant. Il exécutera ensuite les instructions à partir du 2ème octet.

Nous venons de découvrir que chaque transputer peut être relié jusqu'à 4 autres composants via des liens en séries bidirectionnels. Un transputer possède 2 modes de démarrage, par ROM ou par Lien. Au vu de l'architecture et de la présence de `input.bin` sur le lien 0 du transputer T0, il est fortement probable que les transputers soient en mode Lien. Le fichier de `input.bin` contient les données envoyées au transputer T0 via son lien 0 et correspond donc au programme de démarrage.

Analysons maintenant le contenu de `input.bin` pour connaître le code du Transputer T0.

Dans la suite de l'exercice, nous désignons par WORKSTATION la machine connectée en série sur le transputer T0 sur le lien 0 et qui lui envoie le contenu du fichier `input.bin`.

5.2 Analyse du code envoyé au transputer T0

Le transputer T0 est en mode Lien. Cela signifie qu'il attend de recevoir des instructions via l'un de ses liens. Le premier octet définit le comportement que le transputer doit avoir.

```
~/SSTIC2015/stage5/$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> hex(ord(open("input.bin").read()[0]))
'0xf8'
```

Le premier octet n'étant ni 0 ni 1 mais 0xF8, le transputer T0 va lire les 0xF8 prochains octets et commençer son exécution à partir du 2ème octet reçu.

```
ROM:0000     db 0F8h ;                  ; Number of bytes to read on link0
ROM:0001
ROM:0001         ; ================ S U B R O U T I N E =======================================
ROM:0001
ROM:0001
ROM:0001   T0:
ROM:0001     ajw      0FFFFFFB4h
ROM:0003     ldc      0
ROM:0004     stl      1
ROM:0005     ldc      0
ROM:0006     stl      3
ROM:0007     mint
ROM:0009     ldnlp    400h
ROM:000C     gajw
ROM:000E     ajw      0FFFFFFB4h
ROM:0010     ldc      0C9h ; '+'
ROM:0012     ldpi
ROM:0014     mint
ROM:0016     ldc      8
ROM:0017     out
ROM:0018
ROM:0018   loc_18:                                  ; CODE XREF: T0+35
ROM:0018     ldlp     49h ; 'I'
ROM:001A     mint
ROM:001C     ldnlp    4
ROM:001D     ldc      0Ch
ROM:001E     in                         ; Receive data <dst:wp[49] len:0xC> from <IN0:WORKSTATION>
ROM:001F     ldl      49h ; 'I'
ROM:0021     cj       loc_38            ; Jump if wp[49] == 0
ROM:0023     ldc      0CDh ; '-'
ROM:0025     ldpi
ROM:0027     mint
ROM:0029     ldnlp    4
ROM:002A     ldl      49h ; 'I'
ROM:002C     in                         ; Receive data <dst:0xF4, len:wp[49]> from <IN0:WORKSTATION>
ROM:002D     ldc      0C3h ; '+'
ROM:002F     ldpi
ROM:0031     ldl      4Ah ; 'J'
ROM:0033     ldl      49h ; 'I'
ROM:0035     out                        ; Send data <src:0xF4, len:<wp[49]> to <channel_id>
ROM:0036     j        loc_18
ROM:0038         ; ---------------------------------------------------------------------------
```

Cette première partie de code a pour rôle de recevoir le code des transputers T1, T2 et T3 et de le leur envoyer via les diffèrents liens. Il s'agit d'une fonction de dispatcher.

D'abord, le transputer T0 reçoit une structure `CODE_HEADER` de 12 octets.

```
struct CODE_HEADER {
        DWORD length ;      // longueur des donnees a lire
        DWORD channel_id;   // identifiant du lien sur lequel envoyer ces donnees
        DWORD unused;       // inutilise
};
```

Après réception de cette structure, le transputer T0 lit les `CODE_HEADER.length` prochains octets et les envoie sur le lien `CODE_HEADER.channel_id`. La routine s'arrête lorsqu'elle reçoit une structure vide.

Cette opération permet de fournir le code de chargement aux transputers T1, T2 et T3.

La liste complète des paquets d'initialisation reçus et envoyés par cette routine est disponible en annexe (B.1).

```
ROM:0038         ; ---------------------------------------------------------------------------
ROM:0038
ROM:0038   loc_38:                                  ; CODE XREF: T0+20
ROM:0038     ldlp     49h ; 'I'
ROM:003A     mint
ROM:003C     ldnlp    1
ROM:003D     ldc      0Ch
ROM:003E     out                        ; Send data <src:wp[49], len:0xC> to <OUT1:T1>
ROM:003F     ldlp     49h ; 'I'
ROM:0041     mint
ROM:0043     ldnlp    2
ROM:0044     ldc      0Ch
ROM:0045     out                        ; Send data <src:wp[49], len:0xC> to <OUT:T2>
ROM:0046     ldlp     49h ; 'I'
ROM:0048     mint
ROM:004A     ldnlp    3
ROM:004B     ldc      0Ch
ROM:004C     out                        ; Send data <src:wp[49], len:0xC> to <OUT3:T3>
ROM:004D     ldc      94h ; '  '
ROM:004F     ldpi
ROM:0051     mint
ROM:0053     ldc      8
ROM:0054     out                        ; Send data "Code Ok" <src:E5, len:0x8> to <OUT1:WORKSTATION>
ROM:0055     ldlp     2
ROM:0056     mint
ROM:0058     ldnlp    4
ROM:0059     ldc      4
```

```
ROM:005A    in                      ; Receive data <dst:wp[2], len:0x4> from <IN0:WORKSTATION> -> "KEY:"
ROM:005B    ldlp    5
ROM:005C    mint
ROM:005E    ldnlp   4
ROM:005F    ldc     0Ch
ROM:0060    in                      ; Receive data <dst:wp[5], len:0xC> from <IN0:WORKSTATION> ->
ROM:0061    ldc     88h ; '  '      ;                                   FFFFFFFF FFFFFFFF FFFFFFFF
ROM:0063    ldpi
ROM:0065    mint
ROM:0067    ldc     8
ROM:0068    out                     ; Send data "Decrypt" <src:ED, len:0x8> to <OUT1:WORKSTATION>
ROM:0069    ldlp    3
ROM:006A    mint
ROM:006C    ldnlp   4
ROM:006D    ldc     1
ROM:006E    in                      ; Receive data <dst:wp[3], len:0x1> from <IN0:WORKSTATION)
ROM:006F    ldlp    9
ROM:0070    mint
ROM:0072    ldnlp   4
ROM:0073    ldlp    3
ROM:0074    lb
ROM:0075    in                      ; Receive data <dst:wp[9], len:wp[3]> from <IN0:WORKSTATION> ->
ROM:0076    ldc     0               ;                                     "congratulations.tar.bz2"
ROM:0077    stl     4               ; w4 = 0
```

Voici les actions qui sont menées ici :

* T0 envoie une structure `CODE_HEADER` aux transputers T1, T2 et T3
* T0 envoie "Code ok" à WORKSTATION
* WORKSTATION envoie une clé de 12 octets à T0 (clé = 0xFFFFFFFFFFFFFFFFFFFFFFFF")
* T0 envoie "Decrypt" à WORKSTATION
* WORKSTATION envoie "congratulations.tar.bz2" à T0

```
ROM:0078
ROM:0078  loc_78:                                   ; CODE XREF: T0+DA
ROM:0078    ldlp    1
ROM:0079    mint
ROM:007B    ldnlp   4
ROM:007C    ldc     1
ROM:007D    in                      ; Receive data <dst:wp[1], len:1> from <IN0:WORKSTATION>
ROM:007E    ldlp    5
ROM:007F    mint
ROM:0081    ldnlp   1
ROM:0082    ldc     0Ch
ROM:0083    out                     ; Send data <src:wp[5], len:0xC> to <IN1:T1>
ROM:0084    ldlp    5
ROM:0085    mint
ROM:0087    ldnlp   2
ROM:0088    ldc     0Ch
ROM:0089    out                     ; Send data <src:wp[5], len:0xC> to <OUT2:T2>
ROM:008A    ldlp    5
ROM:008B    mint
ROM:008D    ldnlp   3
ROM:008E    ldc     0Ch
ROM:008F    out                     ; Send data <src:wp[5], len:0xC> to <OUT3:T3>
ROM:0090    ldlp    0
ROM:0091    adc     1
ROM:0092    mint
ROM:0094    ldnlp   5
ROM:0095    ldc     1
ROM:0096    in                      ; Receive data <dst:wp[0]+1, len:1> from <IN1:T1>
ROM:0097    ldlp    0
ROM:0098    adc     2
ROM:0099    mint
ROM:009B    ldnlp   6
ROM:009C    ldc     1
ROM:009D    in                      ; Receive data <wp[0]+2, len:1> from <IN2:T2>
ROM:009E    ldlp    0
ROM:009F    adc     3
ROM:00A0    mint
ROM:00A2    ldnlp   7
ROM:00A3    ldc     1
ROM:00A4    in                      ; Receive data <wp[0]+3, len:1> from <IN3:T3>
ROM:00A5    ldlp    0
ROM:00A6    adc     1
ROM:00A7    lb                      ; // A = wp[0][1]
ROM:00A8    ldlp    0
ROM:00A9    adc     2
ROM:00AA    lb                      ; // A = wp[0][2]
ROM:00AB    xor
ROM:00AD    ldlp    0
ROM:00AE    adc     3
ROM:00AF    lb                      ; // A = wp[0][3]
ROM:00B0    xor                     ; // A = wp[0][1]^wp[0][2]^wp[0][3]
ROM:00B2    ldlp    0
ROM:00B3    adc     1               ; // A = wp[0][1], B=wp[0][1]^wp[0][2]^wp[0][3]
ROM:00B4    sb                      ; wp[0][1] = wp[0][1]^wp[0][2]^wp[0][3]
ROM:00B6    ldlp    1
ROM:00B7    lb                      ; A = byte(addr(wp[1]))
ROM:00B8    ldl     4               ; A = word(wp[4]), B = byte(addr(wp[1]))
ROM:00B9    ldlp    5               ; A = addr(wp[5]), B = word(wp[4]), C = byte(addr(wp[1]))
ROM:00BA    bsub                    ; A = addr(wp[5]) + word(wp[4]), B = byte(addr(wp[1]))
ROM:00BB    lb                      ; A = wp[5][wp[4]], B = byte(addr(wp[1]))
ROM:00BC    ldl     4               ; A = wp[4], B = wp[5][wp[4]], C = byte(addr(wp[1]))
ROM:00BD    ssub                    ; A = wp[4] + 2*(wp[5][wp[4]]), B = byte(addr(wp[1]))
ROM:00BF    xor                     ; A = ((wp[4] + 2*(wp[5][wp[4]])) ^ byte(addr(wp[1])))
ROM:00C1    ldlp    0               ; A = addr(wp[0]), B = ((wp[4] + 2*(wp[5][wp[4]])) ^ byte(addr(wp[1])))
ROM:00C2    sb                      ; wp[0] = ((wp[4] + 2*(wp[5][wp[4]])) ^ byte(addr(wp[1])))% 0x100
ROM:00C4    ldlp    0               ; A = addr(wp[0])
ROM:00C5    adc     1               ; A = addr(wp[0]) + 1
ROM:00C6    lb                      ; A = wp[0][1]
```

```
ROM:00C7    ldl     4               ; A = word(wp[4]), B = wp[0][1]
ROM:00C8    ldlp    5               ; A = addr(wp[5]), B = word(wp[4]), C = wp[0][1]
ROM:00C9    bsub                    ; A = wp[5][wp[4]], B = wp[0][1]
ROM:00CA    sb                      ; wp[5][wp[4]] = wp[0][1]
ROM:00CC    ldl     4               ; A = word(wp[4])
ROM:00CD    adc     1
ROM:00CE    dup
ROM:00D0    stl     4               ; wp[4] += 1
ROM:00D1    eqc     0Ch
ROM:00D2    cj      loc_D6
ROM:00D3    adc     0
ROM:00D4    ldc     0
ROM:00D5    stl     4               ; wp[4] = 0
ROM:00D6
ROM:00D6  loc_D6:                               ; CODE XREF: T0+D1
ROM:00D6    ldlp    0
ROM:00D7    mint                    ;
ROM:00D9    ldc     1
ROM:00DA    out                     ; Send data <src:wp[0], len:0x1> to <OUT0:WORKSTATION>
ROM:00DB    j       loc_78
ROM:00DB  ; End of function T0
ROM:00DB
ROM:00DB        ; ---------------------------------------------------------------------------
ROM:00DD    db "Boot ok",0
ROM:00E5    db "Code Ok",0
ROM:00ED    db "Decrypt",0
ROM:00F5        ; ---------------------------------------------------------------------------
ROM:00F5    ajw     4Ch
ROM:00F7    ret
```

Cette dernière routine est la principale de T0.

* T0 lit un octet W1 provenant de WORKSTATION
* T0 envoie la clé de chiffrement à T1, T2 et T3
* T0 reçoit un octet de T1, T2 et T3
* T0 effectue une opération XOR entre les 3 octets B1, B2 et B3 reçus de T1, T2 et T3
* T0 envoie le résultat de l'opération suivante à WORKSTATION :

$$(i + (2 \times ord(key[i])) \oplus ord(W1))\&0xFF$$

* T0 affecte le résultat de l'opération XOR au ième caractère de la clé de chiffrement, i s'incrémentant à chaque tour de boucle et revenant à 0 lorsqu'il atteint la fin de la clé.

Le transputer T0 exécute cette routine tant que WORKSTATION lui envoie des octets à déchiffrer.

Maintenant que nous connaissons le comportement du transputer T0, nous avons besoin de définir le comportement des transputers T1, T2 et T3 afin de connaître l'algorithme de chiffrement.

Analysons maintenant le code envoyé au transputer T1.

5.3 Analyse du code envoyé au transputers T1, T2 et T3

Les transputers T1, T2 et T3 sont en mode Lien. Cela signifie qu'ils attendent de recevoir des instructions via un de leurs liens. Le premier octet définit le comportement que les transputers doivent avoir.

Ils reçoivent leur code de démarrage de la part du transputer T0. Les premières données réveillent les transputers.

Voici les données de démarrage envoyées par T0 pour T1, T2 et T3.

```
[0x71][T1][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
           75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
           fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
           10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
           f0]

[0x71][T2][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
           75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
           fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
```

```
            10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
            f0]

[0x71][T3][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
            75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
            fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
            10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
            f0]
```

Le code de démarrage est identique pour les transputers T1, T2 et T3, ce qui signifie qu'ils ont le même rôle.

Analysons leur code :

```
ROM:0000        ; ================ S U B R O U T I N E =======================================
ROM:0000
ROM:0000
ROM:0000  T1T2T3:
ROM:0000    ajw     0FFFFFFF8h
ROM:0002    mint
ROM:0004    ldnlp   400h
ROM:0007    gajw
ROM:0009    ajw     0FFFFFFF8h
ROM:000B
ROM:000B  loc_B:                                    ; CODE XREF: T1T2T3+24
ROM:000B    ldlp    5
ROM:000C    mint
ROM:000E    ldnlp   4
ROM:000F    ldc     0Ch
ROM:0010    in                      ; Receive data <dst:wp[5], len:0xC> from <IN0:T0) -> KEY(0xC)
ROM:0011    ldl     5
ROM:0012    cj      loc_26          ; if wp[5] == 0
ROM:0014    ldc     54h ; 'T'
ROM:0016    ldpi                    ; 6C
ROM:0018    mint
ROM:001A    ldnlp   4
ROM:001B    ldl     5
ROM:001C    in                      ; Receive data <dst:0x6C, len:wp[5]> from <IN0:T0)
ROM:001D    ldc     4Bh ; 'K'
ROM:001F    ldpi                    ; 6C
ROM:0021    ldl     6
ROM:0022    ldl     5
ROM:0023    out                     ; Send data <src:0x6C, len:wp[5]> to <wp[6], OutLink?, ?>
ROM:0024    j       loc_B
ROM:0026  ; ---------------------------------------------------------------------------
```

D'abord, les transputers T1, T2 et T3 reçoivent une structure `CODE_HEADER` de 12 octets du transputer T0.

```
struct CODE_HEADER {
        DWORD length ;      // longueur des donnees a lire
        DWORD channel_id;   // identifiant du lien sur lequel envoyer ces donnees
        DWORD unused;       // inutilise
};
```

Après réception de cette structure, les transputers T1, T2 et T3 lisent les `CODE_HEADER.length` prochains octets provenant du transputer T0 et les envoient sur le lien `CODE_HEADER.channel_id` La routine s'arrête lorsqu'elle reçoit une structure vide.

Cette opération permet de fournir le code de chargement aux transputers T4, T5, T6, T7, T8, T9, T10, T11 et T12.

```
ROM:0026  ; ---------------------------------------------------------------------------
ROM:0026
ROM:0026  loc_26:                                   ; CODE XREF: T1T2T3+12
ROM:0026                                            ;       T1T2T3+6A
ROM:0026    ldlp    1
ROM:0027    mint
ROM:0029    ldnlp   4
ROM:002A    ldc     0Ch
ROM:002B    in                      ; Receive data <dst:wp[1], len:0xC> from <IN0:T0)
ROM:002C    ldlp    1
ROM:002D    mint
ROM:002F    ldnlp   1
ROM:0030    ldc     0Ch
ROM:0031    out                     ; Send data <src:wp[1], len:0xC> to <OUT1:T4-T7-T10>
ROM:0032    ldlp    1
ROM:0033    mint
ROM:0035    ldnlp   2
ROM:0036    ldc     0Ch
ROM:0037    out                     ; Send data <src:wp[1], len:0xC> to <OUT2:T5-T8-T11>
ROM:0038    ldlp    1
ROM:0039    mint
ROM:003B    ldnlp   3
ROM:003C    ldc     0Ch
ROM:003D    out                     ; Send data <src:wp[1], len:0xC> to <OUT3:T6-T9-T12>
ROM:003E    ldlp    0
ROM:003F    adc     1
ROM:0040    mint
```

```
ROM:0042    ldnlp    5
ROM:0043    ldc      1
ROM:0044    in                          ; Receive data <dst:wp[0][1], len:0x1> from <IN1:T4-T7-T10)
ROM:0045    ldlp     0
ROM:0046    adc      2
ROM:0047    mint
ROM:0049    ldnlp    6
ROM:004A    ldc      1
ROM:004B    in                          ; Receive data <dst:wp[0][2], len:0x1> from <IN2:T5-T8-T12)
ROM:004C    ldlp     0
ROM:004D    adc      3
ROM:004E    mint
ROM:0050    ldnlp    7
ROM:0051    ldc      1
ROM:0052    in                          ; Receive data <dst:wp[0][3], len:0x1> from <IN3:T6-T8-T12)
ROM:0053    ldlp     0
ROM:0054    adc      1
ROM:0055    lb                          ; A = byte(addr(wp[0]) + 1)
ROM:0056    ldlp     0
ROM:0057    adc      2
ROM:0058    lb                          ; A = byte(addr(wp[0]) + 2), B = byte(addr(wp[0]) + 1)
ROM:0059    xor                         ; A = (byte(addr(wp[0]) + 2) ^ byte(addr(wp[0]) + 1))
ROM:005B    ldlp     0
ROM:005C    adc      3
ROM:005D    lb                          ; A=(byte(addr(wp[0]) + 3), B=(byte(addr(wp[0]) + 2) ^ byte(addr(wp[0]) + 1))
ROM:005E    xor                         ; A=(byte(addr(wp[0]) + 3) ^ byte(addr(wp[0]) + 2) ^ byte(addr(wp[0]) + 1))
ROM:0060    dup                         ; A=A, B=A
ROM:0062    ldlp     0
ROM:0063    sb                          ; wp[0] = byte1^byte2^byte3
ROM:0065    ldlp     0
ROM:0066    mint
ROM:0068    ldc      1
ROM:0069    out                         ; Send data <src:wp[0], len:0x1> to <OUT0:T0>
ROM:006A    j        loc_26
ROM:006A ; End of function T1T2T3
ROM:006A
ROM:006C ; ---------------------------------------------------------------------------
ROM:006C    j        loc_6D
ROM:006D ; ---------------------------------------------------------------------------
ROM:006D
ROM:006D loc_6D:                                ; CODE XREF: ROM:006C
ROM:006D    ajw      8
ROM:006E    ret
```

Cette dernière routine est la principale de T1, T2 et T3.

