Solution du challenge SSTIC 2015

Nicolas Iooss

16 avril 2015

Résumé

Cette année le challenge consistait à étudier l'image d'une carte microSD pour y trouver une adresse e-mail en @challenge.sstic.org. Cette carte microSD contient un fichier pour Rubber Ducky qui permet, une fois connecté à un ordinateur, d'exécuter des commandes qui créent une archive zip.

Cette archive contient un fichier chiffré dont la clé de déchiffrement a été écrite sous forme d'énigme dans une carte OpenArena.

Une fois cette énigme résolue, il est possible de déchiffrer le fichier chiffré, qui est une autre archive zip contenant un second fichier chiffré dont il faut récupérer la clé de chiffrement qui a été "enregistrée avec Paint". Après avoir compris ce que ceci signifie et trouvé la clé, le déchiffrement du fichier permet d'obtenir encore une fois une archive zip, mais qui contient cette fois ci une page HTML.

Cette page HTML contient un code Javascript obfusqué qui tente de déchiffrer un bloc de données en utilisant des paramètres de déchiffrement (vecteur d'initialisation et clé) issus de l'agent utilisateur du navigateur.

Une fois ces paramètres devinés, la cinquième étape du challenge consiste à comprendre le fonctionnement d'un réseau de transputers afin de casser l'algorithme de déchiffrement qui est implémenté sur celui-ci.

Le résultat de ce déchiffrement est une archive contenant une image au format JPEG de laquelle il faut extraire successivement des images aux formats PNG, TIFF et GIF qui sont cachées les unes dans les autres. Finalement la dernière image extraite contient l'adresse e-mail qu'il faut trouver, permettant ainsi de valider la fin du challenge.

Table des matières

0 Premier avril

Contrairement à ce qui a été annoncé dans le résumé, cette année le challenge du SSTIC n'a pas commencé directement par l'analyse de l'image d'une carte microSD mais par une annonce sur le compte Twitter du symposium le premier avril :

FIGURE 1 – https://twitter.com/sstic/status/583271514163814400

Ce tweet invite à télécharger http://static.sstic.org/challenge2015/chlg-2015. Une rapide analyse de ce fichier binaire permet de trouver quelques informations intéressantes :

```
% binwalk chlg-2015
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             OpenSSL encryption, salted, salt: 0x441980B6425FD4FF

% xxd chlg-2015 | head -8
00000000: 5361 6c74 6564 5f5f 4419 80b6 425f d4ff  Salted__D...B_..
00000010: 1e5c 83a1 c4d7 24f5 4697 9aac 2571 5f8e  .\....$.F...%q_.
00000020: 64e2 52ad 8947 119e c4ad 929b 6505 4de2  d.R..G......e.M.
00000030: ff44 0add d438 273c 4b8d 760d 416d c883  .D...8'<K.v.Am..
00000040: 290e 8014 1ce7 88a2 b6c7 cd00 d8e9 98bd  ).............
00000050: e40c cb76 4b52 626e 0ee0 c966 b1c8 c24b  ...vKRbn...f...K
00000060: e40c cb76 4b52 626e 0ee0 c966 b1c8 c24b  ...vKRbn...f...K
00000070: 4915 cd03 153c 5210 abe7 c68f c882 f198  I....<R.........
```

Ce fichier commence par "Salted__", qui correspond à l'entête d'un fichier chiffré par la commande openssl. Le mot de passe utilisé pour chiffrer le fichier a été salé par le sel donné dans l'entête, mais ni ce mot de passe ni l'algorithme utilisé ne sont connus.

Néanmoins on remarque la répétition de la séquence `e40c cb76 4b52 626e 0ee0 c966 b1c8 c24b` aux positions hexadécimales 50 et 60. Il est donc pertinent de chercher des séquences de 16 octets qui seraient beaucoup répétées dans le fichier :

```
% xxd -c16 -p chlg-2015 | sort | uniq -c | sort -n -k1 | tail
    210 0b7c95d5a80232a053290ea06d2fb0cf
    225 00a7342de61ab4b16a4683278e01e34f
    239 3d06a766ef80a06a053b80221b2c3d73
    242 3b7257deb0bd655d54e6a19621f5fccf
    244 f43238a9941a0df61eef2b5407f0f75c
    292 ea12dfa3e6261e8f2a80a84bda0caada
    326 96eadc039da237294066f0eed246e26c
    347 ff6757e5b67d8fbdf67d886987ec2512
```

```
   3563 e40ccb764b52626e0ee0c966b1c8c24b
 145511 ad07b6efc462dffead318a9501a14390
```

Ainsi, le fichier a été chiffré par un algorithme qui a produit 3563 fois la séquence `e40c cb76 4b52 626e 0ee0 c966 b1c8 c24b` et 145511 fois `ad07 b6ef c462 dffe ad31 8a95 01a1 4390`. Il s'agit donc probablement d'un algorithme de chiffrement par bloc qui utilise le mode Electronic CodeBook (ECB) [1]. En tentant les algorithmes supportés par openssl qui utilisent le mode ECB (dont la liste dans dans la documentation [2]) et en devinant quel mot de passe peut être utilisé, on trouve que le fichier a été chiffré par AES-128 en mode ECB avec le mot de passe `SSTIC` :

```
% openssl enc -d -aes-128-ecb -in chlg-2015 -out decrypted -k sstic
% file decrypted
decrypted: PC bitmap, Windows 95/NT4 and newer format, 1000 x 1000 x 24
```

Le fichier déchiffré est donc une image bitmap, que voici.

FIGURE 2 – Image déchiffrée

Malheureusement l'adresse e-mail n'a pas pour domaine @challenge.sstic.org. Ce premier petit challenge est donc un poisson d'avril qui permet toutefois d'avoir un avant-goût du véritable challenge, au cours duquel il faut également déchiffrer des fichiers en trouvant des astuces.

1. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB
2. https://www.openssl.org/docs/apps/enc.html#SUPPORTED-CIPHERS

1 Un Rubber Ducky qui parle en PowerShell

1.1 De la carte microSD au script Rubber Ducky

Le vrai challenge commence sur http://communaute.sstic.org/ChallengeSSTIC2015, par le téléchargement d'un fichier qui contient le contenu d'une "carte microSD qui était insérée dans une clé USB étrange".