* T1, T2 et T3 reçoivent la clé de chiffrement envoyée par T0
* T1, T2 et T3 envoient la clé de chiffrement aux transputers T4-T7-T10, T5-T8-T11 et T6-T9-T12.
* T1, T2 et T3 reçoivent un octet B1 provenant respectivement de T4, T7 et T10
* T1, T2 et T3 reçoivent un octet B2 provenant respectivement de T5, T8 et T11
* T1, T2 et T3 reçoivent un octet B3 provenant respectivement de T6, T9 et T12
* T1, T2 et T3 effectuent une opération XOR entre les 3 octets B1, B2 et B3 reçus
* T1, T2 et T3 envoient le résultat de l'opération XOR à T0

Les transputers T1, T2 et T3 exécutent cette routine tant que T0 leur envoie des données.

Maintenant que nous connaissons le comportement des transputers T1, T2 et T3, nous avons besoin de définir le comportement des transputers T4, T5, T6, T7, T8, T9, T10, T11 et T12 afin de connaître l'algorithme de chiffrement.

Analysons maintenant le code envoyé au transputer T4.

5.4 Analyse du code envoyé aux transputers T4, T5, T6, T7, T8, T9,T10, T11 et T12

Le code des transputers T4, T5, T6, T7, T8, T9,T10, T11 et T12 sont similaires au début.

```
ROM:0000 ; ================ S U B R O U T I N E =======================================
ROM:0000
ROM:0000
ROM:0000 transputers_lvl2:
ROM:0000    ajw      0FFFFFFFDh
```

```
ROM:0002    mint
ROM:0004    ldnlp    400h
ROM:0007    gajw
ROM:0009    ajw      0FFFFFFFDh
ROM:000B    ldlp     0
ROM:000C    mint
ROM:000E    ldnlp    4
ROM:000F    ldc      0Ch
ROM:0010    in                          ; Receive data <dst:wp[0], len:0xC> from <IN0:T1-T2-T3)
ROM:0011    ldc      0Bh
ROM:0012    ldpi                        ; 0x1F
ROM:0014    mint
ROM:0016    ldnlp    4
ROM:0017    ldl      0
ROM:0018    in                          ; Receive data <dst:0x1F, len:wp[0]> from <IN0:T1-T2-T3)
ROM:0019    ldc      3
ROM:001A    ldpi                        ; 0x1F
ROM:001C    ldl      2                  ; wp[2]
ROM:001D    bsub
ROM:001E    gcall
ROM:001F    j        loc_20
ROM:0020 ; ---------------------------------------------------------------------------
ROM:0020
ROM:0020 loc_20:                                    ; CODE XREF: transputers_lvl2+1F
ROM:0020    ajw      3
ROM:0021    ret
ROM:0021 ; End of function transputers_lvl2
ROM:0021
ROM:0021 ; ---------------------------------------------------------------------------
ROM:0023    db  20h
```

D'abord, les transputers T4, T5, T6, T7, T8, T9,T10, T11 et T12 reçoivent une structure `CODE_HEADER` de 12 octets du transputer T0.

```
struct CODE_HEADER {
        DWORD length ;      // longueur des donnees a lire
        DWORD unused;       // inutilise
        DWORD exec_rva;     // adresse que l'on doit ex cuter
};
```

Après réception de cette structure, les transputers T4, T5, T6, T7, T8, T9,T10, T11 et T12 lisent les `CODE_HEADER.length` prochains octets provenant des transputers T1-T2-T3. Une fois récupérés, ils exécutent les instructions aux adresses `CODE_HEADER.exec_rva`

```
ROM:0000 ; =============== S U B R O U T I N E =======================================
ROM:0000
ROM:0000
ROM:0000 get_data:                                  ; CODE XREF: T4_main+Fp
ROM:0000    ldl      3
ROM:0001    ldl      2
ROM:0002    ldl      4
ROM:0003    in
ROM:0004    ret
ROM:0004 ; End of function get_data
ROM:0004
ROM:0006
ROM:0006 ; =============== S U B R O U T I N E =======================================
ROM:0006
ROM:0006
ROM:0006 send_data:                                 ; CODE XREF: T4_main+30p
ROM:0006    ldl      3
ROM:0007    ldl      2
ROM:0008    ldl      4
ROM:0009    out
ROM:000A    ret
ROM:000A ; End of function send_data
```

Les transputers T4, T5, T6, T7, T8, T9,T10, T11 et T12 possèdent tous les mêmes fonctions `get_data` et `send_data` dont le but est de respectivement recevoir des données et d'envoyer des données suivant les paramètres qui leurs sont données.

5.5 Analyse du code envoyé au transputer T4

Voici le code désassemblé du transputer T4 :

```
ROM:000C T4_main:
ROM:000C    ajw      0FFFFFFFBh
ROM:000E    ldc      0
ROM:000F    stl      1                  ; wp[1] = 0
ROM:0010    ldc      0
ROM:0011    ldlp     1
ROM:0012    sb                          ; wp[1] = 0
ROM:0014
ROM:0014 loc_14:                                    ; CODE XREF: T4_main+32j
ROM:0014    ldc      0Ch
ROM:0015    stl      0                  ; wp[0] = 0xC
ROM:0016    ldlp     2
ROM:0017    mint
ROM:0019    ldnlp    4
ROM:001A    ldl      6
ROM:001B    call     get_data           ; Receive data <dst:wp[2], len:wp[6]> from <0x80000010, InLink0, T1)
```

```
ROM:001D    ldc     0
ROM:001E    stl     0                   ; wp[0] = 0
ROM:001F
ROM:001F loc_1F:                                    ; CODE XREF: T4_main+28j
ROM:001F    ldl     0                   ; A = wp[0]
ROM:0020    ldlp    2                   ; A = addr(wp[2]), B = wp[0]
ROM:0021    bsub                        ; A = (addr(wp[2]) + wp[0])
ROM:0022    lb                          ; A = (byte(addr(wp[2]) + wp[0]))
ROM:0023    ldlp    1                   ; A = addr(wp[1]), B = (byte(addr(wp[2]) + wp[0]))
ROM:0024    lb                          ; A = wp[1], B = (byte(addr(wp[2]) + wp[0]))
ROM:0025    bsub                        ; A = ((byte(addr(wp[2]) + wp[0])) + wp[1])
ROM:0026    ldc     0FFh
ROM:0028    and                         ; A = (((byte(addr(wp[2]) + wp[0])) + wp[1]) % 0x100)
ROM:002A    ldlp    1
ROM:002B    sb                          ; w1 = (((byte(addr(wp[2]) + wp[0])) + wp[1]) % 0x100)
ROM:002D    ldl     0
ROM:002E    adc     1
ROM:002F    stl     0                   ; wp[0] += 1
ROM:0030    ldc     0Ch
ROM:0031    ldl     0
ROM:0032    gt
ROM:0033    cj      loc_36              ; Jump if wp[0] >= 0xC
ROM:0034    j       loc_1F              ; A = wp[0]
ROM:0036 ; ---------------------------------------------------------------------------
ROM:0036
ROM:0036 loc_36:                                    ; CODE XREF: T4_main+27j
ROM:0036    ldc     1
ROM:0037    stl     0                   ; wp[0] = 1
ROM:0038    ldlp    1
ROM:0039    mint
ROM:003B    ldl     6
ROM:003C    call    send_data           ; Send data <src:wp[1], len:wp[6]> to <0x80000000, OutLink0, T4>
ROM:003E    j       loc_14
ROM:003E ; End of function T4_main
ROM:003E
ROM:003E ; ---------------------------------------------------------------------------
ROM:0041    db  20h
ROM:0042    db  20h
ROM:0043    db  20h
ROM:0043
ROM:0043    end ;
```

Voici les actions effectuées :

* T4 reçoit la clé de chiffrement de T1
* T4 renvoie à T1 la somme de tous les octets de la clé

5.6 Analyse du code envoyé au transputer T5

Voici le code désassemblé du transputer T5 :

```
ROM:000C ; ================ S U B R O U T I N E =======================================
ROM:000C
ROM:000C
ROM:000C transputer5:
ROM:000C    ajw     0FFFFFFFBh
ROM:000E    ldc     0
ROM:000F    stl     1                   ; wp[1] = 0
ROM:0010    ldc     0
ROM:0011    ldlp    1
ROM:0012    sb                          ; wp[1] = 0
ROM:0014
ROM:0014 loc_14:                                    ; CODE XREF: transputer5+32
ROM:0014    ldc     0Ch
ROM:0015    stl     0                   ; wp[0] = 0xC
ROM:0016    ldlp    2
ROM:0017    mint
ROM:0019    ldnlp   4
ROM:001A    ldl     6
ROM:001B    call    get_data            ; Receive data <dst:wp[2], len:wp[6]> from <IN0:T1)
ROM:001D    ldc     0
ROM:001E    stl     0                   ; wp[0] = 0
ROM:001F
ROM:001F loc_1F:                                    ; CODE XREF: transputer5+28
ROM:001F    ldl     0
ROM:0020    ldlp    2
ROM:0021    bsub                        ; A = addr(wp[2]) + wp[0]
ROM:0022    lb                          ; A = byte(addr(wp[2])+wp[0])
ROM:0023    ldl     1                   ; A = wp[1], B = byte(addr(wp[2])+wp[0])
ROM:0024    xor                         ; A = (wp[1] ^ byte(addr(wp[2])+wp[0]))
ROM:0026    ldc     0FFh
ROM:0028    and                         ; A = ((wp[1] ^ byte(addr(wp[2])+wp[0])) % 0x100)
ROM:002A    ldlp    1
ROM:002B    sb                          ; wp[1] = ((wp[1] ^ byte(addr(wp[2])+wp[0])) % 0x100)
ROM:002D    ldl     0
ROM:002E    adc     1
ROM:002F    stl     0                   ; wp[0] += 1
ROM:0030    ldc     0Ch
ROM:0031    ldl     0
ROM:0032    gt
ROM:0033    cj      loc_36              ; Jump if wp[0] >= 0xC
ROM:0034    j       loc_1F
ROM:0036 ; ---------------------------------------------------------------------------
ROM:0036
ROM:0036 loc_36:                                    ; CODE XREF: transputer5+27
ROM:0036    ldc     1
ROM:0037    stl     0                   ; wp[0] = 1
ROM:0038    ldlp    1
ROM:0039    mint
ROM:003B    ldl     6
ROM:003C    call    send_data           ; Send data <src:wp[1], len:wp[6]> to <OUT0:T4>
```

```
ROM:003E    j       loc_14
ROM:003E ; End of function transputer5
ROM:003E
ROM:003E ; ---------------------------------------------------------------------------
ROM:0041    db  20h
ROM:0042    db  20h
ROM:0043    db  20h
ROM:0043
ROM:0043    end ;
```

Voici les actions effectuées :

* T5 reçoit la clé de chiffrement de T1
* T5 renvoie à T1 le XOR de tous les octets de la clé

5.7 Analyse du code envoyé au transputer T6

Voici le code désassemblé du transputer T6 :

```
ROM:000C ; ================ S U B R O U T I N E =======================================
ROM:000C
ROM:000C
ROM:000C sub_C:
ROM:000C    ajw     0FFFFFFF9h
ROM:000E    ldc     0
ROM:000F    stl     2               ; wp[2] = 0
ROM:0010    ldc     0
ROM:0011    stl     1               ; wp[1] = 0
ROM:0012    ldc     0
ROM:0013    stl     3               ; wp[3] = 0
ROM:0014
ROM:0014 loc_14:                                ; CODE XREF: sub_C+70
ROM:0014    ldc     0Ch
ROM:0015    stl     0               ; wp[0] = 0xC
ROM:0016    ldlp    4
ROM:0017    mint
ROM:0019    ldnlp   4
ROM:001A    ldl     8
ROM:001B    call    get_data        ; Receive data <dst:wp[4], len:wp[8]> from <IN0:T1)
ROM:001D    ldl     3
ROM:001E    eqc     0
ROM:001F    cj      loc_3C          ; jump if wp[3] != 0
ROM:0021    ldc     0
ROM:0022    stl     0               ; wp[0] = 0
ROM:0023
ROM:0023 loc_23:                                ; CODE XREF: sub_C+2C
ROM:0023    ldl     0
ROM:0024    ldlp    4
ROM:0025    bsub
ROM:0026    lb                      ; A = byte(addr(wp[4])+w[0])
ROM:0027    ldl     1
ROM:0028    bsub                    ; A = byte(addr(wp[4])+w[0]) + wp[1]
ROM:0029    ldc     0FFFFh
ROM:002D    and                     ; A = ((byte(addr(wp[4])+w[0]) + wp[1]) & 0xFFFF)
ROM:002F    stl     1               ; w1 = ((byte(addr(wp[4])+w[0]) + wp[1]) & 0xFFFF)
ROM:0030    ldl     0
ROM:0031    adc     1
ROM:0032    stl     0               ; wp[0] += 1
ROM:0033    ldc     0Ch
ROM:0034    ldl     0
ROM:0035    gt
ROM:0036    cj      loc_3A          ; if A <= 0xC
ROM:0037    adc     0
ROM:0038    j       loc_23
ROM:003A ; ---------------------------------------------------------------------------
ROM:003A
ROM:003A loc_3A:                                ; CODE XREF: sub_C+2A
ROM:003A    ldc     1
ROM:003B    stl     3               ; wp[3] = 1
ROM:003C
ROM:003C loc_3C:                                ; CODE XREF: sub_C+13
ROM:003C    ldl     1               ; A = wp[1]
ROM:003D    ldc     8000h
ROM:0041    and                     ; A = (wp[1] & 0x8000)
ROM:0043    ldc     0Fh
ROM:0044    shr                     ; A = ((wp[1] & 0x8000) >> 0xF)
ROM:0046    ldl     1
ROM:0047    ldc     4000h
ROM:004B    and
ROM:004D    ldc     0Eh
ROM:004E    shr                     ; A = ((wp[1] & 0x4000) >> 0xE), B = ((wp[1] & 0x8000) >> 0xF)
ROM:0050    xor                     ; A = (((wp[1] & 0x4000) >> 0xE) ^ ((wp[1] & 0x8000) >> 0xF))
ROM:0052    ldc     0FFFFh
ROM:0056    and                     ; A = ((((wp[1] & 0x4000) >> 0xE) ^ ((wp[1] & 0x8000) >> 0xF)) & 0xFFFF)
ROM:0058    ldl     1
ROM:0059    ldc     1
ROM:005A    shl                     ;
ROM:005C    ldc     0FFFFh
ROM:0060    and                     ;
ROM:0062    xor                     ;
ROM:0064    ldc     0FFFFh
ROM:0068    and                     ;
ROM:006A    dup                     ;
ROM:006C    stl     1               ;
ROM:006D    ldc     0FFh
ROM:006F    and                     ;
ROM:0071    ldlp    2
ROM:0072    sb                      ; wp[2] = (((((wp[1]<<1) & 0xFFFF) ^ ((((wp[1] & 0x4000) >> 0xE) ^
ROM:0074    ldc     1               ;           ((wp[1] & 0x8000) >> 0xF)) & 0xFFFF)) & 0xFFFF) & 0xFF)
ROM:0075    stl     0               ; wp[0] = 1
ROM:0076    ldlp    2
```

```
ROM:0077    mint
ROM:0079    ldl     8
ROM:007A    call    send_data       ; Send data <src:wp[2], len:wp[8]> to <OUT0:T4>
ROM:007C    j       loc_14
ROM:007C ; End of function sub_C
ROM:007C
ROM:007C ; ---------------------------------------------------------------------------
ROM:007F    db  20h
ROM:007F
ROM:007F    end ;
```

Voici les actions effectuées :

* T6 reçoit la clé de chiffrement de T1
* Au premier tour de boucle, T6 initialise sa variable `w1` avec la somme de tous les octets de la clé.
* Pour les autres tours de boucles, T6 ajuste sa variable `w1` en suivant l'opération suivante :

$$w1 = ((((w1 \ll 1)\&FFFF) \oplus ((((w1\&4000) \gg E) \oplus ((w1\&8000) \gg F))\&FFFF))\&FFFF)$$

* T6 renvoie à T1 le résultat de l'opération suivante :

$$w1\&FF$$

Code Python correspondant au code de traitement du transputer T6 :

```
def transputer6(key):
    global t6_w1
    global t6_w3

    w4 = key

    if t6_w3 == 0:
        for w0 in range(0xC):
            t6_w1 = (t6_w1 + ord(w4[w0])) & 0xFFFF
        t6_w3 = 1

    t6_w1 = ((((t6_w1 << 1) & 0xFFFF) ^ ((((t6_w1 & 0x4000) >> 0xE) ^
                        ((t6_w1 & 0x8000) >> 0xF)) & 0xFFFF)) & 0xFFFF)
    w2 = t6_w1 & 0xFF

    return w2
```

5.8 Analyse du code envoyé au transputer T7

Voici le code désassemblé du transputer T7 :

```
ROM:000C ; ================ S U B R O U T I N E =======================================
ROM:000C
ROM:000C
ROM:000C Transputer_7:
ROM:000C    ajw     0FFFFFFF9h
ROM:000E    ldc     0
ROM:000F    stl     3               ; wp[3] = 0
ROM:0010
ROM:0010 loc_10:                                ; CODE XREF: sub_C+46
ROM:0010    ldc     0Ch
ROM:0011    stl     0               ; wp[0] = 0xC
ROM:0012    ldlp    4
ROM:0013    mint
ROM:0015    ldnlp   4
ROM:0016    ldl     8
ROM:0017    call    get_data        ; Receive data <dst:wp[4], len:wp[8]> from <IN0:T1)
ROM:0019    ldc     0
ROM:001A    stl     1               ; wp[1] = 0
ROM:001B    ldc     0
ROM:001C    stl     2               ; wp[2] = 0
ROM:001D    ldc     0
ROM:001E    stl     0               ; wp[0] = 0
ROM:001F
ROM:001F loc_1F:                                ; CODE XREF: sub_C+31
ROM:001F    ldl     0               ; A = word(wp[0])
ROM:0020    ldlp    4
ROM:0021    bsub
ROM:0022    lb                      ; A = byte(addr(wp[4])+wp[0])
ROM:0023    ldl     1
ROM:0024    bsub                    ; A = (byte(addr(wp[4])+wp[0]) + wp[1])
ROM:0025    ldc     0FFh
ROM:0027    and                     ; A = ((byte(addr(wp[4])+wp[0]) + wp[1]) & 0xFF)
ROM:0029    stl     1               ; wp[1] = ((byte(addr(wp[4])+wp[0]) + wp[1]) & 0xFF)
```

```
ROM:002A    ldlp    4               ; A = addr(wp[4])
ROM:002B    ldl     0               ; A = wp[0], B = addr(wp[4])
ROM:002C    adc     6               ; A = wp[0] + 6, B = addr(wp[4])
ROM:002D    bsub                    ; A = addr(wp[4]) + wp[0] + 6
ROM:002E    lb                      ; A = byte(addr(wp[4]) + wp[0] + 6)
ROM:002F    ldl     2               ; A = word(wp[2]), B = byte(addr(wp[4]) + wp[0] + 6)
ROM:0030    bsub                    ; A = (byte(addr(wp[4])+wp[0]+6) + wp[2])
ROM:0031    ldc     0FFh
ROM:0033    and                     ; A = ((byte(addr(wp[4])+wp[0]+6) + wp[2]) & 0xFF)
ROM:0035    stl     2               ; wp[2] = (word(wp[2]) + byte(addr(wp[4]) + wp[0] + 6)) % 0x100
ROM:0036    ldl     0
ROM:0037    adc     1
ROM:0038    stl     0               ; wp[0] += 1
ROM:0039    ldc     6
ROM:003A    ldl     0
ROM:003B    gt
ROM:003C    cj      loc_3F          ; Jump if w[0] >= 6
ROM:003D    j       loc_1F          ; A = word(wp[0])
ROM:003F ; ---------------------------------------------------------------------------
ROM:003F
ROM:003F loc_3F:                                    ; CODE XREF: sub_C+30
ROM:003F    ldl     2
ROM:0040    ldl     1
ROM:0041    xor
ROM:0043    ldc     0FFh
ROM:0045    and
ROM:0047    ldlp    3
ROM:0048    sb
ROM:004A    ldc     1
ROM:004B    stl     0               ; wp[0] = 1
ROM:004C    ldlp    3
ROM:004D    mint
ROM:004F    ldl     8
ROM:0050    call    send_data       ; Send data <src:wp[3], len:wp[8]> to <OUT0:T4>
ROM:0052    j       loc_10
ROM:0052 ; End of function sub_C
ROM:0052
ROM:0052 ; ---------------------------------------------------------------------------
ROM:0055    db  20h
ROM:0056    db  20h
ROM:0057    db  20h
ROM:0057
ROM:0057    end ;
```