L'archive zip du challenge [3] contient un seul fichier, nommé `sdcard.img`. Il s'agit probablement d'une copie brute du contenu de la carte microSD qu'il est demandé d'analyser. La commande `file` permet de déterminer qu'il s'agit d'un système de fichier FAT :

```
% file sdcard.img
sdcard.img: DOS/MBR boot sector, code offset 0x3c+2, OEM-ID "mkfs.fat",
sectors/cluster 4, root entries 512, Media descriptor 0xf8, sectors/FAT 244,
sectors/track 32, heads 64, sectors 250000 (volumes > 32 MB) , serial number
0xe50d883b, unlabeled, FAT (16 bit)
```

Ce système de fichier ne contient qu'un seul fichier, nommé `inject.bin`, qui ne contient pas de texte. Par ailleurs, l'exécution de `strings` sur `sdcard.img` révèle un contenu intéressant :

```
% strings -tx -5 sdcard.img | tail -3
  3d221 UILD   SH
  3d260 INJECT  BIN
  41a00 java -jar encoder.jar -i /tmp/duckyscript.txt
```

En observant les octets aux alentours de ces chaînes de caractères on comprend mieux leur contexte :

```
% xxd -a sdcard.img | grep '^0003d'
0003d200: e562 0075 0069 006c 0064 000f 00bd 2e00  .b.u.i.l.d......
0003d210: 7300 6800 0000 ffff ffff 0000 ffff ffff  s.h.............
0003d220: e555 494c 4420 2020 5348 2020 0000 2d1e  .UILD   SH  ..-.
0003d230: 7a46 7a46 0000 2d1e 7a46 0300 2d00 0000  zFzF..-.zF..-...
0003d240: 4169 006e 006a 0065 0063 000f 00e4 7400  Ai.n.j.e.c....t.
0003d250: 2e00 6200 6900 6e00 0000 0000 ffff ffff  ..b.i.n.........
0003d260: 494e 4a45 4354 2020 4249 4e20 0000 3a1e  INJECT  BIN ..:.
0003d270: 7a46 7a46 0000 3a1e 7a46 0400 a2ab 0a02  zFzF..:.zF......
0003d280: 0000 0000 0000 0000 0000 0000 0000 0000  ................

% xxd -a sdcard.img | grep '^00041'
00041a00: 6a61 7661 202d 6a61 7220 656e 636f 6465  java -jar encode
00041a10: 722e 6a61 7220 2d69 202f 746d 702f 6475  r.jar -i /tmp/du
00041a20: 636b 7973 6372 6970 742e 7478 7400 0000  ckyscript.txt...
00041a30: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Le contenu à la position 3d200 ressemble fortement à une table de dossier qui contient deux entrées :
— la première (de 3d200 à 3d23f) correspond à un fichier supprimé qui s'appelait `build.sh`,
— et la seconde (de 3d240 à 3d27f) correspond au fichier `inject.bin` qui a déjà été trouvé.

Le texte à la position 41a00 correspondrait alors au contenu du fichier supprimé.

En résumé, la carte SD qui était insérée dans une clé USB étrange contient un fichier `inject.bin` de format non trivial et un fichier supprimé `build.sh` qui contenait : `java -jar encoder.jar -i /tmp/duckyscript.txt`.

Ceci fait beaucoup d'indices qui laissent penser que la clé USB étrange est un Rubber Ducky [4]. Ce périphérique USB simule un clavier sur lequel les frappes correspondantes au fichier `inject.bin` présent sur la carte microSD embarquée sont entrées. Le contenu de `inject.bin` est le résultat de la compilation d'un script Rubber Ducky par un programme opensource écrit en Java et appelé Encoder [5].

Chaque instruction du Rubber Ducky est composée de deux octets. Le premier indique quelle touche a été enfoncée et le second quels modificateurs ont été utilisés (touches Control, Majuscule, Alt, Windows). Afin de comprendre la séquence de frappes claviers indiquée par `inject.bin`, j'ai écrit un programme Python qui effectue l'inverse du mécanisme d'encodage du programme Encoder :

3. http://static.sstic.org/challenge2015/challenge.zip
4. http://usbrubberducky.com/
5. https://github.com/hak5darren/USB-Rubber-Ducky

```
#!/usr/bin/env python3
"""Decode the Rubber Ducky inject.bin compiled script"""
import struct
import sys

# Build a (opcode, modifier)-to-char dictonary
OM2C = {
    (0x1e, 0): '1', (0x1e, 2): '!',
    (0x1f, 0): '2', (0x1f, 2): '@',
    (0x20, 0): '3', (0x20, 2): '#',
    (0x21, 0): '4', (0x21, 2): '$',
    (0x22, 0): '5', (0x22, 2): '%',
    (0x23, 0): '6', (0x23, 2): '^',
    (0x24, 0): '7', (0x24, 2): '&',
    (0x25, 0): '8', (0x25, 2): '*',
    (0x26, 0): '9', (0x26, 2): '(',
    (0x27, 0): '0', (0x27, 2): ')',
    (0x28, 0): '[ENTER]\n', (0x29, 0): '[ESC]\n',
    (0x2b, 0): '\t', (0x2c, 0): ' ',
    (0x2d, 0): '-', (0x2d, 2): '_',
    (0x2e, 0): '=', (0x2e, 2): '+',
    (0x2f, 0): '[', (0x2f, 2): '{',
    (0x30, 0): ']', (0x30, 2): '}',
    (0x31, 0): '\\', (0x31, 2): '|',
    (0x33, 0): ':', (0x33, 2): ';',
    (0x34, 0): "'", (0x34, 2): '"',
    (0x35, 0): '`', (0x35, 2): '~',
    (0x36, 0): ',', (0x36, 2): '<',
    (0x37, 0): '.', (0x37, 2): '>',
    (0x38, 0): '/', (0x38, 2): '?',
}
# Alphabet
for i in range(26):
    OM2C[(i + 4, 0)] = chr(ord('a') + i)
    OM2C[(i + 4, 2)] = chr(ord('A') + i)

delay_time = 0
with open('inject.bin', 'rb') as f:
    while True:
        injectdata = f.read(2)
        if len(injectdata) == 0:
            break
        opcode, modifier = struct.unpack('BB', injectdata)

        # "DELAY" is encoded with successive opcode-0 commands
        if opcode == 0:
            delay_time += modifier
            continue
        if delay_time:
            print("[DELAY {}]".format(delay_time))
            delay_time = 0

        # WIN key + letter is encoded with modifier 8
        if modifier & 8:
            c = OM2C.get((opcode, modifier & ~8))
            if c is not None:
                print("[WIN {}]".format(c))
                continue

        sys.stdout.write(OM2C.get((opcode, modifier)))
```

Le résultat de ce script commence par :

```
[DELAY 2000]
[WIN r]
[DELAY 500]
[ENTER]
[DELAY 1000]
cmd[ENTER]
[DELAY 50]
powershell -enc ZgB1AG4AYwB0AGkAbwBuACAAdwByAGkAdABlAF8AZgBpAGwAZQBfAGIAeQB0AGUAcwB7AHAAY
...
```

Cette séquence de commande se traduit en français par : lorsque le Rubber Ducky est connecté, il attend tout d'abord deux secondes, puis simule l'appui sur les touches Windows et R du clavier, attend une demi-seconde, appui sur Entrée, attend une seconde, entre "cmd" et la touche Entrée, attend 50 millisecondes, entre "powershell -enc ZgB1AG4AYwB"... On comprend alors que le script Rubber Ducky utilisé cible les systèmes Windows, où la séquence Win-R permet d'exécuter une commande, et que le script fait ouvrir une fenêtre de commande dans lequel il injecte des invocations PowerShell.

1.2 Commandes PowerShell injectées

L'étape suivante consiste donc à décoder les commandes PowerShell qui sont injectées par le Rubber Ducky. L'option "-enc" qui est utilisée dans les invocations de PowerShell correspond à l'option "-EncodedCommand" qui permet d'exécuter une commande encodée en base 64 [6]. Une fois décodées, les commandes PowerShell exécutées sont de la forme (sans les espaces ni sauts de ligne, qui ont été ajoutés pour améliorer la lisibilité) :