Voici les actions effectuées :

* T7 reçoit la clé de chiffrement de T2
* T7 fait le XOR de la somme de chaque moitié de la clé
* T7 envoie à T2 le résultat de l'opération XOR

Code Python correspondant au code de traitement du transputer T7 :

```python
def transputer7(key):
    w1 = 0
    w2 = 0
    t7_w3 = 0

    for w0 in range(6):
        w1 = ((ord(key[w0]) + w1) & 0xFF)
        w2 = ((ord(key[w0+6]) + w2) & 0xFF)

    t7_w3 = (w1 ^ w2) & 0xFF
    w0 = 1

    return t7_w3
```

5.9 Analyse du code envoyé au transputer T8

Voici le code désassemblé du transputer T8 :

```
ROM:000C ; ================ S U B R O U T I N E =======================================
ROM:000C
ROM:000C
ROM:000C Transputer_8:
ROM:000C    ajw     0FFFFFFEFh
ROM:000E    ldc     0
ROM:000F    stl     3               ; wp[3] = 0
ROM:0010    ldc     0
ROM:0011    stl     4               ; wp[4] = 0
ROM:0012    ldc     0
ROM:0013    stl     2               ; wp[2] = 0
ROM:0014
ROM:0014 loc_14:                                    ; CODE XREF: sub_C+24
ROM:0014    ldc     0               ; Init un tableau wp[5] de 4 * 0xC
ROM:0015    stl     0               ; wp[0] = 0
ROM:0016
ROM:0016 loc_16:                                    ; CODE XREF: sub_C+1B
ROM:0016    ldc     0
ROM:0017    ldl     2
ROM:0018    ldc     3               ; A = 3, B = wp[2], C = 0
ROM:0019    prod                    ; A = (3*wp[2]), C = 0
ROM:001A    ldlp    5               ; A = addr(wp[5]), B = (3*wp[2]), C = 0
ROM:001B    wsub                    ; A = addr(wp[5]) + 4*(3*wp[2]), B = 0
ROM:001C    ldl     0               ; A = wp[0], B = addr(wp[5]) + 4*(3*wp[2]), C = 0
```

```
ROM:001D    bsub                    ; A = wp[0] + addr(wp[5]) + 4*(3*wp[2]), B = 0
ROM:001E    sb                      ; addr(wp[5]) + 4*(3*wp[2]) + wp[0]  =  0
ROM:0020    ldl     0
ROM:0021    adc     1
ROM:0022    stl     0               ; wp[0] += 1
ROM:0023    ldc     0Ch
ROM:0024    ldl     0
ROM:0025    gt
ROM:0026    cj      loc_29          ; Jump if wp[0] >= 0xC
ROM:0027    j       loc_16
ROM:0029 ; ---------------------------------------------------------------------------
ROM:0029
ROM:0029 loc_29:                                    ; CODE XREF: sub_C+1A
ROM:0029    ldl     2
ROM:002A    adc     1
ROM:002B    stl     2               ; wp[2] += 1
ROM:002C    ldc     4
ROM:002D    ldl     2
ROM:002E    gt
ROM:002F    cj      loc_32          ; Jump if wp[2] >= 4
ROM:0030    j       loc_14          ; Init un tableau wp[5] de 4 * 0xC
ROM:0032 ; ---------------------------------------------------------------------------
ROM:0032
ROM:0032 loc_32:                                    ; CODE XREF: sub_C+23
ROM:0032                            ; sub_C+7Fj
ROM:0032    ldc     0Ch
ROM:0033    stl     0
ROM:0034    ldl     4
ROM:0035    ldc     3
ROM:0036    prod                    ; A = (3*wp[4])
ROM:0037    ldlp    5
ROM:0038    wsub                    ; A = addr(wp[5]) + 12*wp[4]
ROM:0039    mint
ROM:003B    ldnlp   4
ROM:003C    ldl     12h
ROM:003E    call    get_data        ; Receive data <dst:wp[5][12*wp[4]], len:0xC> from <IN0:T2)
ROM:0040    ldl     4
ROM:0041    adc     1
ROM:0042    dup
ROM:0044    stl     4               ; wp[4] += 1
ROM:0045    eqc     4
ROM:0046    cj      loc_4A          ; Jump if wp[4] != 4
ROM:0047    adc     0
ROM:0048    ldc     0
ROM:0049    stl     4               ; wp[4] = 0
ROM:004A
ROM:004A loc_4A:                                    ; CODE XREF: sub_C+3A
ROM:004A    ldc     0
ROM:004B    ldlp    3
ROM:004C    sb                      ; wp[3] = 0
ROM:004E    ldc     0
ROM:004F    stl     2               ; wp[2] = 0
ROM:0050
ROM:0050 loc_50:                                    ; CODE XREF: sub_C+74
ROM:0050    ldc     0
ROM:0051    stl     1
ROM:0052    ldc     0
ROM:0053    stl     0
ROM:0054
ROM:0054 loc_54:                                    ; CODE XREF: sub_C+5F
ROM:0054    ldl     2
ROM:0055    ldc     3
ROM:0056    prod                    ; A = 3*wp[2]
ROM:0057    ldlp    5               ; A = addr(wp[5]), B = 3*wp[2]
ROM:0058    wsub                    ; A = addr(wp[5]) + 12*wp[2]
ROM:0059    ldl     0               ; A = wp[0], B = addr(wp[5]) + 12*wp[2]
ROM:005A    bsub                    ; A = wp[0] + addr(wp[5]) + 12*wp[2]
ROM:005B    lb                      ; A = byte(wp[0] + addr(wp[5]) + 12*wp[2])
ROM:005C    ldl     1               ; A = wp[1], B = byte(wp[0] + addr(wp[5]) + 12*wp[2])
ROM:005D    bsub                    ; A = wp[1] + byte(wp[0] + addr(wp[5]) + 12*wp[2])
ROM:005E    ldc     0FFh
ROM:0060    and                     ; A = (wp[1] + byte(wp[0] + addr(wp[5]) + 12*wp[2])) & 0xFF
ROM:0062    stl     1               ; w1 = (wp[1] + byte(wp[0] + addr(wp[5]) + 12*wp[2])) & 0xFF
ROM:0063    ldl     0
ROM:0064    adc     1
ROM:0065    stl     0               ; wp[0] += 1
ROM:0066    ldc     0Ch
ROM:0067    ldl     0
ROM:0068    gt
ROM:0069    cj      loc_6D          ; Jump if wp[0] >= 0xC
ROM:006A    adc     0
ROM:006B    j       loc_54
ROM:006D ; ---------------------------------------------------------------------------
ROM:006D
ROM:006D loc_6D:                                    ; CODE XREF: sub_C+5D
ROM:006D    ldl     3
ROM:006E    ldl     1
ROM:006F    xor
ROM:0071    ldc     0FFh
ROM:0073    and                     ; A = (wp[1] ^ wp[3]) % 0x100
ROM:0075    ldlp    3
ROM:0076    sb                      ; wp[3] = (wp[1] ^ wp[3]) % 0x100
ROM:0078    ldl     2
ROM:0079    adc     1
ROM:007A    stl     2               ; wp[2] += 1
ROM:007B    ldc     4
ROM:007C    ldl     2
ROM:007D    gt
ROM:007E    cj      loc_82          ; Jump if wp[2] >= 4
ROM:007F    adc     0
ROM:0080    j       loc_50
ROM:0082 ; ---------------------------------------------------------------------------
ROM:0082
ROM:0082 loc_82:                                    ; CODE XREF: sub_C+72
ROM:0082    ldc     1
ROM:0083    stl     0               ; wp[0] = 1
ROM:0084    ldlp    3
```

```
ROM:0085    mint
ROM:0087    ldl     12h
ROM:0089    call    send_data       ; Send data <src:wp[3], len:0xC> to <OUT0:sT2>
ROM:008B    j       loc_32
ROM:008B ; End of function sub_C
ROM:008B
ROM:008B ; ---------------------------------------------------------------------------
ROM:008E    db  20h
ROM:008F    db  20h
ROM:008F
ROM:008F    end ;
```

T8 implémente un tableau pouvant contenir 4 clés de chiffrement différentes. A chaque fois qu'il reçoit une clé, il la met dans la ième case du tableau et incrémente i. Si i dépasse du tableau, il revient à 0.

Voici les actions effectuées :

* T8 reçoit la clé de chiffrement de T2
* T8 fait le XOR de la somme de chacune des 4 clés stockées
* T8 envoie à T2 le résultat de l'opération XOR

Code Python correspondant au code de traitement du transputer T8 :

```
def transputer8(key):
    global t8_w4
    global t8_w5

    w0 = 0xC
    t8_w5 = t8_w5[:12*t8_w4] + key + t8_w5[(12*t8_w4)+0xC:]

    t8_w4 += 1
    if t8_w4 >= 4:
        t8_w4 = 0

    w3 = 0
    for w2 in range(4):
        w0 = 0
        w1 = 0
        for w0 in range(0xC):
            w1 += ord(t8_w5[w0+(12*w2)])
            w1 &= 0xFF
        w3 ^= w1
        w3 &= 0xFF

    w0 = 1
    return w3
```

5.10 Analyse du code envoyé au transputer T9

Voici le code désassemblé du transputer T9 :

```
ROM:000C ; ================ S U B R O U T I N E =======================================
ROM:000C
ROM:000C
ROM:000C Transputer_9:
ROM:000C    ajw     0FFFFFFFBh
ROM:000E    ldc     0
ROM:000F    stl     1               ; wp[1] = 0
ROM:0010
ROM:0010 loc_10:                                ; CODE XREF: sub_C+38
ROM:0010    ldc     0Ch
ROM:0011    stl     0
ROM:0012    ldlp    2
ROM:0013    mint
ROM:0015    ldnlp   4
ROM:0016    ldl     6
ROM:0017    call    get_data        ; Receive data <dst:wp[2], len:wp[6]> from <IN0:T2)
ROM:0019    ldc     0
ROM:001A    ldlp    1
ROM:001B    sb                      ; wp[1] = 0
ROM:001D    ldc     0
ROM:001E    stl     0               ; wp[0] = 0
ROM:001F
ROM:001F loc_1F:                                ; CODE XREF: sub_C+2E
ROM:001F    ldl     0               ; A = wp[0]
ROM:0020    ldlp    2               ; A = addr(wp[2]), B = wp[0]
ROM:0021    bsub                    ; A = addr(wp[2]) + wp[0]
ROM:0022    lb                      ; A = byte(addr(wp[2]) + wp[0])
ROM:0023    ldl     0               ; A = wp[0], B = addr(wp[2]) + wp[0]
ROM:0024    ldc     7               ; A = 7, B = wp[0], C = addr(wp[2]) + wp[0]
ROM:0025    and                     ; A = (7&wp[0]), B = (addr(wp[2])+wp[0])
ROM:0027    shl                     ; A = ((addr(wp[2])+wp[0]) << (7&wp[0]))
ROM:0029    ldl     1               ; A = wp[1], B = ((addr(wp[2])+wp[0]) << (7&wp[0]))
ROM:002A    xor                     ; A = (wp[1] ^ ((addr(wp[2])+wp[0]) << (7&wp[0])))
ROM:002C    ldc     0FFh
ROM:002E    and                     ; A = ((wp[1] ^ ((addr(wp[2])+wp[0]) << (7&wp[0]))) & 0xFF)
ROM:0030    ldlp    1
ROM:0031    sb                      ; wp[1] = ((wp[1] ^ ((addr(wp[2])+wp[0]) << (7&wp[0]))) & 0xFF)
```

```
ROM:0033    ldl     0
ROM:0034    adc     1
ROM:0035    stl     0               ; wp[0] += 1
ROM:0036    ldc     0Ch
ROM:0037    ldl     0
ROM:0038    gt
ROM:0039    cj      loc_3C          ; Jump wp[0] >= 0xC
ROM:003A    j       loc_1F          ; A = wp[0]
ROM:003C ; ---------------------------------------------------------------------------
ROM:003C
ROM:003C loc_3C:                                    ; CODE XREF: sub_C+2D
ROM:003C    ldc     1
ROM:003D    stl     0
ROM:003E    ldlp    1
ROM:003F    mint
ROM:0041    ldl     6
ROM:0042    call    send_data       ; Send data <src:wp[3], len:0xC> to <OUT0:sT2>
ROM:0044    j       loc_10
ROM:0044 ; End of function Transputer_9
ROM:0044
ROM:0044 ; ---------------------------------------------------------------------------
ROM:0047    db  20h
ROM:0047
ROM:0047    end ;
```

Voici les actions effectuées :

* T9 reçoit la clé de chiffrement de T2
* T9 fait l'opération suivante sur chaque octet i de la clé avec w1=0 :

$$w1 \oplus ord(key[i]) \ll (7\&i)$$

* T9 envoie à T2 le résultat de l'opération

Code Python correspondant au code de traitement du transputer T9 :

```
def transputer9(key):
    t9_w1 = 0

    for w0 in range(0xC): # w0
        t9_w1 ^= ord(key[w0]) << (7 & w0)
        t9_w1 &= 0xFF

    w0 = 1
    return t9_w1
```

5.11 Analyse du code envoyé au transputer T10

Voici le code désassemblé du transputer T10 :

```
ROM:000C Transputer_10:
ROM:000C    ajw     0FFFFFFF0h
ROM:000E    ldc     0
ROM:000F    stl     3               ; wp[3] = 0
ROM:0010    ldc     0
ROM:0011    stl     2               ; wp[2] = 0
ROM:0012    ldc     0
ROM:0013    stl     0               ; wp[0] = 0
ROM:0014
ROM:0014 loc_14:                                    ; CODE XREF: Transputer_10+24
ROM:0014    ldc     0               ; Init un tableau wp[4] de 4 * 0xC
ROM:0015    stl     1
ROM:0016
ROM:0016 loc_16:                                    ; CODE XREF: Transputer_10+1B
ROM:0016    ldc     0
ROM:0017    ldl     0
ROM:0018    ldc     3
ROM:0019    prod                    ; A = (3*wp[0]), B = 0
ROM:001A    ldlp    4
ROM:001B    wsub
ROM:001C    ldl     1
ROM:001D    bsub
ROM:001E    sb                      ; addr(wp[4]) + 4*(3*wp[0]) + wp[1]  =  0
ROM:0020    ldl     1
ROM:0021    adc     1
ROM:0022    stl     1               ; wp[1] += 1
ROM:0023    ldc     0Ch
ROM:0024    ldl     1
ROM:0025    gt
ROM:0026    cj      loc_29          ; Jump if wp[1] >= 0xC
ROM:0027    j       loc_16
ROM:0029 ; ---------------------------------------------------------------------------
ROM:0029
ROM:0029 loc_29:                                    ; CODE XREF: Transputer_10+1A
ROM:0029    ldl     0
ROM:002A    adc     1
ROM:002B    stl     0
ROM:002C    ldc     4
ROM:002D    ldl     0
ROM:002E    gt
ROM:002F    cj      loc_32          ; jump if wp[0] >= 4
ROM:0030    j       loc_14          ; Init un tableau wp[4] de 4 * 0xC
ROM:0032 ; ---------------------------------------------------------------------------
```

```
ROM:0032
ROM:0032 loc_32:                                         ; CODE XREF: Transputer_10+23
ROM:0032                                                 ; Transputer_10+7A
ROM:0032    ldc     0Ch
ROM:0033    stl     0
ROM:0034    ldl     2
ROM:0035    ldc     3
ROM:0036    prod                    ; A = (3*wp[2])
ROM:0037    ldlp    4
ROM:0038    wsub                    ; A = addr(wp[4]) + 4*(3*wp[2])
ROM:0039    mint
ROM:003B    ldnlp   4
ROM:003C    ldl     11h
ROM:003E    call    get_data        ; Receive data <dst:wp[4][12*wp[2]], len:wp[0x11]> from <IN0:T3)
ROM:0040    ldl     2
ROM:0041    adc     1
ROM:0042    dup
ROM:0044    stl     2               ; wp[2] += 1
ROM:0045    eqc     4
ROM:0046    cj      loc_4A          ; Jump if wp[2] != 4
ROM:0047    adc     0
ROM:0048    ldc     0
ROM:0049    stl     2               ; wp[2] = 0
ROM:004A
ROM:004A loc_4A:                                         ; CODE XREF: Transputer_10+3A
ROM:004A    ldc     0
ROM:004B    stl     1               ; wp[1] = 0
ROM:004C    ldc     0
ROM:004D    stl     0               ; wp[0] = 0
ROM:004E
ROM:004E loc_4E:                                         ; CODE XREF: Transputer_10+57
ROM:004E    ldl     0               ; A = wp[0]
ROM:004F    ldc     3               ; A = 3, B = wp[0]
ROM:0050    prod                    ; A = (3*wp[0])
ROM:0051    ldlp    4
ROM:0052    wsub                    ; A = addr(wp[4]) + 4*(3*wp[0])
ROM:0053    lb                      ; A = byte(addr(wp[4]) + 4*(3*wp[0]))
ROM:0054    ldl     1
ROM:0055    bsub                    ; A = byte(addr(wp[4]) + 4*(3*wp[0])) + wp[1]
ROM:0056    ldc     0FFh
ROM:0058    and                     ; A = (byte(addr(wp[4]) + 4*(3*wp[0])) + wp[1]) & 0xFF
ROM:005A    stl     1               ; wp[1] = (byte(addr(wp[4]) + 4*(3*wp[0])) + wp[1]) & 0xFF
ROM:005B    ldl     0
ROM:005C    adc     1
ROM:005D    stl     0               ; wp[0] ++ 1
ROM:005E    ldc     4
ROM:005F    ldl     0
ROM:0060    gt
ROM:0061    cj      loc_65          ; Jump if wp[0] >= 4
ROM:0062    adc     0
ROM:0063    j       loc_4E          ; A = wp[0]
ROM:0065 ; ---------------------------------------------------------------------------
ROM:0065
ROM:0065 loc_65:                                         ; CODE XREF: Transputer_10+55
ROM:0065    ldl     1
ROM:0066    ldc     3
ROM:0067    and                     ; A = (wp[1] & 3)
ROM:0069    ldc     3
ROM:006A    prod                    ; A = (3*(wp[1] & 3))
ROM:006B    ldlp    4
ROM:006C    wsub                    ; A = (addr(wp[4]) + 4*(3*(wp[1] & 3)))
ROM:006D    ldl     1
ROM:006E    ldc     4
ROM:006F    shr                     ; A = (wp[1] >> 4), B = (addr(wp[4]) + 4*(3*(wp[1] & 3)))
ROM:0071    ldc     0Ch             ; A = 0xC, B = (wp[1] >> 4), C = (addr(wp[4]) + 4*(3*(wp[1] & 3)))
ROM:0072    rem                     ; A = ((wp[1] >> 4) % 0xC) , B = (addr(wp[4]) + 4*(3*(wp[1] & 3)))
ROM:0074    ldc     0FFh
ROM:0076    and                     ; A = (((wp[1] >> 4) % 0xC) & 0xFF), B = (addr(wp[4]) + 4*(3*(wp[1] & 3)))
ROM:0078    bsub                    ; A = ((((wp[1] >> 4) % 0xC) % 0x100) + (addr(wp[4]) + 4*(3*(wp[1] & 3))))
ROM:0079    lb                      ; A = byte((((wp[1] >> 4) % 0xC) % 0x100) + (addr(wp[4]) + 4*(3*(wp[1] & 3))))
ROM:007A    ldlp    3               ; w3 = byte((((wp[1] >> 4) % 0xC) % 0x100) + (addr(wp[4]) + 4*(3*(wp[1] & 3))))
ROM:007B    sb
ROM:007D    ldc     1
ROM:007E    stl     0               ; wp[0] = 1
ROM:007F    ldlp    3
ROM:0080    mint
ROM:0082    ldl     11h
ROM:0084    call    send_data       ; Send data <src:wp[3], len:wp[11]> to <OUT0:T3>
ROM:0086    j       loc_32
ROM:0086 ; End of function Transputer_10
ROM:0086
ROM:0086 ; ---------------------------------------------------------------------------
ROM:0089    db  20h
ROM:008A    db  20h
ROM:008B    db  20h
ROM:008B
ROM:008B    end ;
```

T10 implémente un tableau pouvant contenir 4 clés de chiffrement différentes. A chaque fois qu'il reçoit une clé, il la met dans la ième case du tableau et incrémente i. Si i dépasse du tableau, il revient à 0.

Voici les actions effectuées :

* T10 reçoit la clé de chiffrement de T3
* T10 fait la somme S1 des premiers octets de chacune des 4 clés
* T10 choisit une clé parmi les 4 en faisant l'opération

$$S1\&3$$

* T10 choisit l'indice du caractère de la clé en faisant l'opération

$$(S1 \gg 4)\%0xC$$

* T10 envoie à T3 le caractère choisi

Code Python correspondant au code de traitement du transputer T10 :

```
def transputer10(key):
    global t10_w1
    global t10_w2
    global t10_w3
    global t10_w4

    t10_w4 = t10_w4[:12*t10_w2] + key + t10_w4[(12*t10_w2)+0xC:]
    t10_w2 += 1
    if t10_w2 == 4:
        t10_w2 = 0

    t10_w1 = 0

    for i in range(4):
        t10_w1 += ord( t10_w4[12*i] )
        t10_w1 &= 0xFF

    t10_w3 = ord(t10_w4[(12*(t10_w1&3))+(((t10_w1>>4)%0xC)&0xFF)])

    return t10_w3
```

5.12 Analyse du code envoyé au transputer T11

Voici le code désassemblé du transputer T11 :

```
ROM:000C Transputer_11:
ROM:000C    ajw     0FFFFFFFAh
ROM:000E    ldc     0
ROM:000F    stl     1               ; wp[1] = 0
ROM:0010    ldc     0
ROM:0011    stl     2               ; wp[2] = 0
ROM:0012
ROM:0012 loc_12:                                    ; CODE XREF: Transputer_11+54
ROM:0012    ldc     0Ch
ROM:0013    stl     0               ; wp[0] = 0xC
ROM:0014    ldlp    3
ROM:0015    mint
ROM:0017    ldnlp   4
ROM:0018    ldl     7
ROM:0019    call    get_data        ; Receive data <dst:wp[3], len:wp[7]> from <IN0:T3)
ROM:001B    ldc     0
ROM:001C    ldlp    1
ROM:001D    sb                      ; wp[1] = 0
ROM:001F    ldlp    3
ROM:0020    lb                      ; A = byte(wp[3])
ROM:0021    ldlp    3               ; A =addr(wp[3]), B = byte(wp[3])
ROM:0022    adc     3
ROM:0023    lb                      ; A = (byte(addr(wp[3])+3)), B = byte(wp[3])
ROM:0024    xor                     ; A = ((byte(addr(wp[3])+3)) ^ byte(wp[3]))
ROM:0026    ldlp    3
ROM:0027    adc     7
ROM:0028    lb                      ; A = (byte(addr(wp[3])+7)), B = ((byte(addr(wp[3])+3)) ^ byte(wp[3]))
ROM:0029    xor                     ; A = ((byte(addr(wp[3])+7)) ^ ((byte(addr(wp[3])+3)) ^ byte(wp[3])))
ROM:002B    ldc     0FFh
ROM:002D    and                     ; A = (((byte(addr(wp[3])+7)) ^ ((byte(addr(wp[3])+3)) ^ byte(wp[3]))) % 0x100)
ROM:002F    ldlp    1
ROM:0030    sb                      ; w1 = (((byte(addr(wp[3])+7)) ^ ((byte(addr(wp[3])+3)) ^ byte(wp[3]))) % 0x100)
ROM:0032    ldc     1
ROM:0033    stl     0               ; wp[0] = 1
ROM:0034    ldlp    1
ROM:0035    mint
ROM:0037    ldnlp   1
ROM:0038    ldl     7
ROM:0039    call    send_data       ; Send data <src:wp[1], len:wp[7]> to <0x80000004, OutLink0, T12>
ROM:003B    ldc     1
ROM:003C    stl     0               ; wp[0] = 1
ROM:003D    ldlp    1
ROM:003E    mint
ROM:0040    ldnlp   5
ROM:0041    ldl     7
ROM:0042    call    get_data        ; Receive data <dst:wp[1], len:wp[7]> from <0x80000014, InLink0, T12)
ROM:0044    ldlp    1
ROM:0045    lb                      ; A = wp[1]
ROM:0046    ldc     0Ch
ROM:0047    rem                     ; A = (wp[1] % 0xC)
ROM:0049    ldc     0FFh
ROM:004B    and                     ; A = ((wp[1] % 0xC) % 0x100)
ROM:004D    ldlp    1
ROM:004E    sb                      ; wp[1] = ((wp[1] % 0xC) % 0x100)
ROM:0050    ldlp    1
ROM:0051    lb                      ; A = byte(addr(wp[1]))
ROM:0052    ldlp    3               ; A = addr(wp[3]), B = byte(addr(wp[1]))
```

```
ROM:0053    bsub
ROM:0054    lb                      ; A = (addr(wp[3]) + wp[1])
ROM:0055    ldlp    2
ROM:0056    sb                      ; wp[2] = (addr(wp[3]) + wp[1])
ROM:0058    ldc     1
ROM:0059    stl     0               ; w[0] = 1
ROM:005A    ldlp    2
ROM:005B    mint
ROM:005D    ldl     7
ROM:005E    call    send_data       ; Send data <src:wp[2], len:wp[7]> to <OUT0:T3>
ROM:0060    j       loc_12
ROM:0060 ; End of function Transputer_11
ROM:0060
ROM:0060 ; ---------------------------------------------------------------------------
ROM:0063    db  20h
ROM:0063
ROM:0063    end ;
```

Voici les actions effectuées :

* T11 reçoit la clé de chiffrement de T3
* T11 calcule un octet X à partir de la clé en faisant l'opération suivante :

$$(ord(key[0]) \oplus ord(key[3]) \oplus ord(key[7]))\&0xFF$$

* T11 envoie à T12 l'octet X
* T11 reçoit un octet Y de T12
* T11 envoie à T3 le caractère en position Y de la clé

Code Python correspondant au code de traitement du transputer T11. Le code a été modifié en incorporant une partie du code du transputer T12 afin d'éviter la communication T11-T12.

```
def transputer11(key):
    global t12_w3
    w3 = key

    '''
    w1 = ( ord(w3[7]) ^ ord(w3[3]) ^ ord(w3[0]) ) & 0xFF
    snd("T11", "T12", chr(w1))
    w1 = rcv("T12", "T11", 0x1)
    '''
    # From t12
    w1 = (ord(t12_w3[1]) ^ ord(t12_w3[5]) ^ ord(t12_w3[9])) & 0xFF
    w1 = (w1 % 0xC) & 0xFF
    w2 = ord(w3[w1])

    return w2
```

5.13 Analyse du code envoyé au transputer T12

Voici le code désassemblé du transputer T12 :

```
ROM:000C Transputer_12:
ROM:000C    ajw     0FFFFFFFAh
ROM:000E    ldc     0
ROM:000F    stl     2               ; wp[2] = 0
ROM:0010    ldc     0
ROM:0011    stl     1               ; wp[1] = 0
ROM:0012    ldc     0
ROM:0013    stl     0               ; wp[0] = 0
ROM:0014
ROM:0014 lo                         ; CODE XREF: Transputer_12+15
ROM:0014    ldc     0               ; Init un tableau w3 avec 12 * "\x00"
ROM:0015    ldl     0
ROM:0016    ldlp    3
ROM:0017    bsub
ROM:0018    sb                      ; addr(wp[3])+wp[0] = 0
ROM:001A    ldl     0
ROM:001B    adc     1
ROM:001C    stl     0               ; wp[0] += 1
ROM:001D    ldc     0Ch
ROM:001E    ldl     0
ROM:001F    gt
ROM:0020    cj      loc_23
ROM:0021    j       loc_14          ; Init un tableau w3 avec 12 * "\x00"
ROM:0023 ; ---------------------------------------------------------------------------
ROM:0023
ROM:0023 loc_23:                                    ; CODE XREF: Transputer_12+14
ROM:0023                                            ; Transputer_12+66
ROM:0023    ldc     0
ROM:0024    ldlp    1
ROM:0025    sb                      ; wp[1] = 0
ROM:0027    ldlp    3
ROM:0028    adc     1
ROM:0029    lb                      ; A = (addr(wp[3])+1)
```

```
ROM:002A    ldlp    3
ROM:002B    adc     5
ROM:002C    lb                      ; A = (addr(wp[3])+5), B = (addr(wp[3])+1)
ROM:002D    xor                     ; A = ((addr(wp[3])+5) ^ (addr(wp[3])+1))
ROM:002F    ldlp    3
ROM:0030    adc     9
ROM:0031    lb
ROM:0032    xor                     ; A = ((addr(wp[3])+9) ^ (addr(wp[3])+5) ^ (addr(wp[3])+1))
ROM:0034    ldc     0FFh
ROM:0036    and                     ; A = (((addr(wp[3])+9) ^ (addr(wp[3])+5) ^ (addr(wp[3])+1)) % 0x100)
ROM:0038    ldlp    1
ROM:0039    sb                      ; wp[1] = (((addr(wp[3])+9) ^ (addr(wp[3])+5) ^ (addr(wp[3])+1)) % 0x100)
ROM:003B    ldc     0Ch
ROM:003C    stl     0               ; wp[0] = 0xC
ROM:003D    ldlp    3
ROM:003E    mint
ROM:0040    ldnlp   4
ROM:0041    ldl     7
ROM:0042    call    get_data        ; Receive data <dst:wp[3], len:wp[7]> from <IN0:T3)
ROM:0044    ldc     1
ROM:0045    stl     0               ; wp[0] = 1
ROM:0046    ldlp    2
ROM:0047    mint
ROM:0049    ldnlp   5
ROM:004A    ldl     7
ROM:004B    call    get_data        ; Receive data <dst:wp[2], len:wp[7]> from <IN1:T11)
ROM:004D    ldc     1
ROM:004E    stl     0               ; wp[0] = 1
ROM:004F    ldlp    1
ROM:0050    mint
ROM:0052    ldnlp   1
ROM:0053    ldl     7
ROM:0054    call    send_data       ; Send data <src:wp[1], len:wp[7]> to <OUT0:T11>
ROM:0056    ldlp    2
ROM:0057    lb
ROM:0058    ldc     0Ch
ROM:0059    rem
ROM:005B    ldc     0FFh
```

Voici les actions effectuées :

* T12 calcule un octet X à partir de la clé en faisant l'opération suivante :

$$(ord(key[1]) \oplus ord(key[5]) \oplus ord(key[9]))\&0xFF$$

* T12 reçoit la clé de chiffrement de T3
* T12 reçoit un octet Y de T11
* T12 envoie à T11 l'octet X
* T12 envoie à T3 la caractère en position Y de la clé

Code Python correspondant au code de traitement du transputer T12. Le code a été modifié en incorporant une partie du code du transputer T11 afin d'éviter la communication T12-T11.

```
def transputer12(key):
    global t12_w3
    '''
    w1 = (ord(w3[1]) ^ ord(w3[5]) ^ ord(w3[9])) & 0xFF
    t12_w3 = rcv("T3", "T12", 0xC)
    w2 = rcv("T11", "T12", 0x1)
    '''
    t12_w3 = key

    # From t11
    w2 = ( ord(key[7]) ^ ord(key[3]) ^ ord(key[0]) ) & 0xFF

    #snd("T12", "T11", chr(w1))

    w2 = (w2 % 0xC) & 0xFF
    w1 = ord(t12_w3[w2])
    return w1
```

5.14 Cassage de la clé par bruteforce

Nous pouvons à présent simuler le comportement de tous les transputers. Globalement, le code founi aux transputers par `input.bin` prend en entrée des données chiffrées et une clé de chiffrement, et renvoie en sortie les données déchiffrées.

Le code Python correspondant au code fourni aux transputers est disponible en annexe (B.2).

Nous devons à présent trouver la clé de chiffrement utilisée.

N'ayant aucun indice, nous allons analyser l'algorithme de chiffrement pour vérifier sa robustesse.

```
while 1:
        # For each characters in the key
        for i in range(0xC):

                # Return results when there is no more data to decrypt
                if len(encrypted) <= 0:
                        return out, key

                # Get the following encrypted byte
                data = encrypted[0]
                encrypted = encrypted[1:]

                # Send key to each transputer
                b4 = transputer4(key)
                b5 = transputer5(key)
                b6 = transputer6(key)

                b7 = transputer7(key)
                b8 = transputer8(key)
                b9 = transputer9(key)

                b10 = transputer10(key)
                b11 = transputer11(key)
                b12 = transputer12(key)

                # Combine transputers's results
                k = b4 ^ b5 ^ b6 ^ b7 ^ b8 ^ b9 ^ b10 ^ b11 ^ b12

                # Decrypt the encrypted byte
                out += chr((i + (2 * ord(key[i])) ^ ord(data)) & 0xFF)

                # Update the key
                key = key[:i] + chr(k) + key[i + 1:]
```

Dans cet algorithme, nous pouvons voir que l'octet chiffré est déchiffré avant que la clé ne soit mise à jour avec les octets reçus des transputers. La clé n'est mise à jour qu'au 2ème tour de boucle. Cela signifie qu'en connaissant les 12 premiers octets du message déchiffré, nous pouvons définir l'ensemble des clés permettant d'arriver à un résultat correspondant à ces 12 octets.

Il nous faut maintenant trouver un moyen de connaître les 12 premiers octets du message déchiffré.

Pour cela, un indice est fourni par WORKSTATION au transputer T0. Il s'agit du nom du fichier pour les données déchiffrées, `congratulations.tar.bz2`.

Regardons de plus près le format d'un fichier TAR.BZ2 (`http://en.wikipedia.org/wiki/Bzip2`) :

```
.magic:16                       = 'BZ' signature/magic number
.version:8                      = 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1 (deprecated)
.hundred_k_blocksize:8          = '1'..'9' block-size 100 kB-900 kB (uncompressed)

.compressed_magic:48            = 0x314159265359 (BCD (pi))
.crc:32                         = checksum for this block
.randomised:1                   = 0=>normal, 1=>randomised (deprecated)
.origPtr:24                     = starting pointer into BWT for after untransform
.huffman_used_map:16            = bitmap, of ranges of 16 bytes, present/not present
.huffman_used_bitmaps:0..256    = bitmap, of symbols used, present/not present (multiples of 16)
.huffman_groups:3               = 2..6 number of different Huffman tables in use
.selectors_used:15              = number of times that the Huffman tables are swapped (each 50 bytes)
*.selector_list:1..6            = zero-terminated bit runs (0..62) of MTF'ed Huffman table (*selectors
.start_huffman_length:5         = 0..20 starting bit length for Huffman deltas
*.delta_bit_length:1..40        = 0=>next symbol; 1=>alter length
                                                { 1=>decrement length;  0=>increment length } (*(symb
.contents:2..                     = Huffman encoded data stream until end of block (max. 7372800 bit)

.eos_magic:48                   = 0x177245385090 (BCD sqrt(pi))
.crc:32                         = checksum for whole stream
.padding:0..7                   = align to whole byte
```

Nous avons des 12 premiers octets d'un fichier TAR.BZ2, en espérant qu'ils soient génériques. Voyons ça :

* Octets 1 et 2 : MAGIC (toujours égale à "BZ")
* Octet 3 : Numéro de version. (S'agissant d'un fichier BZ2, la valeur est "h")
* Octet 4 : Le taux de compression. La valeur peut aller de 1 à 9. En comparant plusieurs fichiers TAR.BZ2, nous observons que le taux de compression est toujours à 9. Cela doit être le comportement par défaut.
* Octets 5 à 10 : COMPRESSED_MAGIC (toujours égale à 0x314159265359)
* Octets 10 à 14 : le CRC. Sa valeur dépend de l'ensemble des données déchiffrées.

Au vu du format de l'entête TAR.BZ2 et en émettant l'hypothèse que le taux de compression utilisé est celui par défaut, nous pouvons deviner les 10 premiers octets du message déchiffré, soit :

* BZh91AY&SY

Il nous manque 2 octets du message déchiffré pour pouvoir avoir une liste complète de clés potentielles de 12 octets.

Ce n'est pas grave. A partir des 10 premiers octets du message déchiffré, nous pouvons trouver toutes les clés valides de 10 octets. Avec ces 'graines' de clés, il faudra ensuite en déduire toutes les clés possibles et bruteforcer les 2 derniers octets.

Commençons par trouver toutes les graines de 10 octets possibles.

Pour cela, faisons un script Python qui va, pour chaque caractère de la clé, tester tous les octets possibles jusqu'à avoir l'octet déchiffré attendu. Une fois que nous connaîtrons la liste des octets valides pour chaque caractère de la clé, nous en déduirons l'ensemble des graines possibles.

```
def crack_seeds(header="BZh91AY&SY"):
    ''' Generate all seeds which decrypt data starting
                by BZh91AY&SY and put them into seeds.txt
        '''

    start_time = time.time()

    # Extract the 36 first bytes of encrypted data from input.bin
    f = open("input.bin")
    encrypted = f.read()[0x9AD:0x9AD + 0xC * 3]
    f.close()
    key_found = list()

    for j in range(len(header)):
        possible_bytes = list()

        for i in range(0x100):

            # Decrypt the byte
            decrypted_byte = ((j + (2 * i)) ^ ord(encrypted[j])) & 0xFF

            # If the decrypted byte match the TAR.BZ2 header,
            # add it to the list of possible valid bytes
            if chr(decrypted_byte) == header[j]:
                possible_bytes.append(i)

        key_found.append(possible_bytes)

    print "Possible bytes found:"
    for i, k in enumerate(key_found):
        print i, [hex(x)[2:] for x in k]

    # Generate all possible seeds from all possible bytes
    possible_seeds = [""]
    for j in range(len(header)):

        possible_seeds_2 = list()
        for seed in possible_seeds:

            for i in key_found[j]:
                possible_seeds_2.append(seed + chr(i))

        if len(possible_seeds_2) == 0:
            return "No match found at step %s" % j
```

```
        possible_seeds = possible_seeds_2
        del possible_seeds_2

    print "Possible seeds found: %s " % len(possible_seeds)
    f = open("seeds.txt", "w+")
    for k in possible_seeds:
        f.write(" ".join([hex(ord(x))[2:] for x in k]) + "\n")
    f.close()
    print "Seeds written in file seeds.txt"
    print "Executed in %s seconds" % (time.time() - start_time)
```

```
~/SSTIC2015/stage5/$ pypy bf.py crack_seeds
Possible bytes found:
0 ['5e', 'de']
1 ['54', 'd4']
2 ['1b', '9b']
3 ['71', 'f1']
4 ['56', 'd6']
5 ['7c', 'fc']
6 ['64', 'e4']
7 ['7d', 'fd']
8 ['69', 'e9']
9 ['76', 'f6']
Possible seeds found: 1024
Seeds written in file seeds.txt
Executed in 0.0622818470001 seconds
```

Les résultats montrent qu'il y a 2 octets possibles pour chaque caractère de la clé. Cela nous fait 2 ^ 10 possibilités, soit 1024 graines possibles.

Avant de commencer le bruteforce de chacune des graines, vérifions que notre algorithme de déchiffrement fonctionne correctement grâce aux données de test fournies dans le PDF.