```
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('UEsDBAoDAAAAAD....gUsBzWnXw='));
} else {
    write_file_bytes([Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));
}
```

... où la séquence `UEsDBAoDAAAAAD....gUsBzWnXw=` varie selon les lignes. Ainsi, les commandes vérifient que la concaténation du dossier courant et du nom d'utilisateur est `challenge2015sstic`. Si c'est le cas, du contenu qui est encodé en base64 est ajouté à la fin du fichier `stage2.zip`. Sinon, c'est le contenu encodé "VAByAHkA-SABhAHIAZABlAHIA" qui est ajouté, et ceci correspond au texte "TryHarder".

La dernière invocation PowerShell utilise un script différent des précédents :

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

6. https://technet.microsoft.com/fr-fr/library/hh847736.aspx

```
    echo "You LOSE";
}
```

Ce script calcule la somme de contrôle SHA-1 du fichier `stage2.zip`, la compare à une valeur de référence, et indique si cette comparaison est bonne. Avec tout cela il est possible de construire `stage2.zip` et de vérifier le résultat en calculant la somme de contrôle SHA-1 (le code du script Python utilisé est disponible en A.1).

```
% sha1sum stage2.zip
ea9b8a6f5b527e72652019313c25b56ad27c7ec6  stage2.zip
```

Ainsi la carte microSD contient un programme Rubber Ducky qui crée sur un système Windows qui vérifie certaines conditions (en particulier le fait que lancer `cmd` crée une interface de ligne de commande dans un dossier dont le nom concaténé avec le nom d'utilisateur est `challenge2015sstic`) une archive `stage2.zip`. La partie suivante du challenge consiste logiquement à étudier le contenu de cette archive.

2 Carte OpenArena

2.1 Découverte de l'archive

L'archive `stage2.zip` récupérée à l'étape précédente contient trois fichiers :
- `encrypted`
- `memo.txt`
- `sstic.pk3`

`memo.txt` contient des indications sur le contenu des deux autres fichiers :

```
Cipher: AES-OFB
IV: 0x53535449433230313352d537461676532
Key: Damn... I ALWAYS forget it. Fortunately I found a way to hide it into my favorite game !

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

Le fichier `encrypted` est donc chiffré en utilisant l'algorithme AES-OFB (Advanced Encryption Standard, Output FeedBack), le vecteur d'initialisation `0x53535449433230313352d537461676532` (soit "SSTIC2015-Stage2"), et une clé cachée dans un jeu qu'il reste à déterminer.

L'extension `.pk3` est utilisée dans certains jeux (apparemment ceux dérivés de Quake III selon Wikipedia [7]) pour indiquer des fichiers zip. Il est ainsi possible d'extraire le contenu de `sstic.pk3`, qui contient :
- un fichier `README` qui indique : "Copy the pk3 in your baseoa directory. In the game, open the console (²) and type \map sstic.",
- un fichier `AUTHORS` qui crédite "Open Iconic v1.1.1 – useiconic.com" et "085am_underworks2 v0.8.5 by Neon_Knight – openarena.wikia.com",
- une image `levelshots/sstic.tga` qui contient les logos du SSTIC et du jeu OpenArena,
- un fichier `maps/sstic.bsp` qui contient des chaînes de caractères intéressantes comme "Yes !\n You found my key !", mais pas d'adresse e-mail,
- un fichier `scripts/sstic.arena` au contenu peu intéressant,
- un fichier son `sound/world/bj3.wav`, et
- un dossier `textures/sstic/` rempli de textures au format TGA.

On comprend ainsi que le jeu dont il est question dans `memo.txt` est OpenArena, et que `sstic.pk3` est une carte OpenArena dérivée de 085am_underworks2 [8]. Pour utiliser cette carte, il suffit d'installer OpenArena (qui est un jeu libre) et de copier `sstic.pk3` dans le dossier `.openarena/baseoa/` du répertoire personnel, sur Linux. Ensuite, au lancement du jeu, il faut activer la console (avec la touche ² ou en appuyant sur Majuscule-Échap selon la configuration) et entrer la commande `\map sstic`.

2.2 Exploration de la carte

Au lancement de la carte `sstic` dans OpenArena, le joueur est placé dans une petite pièce dans laquelle sur un mur se trouve un carré sur fond bleu (que j'appelle "tuile") avec trois nombres hexadécimaux en couleur et un soleil noir (figure 3).

Un peu d'exploration de la carte révèle une grande salle avec des colonnes et des plateformes en hauteur avec des échelles pour y accéder, une autre petite pièce avec une grande cage au milieu, un couloir "au sous-sol" rempli d'eau qui contourne la seconde pièce. L'exploration permet de trouver d'autres tuiles similaires à celle observée dans la salle de départ, ainsi que deux boutons. Un bouton est situé au niveau des plateformes en hauteur de la grande salle et son activation a pour effet l'affichage d'un message "15 seconds ..." au centre de l'écran. L'autre bouton est situé sur un mur de la pièce contenant la cage, son activation entraîne l'affichage du message "The secret area is now open during 30 seconds !" au centre de l'écran. La "zone secrète" qui est ainsi ouverte correspond à un panneau derrière un gros logo SSTIC dans la grande salle, qui se lève pour révéler un troisième bouton pendant les 30 secondes suivant l'activation du second bouton (figure 4).

En tirant sur le troisième bouton, le joueur est téléporté dans une zone en dehors de la carte initiale, au milieu des étoiles, sur un chemin rectiligne traversé par des bassins de ce qu'il semble être de la lave. Après avoir sauté au dessus de deux bassin, le joueur obtient un lance-roquette et le message "Time to Rocket Jump ? !" s'affiche à l'écran (figure 5).

Le jeu semble indiquer qu'il faille utiliser un Trickjump [9] avec le lance-roquette [10]. Néanmoins ce saut est très difficile à réaliser et il ne faut pas oublier que ce jeu n'est qu'une étape d'un challenge de sécurité informatique (par opposition à un jeu réel). Il est possible d'utiliser quelques codes de triche [11] pour réussir le saut.

7. https://en.wikipedia.org/wiki/PK3_%28file_extension%29
8. http://www.mapraider.com/maps/openarena/domination/5910/am_underworks2-(0-8-5)
9. https://en.wikipedia.org/wiki/Trickjump
10. http://www.trickingq3.com/Tutorials/Rocket_Tutorial.html
11. http://quake.wikia.com/wiki/Cheats_and_Console_Commands

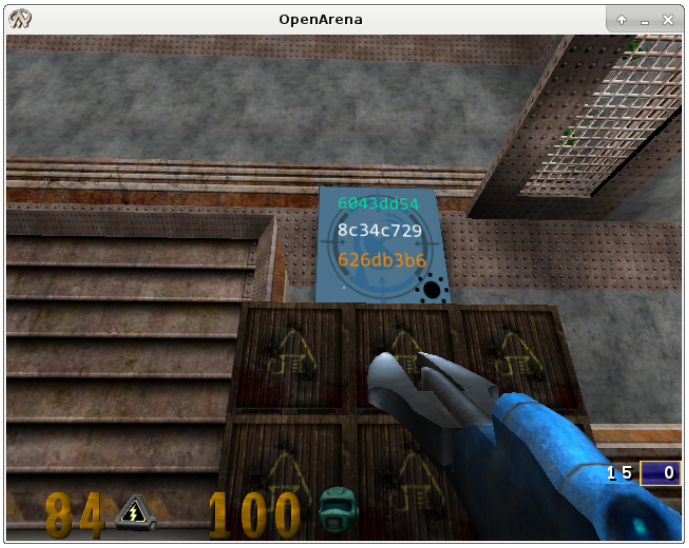


FIGURE 3 – Tuile hexadécimale dans la salle de départ

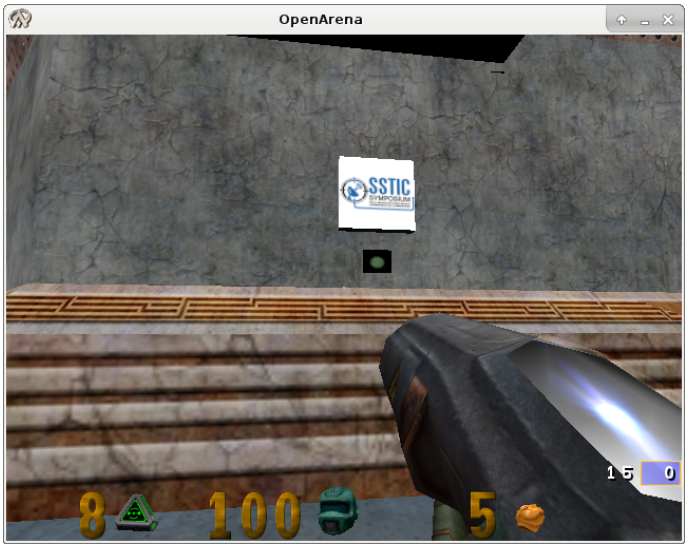


FIGURE 4 – Troisième bouton dans la “zone secrète”

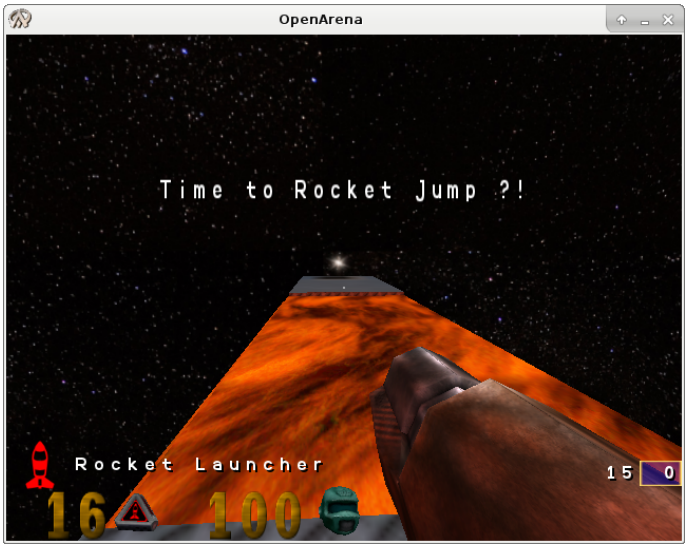


FIGURE 5 – Le défi du saut roquette

Par exemple :

— `\devmap sstic` permet de charger la carte avec les codes de triche activés,
— `\god` permet d'être invincible et donc de ne pas mourir dans la lave,
— `\noclip` permet de voler à travers la carte et de traverser les murs,
— `\setviewpos -1900 610 -440 90` permet de se téléporter juste après le saut difficile.

Après le saut, le joueur trouve un quatrième bouton en hauteur. Tirer dessus a pour effet d'être téléporté dans une salle qui contient sur le mur un assemblage de symboles et de couleurs (figure 6).

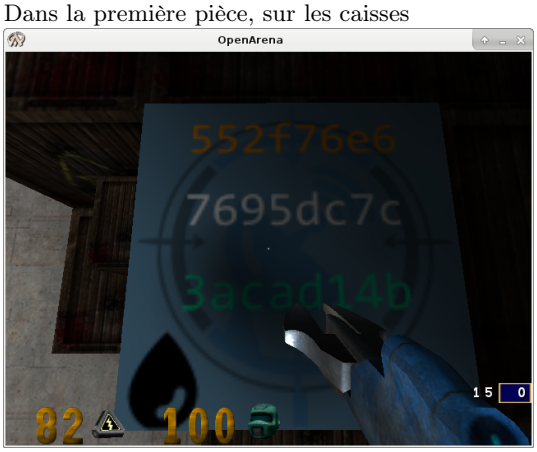
FIGURE 6 – La clé de chiffrement dans le jeu

Cela signifie certainement qu'il faut trouver dans la carte chaque tuile indiquée par les symboles et combiner les nombres hexadécimaux des couleurs correspondantes pour obtenir la clé.

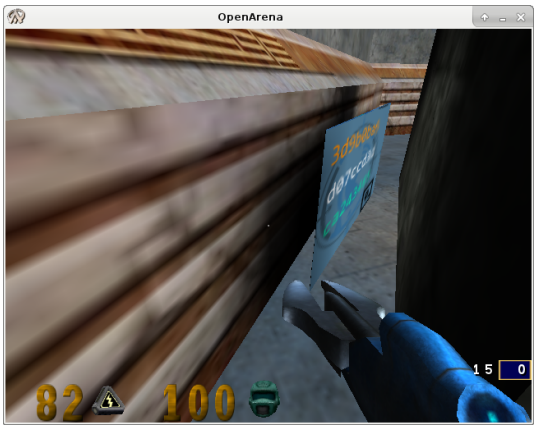
2.3 À la recherche des tuiles

Voici où se trouve chaque tuile.

— Dans la première pièce, sur les caisses

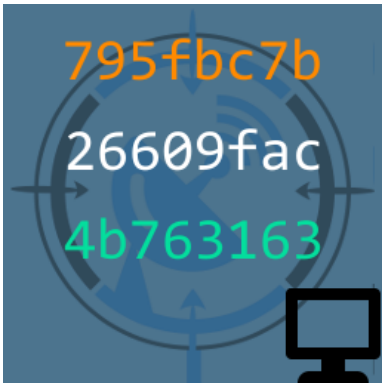


— Dans la grande salle, entre un bloc et un mur



— Dans la grande salle, sur la plateforme mouvante

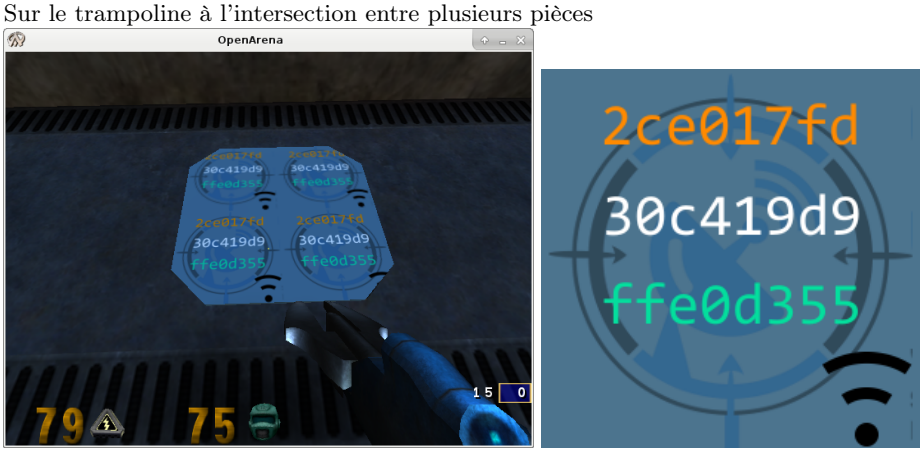
— Dans la petite pièce avec la cage, au dessus du bloc dans la cage