Données de test :

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

Nous effectuons 2 fois le test de déchiffrement, pour vérifier que les données d'initialisation de chaque transputer sont bien remises à 0 entre 2 déchiffrements.

Notre fonction de déchiffrement semble bien fonctionner.

Commençons maintenant le bruteforce de chacune de nos graines. Il manque 2 octets par graine pour former une clé, ce qui représente 65535 (0xFFFF) combinaisons possibles par graine.

Il y a 1024 graines possibles, soit au total 67 107 840 clés possibles. Le temps moyen de déchiffrement des données chiffrées s'élevent à 52 secondes.

Il faudrait donc 52 * 67 107 840 = 3489607680 secondes, soit 58160128 minutes, soit 969335 heures, soit 40388 jours, soit environ 110 ans pour bruteforcer toutes les clés possibles.

Nous allons essayer de résoudre ce challenge dans un laps de temps un peu moins élevé.

Pour cela, nous allons étudier le format de l'entête TAR.BZ2 et ne déchiffrer que les 36 premiers octets en mettant en place des heuristiques. Cette méthode nous permettra de 'cribler' plus rapidement les clés et de ne laisser que les clés déchiffrant une entête TAR.BZ2 plus ou moins correctement.

Reprenons les parties intéressantes du format TAR.BZ2 :

```
.randomised:1                   = 0=>normal, 1=>randomised (deprecated)
.origPtr:24                     = starting pointer into BWT for after untransform
.huffman_used_map:16            = bitmap, of ranges of 16 bytes, present/not present
```

Le bit `randomized` doit être à 0, vu que l'utilisation de la valeur 1 est `deprecated`. La valeur du champs `ORIG_PTR` doit être un offset dans le fichier décompressé. Sa valeur doit donc être inférieure à la taille des données décompressées. Nous prendrons arbitrairement la valeur 0x20000. La valeur du champs `HUFFMAN_USED_MAP` est à 0xFFFF dans tous les autres fichiers TAR.BZ2 que nous avons rencontrés.

Nous supposons que ce sera aussi le cas ici.

Nous avons maintenant nos heuristiques pour détecter des données déchiffrées potentiellement valides. Codons notre fonction de crible.

```
def heuristic_is_tarbz2(data, header="BZh91AY&SY", max_ptr=0x20000):
    # Check if data have the correct header
    if data[:10] != header:
        return False

    # Transform binary data to binary array (0101110000...)
    data = ''.join(format(x, 'b').zfill(8) for x in bytearray(data))

    # 1st heuristic: test random bit (OFFSET: 112, EXPECTED_VALUE: 0)
    random = int(data[112:113], 2)
    if random != 0:
        return False

    # 2nd heuristic: test the Origin Pointer (OFFSET: 113, EXPECTED_VALUE: >0x20000)
    orig_ptr = int(data[113:137], 2)
    if orig_ptr > max_ptr:
        return False

    # 3rd heuristic: test the HUFFMAN_USED_MAP (OFFSET: 137, EXPECTED_VALUE: 0xFFFF)
    humap = int(data[137:153], 2)
    if humap != 0xFFFF:
        return False

    return True


def crible():
    # Extract the 36 first bytes of encrypted data from input.bin
    encrypted = open("input.bin").read()[0x9AD:0x9AD + 0xC * 3]

    while True:
        # Load all seeds which have to be tested and randomly select one
        f = open("seeds.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        seeds = re.split("\n", data)
        seed = seeds[randint(0, len(seeds) - 1)]

        # Load all seeds which have been already tested
        f = open("seeds_failed.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        seeds_failed = re.split("\n", data)

        # If the seed has already been tested, skip
        if seed in seeds_failed:
            continue

        start_time = time.time()

        # Transform the seed from hexadecimal string to binary string
        seed1 = "".join([chr(int(x, 16)) for x in re.split(" ", seed)])

        # Test all possible combinations from the seed by adding 2 bytes to it
        found = False
        for j in range(0x100):
            for i in range(0x100):

                # Craft the key from the seed and 2 bytes (i, j)
                key = seed1 + chr(i) + chr(j)

                # Use the key for decrypting the first 36 bytes of data
                (decrypted_data, key_out) = one_shot_transputer(encrypted, key)

                # Test if decrypted data are part of a TAR.BZ2 file (heuristic)
                if heuristic_is_tarbz2(decrypted_data) is True:
                    # If heuristic matches, add the key to the file
                    # containing keys which has to be fully tested
                    f = open("keys.txt", "a+")
                    f.write(" ".join([hex(ord(x))[2:].zfill(2) for x in key]) + "\n")
                    f.close()
                    found = True
        # If no keys has been found from the seed, add the seed to the file containing all
```

```
        # the seeds which have been already tested
        if not found:
            f = open("seeds_failed.txt", "a+")
            f.write(seed + "\n")
            f.close()

        print "Seed bruteforced in %s seconds" % (time.time() - start_time)
```

```
~/SSTIC2015/stage5/$ pypy bf.py crible 21
Seed bruteforced in 25.0650579929 seconds
Seed bruteforced in 27.5294489861 seconds
Seed bruteforced in 27.7235760689 seconds
Seed bruteforced in 27.8970401287 seconds
Seed bruteforced in 28.6616799831 seconds
Seed bruteforced in 29.357462883 seconds
[...]
```

Astuce : Utiliser pypy (`http://pypy.org/`) au lieu de Python pour exécuter les codes coûteux en CPU.

Le temps moyen nécessaire pour bruteforcer les 65535 possibilités d'une graîne est d'environ 30 secondes. Il y a 1024 graines à tester. Le temps d'exécution maximum de notre script est donc égal à 1024 * 30 secondes, soit 512 minutes, soit environ 8h30.

Notre code étant parallèlisé et bénéficiant de 21 coeurs, cela devrait nous prendre au maximum 25 minutes.

En parallèle, nous devons tester toutes les clés sortant du crible. Ces clés seront utilisées pour déchiffrer totalement les données, ce qui permettra de vérifier intégralement la validité des données.

Voici le code Python qui fait cela :

```
def is_tarbz2(data):
    ''' Return True if data are a valid TAR.BZ2 file '''
    try:
        bz2.decompress(data)
    except:
        return False
    return True


def test_keys():
    ''' Multiprocessed function - Test key validity by testing
            fully decompressed data validity
        '''

    start_time = time.time()

    # Extract encrypted data from input.bin
    encrypted = open("input.bin").read()[0x9AD:]

    while True:

        # Load all keys which have to be tested and randomly select one
        f = open("keys.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        keys = re.split("\n", data)
        key = keys[randint(0, len(keys) - 1)]

        # Load all keys which have been already tested
        f = open("keys_failed.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        keys_failed = re.split("\n", data)

        # If the key has already been tested, skip
        if key in keys_failed or key == "":
            continue

        # Transform the key from hexadecimal string to binary string
```

```
        key2 = "".join([chr(int(x, 16)) for x in re.split(" ", key)])

        # Use the key for decrypting a part of data
        start_dec_time = time.time()
        (data_out, key_out) = one_shot_transputer(encrypted, key2)
        print "File decrypted in %s seconds" % (time.time() - start_dec_time)

        # Test if decrypted data are part of a TAR.BZ2 file
        if is_tarbz2(data_out):
            print "VALID KEY FOUND: %s " % key
            # Add the key to the file containing all possible valid keys
            f = open("keys_final.txt", "a+")
            f.write(key + "\n")
            f.close()
            print "Valid key found in %s seconds" % (time.time() - start_time)
        else:
            # Add the key to the file containing all the tested keys
            f = open("keys_failed.txt", "a+")
            f.write(key + "\n")
            f.close()
```

```
~/SSTIC2015/stage5/$ pypy bf.py testkeys 3
File decrypted in 52.6092410088 seconds
VALID KEY FOUND: 5e d4 9b 71 56 fc e4 7d e9 76 da c5
Valid key found in 351.569871902 seconds
```

```
~/SSTIC2015/stage5/$ python bf.py decrypt "5e d4 9b 71 56 fc e4 7d e9 76 da c5"
                     "congratulations.tar.bz2"

~/SSTIC2015/stage5/$ file congratulations.tar.bz2
congratulations.tar.bz2: bzip2 compressed data, block size = 900k

~/SSTIC2015/stage5/$ sha256sum congratulations.tar.bz2
9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321  congratulations.tar.bz2
```

Le hash SHA-256 du fichier `congratulations.tar.bz2` correspond bien au hash fourni dans le fichier PDF.

L'épreuve n° 5 se termine avec `congratulations.tar.bz2` et nous passons maintenant à l'épreuve n° 6.

6 Épreuve 6 : Stéganalyse d'une image JPG

6.1 Découverte de l'épreuve

L'épreuve n° 6 commence avec le fichier `congratulations.tar.bz2` récupéré à l'issue de l'épreuve n° 5. Voyons ce qu'il contient.

```
~/SSTIC2015/stage6/$ file congratulations.tar.bz2
congratulations.tar.bz2: bzip2 compressed data, block size = 900k

~/SSTIC2015/stage6$ tar xvf congratulations.tar.bz2
congratulations.jpg

~/SSTIC2015/stage6$ file *
congratulations.jpg:      JPEG image data, JFIF standard 1.01
congratulations.tar.bz2: bzip2 compressed data, block size = 900k

~/SSTIC2015/stage6$ sha256sum *
8b381121312a5941cdee21447bc525f3add35af0792e5d029ed3bb6b51d00ca5  congratulations.jpg
9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321  congratulations.tar.bz2
```

L'archive TAR.BZ2 `congratulations.tar.bz2` contient 1 fichier `congratulations.jpg`.

Au regard du nom du fichier, nous nous attendons à voir apparaître sous nos yeux l'adresse e-mail de validation.

Et bien non ! Il reste encore apparemment un dernier petit effort à fournir avant d'en finir. Il semble que cet effort consiste à analyser l'image `congratulations.jpg`.

6.2 Stéganalyse du dernier effort !

Nous devons à présent analyser l'image `congratulations.jpg`. Nous commençons par regarder si des fichiers ne sont pas embarqués quelque part dans l'image. Pour cela, nous utilisons un outil nommé `binwalk` qui recherche la présence d'entêtes particulières lui permettant d'identifier des fichiers cachés.

```
~/SSTIC2015/stage6/$ binwalk congratulations.jpg

DECIMAL       HEX             DESCRIPTION
--------------------------------------------------------------------------------
55248         0xD7D0          bzip2 compressed data, block size = 900k
```

L'image `congratulations.jpg` semble contenir une archive `TAR.BZ2`. Nous allons l'extraire avec `binwalk`.

```
~/SSTIC2015/stage6$ binwalk -e congratulations.jpg

DECIMAL       HEX             DESCRIPTION
--------------------------------------------------------------------------------
55248         0xD7D0          bzip2 compressed data, block size = 900k

~/SSTIC2015/stage6$ file *
congratulations.jpg:      JPEG image data, JFIF standard 1.01
congratulations.tar.bz2:  bzip2 compressed data, block size = 900k
D7D0:                     POSIX tar archive (GNU)

~/SSTIC2015/stage6$ tar xvf D7D0
congratulations.png

~/SSTIC2015/stage6$ file congratulations.png
congratulations.png: PNG image data, 636 x 474, 8-bit/color RGBA, non-interlaced

~/SSTIC2015/stage6$ sha256sum congratulations.png
554ed53550fc79c6f257bec8b2af8d5304e30598ba1d687e694a41b3af14287c  congratulations.png
```

L'image `congratulations.jpg` contenait une archive TAR.BZ2 qui, une fois décompressée, dévoile une nouvelle image `congratulations.png`

6.3 Stéganalyse du 2ème dernier effort !!

Nous devons à présent analyser l'image `congratulations.png`.

Nous commençons par regarder le format de l'image PNG avec un parser en Python :

```
~/SSTIC2015/stage6$ python png_parser.py congratulations.png
-------------------------------------------------------------
| HEADER   |   OFFSET   |   LENGTH   |      ENTROPY      |
```

IHDR	0x0	0xd	2.1887870678
bKGD	0x19	0x6	1.0
pHYs	0x2b	0x9	1.83659166811
tIME	0x40	0x7	2.80735492206
sTic	0x53	0x1337	7.96024868617
sTic	0x1396	0x1337	7.95843583237
sTic	0x26d9	0x1337	7.94448945337
sTic	0x3a1c	0x1337	7.94581365868
sTic	0x4d5f	0x1337	7.93109892472
sTic	0x60a2	0x1337	7.94473445384
sTic	0x73e5	0x1337	7.95073513771
sTic	0x8728	0x1337	7.94772962756
sTic	0x9a6b	0x1337	7.9546935447
sTic	0xadae	0x1337	7.95649238218
sTic	0xc0f1	0x1337	7.94690837566
sTic	0xd434	0x1337	7.94706137966
sTic	0xe777	0x1337	7.94318515691
sTic	0xfaba	0x1337	7.95103028025
sTic	0x10dfd	0x1337	7.95676251986
sTic	0x12140	0x1337	7.95359362404
sTic	0x13483	0x1337	7.95476891771
sTic	0x147c6	0x1337	7.9512706185
sTic	0x15b09	0x1337	7.95644476879
sTic	0x16e4c	0x1337	7.93201099372
sTic	0x1818f	0x1337	7.88634086901
sTic	0x194d2	0x1337	7.90734419397
sTic	0x1a815	0x1337	7.92134208865
sTic	0x1bb58	0x1337	7.952149995
sTic	0x1ce9b	0x1337	7.95202487051
sTic	0x1e1de	0x1337	7.96236224821
sTic	0x1f521	0x1337	7.92108606418
sTic	0x20864	0x26	5.14266435555
IDAT	0x20896	0x2000	7.92828131191
IDAT	0x228a2	0x2000	7.93149480711
IDAT	0x248ae	0x2000	7.9677403043
IDAT	0x268ba	0x2000	7.96519976014
IDAT	0x288c6	0x2000	7.96208566612
IDAT	0x2a8d2	0x2000	7.95168421077
IDAT	0x2c8de	0x2000	7.92551873081
IDAT	0x2e8ea	0x1aab	7.91617555482
IEND	0x303a1	0x0	0.0

Le nom et la taille de certaines entêtes de l'image sont bizarres. En effet, les entêtes "sTic" de taille 0x1337 ne sont pas très courantes. Ces entêtes sont des entêtes de "chunks". Les chunks contiennent des données relatives à l'image.

Le dernier chunk "sTic" de taille 0x26 nous laissent penser qu'une donnée initiale a été découpée et transformée en chunks. Nous allons extraire cette donnée initiale en concaténant les données de chaque chunks, puis les mettre dans un fichier `sstic.data`.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import math
import array
import struct


def entropy(data):
    ''' Return the data entropy '''
    if len(data) == 0:
        return 0.0

    occurences = array.array('L', [0] * 256)
    for x in data:
        occurences[ord(x)] += 1
    entropy = 0
    for x in occurences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x * math.log(p_x, 2)
    return entropy
```

```
def parse(data):
    data = data[8:]
    n = len(data)
    while data:
        m = n - len(data)
        chunk_length = struct.unpack('>I', data[:4])[0]
        data = data[4:]

        chunk_type = data[:4]  # id du chunk = name
        data = data[4:]

        chunk_data = data[:chunk_length]
        data = data[chunk_length:]

        chunk_crc = struct.unpack('>I', data[:4])[0]
        data = data[4:]
        yield chunk_type, chunk_data, m, chunk_length, chunk_crc


def main():
    f = open(sys.argv[1], 'rb')
    data = f.read()
    f.close()

    sstic_data = ""

    # Parse PNG chunks
    for chunk_type, chunk_data, chunk_offset, length, chunk_crc in parse(data):
        print "Header:", chunk_type, hex(chunk_offset), hex(length), entropy(chunk_data)
        if chunk_type == "sTic":
            sstic_data += chunk_data

    # Save sTic data to file sstic.data
    f = open("sstic.data", "w+")
    f.write(sstic_data)
    f.close()
    print "sTic data saved in file sstic.data"
    print "Entropy of sTic data: %s" % entropy(sstic_data)

if __name__ == '__main__':
    main()
```

```
~/SSTIC2015/stage6$ file sstic.data
sstic.data: data
```

Les données extraites sont des données brutes, dont l'entropie est égale à 7.99. L'entropie maximale étant de 8, nous pouvons considérer que ces données possèdent une forte entropie. Cela peut signifier que ces données sont compressées ou bien chiffrées.

N'ayant aucune information à propos d'un algorithme de chiffrement ou d'une clé, essayons de décompresser ces données

```
~/SSTIC2015/stage6$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> import zlib
>>> f = open("sstic.data")
>>> data = f.read()
>>> f.close()
>>>
>>> f = open("sstic.data.decompressed", "w+")
>>> f.write(zlib.decompress(data))
>>> f.close()

~/SSTIC2015/stage6$ file sstic.data.decompressed
sstic.data.decompressed: bzip2 compressed data, block size = 900k

~/SSTIC2015/stage6$ tar xvf sstic.data.decompressed
congratulations.tiff
```

Une fois décompressées, les données présentes dans les chunks de l'image PNG se révèlent être un fichier au format TAR.BZ2. L'archive contient une nouvelle image

nommée `congratulations.tiff`.

6.4 Stéganalyse du 3ème dernier effort !!!

Nous devons à présent analyser l'image `congratulations.tiff`.

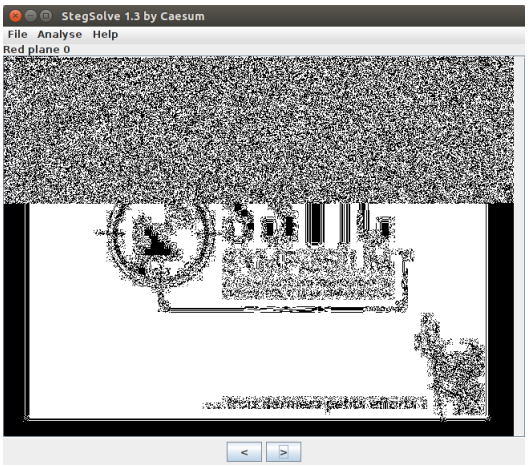
Regardons le code héxadécimal de l'image TIFF. Les données n'étant pas compressées, le contenu de l'image est une succession de 3 octets pour chaque pixel (RGB).