— Sous l'escalier qui mène au couloir rempli d'eau

— Près de l'escalier, une fois le premier bouton activé

— Sur le trampoline à l'intersection entre plusieurs pièces



Ceci permet de décoder la clé :

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

Finalement le fichier chiffré peut être déchiffré avec une commande `openssl`, en enlevant manuellement le padding.

```
% openssl enc -d -aes-256-ofb -in encrypted \
    -iv 5353544943323031352d537461676532 \
    -K 9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac | \
  head -c -16 > decrypted

% sha256sum decrypted
845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187  decrypted

% file decrypted
decrypted: Zip archive data, at least v1.0 to extract
```

La somme de contrôle du fichier déchiffré correspond bien à celle indiquée dans `memo.txt`, et ce fichier semble être une archive zip. La partie suivante analyse le contenu de cette archive.

3 Capture le Paint !

La partie précédente a permis de déchiffrer un fichier en retrouvant la clé caché dans une carte OpenArena. Le fichier déchiffré est une archive zip qui contient trois fichiers :

- `encrypted`
- `memo.txt`
- `paint.cap`

Encore une fois, `memo.txt` contient des indications sur le contenu des deux autres fichiers :

```
Cipher: Serpent-1-CBC-With-CTS
IV: 0x53535449433230313352d537461676533
Key: Well, definitely can't remember it... So this time I securely stored it with Paint.

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

Ce fichier indique donc l'algorithme de chiffrement utilisé, le vecteur d'initialisation (qui est "SSTIC2015-Stage3" en hexadécimal) et deux sommes de contrôle, une pour le fichier chiffré (qui correspond bien au `encrypted` extrait de l'archive) et une pour le fichier déchiffré.

Pour comprendre ce que l'auteur de `memo.txt` entend par "enregistrer de manière sécurisée avec Paint", il faut analyser `paint.cap`. La commande `file` indique qu'il s'agit d'une capture de paquets USB :

```
% file paint.cap
paint.cap: tcpdump capture file (little-endian) - version 2.4
(Memory-mapped Linux USB, capture length 262144)
```

Un tel fichier peut être analysé avec Wireshark [12]. Ceci permet de s'apercevoir que ce fichier contient :

- 6 paquets dans lesquels l'hôte demande les descripteurs de périphérique à 3 clients et dans lesquels ceux-ci répondent, et en particulier le périphérique 3 annonce être une souris USB.
- Ensuite beaucoup de paquets dans lesquels le périphérique 3 envoie 4 octets et l'hôte accuse réception.

Les sources du noyau Linux permettent de comprendre la signification des 4 octets envoyés par la souris dans chaque paquet (fonction `usb_mouse_irq` de `drivers/hid/usbhid/usbmouse.c` [13]) :

- le premier octet encode l'état des boutons (le bit de poids faible correspond au bouton gauche),
- le second encode le déplacement relatif selon la coordonnée X (de -128 à 127),
- le troisième encode le déplacement relatif selon la coordonnée Y (de -128 à 127), et
- le dernier encode le déplacement relatif selon la molette.

L'auteur a donc certainement enregistré les messages envoyés par sa souris pendant qu'il dessinait la clé de chiffrement dans Paint, ce qui explique son message dans `memo.txt`.

Pour reconstituer l'image dessinée, il suffit de suivre les mouvements de la souris en additionnant les déplacements relatifs, et de dessiner dans une image un point noir quand la souris se déplace avec le bouton gauche appuyé.

Voici un script Python qui utilise Scapy [14] pour décoder les paquets USB et Pillow [15] pour dessiner. La figure 7 présente le résultat obtenu.

```python
#!/usr/bin/env python2
"""Decode the mouse movements in paint.cap"""
from scapy.all import *
from PIL import Image, ImageDraw

class URB(Packet):
    fields_desc = [
        LELongField('URB_id', 0),
        ByteField('URB_type', 0),
        ByteField('URB_transfer_type', 0),
        BitField('URB_direction', 0, 1),
        BitField('URB_endpoint', 0, 7),
        ByteField('URB_device', 0),
        LEShortField('URB_busid', 0),
        ByteField('device_setup_rq', 0),
        ByteField('data_ispresent', 0),
        LELongField('URB_sec', 0),
```

12. https://www.wireshark.org/
13. http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/hid/usbhid/usbmouse.c?id=v3.19#n81
14. http://www.secdev.org/projects/scapy/
15. http://python-pillow.github.io/

```
        LEIntField('URB_usec', 0),
        LEIntField('URB_status', 0),
        LEIntField('URB_length', 0),
        LEIntField('data_length', 0),
        XLongField('UNKNOWN', 0),
        LEIntField('interval', 0),
        LEIntField('start_frame', 0),
        LEIntField('copy_transfer_flags', 0),
        LEIntField('num_iso_desc', 0),
    ]

class SignedByteField(Field):
    def __init__(self, name, default):
        Field.__init__(self, name, default, fmt="b")

class URBMouseInput(Packet):
    fields_desc = [
        ByteField('buttons', 0),
        SignedByteField('rel_x', 0),
        SignedByteField('rel_y', 0),
        SignedByteField('rel_z', 0),
    ]

conf.l2types.register(220, URB)

# List of mouse positions (btn, x, y) relative to the initial position
mousemove_list = []
packets = rdpcap('paint.cap')
x = y = 0
for urbpkt in packets[6:]:  # Skip the first 6 packets
    if urbpkt.data_length == 0:
        continue
    mouse_in = URBMouseInput(urbpkt.payload.load)
    x += mouse_in.rel_x
    y += mouse_in.rel_y
    mousemove_list.append((mouse_in.buttons, x, y))

# Replay the mouse move movements in an image
xmin = min(mm[1] for mm in mousemove_list)
xmax = max(mm[1] for mm in mousemove_list)
ymin = min(mm[2] for mm in mousemove_list)
ymax = max(mm[2] for mm in mousemove_list)
im = Image.new('RGB', (xmax + 1 - xmin, ymax + 1 - ymin), 'white')
draw = ImageDraw.Draw(im)
for btn, x, y in mousemove_list:
    if btn:
        draw.point((x - xmin, y - ymin), fill='black')
im.save('paint-out.png', 'PNG')
```

Ainsi, la clé de chiffrement est le résultat de `BLAKE256("The quick brown fox jumps over the lobster dog")`. La fonction BLAKE est un algorithme de hachage qui était finaliste dans le concours qui a déterminé SHA-3 [16] et même s'il n'est pas implémenté dans `openssl`, l'implémentation de référence est disponible sous licence CC0 [17]. Ceci permet de trouver la clé de chiffrement :

```
% printf 'The quick brown fox jumps over the lobster dog' | ./blake256 /dev/stdin
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d /dev/stdin
```

L'algorithme de chiffrement utilisé, Serpent, a été finaliste dans le concours qui a déterminé AES et n'est pas implémenté dans `openssl`. Il faut donc écrire un programme court pour déchiffrer le fichier `encrypted`. J'ai trouvé deux implémentations facilement utilisables de Serpent, l'une dans un module PHP [18], l'autre dans la

16. https://en.wikipedia.org/wiki/BLAKE_%28hash_function%29
17. https://github.com/veorq/BLAKE
18. http://php.net/manual/en/mcrypt.ciphers.php

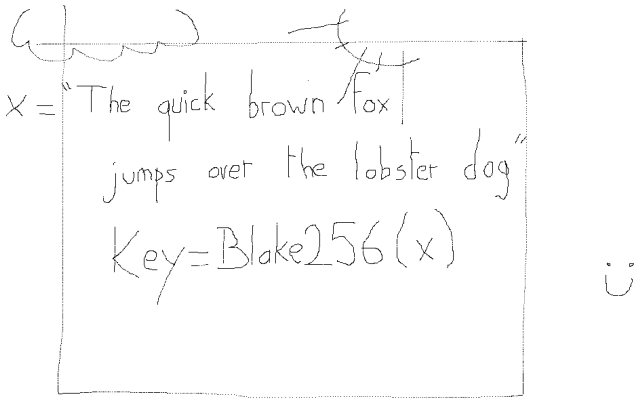


FIGURE 7 – Dessin reconstitué à partir de la capture des paquets USB

bibliothèque C++ Crypto++ [19]. Comme je n'avais encore jamais écrit de programme avec Crypto++, j'ai utilisé cette bibliothèque dans la résolution du challenge. Voici le programme C++ qui permet de déchiffrer `encrypted`.

```
#include <cryptopp/serpent.h>
#include <cryptopp/modes.h>
#include <cryptopp/filters.h>
#include <fstream>
#include <iostream>

int main()
{
    const unsigned char key[32] = {
        0x66, 0xc1, 0xba, 0x5e, 0x8c, 0xa2, 0x9a, 0x8a, 0xb6, 0xc1, 0x05, 0xa9,
        0xbe, 0x9e, 0x75, 0xfe, 0x0b, 0xa0, 0x79, 0x97, 0xa8, 0x39, 0xff, 0xea,
        0xe9, 0x70, 0x0b, 0x00, 0xb7, 0x26, 0x9c, 0x8d
    };
    const unsigned char iv[16] = {
        0x53, 0x53, 0x54, 0x49, 0x43, 0x32, 0x30, 0x31, 0x35, 0x2d, 0x53, 0x74,
        0x61, 0x67, 0x65, 0x33
    };

    /* Read encrypted */
    std::ifstream encrypted_file("encrypted", std::ios::in|std::ios::binary);
    char enc_data[296798];
    encrypted_file.read(enc_data, sizeof(enc_data));
    encrypted_file.close();
    std::string encrypted(enc_data, sizeof(enc_data));

    /* Decrypt with Serpent-1-CBC-With-CTS */
    CryptoPP::CBC_CTS_Mode<CryptoPP::Serpent>::Decryption decipher;
    decipher.SetKeyWithIV(key, sizeof(key), iv, sizeof(iv));
    std::string decrypted;
    CryptoPP::StringSource ss(
        encrypted, true,
        new CryptoPP::StreamTransformationFilter(decipher, new CryptoPP::StringSink(decrypted))
    );

    /* Write decrypted */
    std::ofstream decrypted_file("decrypted", std::ios::out|std::ios::binary|std::ios::trunc);
    decrypted_file << decrypted;
    decrypted_file.close();
    return 0;
}
```

19. http://www.cryptopp.com/

J'ai compilé puis exécuté ce programme sur Linux avec les commandes suivantes :

```
% g++ -Wall -Wextra decrypt.cpp -o decrypt -lcryptopp -lpthread
% ./decrypt
```

Ceci permet d'obtenir le fichier `decrypted` dont la somme de contrôle SHA-256 était dans `memo.txt`. Il est donc possible de vérifier cette somme de contrôle et de se rendre compte que ce fichier est, encore une fois, une archive zip avec le contenu de l'étape suivante.

```
% sha256sum decrypted
7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2  decrypted

% file decrypted
decrypted: Zip archive data, at least v2.0 to extract
```

4 Cryptographie web

4.1 Décodage de Javascript obfusqué

L'archive zip précédemment déchiffrée contient un seul fichier, nommé `stage4.html`. Ce fichier commence par définir une feuille de style CSS puis contient entre les balises `<body>` et `</body>` trois lignes de Javascript particulièrement longues :

— la première définit une variable `data` comme une chaîne de 528032 caractères hexadécimaux,
— la seconde est `var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";`, et
— la troisième contient du code Javascript sur 11580 caractères, sans aucun qui soit alphanumérique.

Les 5 premières instructions de la troisième ligne correspondent à du code Javascript obfusqué qui permet de construire des chaînes de caractères. Voici ces instructions, avec des espaces et des commentaires pour mieux les comprendre.

```
$ = ~[];                    // $ = -1
$ = {
    ___: ++$,               // $.___ = 0
    $$$$: (![] + "")[$],    // $.$$$$ = "false"[0] = "f"
    __$: ++$,               // $.__$ = 1
    $_$_: (![] + "")[$],    // $.$_$_ = "a"
    _$_: ++$,               // $._$_ = 2
    $_$$: ({} + "")[$],     // $.$_$$ = "[object Object]"[2] = "b"
    $$_$: ($[$] + "")[$],   // $.$$_$ = ""undefined"[2] = "d"
    _$$: ++$,               // $._$$ = 3
    $$$_: (!"" + "")[$],    // $.$$$_ = "true"[3] = "e"
    $__: ++$,               // $.$__ = 4
    $_$: ++$,               // $.$_$ = 5
    $$__: ({} + "")[$],     // $.$$__ = "c"
    $$_: ++$,               // $.$$_ = 6
    $$$: ++$,               // $.$$$ = 7
    $___: ++$,              // $.$___ = 8
    $__$: ++$               // $.$__$ = 9
};
$.$_ = ($.$_ = $ + "")[$.$_$] +      // $.$_ = '[object Object]', add "c"
    ($._$ = $.$_[$.__$]) +           // $._$ = "o"
    ($.$$ = ($.$ + "")[$.__$]) +     // $.$$ = "n"
    ((!$) + "")[$._$$] +             // add "false"[3] = "s"
    ($.__ = $.$_[$.$$_]) +           // $.__ = "t"
    ($.$ = (!"" + "")[$.__$]) +      // $.$ = "r"
    ($._ = (!"" + "")[$._$_]) +      // $._ = "u"
    $.$_[$.$_$] + $.__ + $._$ + $.$; // add "ctor", $.$_ = "constructor"
$.$$ = $.$ + (!"" + "")[$._$$] + $.__ + $._ + $.$ + $.$$; // $.$$ = "return"
$.$ = ($.___)[$.$_][$.$_];           // $.$ = 0["constructor"]["constructor"]
```

Après l'exécution de ces instructions, la variable `$.$_` contient la chaîne de caractère `constructor` et `$.$$` `return`. Le contenu de `$.$` est plus complexe à comprendre. Comme `0["constructor"]` renvoie la fonction interne `Number`[20], `0["constructor"]` renvoie l'attribut `constructor` de `Number`, qui est la fonction interne `Function`. Ainsi, `$.$` contient en quelque sorte un pointeur vers la fonction `Function`.

La sixième instruction, qui occupe le reste du code Javascript, est de la forme :