```
00000500    01 01 00 01   01 00 00 01   00 00 01 00   01 01 00 01   ................
00000510    01 00 00 01   00 01 01 00   00 00 00 01   01 00 01 00   ................
00000520    00 01 00 00   01 00 00 01   01 00 01 00   00 01 01 00   ................
00000530    01 01 00 01   00 00 01 00   00 01 01 00   01 01 00 00   ................
00000540    01 00 01 00   00 01 01 00   01 01 00 00   00 00 01 01   ................
[...]
00011870    FE FF FF FF   FF FF FF FF   FF FF FF FF   FF FE FF FF   ................
00011880    FE FF FF FE   FF FE FE FF   FF FF FF FE   FF FF FF FE   ................
00011890    FF FE FF FF   FE FE FF FE   FF FF FF FF   FF FE FF FF   ................
000118A0    FE FE FF FF   FF FF FE FF   FF FF FE FF   FF FE FF FE   ................
000118B0    FF FF FF FE   FF FF FE FF   FF FF FF FF   FE FF FE FF   ................
000118C0    FF FE FF FF   FF FF FF FF   FF FF FE FE   FF FF FF FF   ................
```

Nous observons ici quelque chose de suspect qui se répète dans une partie majeure du code de l'image. L'adresse 0x500 semble représenter le haut de l'image, qui forme un bandeau noir. Le code RGB pour le noir est (0, 0, 0).

Il semblerait que certains bits aient été mis à 0 et d'autres à 1 après que l'image ait été générée. C'est une technique que l'on apelle le Low Significant Bit, qui consiste à cacher des informations dans les bits de poids faible de chaque pixel.

Nous allons vérifier cela avec un outil écrit en Java s'apellant StegSolve (`http://www.caesum.com/handbook/stego.htm`).



Sur ce plan "Red plane 0", le vert et le bleu ont été supprimés (mis à 0). Les pixels ont une couleur rouge si le LSB est à 0 et une couleur noire si il est à 1. Nous voyons sur ce plan que le tiers supérieur de l'image a été altéré. C'est donc bien du LSB. (L'image est ici en niveau de gris).

Sur ce plan "Green plane 0", le rouge et le bleu ont été supprimés (mis à 0). Les pixels ont une couleur verte si le LSB est à 0 et une couleur noire si il est à 1. Nous voyons sur ce plan que le tiers supérieur de l'image a été altéré. C'est donc bien du LSB. (L'image est ici en niveau de gris).

Sur ce plan "Blue plane 0", le vert et le rouge ont été supprimés (mis à 0). Les pixels ont une couleur bleue si le LSB est à 0 et une couleur noire si il est à 1. Nous ne voyons pas d'altération similaire aux 2 autres plans. Il n'y a pas de LSB. (L'image est ici en niveau de gris).

À la vue de ces 3 plans, il semblerait que de l'information ait été cachée dans les bits de poids faible des couleurs rouge et verte. Nous allons faire un script qui récupère la valeur de chacun de ces bits, qui les concatène et qui les met dans un fichier.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
from libtiff import TIFF

def lsb_rg():

    # to open a tiff file for reading:
    tiff_img = TIFF.open(sys.argv[1], mode='r')

    lsb_data = ""
    # For each image in the TIFF
    for image in tiff_img.iter_images():
        # For each pixel row
        for i, row in enumerate(image):
            # For each pixel column
            for j, pix in enumerate(row):
                # Extract LSB from Red and Green value
                lsb_data += bin(pix[0])[-1] + bin(pix[1])[-1]

    # Convert bits string into a binary string
    data_out = ""
    while len(lsb_data) > 0:
        o = lsb_data[:8]
        if len(o) != 8:
            break
        lsb_data = lsb_data[8:]
        data_out += chr(int(o, 2))

    f = open("sstic.lsb", "w+")
    f.write(data_out)
    f.close()

lsb_rg()
```

```
~/SSTIC2015/stage6$ python extract_lsb_rg.py congratulations.tiff

~/SSTIC2015/stage6$ file sstic.lsb
sstic.lsb: bzip2 compressed data, block size = 900k

~/SSTIC2015/stage6$ sha256sum sstic.lsb
46502f2e782c3aaf614dab1598a634484bf3ac307cc91f30f0fb88cc1bbcd443  sstic.lsb

~/SSTIC2015/stage6$ tar xvf sstic.lsb
bzip2: (stdin): trailing garbage after EOF ignored
congratulations.gif
```

Les données cachées dans le LSB des couleurs rouge et verte sont à l'origine une archive TAR.BZ2. Cette archive contient une nouvelle image `congratulations.gif`.

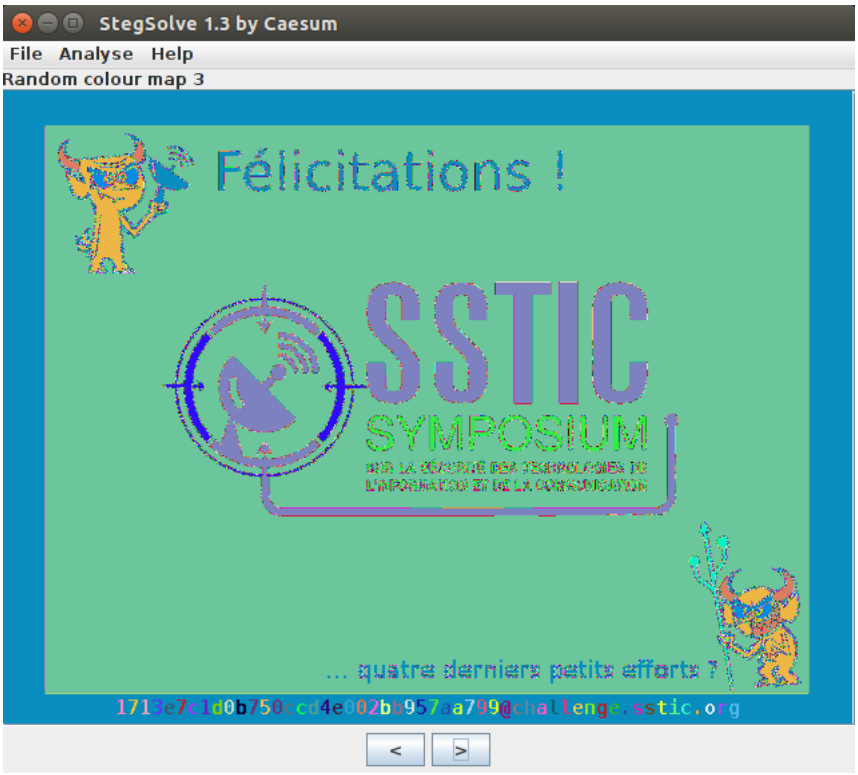
6.5 Stéganalyse du 4ème dernier effort ! ! ! !

Nous devons à présent analyser l'image `congratulations.gif`.

Les images GIF possèdent une palette de 256 couleurs. Ces couleurs sont choisies au préalable parmi un ensemble de 16 777 213 nuances.

L'outil StegSolve permet de transformer une image GIF en modifiant aléatoirement les couleurs de sa palette grâce au mode "Random Colour Map". Cela peut mettre en évidence certaines données cachées.

Voyons le résultat :



Après la modification aléatoire des couleurs de la palette de l'image `congratulations.gif`, nous voyons apparaître une adresse e-mail en bas de l'image.

Nous venons à présent de découvrir l'adresse e-mail de validation.

`1713e7c1d0b750ccd4e002bb957aa799@challenge.sstic.org`

7 Conclusion

Le challenge SSTIC 2015 est une succession d'épreuves nous faisant travailler plusieurs domaines de compétences, que ce soit (brièvement) du host forensic avec l'analyse de l'image de la SDCard, du network forensic avec l'analyse de la trace USB, de la désobfuscation de code avec le JavaScript, de la rétro-conception sur une architecture pas très connue ou encore de la stéganalyse sur divers types d'images.

Je tiens à remercier les auteurs de ces différentes épreuves. Elles m'ont permis d'acquérir de nouvelles connaissances sur des domaines que je n'aurais peut-être jamais connus autrement, notamment les Ducky Scripts, le format d'échange USB, les User-Agents de Mozilla Firefox, les architectures distribuées ST-20 et diffèrentes techniques de stéganalyse.

Merci aux concepteurs des challenges, au comité d'organisation et à ceux m'ayant soutenus durant cette épreuve.

A Désobfuscation du code javaScript

A.1 Code source du javascript désofusqué

```
<script>
__=document
$$$='stage5'
$$_$='load'
$_$$=' '
_$$$$$='user'
_$$$='div'
$$_$$$='navigator'
$$_$$='preferences'
$_$$$='to'
$$$_$='href'
$$$$_='='
$$$$$='chrome'
_$$$$='"'
$_$$$$='Agent'
$$$_$$='down'
$$$$_$='import'
$='<b>Failed'+$_$$+$_$$$+$_$$+$$_$+$_$$+$$$+'</b>'
___='write'
____='getElementById'
_$_ = "raw"
$$=window
__$=$$.crypto.subtle
__$_='decrypt'
___$='status'
$____=$$$$_$+'Key'
_______=0
__$__='then'
_$____='digest'
__$___='innerHTML'
___$__={name:'SHA-1'}
____$_=data
_____$=hash
_$_____=Blob
___$___=URL
____$__='createObjectURL'
______$='type'
$_______='application/octet-stream'
_$______='name'
__$_____='AES-CBC'
___$____='iv'
____$___='<a'+$_$$+$$$_$+$$$$_+_$$$$
_____$__='"'+$_$$+$$$_$$+$$_$+$$$$_+_$$$$+$$$+'.zip'+_$$$$+'>'+$$$_$$+$$_$+$_$$+$$$+'</a>'
______$_='('
     _______$=')'
$________='setTimeout'
_________=parseInt
__________=$$[$$_$$$][_$$$$$+$_$$$$]
___________='length'
____________='substr'
______________=_______+1
________________=_______________*2
________________=________________*4
_________________=________________*_______________
$_$=125*_________________
___________________='indexOf'
____________________='charCodeAt'
_____________________='push'
______________________=Uint8Array
_______________________=''
__________________________='byteLength'
___________________________=$_$$$+'String'
__[___]('<h'+_____________+'>Down'+$$_$+$_$$+'manager</h'+_____________+'>')
__[___]('<'+_$$$+$_$$+'id'+$$$$_+_$$$$+___$+_$$$$+'><i>'+$$_$+'ing...</i></'+_$$$+'>')
__[___]('<'+_$$$+$_$$+'style'+$$$$_+_$$$$+'display:none'+_$$$$+'><a'+$_$$+'target'+
                $$$$_+_$$$$+'blank'+_$$$$+$_$$+$$$_$+$$$$_+_$$$$+$$$$$+'://browser/content/'+
                $$_$$+'/'+$$_$$+'.xul'+_$$$$+'>Back'+$_$$+$_$$$+$_$$+$$_$$+'</a></'+_$$$+'>')
function _____(______){_=[]
     for(___________=_______
     ___________<______[____________]
++___________)_[_____________________](______[____________________](___________))
     return new ______________________(_)
}function ____________________(______){_=[]
     for(___________=_______
```

```
    ----------<------[------------]/---------------
++----------)-[--------------------](---------(------[--------------]
(----------*---------------,---------------),------------------))
    return new ----------------------(-)
}function ----------------------(---------){------=------------------------
    for(----------=-------
    ----------<--------[-------------------------]
++----------){---=--------[----------][--------------------------](-----------------)
    if(---[------------]<----------------)------+=-------
    ------+=---
}return ------
}function ---------------------------(){$_=-----(----------[-------------]
    (----------[------------------](------$_)+---------------,------------------))
    _$__=-----(----------[-------------](----------[------------------]
        (-------$)-----------------,-----------------))
_$={}
_$[_$------]=__$-----
_$[___$____]=$_
_$[------------]=_$__[------------]*----------------
__$[$____](_$_,_$__,_$,false,[__$_])[__$__](function(_$___){__$[__$_](_$,_$___,
    --------------------(____$_))[__$__](function(___$_){____$ =
    new ----------------------(___$_)
    __$[_$____](___$__, ____$)[__$__](function(____$$){if (-----$==----------------------
        (new ----------------------(____$$))){-----$_={}
    -----$_[------$]=$-------
-----$=new _$-----([____$],-----$_)
__$____ = ___$___[____$__](-----$)
__[____](___$)[__$___]=____$___+__$____+-----$__
}else{__[____](___$)[__$___]=$
}})
}).catch(function(){__[____](___$)[__$___]=$
})
}).catch(function(){__[____](___$)[__$___]=$
})
}$$[$--------](--------------------------,$_$)

</script>
```

A.2 Code source du bruteforcer

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import re
import hashlib
import struct
import string
from Crypto.Cipher import AES

def generate_ua():
        platform = ['Windows NT', 'Windows NT 10.0', 'Windows NT 10.0; WOW64',
                'Windows NT 10.0; Win64; x64', 'Windows NT 3.1', 'Windows NT 3.1; WOW64',
                'Windows NT 3.1; Win64; x64', 'Windows NT 3.5', 'Windows NT 3.51',
                'Windows NT 3.51; WOW64', 'Windows NT 3.51; Win64; x64',
                'Windows NT 3.5; WOW64', 'Windows NT 3.5; Win64; x64',
                'Windows NT 4.0', 'Windows NT 4.0; WOW64', 'Windows NT 4.0; Win64; x64',
                'Windows NT 5.0', 'Windows NT 5.01', 'Windows NT 5.01; WOW64',
                'Windows NT 5.01; Win64; x64',  'Windows NT 5.0; WOW64',
                'Windows NT 5.0; Win64; x64', 'Windows NT 5.1', 'Windows NT 5.1; WOW64',
                'Windows NT 5.1; Win64; x64', 'Windows NT 5.2', 'Windows NT 5.2; WOW64',
                'Windows NT 5.2; Win64; x64', 'Windows NT 6.0', 'Windows NT 6.0; WOW64',
                'Windows NT 6.0; Win64; x64', 'Windows NT 6.1', 'Windows NT 6.1.1',
                'Windows NT 6.1.1; WOW64', 'Windows NT 6.1.1; Win64; x64',
                'Windows NT 6.1; WOW64', 'Windows NT 6.1; Win64; x64',
                'Windows NT 6.2', 'Windows NT 6.2; WOW64', 'Windows NT 6.2; Win64; x64',
                'Windows NT 6.3', 'Windows NT 6.3; WOW64', 'Windows NT 6.3; Win64; x64',
                'Windows NT; WOW64', 'Windows NT; Win64; x64', "WindowsCE 6.0"]

        platform.extend(['Macintosh; Intel Mac OS X 0.97', 'Macintosh; Intel Mac OS X 1.0',
                'Macintosh; Intel Mac OS X 1.1', 'Macintosh; Intel Mac OS X 10.0',
                'Macintosh; Intel Mac OS X 10.0.0', 'Macintosh; Intel Mac OS X 10.0.1',
                'Macintosh; Intel Mac OS X 10.0.2', 'Macintosh; Intel Mac OS X 10.0.3',
                'Macintosh; Intel Mac OS X 10.0.4', 'Macintosh; Intel Mac OS X 10.1',
                'Macintosh; Intel Mac OS X 10.1.0', 'Macintosh; Intel Mac OS X 10.1.1',
                'Macintosh; Intel Mac OS X 10.1.2', 'Macintosh; Intel Mac OS X 10.1.3',
                'Macintosh; Intel Mac OS X 10.1.4', 'Macintosh; Intel Mac OS X 10.1.5',
                'Macintosh; Intel Mac OS X 10.10 ', 'Macintosh; Intel Mac OS X 10.10.0',
```

```
'Macintosh; Intel Mac OS X 10.10.0', 'Macintosh; Intel Mac OS X 10.10.1',
'Macintosh; Intel Mac OS X 10.10.2', 'Macintosh; Intel Mac OS X 10.2',
'Macintosh; Intel Mac OS X 10.2.0', 'Macintosh; Intel Mac OS X 10.2.1',
'Macintosh; Intel Mac OS X 10.2.2', 'Macintosh; Intel Mac OS X 10.2.3',
'Macintosh; Intel Mac OS X 10.2.4', 'Macintosh; Intel Mac OS X 10.2.5',
'Macintosh; Intel Mac OS X 10.2.6', 'Macintosh; Intel Mac OS X 10.2.7',
'Macintosh; Intel Mac OS X 10.2.8', 'Macintosh; Intel Mac OS X 10.3',
'Macintosh; Intel Mac OS X 10.3.0', 'Macintosh; Intel Mac OS X 10.3.1',
'Macintosh; Intel Mac OS X 10.3.2', 'Macintosh; Intel Mac OS X 10.3.3',
'Macintosh; Intel Mac OS X 10.3.4', 'Macintosh; Intel Mac OS X 10.3.5',
'Macintosh; Intel Mac OS X 10.3.6', 'Macintosh; Intel Mac OS X 10.3.7',
'Macintosh; Intel Mac OS X 10.3.8', 'Macintosh; Intel Mac OS X 10.3.9',
'Macintosh; Intel Mac OS X 10.4', 'Macintosh; Intel Mac OS X 10.4.0 ',
'Macintosh; Intel Mac OS X 10.4.1 ', 'Macintosh; Intel Mac OS X 10.4.10',
'Macintosh; Intel Mac OS X 10.4.11', 'Macintosh; Intel Mac OS X 10.4.2 ',
'Macintosh; Intel Mac OS X 10.4.3 ', 'Macintosh; Intel Mac OS X 10.4.4 ',
'Macintosh; Intel Mac OS X 10.4.5 ', 'Macintosh; Intel Mac OS X 10.4.6 ',
'Macintosh; Intel Mac OS X 10.4.7 ', 'Macintosh; Intel Mac OS X 10.4.8 ',
'Macintosh; Intel Mac OS X 10.4.9 ', 'Macintosh; Intel Mac OS X 10.5',
'Macintosh; Intel Mac OS X 10.5.0', 'Macintosh; Intel Mac OS X 10.5.1',
'Macintosh; Intel Mac OS X 10.5.2', 'Macintosh; Intel Mac OS X 10.5.3',
'Macintosh; Intel Mac OS X 10.5.4', 'Macintosh; Intel Mac OS X 10.5.5',
'Macintosh; Intel Mac OS X 10.5.6', 'Macintosh; Intel Mac OS X 10.5.7',
'Macintosh; Intel Mac OS X 10.5.8', 'Macintosh; Intel Mac OS X 10.6',
'Macintosh; Intel Mac OS X 10.6.0', 'Macintosh; Intel Mac OS X 10.6.1',
'Macintosh; Intel Mac OS X 10.6.2', 'Macintosh; Intel Mac OS X 10.6.3',
'Macintosh; Intel Mac OS X 10.6.4', 'Macintosh; Intel Mac OS X 10.6.5',
'Macintosh; Intel Mac OS X 10.6.6', 'Macintosh; Intel Mac OS X 10.6.7',
'Macintosh; Intel Mac OS X 10.6.8', 'Macintosh; Intel Mac OS X 10.7',
'Macintosh; Intel Mac OS X 10.7', 'Macintosh; Intel Mac OS X 10.7.0',
'Macintosh; Intel Mac OS X 10.7.1', 'Macintosh; Intel Mac OS X 10.7.2',
'Macintosh; Intel Mac OS X 10.7.3', 'Macintosh; Intel Mac OS X 10.7.4',
'Macintosh; Intel Mac OS X 10.7.5', 'Macintosh; Intel Mac OS X 10.8',
'Macintosh; Intel Mac OS X 10.8.0', 'Macintosh; Intel Mac OS X 10.8.1',
'Macintosh; Intel Mac OS X 10.8.2', 'Macintosh; Intel Mac OS X 10.8.3',
'Macintosh; Intel Mac OS X 10.8.4', 'Macintosh; Intel Mac OS X 10.8.5',
'Macintosh; Intel Mac OS X 10.9', 'Macintosh; Intel Mac OS X 10.9.0',
'Macintosh; Intel Mac OS X 10.9.1', 'Macintosh; Intel Mac OS X 10.9.2',
'Macintosh; Intel Mac OS X 10.9.3', 'Macintosh; Intel Mac OS X 10.9.4',
'Macintosh; Intel Mac OS X 10.9.5', 'Macintosh; Intel Mac OS X 2.0',
'Macintosh; Intel Mac OS X 2.1', 'Macintosh; Intel Mac OS X 3.0',
'Macintosh; Intel Mac OS X 3.1', 'Macintosh; Intel Mac OS X 3.2',
'Macintosh; Intel Mac OS X 3.3', 'Macintosh; Intel Mac OS X 3.4',
'Macintosh; Intel Mac OS X 4.0', 'Macintosh; Intel Mac OS X 4.1',
'Macintosh; Intel Mac OS X 4.2', 'Macintosh; Intel Mac OS X 4.3',
'Macintosh; Intel Mac OS X 6.0', 'Macintosh; Intel Mac OS X 6.0.1',
'Macintosh; Intel Mac OS X 6.0.2', 'Macintosh; Intel Mac OS X 6.0.3',
'Macintosh; Intel Mac OS X 6.0.4', 'Macintosh; Intel Mac OS X 6.0.5',
'Macintosh; Intel Mac OS X 6.0.6', 'Macintosh; Intel Mac OS X 6.0.7',
'Macintosh; Intel Mac OS X 6.0.8', 'Macintosh; Intel Mac OS X 6.0.8L',
'Macintosh; Intel Mac OS X 7.0', 'Macintosh; Intel Mac OS X 7.0.1',
'Macintosh; Intel Mac OS X 7.0.1P', 'Macintosh; Intel Mac OS X 7.1',
'Macintosh; Intel Mac OS X 7.1.1', 'Macintosh; Intel Mac OS X 7.1.2',
'Macintosh; Intel Mac OS X 7.1.2P', 'Macintosh; Intel Mac OS X 7.1P',
'Macintosh; Intel Mac OS X 7.1P1', 'Macintosh; Intel Mac OS X 7.1P2',
'Macintosh; Intel Mac OS X 7.1P3', 'Macintosh; Intel Mac OS X 7.1P4',
'Macintosh; Intel Mac OS X 7.1P5', 'Macintosh; Intel Mac OS X 7.1P6',
'Macintosh; Intel Mac OS X 7.5', 'Macintosh; Intel Mac OS X 7.5',
'Macintosh; Intel Mac OS X 7.5.1', 'Macintosh; Intel Mac OS X 7.5.2',
'Macintosh; Intel Mac OS X 7.5.3', 'Macintosh; Intel Mac OS X 7.5.3',
'Macintosh; Intel Mac OS X 7.5.3', 'Macintosh; Intel Mac OS X 7.5.3',
'Macintosh; Intel Mac OS X 7.5.3L', 'Macintosh; Intel Mac OS X 7.5.4',
'Macintosh; Intel Mac OS X 7.5.5', 'Macintosh; Intel Mac OS X 7.6',
'Macintosh; Intel Mac OS X 7.6.1', 'Macintosh; Intel Mac OS X 8',
'Macintosh; Intel Mac OS X 8.0', 'Macintosh; Intel Mac OS X 8.1',
'Macintosh; Intel Mac OS X 8.5', 'Macintosh; Intel Mac OS X 8.5.1',
'Macintosh; Intel Mac OS X 8.6', 'Macintosh; Intel Mac OS X 9',
'Macintosh; Intel Mac OS X 9.0.0', 'Macintosh; Intel Mac OS X 9.0.2',
'Macintosh; Intel Mac OS X 9.0.3', 'Macintosh; Intel Mac OS X 9.0.4',
'Macintosh; Intel Mac OS X 9.1', 'Macintosh; Intel Mac OS X 9.2.0',
'Macintosh; Intel Mac OS X 9.2.1', 'Macintosh; Intel Mac OS X 9.2.2',
'Macintosh; PPC Mac OS X 0.97', 'Macintosh; PPC Mac OS X 1.0',
'Macintosh; PPC Mac OS X 1.1', 'Macintosh; PPC Mac OS X 10.0',
'Macintosh; PPC Mac OS X 10.0.0', 'Macintosh; PPC Mac OS X 10.0.1',
'Macintosh; PPC Mac OS X 10.0.2', 'Macintosh; PPC Mac OS X 10.0.3',
'Macintosh; PPC Mac OS X 10.0.4', 'Macintosh; PPC Mac OS X 10.1',
'Macintosh; PPC Mac OS X 10.1.0', 'Macintosh; PPC Mac OS X 10.1.1',
'Macintosh; PPC Mac OS X 10.1.2', 'Macintosh; PPC Mac OS X 10.1.3',
'Macintosh; PPC Mac OS X 10.1.4', 'Macintosh; PPC Mac OS X 10.1.5',
'Macintosh; PPC Mac OS X 10.10 ', 'Macintosh; PPC Mac OS X 10.10.0',
```

```
            'Macintosh; PPC Mac OS X 10.10.0', 'Macintosh; PPC Mac OS X 10.10.1',
            'Macintosh; PPC Mac OS X 10.10.2', 'Macintosh; PPC Mac OS X 10.2',
            'Macintosh; PPC Mac OS X 10.2.0', 'Macintosh; PPC Mac OS X 10.2.1',
            'Macintosh; PPC Mac OS X 10.2.2', 'Macintosh; PPC Mac OS X 10.2.3',
            'Macintosh; PPC Mac OS X 10.2.4', 'Macintosh; PPC Mac OS X 10.2.5',
            'Macintosh; PPC Mac OS X 10.2.6', 'Macintosh; PPC Mac OS X 10.2.7',
            'Macintosh; PPC Mac OS X 10.2.8', 'Macintosh; PPC Mac OS X 10.3',
            'Macintosh; PPC Mac OS X 10.3.0', 'Macintosh; PPC Mac OS X 10.3.1',
            'Macintosh; PPC Mac OS X 10.3.2', 'Macintosh; PPC Mac OS X 10.3.3',
            'Macintosh; PPC Mac OS X 10.3.4', 'Macintosh; PPC Mac OS X 10.3.5',
            'Macintosh; PPC Mac OS X 10.3.6', 'Macintosh; PPC Mac OS X 10.3.7',
            'Macintosh; PPC Mac OS X 10.3.8', 'Macintosh; PPC Mac OS X 10.3.9',
            'Macintosh; PPC Mac OS X 10.4', 'Macintosh; PPC Mac OS X 10.4.0 ',
            'Macintosh; PPC Mac OS X 10.4.1 ', 'Macintosh; PPC Mac OS X 10.4.10',
            'Macintosh; PPC Mac OS X 10.4.11', 'Macintosh; PPC Mac OS X 10.4.2 ',
            'Macintosh; PPC Mac OS X 10.4.3 ', 'Macintosh; PPC Mac OS X 10.4.4 ',
            'Macintosh; PPC Mac OS X 10.4.5 ', 'Macintosh; PPC Mac OS X 10.4.6 ',
            'Macintosh; PPC Mac OS X 10.4.7 ', 'Macintosh; PPC Mac OS X 10.4.8 ',
            'Macintosh; PPC Mac OS X 10.4.9 ', 'Macintosh; PPC Mac OS X 10.5',
            'Macintosh; PPC Mac OS X 10.5.0', 'Macintosh; PPC Mac OS X 10.5.1',
            'Macintosh; PPC Mac OS X 10.5.2', 'Macintosh; PPC Mac OS X 10.5.3',
            'Macintosh; PPC Mac OS X 10.5.4', 'Macintosh; PPC Mac OS X 10.5.5',
            'Macintosh; PPC Mac OS X 10.5.6', 'Macintosh; PPC Mac OS X 10.5.7',
            'Macintosh; PPC Mac OS X 10.5.8', 'Macintosh; PPC Mac OS X 10.6',
            'Macintosh; PPC Mac OS X 10.6.0', 'Macintosh; PPC Mac OS X 10.6.1',
            'Macintosh; PPC Mac OS X 10.6.2', 'Macintosh; PPC Mac OS X 10.6.3',
            'Macintosh; PPC Mac OS X 10.6.4', 'Macintosh; PPC Mac OS X 10.6.5',
            'Macintosh; PPC Mac OS X 10.6.6', 'Macintosh; PPC Mac OS X 10.6.7',
            'Macintosh; PPC Mac OS X 10.6.8', 'Macintosh; PPC Mac OS X 10.7',
            'Macintosh; PPC Mac OS X 10.7', 'Macintosh; PPC Mac OS X 10.7.0',
            'Macintosh; PPC Mac OS X 10.7.1', 'Macintosh; PPC Mac OS X 10.7.2',
            'Macintosh; PPC Mac OS X 10.7.3', 'Macintosh; PPC Mac OS X 10.7.4',
            'Macintosh; PPC Mac OS X 10.7.5', 'Macintosh; PPC Mac OS X 10.8',
            'Macintosh; PPC Mac OS X 10.8.0', 'Macintosh; PPC Mac OS X 10.8.1',
            'Macintosh; PPC Mac OS X 10.8.2', 'Macintosh; PPC Mac OS X 10.8.3',
            'Macintosh; PPC Mac OS X 10.8.4', 'Macintosh; PPC Mac OS X 10.8.5',
            'Macintosh; PPC Mac OS X 10.9', 'Macintosh; PPC Mac OS X 10.9.0',
            'Macintosh; PPC Mac OS X 10.9.1', 'Macintosh; PPC Mac OS X 10.9.2',
            'Macintosh; PPC Mac OS X 10.9.3', 'Macintosh; PPC Mac OS X 10.9.4',
            'Macintosh; PPC Mac OS X 10.9.5', 'Macintosh; PPC Mac OS X 2.0',
            'Macintosh; PPC Mac OS X 2.1', 'Macintosh; PPC Mac OS X 3.0',
            'Macintosh; PPC Mac OS X 3.1', 'Macintosh; PPC Mac OS X 3.2',
            'Macintosh; PPC Mac OS X 3.3', 'Macintosh; PPC Mac OS X 3.4',
            'Macintosh; PPC Mac OS X 4.0', 'Macintosh; PPC Mac OS X 4.1',
            'Macintosh; PPC Mac OS X 4.2', 'Macintosh; PPC Mac OS X 4.3',
            'Macintosh; PPC Mac OS X 6.0', 'Macintosh; PPC Mac OS X 6.0.1',
            'Macintosh; PPC Mac OS X 6.0.2', 'Macintosh; PPC Mac OS X 6.0.3',
            'Macintosh; PPC Mac OS X 6.0.4', 'Macintosh; PPC Mac OS X 6.0.5',
            'Macintosh; PPC Mac OS X 6.0.6', 'Macintosh; PPC Mac OS X 6.0.7',
            'Macintosh; PPC Mac OS X 6.0.8', 'Macintosh; PPC Mac OS X 6.0.8L',
            'Macintosh; PPC Mac OS X 7.0', 'Macintosh; PPC Mac OS X 7.0.1',
            'Macintosh; PPC Mac OS X 7.0.1P', 'Macintosh; PPC Mac OS X 7.1',
            'Macintosh; PPC Mac OS X 7.1.1', 'Macintosh; PPC Mac OS X 7.1.2',
            'Macintosh; PPC Mac OS X 7.1.2P', 'Macintosh; PPC Mac OS X 7.1P',
            'Macintosh; PPC Mac OS X 7.1P1', 'Macintosh; PPC Mac OS X 7.1P2',
            'Macintosh; PPC Mac OS X 7.1P3', 'Macintosh; PPC Mac OS X 7.1P4',
            'Macintosh; PPC Mac OS X 7.1P5', 'Macintosh; PPC Mac OS X 7.1P6',
            'Macintosh; PPC Mac OS X 7.5', 'Macintosh; PPC Mac OS X 7.5',
            'Macintosh; PPC Mac OS X 7.5.1', 'Macintosh; PPC Mac OS X 7.5.2',
            'Macintosh; PPC Mac OS X 7.5.3', 'Macintosh; PPC Mac OS X 7.5.3',
            'Macintosh; PPC Mac OS X 7.5.3', 'Macintosh; PPC Mac OS X 7.5.3',
            'Macintosh; PPC Mac OS X 7.5.3L', 'Macintosh; PPC Mac OS X 7.5.4',
            'Macintosh; PPC Mac OS X 7.5.5', 'Macintosh; PPC Mac OS X 7.6',
            'Macintosh; PPC Mac OS X 7.6.1', 'Macintosh; PPC Mac OS X 8',
            'Macintosh; PPC Mac OS X 8.0', 'Macintosh; PPC Mac OS X 8.1',
            'Macintosh; PPC Mac OS X 8.5', 'Macintosh; PPC Mac OS X 8.5.1',
            'Macintosh; PPC Mac OS X 8.6', 'Macintosh; PPC Mac OS X 9',
            'Macintosh; PPC Mac OS X 9.0.0', 'Macintosh; PPC Mac OS X 9.0.2',
            'Macintosh; PPC Mac OS X 9.0.3', 'Macintosh; PPC Mac OS X 9.0.4',
            'Macintosh; PPC Mac OS X 9.1', 'Macintosh; PPC Mac OS X 9.2.0',
            'Macintosh; PPC Mac OS X 9.2.1', 'Macintosh; PPC Mac OS X 9.2.2'])

    platform.extend(['Android; Mobile', 'Android; Tablet'])
platform.extend(['Mobile', 'Tablet', 'Mobile; nnnn', 'Mobile; ZTEOPEN'])
platform.extend(['Maemo; Linux armv7l', 'Macintosh; Intel Mac OS X 10.5',
            'Windows NT 5.2', 'Macintosh; Intel Mac OS X 10.5', "Android; Linux armv71"])
platform.extend(['X11; Linux i686', 'X11; Linux x86_64', 'X11; Linux i686 on x86_64',
            'Maemo; Linux armv7l'])

    # Gecko versions
```

```
    rv = ['0.6', '0.8', '0.9.2', '0.9.4', '0.9.4.1', '0.9.5', '0.9.7', '1', '1.0', '1.0.0',
        '1.0.1', '1.0.2', '1.1', '1.2.1; MultiZilla v1.1.32 final', '1.2b', '1.3a', '1.4',
        '1.4.1', '1.4; MultiZilla v1.5.0.0f', '1.4a', '1.5', '1.6', '1.7', '1.7.10',
        '1.7.12', '1.7.13', '1.7.2', '1.7.3', '1.7.5', '1.7.6', '1.7.7', '1.7.8', '1.7.9',
        '1.8', '1.8.0', '1.8.0.1', '1.8.0.10', '1.8.0.10pre', '1.8.0.11', '1.8.0.12',
        '1.8.0.13pre', '1.8.0.2', '1.8.0.3', '1.8.0.4', '1.8.0.5', '1.8.0.6', '1.8.0.7',
        '1.8.0.8', '1.8.0.9', '1.8.1', '1.8.1.1', '1.8.1.10', '1.8.1.11', '1.8.1.12',
        '1.8.1.12pre', '1.8.1.13', '1.8.1.14', '1.8.1.15', '1.8.1.16', '1.8.1.17',
        '1.8.1.17pre', '1.8.1.18', '1.8.1.19', '1.8.1.2', '1.8.1.20', '1.8.1.21',
        '1.8.1.21pre', '1.8.1.22pre', '1.8.1.2pre', '1.8.1.3', '1.8.1.3pre', '1.8.1.4',
        '1.8.1.4pre', '1.8.1.5', '1.8.1.6', '1.8.1.7', '1.8.1.8', '1.8.1.9', '1.8.16',
        '1.8.1b1', '1.8.1b2', '1.8b4', '1.8b5', '1.9', '1.9.0', '1.9.0.1', '1.9.0.10',
        '1.9.0.10pre', '1.9.0.11', '1.9.0.12', '1.9.0.13', '1.9.0.14', '1.9.0.15',
        '1.9.0.16', '1.9.0.17', '1.9.0.18', '1.9.0.19', '1.9.0.1pre', '1.9.0.2',
        '1.9.0.2pre', '1.9.0.3', '1.9.0.3pre', '1.9.0.4', '1.9.0.4pre', '1.9.0.5',
        '1.9.0.6', '1.9.0.6pre', '1.9.0.7', '1.9.0.8', '1.9.0.9', '1.9.1', '1.9.1.1',
        '1.9.1.10', '1.9.1.11', '1.9.1.12', '1.9.1.13', '1.9.1.15', '1.9.1.16',
        '1.9.1.2', '1.9.1.2pre', '1.9.1.3', '1.9.1.4', '1.9.1.5', '1.9.1.6', '1.9.1.7',
        '1.9.1.8', '1.9.1.8pre', '1.9.1.9', '1.9.1a2pre', '1.9.1b1', '1.9.1b2',
        '1.9.1b2pre', '1.9.1b3', '1.9.1b3pre', '1.9.1b4', '1.9.1b4pre', '1.9.1b5pre',
        '1.9.2', '1.9.2.1', '1.9.2.10', '1.9.2.10pre', '1.9.2.11', '1.9.2.12', '1.9.2.13',
        '1.9.2.14', '1.9.2.14pre', '1.9.2.15', '1.9.2.16', '1.9.2.16pre', '1.9.2.17',
        '1.9.2.18', '1.9.2.19', '1.9.2.2', '1.9.2.20', '1.9.2.21', '1.9.2.22', '1.9.2.23',
        '1.9.2.24', '1.9.2.25', '1.9.2.28', '1.9.2.2pre', '1.9.2.3', '1.9.2.3pre',
        '1.9.2.4', '1.9.2.6', '1.9.2.7', '1.9.2.8', '1.9.2.9', '1.9.2a1', '1.9.2a1pre',
        '1.9.2a2pre', '1.9.2b1', '1.9.2b4', '1.9.2b5', '1.9.3a3pre', '1.9.3a5pre',
        '1.9a1', '1.9a2', '1.9b1', '1.9b2', '1.9b3', '1.9b3pre', '1.9b4', '1.9b4pre',
        '1.9b5', '1.9b5pre', '1.9pre', '10', '10.0', '10.0.9', '11', '11.0', '12',
        '12.0', '13', '13.0', '14', '14.0', '15', '15.0', '16', '16.0', '16.0.1', '17',
        '17.0', '18', '18.0', '18.1', '19', '19.0', '2', '2.0', '2.0.0.0', '2.0.1', '2.0.4',
        '2.0b10', '2.0b10pre', '2.0b11pre', '2.0b12pre', '2.0b13pre', '2.0b3pre',
        '2.0b4', '2.0b6pre', '2.0b7', '2.0b7pre', '2.0b8', '2.0b8pre', '2.0b9pre',
        '2.2a1pre', '2.7.0', '20', '20.0', '21', '21.0', '21.0.0', '22', '22.0', '23',
        '23.0', '24', '24.0', '24.8', '25.0', '26', '26.0', '27.0', '27.3', '28',
        '28.0', '29.0', '30', '30.0', '31.0', '32', '32.0', '33', '33.0', '34', '34.0',
        '35.0', '36.0', '37.0', '38.0', '39.0', '40.0', '41.0', '42.0', '43.0', '44.0',
        '45.0', '46.0', '5', '5.0', '5.0.1.6', '5.0a2', '6', '6.0', '6.0a2', '7', '7.0',
        '8', '8.0', '8.0; en_us', '9', '9.0', '9.0.1', '9.0a2']

    ua_list = list()
    for p in platform:
        for r in rv:
            ua_list.append("%s; rv:%s" % (p, r))
            ua_list.append("%s; Mobile; rv:%s" % (p, r))
            ua_list.append("%s; Tablet; rv:%s" % (p, r))
    return ua_list


def heur_is_valid_zip(data, max_compressed_size=0x493E0, max_file_name_size=0x20):
    if len(data) < 3 * 16:
        print "Invalid data for zip heuristic"
        return False

    # Check the ZIP header (PK)
    if data[:2] != "PK":
        return False

    # Check the uncompressed data length (<= COMPRESSED_MAX)
    compressed_size = struct.unpack("<L", data[18:22])[0]
    if compressed_size > max_compressed_size:
        return False

    # Check the filename length (<= FILENAME_MAX)
    file_name_size = struct.unpack("<L", data[26:30])[0]
    if file_name_size > max_file_name_size:
        return False

    # Check the filename content (only PRINTABLE characters)
    file_name = data[30:30+file_name_size]
    is_text = all(c in string.printable for c in file_name)
    if is_text is False:
        return False

    return True


def decrypt(key, encrypted_data, iv):
    obj = AES.new(key, AES.MODE_CBC, iv)
    return obj.decrypt(encrypted_data)
```

```
def main():
    # Genere tous les UserAgents possibles
    uas = generate_ua()

    # Ouvre le fichier contenant les donnees chiffrees
    encrypted = open("payload.bin").read()
    encrypted_data = encrypted[:0x40]

    for ua in uas:
        key = ua[-16:]
        iv = ua[:16]
        if len(key) != 16 or len(iv) != 16:
            continue
        decrypted = decrypt(key, encrypted_data, iv=iv)
        if heur_is_valid_zip(decrypted) is True:
            return ua, key, decrypted
    return None, None, None


(ua, key, decrypted) = main()

# Resultats
print 'UserAgent valide trouve: %s' % (ua)

# Ecriture des donnees d chiffrees
encrypted = open("payload.bin").read()
open("stage5.zip", "w+").write(decrypt(ua[-16:], encrypted, iv=ua[:16]))
```

B Architecture ST20

B.1 Paquets de chargement reçus et envoyés par T0 à T1, T2 et T3

```
[0x71][T1][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
           75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
           fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
           10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
           f0]

[0x71][T2][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
           75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
           fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
           10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
           f0]

[0x71][T3][70 60 b8 24 f2 24 20 50 23 fc 60 b8 15 24 f2 54 4c f7 75 21 a2 25 44 21 fb 24 f2 54
           75 f7 24 4b 21 fb 76 75 fb 61 05 11 24 f2 54 4c f7 11 24 f2 51 4c fb 11 24 f2 52 4c
           fb 11 24 f2 53 4c fb 10 81 24 f2 55 41 f7 10 82 24 f2 56 41 f7 10 83 24 f2 57 41 f7
           10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 25 fa 10 23 fb 10 24 f2 41 fb 64 0a 00 b8 22
           f0]