```
$.$($.$($.$$+"\""+ ... +"\"")())();
```

... ce qui peut donc se décoder en :

```
Function( Function("return" + "\"" + ... + "\"")() )();
```

La partie correspondant aux points de suspensions permet donc de construire un code Javascript qui sera renvoyé sous forme de chaîne de caractère par le premier appel à `Function`, puis exécuté juste après le second appel de `Function`.

Au passage, ces instructions correspondent exactement à celles qui sont décrites dans l'article *Malware : Dollar Equals Tilde Square Brackets* du blog d'Avast[21], datant de février 2013, mais cet article ne donne pas de détail sur la manière de décoder le code exécuté.

Pour effectuer ce décodage, j'ai commencé par isoler les trois lignes de code Javascript obfusqué dans un fichier à part, nommé `stage4.js`. Ensuite, j'ai créé un fichier `show_jscode.html` dans le même dossier contenant :

20. http://www.w3schools.com/jsref/jsref_constructor_number.asp
21. https://blog.avast.com/2013/02/14/malware-dollar-equals-tilde-square-brackets/

```
<html><body>
<div id="code"></div>
<script>
oldFunction = 0['constructor']['constructor'];
0['constructor']['constructor'] = function (code) {
    document.getElementById('code').innerText = oldFunction(code)();
}
</script>
<script src="stage4.js"></script>
</body></html>
```

Le code Javascript de cette page HTML a pour effet d'appeler une fois la fonction `Function` pour obtenir une chaîne de caractère contenant le code Javascript décodé, et l'affiche tel quel.

Le code obtenu commence par `__=document;$$$='stage5';$$_$='load';` et définit ainsi un ensemble de variables nommées par des dollars et des soulignés. Après la définition de ces variables, quatre fonctions sont définies. La première ressemble à ceci (en ajoutant l'indentation) :

```
function _____(______) {
    _ = [];
    for (__________ = _______; __________ < ______[___________]; ++__________)
        _[____________________](______[____________________](__________));
    return new ______________________(_);
}
```

Comme ce code est absolument illisible, il est nécessaire de renommer les variables constituées de soulignés par exemple par le nombre de soulignés qu'elles contiennent, préfixé par la lettre `v`. Ceci peut être effectué automatiquement par cette commande shell :

```
% for i in $(seq 28 -1 3) ; do sed 's/_\{'$i'\}/v'$i'/g' -i decoded.js ; done
```

Après cette transformation, la fonction précédente devient :

```
function v5(v6) {
    _ = [];
    for (v11 = v7; v11 < v6[v12]; ++v11)
        _[v20](v6[v19](v11));
    return new v22(_);
}
```

En injectant également les définitions de variables précédentes, on obtient :

```
function v5(v6) {
    _ = [];
    for (v11 = 0; v11 < v6['length']; ++v11)
        _['push'](v6['charCodeAt'](v11));
    return new Uint8Array(_);
}
```

Il s'agit donc d'une fonction permettant de transformer une chaîne de caractères en tableau d'entiers (de type `Uint8Array`) contenant les codes numériques de chaque caractère. J'ai donc nommé cette fonction `str_to_u8arr` au cours de mon analyse.

En poursuivant le décodage du code en substituant les constantes là où elles sont utilisées, en transformant les accès aux attributs d'un object de `obj['attr']` en la notation équivalente `obj.attr`, en renommant quelques variables, et en ajoutant des commentaires, j'ai obtenu le code suivant.

```
$ = '<b>Failed to load stage5</b>';
useragent = 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>');
```

```
/* Convert a string to an Uint8Array */
function str_to_u8arr(v6) {
    _ = [];
    for (v11 = 0; v11 < v6.length; ++v11)
        _.push(v6.charCodeAt(v11));
    return new Uint8Array(_);
}
/* Decode an hexadecimal string to an Uint8Array */
function hexstr_to_u8arr(v6) {
    _ = [];
    for (v11 = 0; v11 < v6.length / 2; ++v11)
        _.push(parseInt(v6.substr(v11 * 2, 2), 16));
    return new Uint8Array(_);
}
/* Encode an Uint8Array to an hexadecimal string */
function u8arr_to_hexstr(v8) {
    v6 = '';
    for (v11 = 0; v11 < v8.byteLength; ++v11) {
         v3 = v8[v11].toString(16);
         if (v3.length < 2)
            v6 += 0;
         v6 += v3;
    }
    return v6;
}

function v27() {
    iv = str_to_u8arr(useragent.substr(useragent.indexOf('(') + 1, 16));
    key = str_to_u8arr(useragent.substr(useragent.indexOf(')') - 16, 16));
    _$ = {};
    _$['name'] = 'AES-CBC';
    _$['iv'] = iv;
    _$['length'] = key.length * 8;
    window.crypto.subtle.importKey("raw", key, _$, false, ['decrypt'])
    .then(function(importedKey) {
        window.crypto.subtle.decrypt(_$, importedKey, hexstr_to_u8arr(data))
        .then(function(result) {
            decrypted = new Uint8Array(result);
            window.crypto.subtle.digest({name: 'SHA-1'}, decrypted)
            .then(function(digest) {
                if (hash == u8arr_to_hexstr(new Uint8Array(digest))) {
                    myBlobAttributes = {};
                    myBlobAttributes['type'] = 'application/octet-stream';
                    myBlob = new Blob([decrypted], myBlobAttributes);
                    myURL = URL.createObjectURL(myBlob);
                    document.getElementById('status').innerHTML =
                        '<a href="' + myURL + '" download="stage5.zip">' +
                        'download stage5</a>';
                } else {
                    document.getElementById('status').innerHTML = $;
                }
            });
        }).catch(function() {
            document.getElementById('status').innerHTML = $;
        });
    }).catch(function() {
        document.getElementById('status').innerHTML = $;
    });
}
window.setTimeout(v27, 1000);
```

En résumé :

— Ce script commence par afficher “Download manager” et “loading...”.
— Il affiche également un lien caché vers `chrome://browser/content/preferences/preferences.xul`.
— Après une seconde, il exécute la fonction nommée avec 27 soulignés (nommée `v27` précédemment).
— Cette fonction détermine un vecteur d'initialisation et une clé de chiffrement en copiant respectivement les 16 premiers et les 16 derniers caractères de la première section entre parenthèses de l'agent utilisateur (obtenu par `window.navigator.userAgent`).
— Ensuite elle déchiffre le contenu de la variable `data` lue comme une chaîne de caractères hexadécimaux avec l'algorithme AES en mode CBC.
— La somme de contrôle SHA-1 du contenu déchiffré est comparée à la variable `hash`.
— Si la somme de contrôle correspond, le message “loading...“ est remplacé par un lien de téléchargement portant le nom `stage5.zip`.
— Si une erreur est survenue précédemment, le message “loading...“ est remplacé par le texte “Failed to load stage5”.

4.2 Déchiffrement

Afin de déchiffrer le contenu de la variable `data`, il faut deviner le contenu à l'intérieur de la première parenthèse de l'agent utilisateur d'un navigateur particulier, ce qui permet d'obtenir le vecteur d'initialisation et la clé utilisée.

Pour cela, voici une liste d'hypothèses réalistes qu'il est possible d'établir :

— Le navigateur utilisé est Firefox ou un de ses dérivés, car le lien caché `chrome://browser/content/preferences/preferences.xul` n'existe que dans ces types de navigateurs.
— Le système d'exploitation sur lequel le navigateur est installé est Windows, pour que le challenge soit cohérent avec la première étape, dans laquelle le script Rubber Ducky est spécifique à Windows.
— La version du navigateur utilisé est assez récente.
— Le fichier déchiffré est une archive zip, car le lien de téléchargement est nommé `stage5.zip`.

Ces hypothèses permettent de restreindre la recherche de l'agent utilisateur concerné aux versions de Firefox pour Windows situées entre les versions 29.0 et 37.0 (qui correspondent à la période de mi-2014 à avril 2015 [22]).

Un tel agent utilisateur ressemble à :