[0x31][T1][25 00 00 00 04 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T1][25 00 00 00 08 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T1][25 00 00 00 0c 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T2][25 00 00 00 04 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T2][25 00 00 00 08 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T2][25 00 00 00 0c 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T3][25 00 00 00 04 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x31][T3][25 00 00 00 08 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]
```

```
[0x31][T3][25 00 00 00 0c 00 00 80 00 00 00 00 24 60 bd 24 f2 24 20 50 23 fc 60 bd 10 24 f2 54
           4c f7 4b 21 fb 24 f2 54 70 f7 43 21 fb 72 f2 f6 00 b3 22 f0 20]

[0x5c][T1][50 00 00 00 04 00 00 80 0c 00 00 00 44 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 bb 40 d1 40 11 23 fb 4c d0 12 24 f2 54 76 61 93 40 d0 70
           12 f2 f1 11 f1 f2 2f 4f 24 f6 11 23 fb 70 81 d0 4c 70 f9 a2 61 09 41 d0 11 24 f2 76
           63 98 20 62 03 20 20 20]

[0x5c][T1][50 00 00 00 08 00 00 80 0c 00 00 00 44 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 bb 40 d1 40 11 23 fb 4c d0 12 24 f2 54 76 61 93 40 d0 70
           12 f2 f1 71 23 f3 2f 4f 24 f6 11 23 fb 70 81 d0 4c 70 f9 a2 61 09 41 d0 11 24 f2 76
           63 98 20 62 03 20 20 20]

[0x98][T1][8c 00 00 00 0c 00 00 80 0c 00 00 00 80 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 b9 40 d2 40 d1 40 d3 4c d0 14 24 f2 54 78 61 93 73 c0 21
           ab 40 d0 70 14 f2 f1 71 f2 2f 2f 2f 4f 24 f6 d1 70 81 d0 4c 70 f9 a3 80 61 09 41 d3
           71 28 20 20 40 24 f6 4f 24 f0 71 24 20 20 40 24 f6 4e 24 f0 23 f3 2f 2f 2f 4f 24 f6
           71 41 24 f1 2f 2f 2f 4f 24 f6 23 f3 2f 2f 2f 4f 24 f6 25 fa d1 2f 4f 24 f6 12 23 fb
           41 d0 12 24 f2 78 67 9a 20 66 05 20]

[0x70][T2][64 00 00 00 04 00 00 80 0c 00 00 00 58 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 b9 40 d3 4c d0 14 24 f2 54 78 61 97 40 d1 40 d2 40 d0 70
           14 f2 f1 71 f2 2f 4f 24 f6 d1 14 70 86 f2 f1 72 f2 2f 4f 24 f6 d2 70 81 d0 46 70 f9
           a2 61 00 72 71 23 f3 2f 4f 24 f6 13 23 fb 41 d0 13 24 f2 78 64 94 20 64 0b 20 20 20]

[0xa8][T2][9c 00 00 00 08 00 00 80 0c 00 00 00 90 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 61 bf 40 d3 40 d4 40 d2 40 d0 40 72 43 f8 15 fa 70 f2 23 fb
           70 81 d0 4c 70 f9 a2 61 0d 72 81 d2 44 72 f9 a2 61 02 4c d0 74 43 f8 15 fa 24 f2 54
           21 72 63 90 74 81 25 fa d4 c4 a3 80 40 d4 40 13 23 fb 40 d2 40 d1 40 d0 72 43 f8 15
           fa 70 f2 f1 71 f2 2f 4f 24 f6 d1 70 81 d0 4c 70 f9 a3 80 61 07 73 71 23 f3 2f 4f 24
           f6 13 23 fb 72 81 d2 44 72 f9 a3 80 63 0e 41 d0 13 24 f2 21 72 68 9b 20 65 04 20 20]

[0x60][T2][54 00 00 00 0c 00 00 80 0c 00 00 00 48 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 bb 40 d1 4c d0 12 24 f2 54 76 61 97 40 11 23 fb 40 d0 70
           12 f2 f1 70 47 24 f6 24 f1 71 23 f3 2f 4f 24 f6 11 23 fb 70 81 d0 4c 70 f9 a2 61 03
           41 d0 11 24 f2 76 63 92 20 63 09 20]

[0xa4][T3][98 00 00 00 04 00 00 80 0c 00 00 00 8c 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 b0 40 d3 40 d2 40 d0 40 d1 40 70 43 f8 14 fa 71 f2 23 fb
           71 81 d1 4c 71 f9 a2 61 0d 70 81 d0 44 70 f9 a2 61 02 4c d0 72 43 f8 14 fa 24 f2 54
           21 71 63 90 72 81 25 fa d2 c4 a3 80 40 d2 40 d1 40 d0 70 43 f8 14 fa f1 71 f2 2f 4f
           24 f6 d1 70 81 d0 44 70 f9 a3 80 61 09 71 43 24 f6 43 f8 14 fa 71 44 24 f0 4c 21 ff
           2f 4f 24 f6 f2 f1 13 23 fb 41 d0 13 24 f2 21 71 67 90 20 65 09 20 20 20]

[0x7c][T3][70 00 00 00 08 00 00 80 0c 00 00 00 64 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 ba 40 d1 40 d2 4c d0 13 24 f2 54 77 61 95 40 11 23 fb 13
           f1 13 83 f1 23 f3 13 87 f1 23 f3 2f 4f 24 f6 11 23 fb 41 d0 11 24 f2 51 77 63 9b 41
           d0 11 24 f2 55 77 64 9c 11 f1 4c 21 ff 2f 4f 24 f6 11 23 fb 11 f1 13 f2 f1 12 23 fb
           41 d0 12 24 f2 77 65 96 20 65 0f 20]

[0x90][T3][84 00 00 00 0c 00 00 80 0c 00 00 00 78 00 00 00 00 00 00 00 0c 00 00 00 73 72 74 f7
           22 f0 73 72 74 fb 22 f0 60 ba 40 d2 40 d1 40 d0 40 70 13 f2 23 fb 70 81 d0 4c 70 f9
           a2 60 01 40 11 23 fb 13 81 f1 13 85 f1 23 f3 13 89 f1 23 f3 2f 4f 24 f6 11 23 fb 4c
           d0 13 24 f2 54 77 64 9c 41 d0 12 24 f2 55 77 64 93 41 d0 11 24 f2 51 77 64 90 12 f1
           4c 21 ff 2f 4f 24 f6 12 23 fb 12 f1 13 f2 f1 11 23 fb 41 d0 11 24 f2 77 66 94 20 65
           0e 20 20 20]
```

B.2 Code source du bruteforcer

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import re
import bz2
import time
from random import randint


# Global varibales used by transputers
t4_w1 = 0
t5_w1 = 0
t6_w1 = 0
t6_w3 = 0
t8_w4 = 0
t8_w5 = 0xC * 4 * "\x00"
t10_w2 = 0
```

```
t10_w4 = 0xC * 4 * "\x00"
t12_w3 = 0xC * "\x00"


def one_shot_transputer(encrypted, key):
    global t4_w1
    global t5_w1
    global t6_w1
    global t6_w3
    global t8_w4
    global t8_w5
    global t10_w2
    global t10_w4
    global t12_w3
    global t12_w3

    t4_w1 = 0
    t5_w1 = 0
    t6_w1 = 0
    t6_w3 = 0
    t8_w4 = 0
    t8_w5 = 0xC * 4 * "\x00"
    t10_w2 = 0
    t10_w4 = 0xC * 4 * "\x00"
    t12_w3 = 0xC * "\x00"

    def transputer4(key):
        global t4_w1
        w2 = key
        for i in range(0xc):
            t4_w1 += ord(w2[i])
            t4_w1 &= 0xFF
        return t4_w1

    def transputer5(key):
        global t5_w1
        w2 = key
        for w0 in range(0xC):
            t5_w1 = ((t5_w1 ^ ord(w2[w0])) & 0xFF)
        return t5_w1

    def transputer6(key):
        global t6_w1
        global t6_w3

        w4 = key
        if t6_w3 == 0:
            for w0 in range(0xC):
                t6_w1 = (t6_w1 + ord(w4[w0])) & 0xFFFF
            t6_w3 = 1

        t6_w1 = ((((t6_w1 << 1) & 0xFFFF) ^ ((((t6_w1 & 0x4000) >> 0xE) ^
                 ((t6_w1 & 0x8000) >> 0xF)) & 0xFFFF)) & 0xFFFF)
        w2 = t6_w1 & 0xFF
        return w2

    def transputer7(key):
        w1 = 0
        w2 = 0
        for i in range(6):
            w1 = ((ord(key[i]) + w1) & 0xFF)
            w2 = ((ord(key[i + 6]) + w2) & 0xFF)
        return (w1 ^ w2) & 0xFF

    def transputer8(key):
        global t8_w4
        global t8_w5

        t8_w5 = t8_w5[:12 * t8_w4] + key + t8_w5[(12 * t8_w4) + 0xC:]

        t8_w4 += 1
        if t8_w4 >= 4:
            t8_w4 = 0

        w3 = 0
        for i in range(4):
            w1 = 0
            for j in range(0xC):
                w1 += ord(t8_w5[j + (12 * i)])
                w1 &= 0xFF
```

```
            w3 ^= w1
            w3 &= 0xFF

        return w3

    def transputer9(key):
        t9_w1 = 0
        for i in range(0xC):
            t9_w1 ^= ord(key[i]) << (7 & i)
            t9_w1 &= 0xFF
        return t9_w1

    def transputer10(key):
        global t10_w2
        global t10_w4

        t10_w4 = t10_w4[:12 * t10_w2] + key + t10_w4[(12 * t10_w2) + 0xC:]

        t10_w2 += 1
        if t10_w2 == 4:
            t10_w2 = 0

        t10_w1 = 0
        for i in range(4):
            t10_w1 += ord(t10_w4[12 * i])
            t10_w1 &= 0xFF

        return ord(t10_w4[(12 * (t10_w1 & 3)) + (((t10_w1 >> 4) % 0xC) & 0xFF)])

    def transputer11(key):
        global t12_w3
        '''
        w1 = ( ord(w3[7]) ^ ord(w3[3]) ^ ord(w3[0]) ) & 0xFF
        snd("T11", "T12", chr(w1))
        w1 = rcv("T12", "T11", 0x1)
        '''
        # From t12
        w1 = (ord(t12_w3[1]) ^ ord(t12_w3[5]) ^ ord(t12_w3[9])) & 0xFF
        w1 = (w1 % 0xC) & 0xFF
        return ord(key[w1])

    def transputer12(key):
        global t12_w3
        '''
        w1 = (ord(w3[1]) ^ ord(w3[5]) ^ ord(w3[9])) & 0xFF
        t12_w3 = rcv("T3", "T12", 0xC)
        w2 = rcv("T11", "T12", 0x1)
        '''
        t12_w3 = key

        # From t11
        w2 = (ord(key[7]) ^ ord(key[3]) ^ ord(key[0])) & 0xFF

        # snd("T12", "T11", chr(w1))
        w2 = (w2 % 0xC) & 0xFF
        w1 = ord(t12_w3[w2])
        return w1

    out = ""

    while 1:
        # For each characters in the key
        for i in range(0xC):

            # Return results when there is no more data to decrypt
            if len(encrypted) <= 0:
                return out, key

            # Get the following encrypted byte
            data = encrypted[0]
            encrypted = encrypted[1:]

            # Send key to each transputer
            b4 = transputer4(key)
            b5 = transputer5(key)
            b6 = transputer6(key)

            b7 = transputer7(key)
            b8 = transputer8(key)
            b9 = transputer9(key)
```

```
            b10 = transputer10(key)
            b11 = transputer11(key)
            b12 = transputer12(key)

            # Combine transputers's results
            k = b4 ^ b5 ^ b6 ^ b7 ^ b8 ^ b9 ^ b10 ^ b11 ^ b12

            # Decrypt the encrypted byte
            out += chr((i + (2 * ord(key[i])) ^ ord(data)) & 0xFF)

            # Update the key
            key = key[:i] + chr(k) + key[i + 1:]

            # Return results when there is no more data to decrypt
            if len(encrypted) == 0:
                return out, key


def crack_seeds(header="BZh91AY&SY"):
    ''' Generate all seeds which decrypt data starting
            by BZh91AY&SY and put them into seeds.txt
        '''

    start_time = time.time()

    # Extract the 36 first bytes of encrypted data from input.bin
    f = open("input.bin")
    encrypted = f.read()[0x9AD:0x9AD + 0xC * 3]
    f.close()
    key_found = list()

    for j in range(len(header)):
        possible_bytes = list()

        for i in range(0x100):

            # Decrypt the byte
            decrypted_byte = ((j + (2 * i)) ^ ord(encrypted[j])) & 0xFF

            # If the decrypted byte match the TAR.BZ2 header,
            # add it to the list of possible valid bytes
            if chr(decrypted_byte) == header[j]:
                possible_bytes.append(i)

        key_found.append(possible_bytes)

    print "Possible bytes found:"
    for i, k in enumerate(key_found):
        print i, [hex(x)[2:] for x in k]

    # Generate all possible seeds from all possible bytes
    possible_seeds = [""]
    for j in range(len(header)):

        possible_seeds_2 = list()
        for seed in possible_seeds:

            for i in key_found[j]:
                possible_seeds_2.append(seed + chr(i))

        if len(possible_seeds_2) == 0:
            return "No match found at step %s" % j

        possible_seeds = possible_seeds_2
        del possible_seeds_2

    print "Possible seeds found: %s " % len(possible_seeds)
    f = open("seeds.txt", "w+")
    for k in possible_seeds:
        f.write(" ".join([hex(ord(x))[2:] for x in k]) + "\n")
    f.close()
    print "Seeds written in file seeds.txt"
    print "Executed in %s seconds" % (time.time() - start_time)


def heuristic_is_tarbz2(data, header="BZh91AY&SY", max_ptr=0x20000):
    # 1st heuristic: test MAGIC (OFFSET: 0, EXPECTED_VALUE: BZh91AY&SY)
    if data[:10] != header:
        return False
```

```
    # Transform binary data to binary array (0101110000...)
    data = ''.join(format(x, 'b').zfill(8) for x in bytearray(data))

    # 2nd heuristic: test random bit (OFFSET: 112, EXPECTED_VALUE: 0)
    random = int(data[112:113], 2)
    if random != 0:
        return False

    # 3rd heuristic: test the Origin Pointer (OFFSET: 113, EXPECTED_VALUE: >0x20000)
    orig_ptr = int(data[113:137], 2)
    if orig_ptr > max_ptr:
        return False

    # 4th heuristic: test the HUFFMAN_USED_MAP (OFFSET: 137, EXPECTED_VALUE: 0xFFFF)
    humap = int(data[137:153], 2)
    if humap != 0xFFFF:
        return False

    return True


def crible():
    # Extract the 36 first bytes of encrypted data from input.bin
    encrypted = open("input.bin").read()[0x9AD:0x9AD + 0xC * 3]

    while True:
        # Load all seeds which have to be tested and randomly select one
        f = open("seeds.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        seeds = re.split("\n", data)
        seed = seeds[randint(0, len(seeds) - 1)]

        # Load all seeds which have been already tested
        f = open("seeds_failed.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        seeds_failed = re.split("\n", data)

        # If the seed has already been tested, skip
        if seed in seeds_failed:
            continue

        start_time = time.time()

        # Transform the seed from hexadecimal string to binary string
        seed1 = "".join([chr(int(x, 16)) for x in re.split(" ", seed)])

        # Test all possible combinations from the seed by adding 2 bytes to it
        found = False
        for j in range(0x100):
            for i in range(0x100):

                # Craft the key from the seed and 2 bytes (i, j)
                key = seed1 + chr(i) + chr(j)

                # Use the key for decrypting the first 36 bytes of data
                (decrypted_data, key_out) = one_shot_transputer(encrypted, key)

                # Test if decrypted data are part of a TAR.BZ2 file (heuristic)
                if heuristic_is_tarbz2(decrypted_data) is True:
                    # If heuristic matches, add the key to the file
                    #  containing   keys which has to be fully tested
                    f = open("keys.txt", "a+")
                    f.write(" ".join([hex(ord(x))[2:].zfill(2) for x in key]) + "\n")
                    f.close()
                    found = True
        # If no keys has been found from the seed, add the seed to the file containing all
        # the seeds which have been already tested
        if not found:
            f = open("seeds_failed.txt", "a+")
            f.write(seed + "\n")
            f.close()

        print "Seed bruteforced in %s seconds" % (time.time() - start_time)
```

```
def is_tarbz2(data):
    ''' Return True if data are a valid TAR.BZ2 file '''
    try:
        bz2.decompress(data)
    except:
        return False
    return True


def test_keys():
    ''' Multiprocessed function - Test key validity by testing
            fully decompressed data validity
        '''

    start_time = time.time()

    # Extract encrypted data from input.bin
    encrypted = open("input.bin").read()[0x9AD:]

    while True:

        # Load all keys which have to be tested and randomly select one
        f = open("keys.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        keys = re.split("\n", data)
        key = keys[randint(0, len(keys) - 1)]

        # Load all keys which have been already tested
        f = open("keys_failed.txt")
        data = f.read()
        f.close()
        data = data.replace("\r", "")
        keys_failed = re.split("\n", data)

        # If the key has already been tested, skip
        if key in keys_failed or key == "":
            continue

        # Transform the key from hexadecimal string to binary string
        key2 = "".join([chr(int(x, 16)) for x in re.split(" ", key)])

        # Use the key for decrypting a part of data
        start_dec_time = time.time()
        (data_out, key_out) = one_shot_transputer(encrypted, key2)
        print "File decrypted in %s seconds" % (time.time() - start_dec_time)

        # Test if decrypted data are part of a TAR.BZ2 file
        if is_tarbz2(data_out):
            print "VALID KEY FOUND: %s " % key
            # Add the key to the file containing all possible valid keys
            f = open("keys_final.txt", "a+")
            f.write(key + "\n")
            f.close()
            print "Valid key found in %s seconds" % (time.time() - start_time)
        else:
            # Add the key to the file containing all the tested keys
            f = open("keys_failed.txt", "a+")
            f.write(key + "\n")
            f.close()


def main():
    import sys

    if sys.argv[1] == "test_transputers":
        ''' Test ST20 simulator with test data given by the PDF '''
        start_time = time.time()
        key = "*SSTIC-2015*"
        encrypted = "1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e".decode('hex')
        (decrypted_data, key_out) = one_shot_transputer(encrypted, key)

        if decrypted_data != "I love ST20 architecture":
            print "Failed -> %s" % decrypted_data
        else:
            print "Success -> %s" % decrypted_data

        # Test a second time
        (decrypted_data, key_out) = one_shot_transputer(encrypted, key)
```

```
        if decrypted_data != "I love ST20 architecture":
            print "Failed -> %s" % decrypted_data
        else:
            print "Success -> %s" % decrypted_data
        print "Executed in %s seconds" % (time.time() - start_time)

    elif sys.argv[1] == "crack_seeds":
        ''' Generate all possible seeds into the file keys.txt '''
        crack_seeds()

    elif sys.argv[1] == "crible":
        ''' Eliminate invalid keys following heuristics '''
        from multiprocessing import Process
        f = open("seeds.txt", "a+")
        f.close()
        f = open("seeds_failed.txt", "a+")
        f.close()
        for i in range(int(sys.argv[2])):
            p = Process(target=crible)
            p.start()

    elif sys.argv[1] == "testkeys":
        ''' Test possible '''
        from multiprocessing import Process
        f = open("keys.txt", "a+")
        f.close()
        f = open("keys_failed.txt", "a+")
        f.close()
        for i in range(int(sys.argv[2])):
            p = Process(target=test_keys)
            p.start()

    elif sys.argv[1] == "decrypt":
        ''' Decrypt encrypted data with key given in argument
            and store them in file given in argument
        '''
        key = "".join([chr(int(x, 16)) for x in sys.argv[2].split(" ")])
        encrypted = open("input.bin").read()[0x9AD:]
        (decrypted_data, key_out) = one_shot_transputer(encrypted, key)

        f = open(sys.argv[3], "w+")
        f.write(decrypted_data)
        f.close()

main()
```