```
Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0
```

Pour cet agent particulier, le vecteur d'initialisation (qui contient les 16 premiers caractères de ce qui est entre parenthèses) est “`Windows NT 6.3; `”, et la clé (qui contient les 16 derniers caractères de la parenthèse) est “ `NT 6.3; rv:36.0`”.

De rapides essais montrent que les quelques couples (vecteur d'initialisation, clé) que j'ai tentés sont mauvais. Il faut donc élargir le bruteforce. Mais avant cela, il est pertinent de remarquer que le mode CBC combiné au chiffrement d'une archive zip permet de tester les clés indépendamment du vecteur d'initialisation. En effet, les données déchiffrées après le premier bloc (de 16 octets) ne dépendent pas du vecteur d'initialisation mais uniquement de la clé, comme indiqué par le schéma suivant.

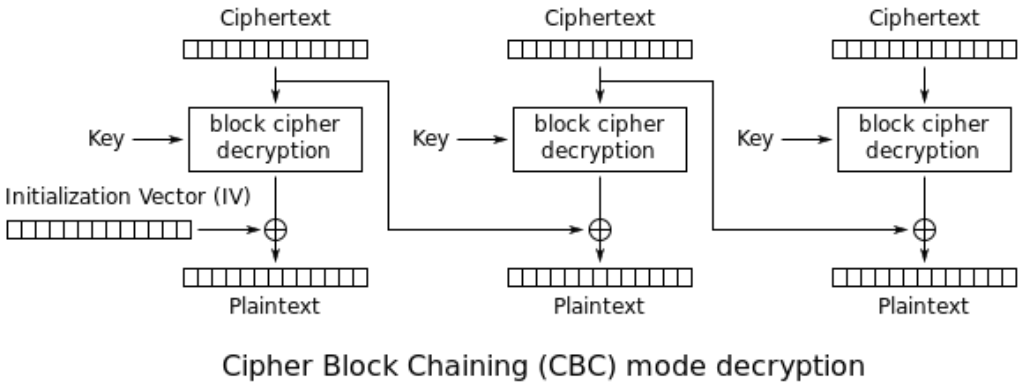


FIGURE 8 – Mode d'opération CBC. Source : https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CBC

De plus, une archive zip est décodée à partir de la fin du fichier, et il est possible de vérifier la présence de la signature `PK` aux positions relatives -22 et -21 pour tester si une clé a une chance d'être correcte.

En prenant également en compte le rembourrage qui est ajouté au fichier clair avant chiffrement, le script suivant permet de tester un ensemble d'agents utilisateurs.

22. https://en.wikipedia.org/wiki/Firefox_release_history

```python
#!/usr/bin/env python3
"""Brute-force the User-Agent used to encrypt a zip archive"""
from Crypto.Cipher import AES
from hashlib import sha1

from . import generate_useragents

# Read the encrypted data and the SHA-1 hash of the decrypted data
with open('stage4.js', 'r') as f:
    encrypted_hex = f.readline().strip().lstrip('var data = "').rstrip('";')
    encrypted = bytes.fromhex(encrypted_hex)
    hash_hex = f.readline().strip().lstrip('var hash = "').rstrip('";')

for useragent in generate_useragents.gen_ua():
    iv = useragent[useragent.index('(') + 1:][:16].encode('ascii')
    key = useragent[useragent.index(')') - 16:][:16].encode('ascii')
    assert len(iv) == len(key) == 16
    decrypted = AES.new(key, AES.MODE_CBC, iv).decrypt(encrypted)
    if sha1(decrypted).hexdigest() == hash_hex:
        print("SUCCESS!", useragent)
        break
    # Remove padding, if valid
    padlen = decrypted[-1]
    if padlen <= 16 and all(d == padlen for d in decrypted[-padlen:]):
        decrypted = decrypted[:-padlen]
    # Re-test the hash
    if sha1(decrypted).hexdigest() == hash_hex:
        print("SUCCESS!", useragent)
        break
    # Test if the key is correct
    if decrypted[-22:-20] == b'PK':
        print("Potential ZIP, key may be good...", key, useragent)
```

Avec ce script, il s'avère qu'aucun agent utilisateur vérifiant les conditions établies ne fonctionne. Par contre, en enlevant la contrainte du système d'exploitation, on obtient quasiment une archive zip (sans les 16 premiers octets) pour la clé " `X 10.6; rv:35.0`", correspondant à Firefox 35.0 sur MacOS X 10.6. Alors il devient très simple de deviner le reste du contenu de l'agent utilisateur, "`Macintosh; Intel Mac OS X 10.6; rv:35.0`", et donc le vecteur d'initialisation, `Macintosh; Intel`.

Pour obtenir le fichier zip déchiffré, il est possible de modifier le script Python pour enregistrer le contenu déchiffré, mais il est également possible d'utiliser le code obfusqué original (copié précédemment dans `stage4.js`) en modifiant l'agent utilisateur avant que le code soit exécuté. Voici une page HTML qui effectue ceci.

```html
<html><body>
<script>
window.navigator.__defineGetter__('userAgent', function() {
    return "(Macintosh; Intel Rainbow Unicorn X 10.6; rv:35.0)";
});
</script>
<script src="stage4.js"></script>
</body></html>
```

Cette page affiche un lien "download stage5" aussi bien avec Firefox que Chromium, sur Linux, et cliquer sur ce lien permet d'enregistrer un fichier nommé `stage5.zip`.

5 Transputers

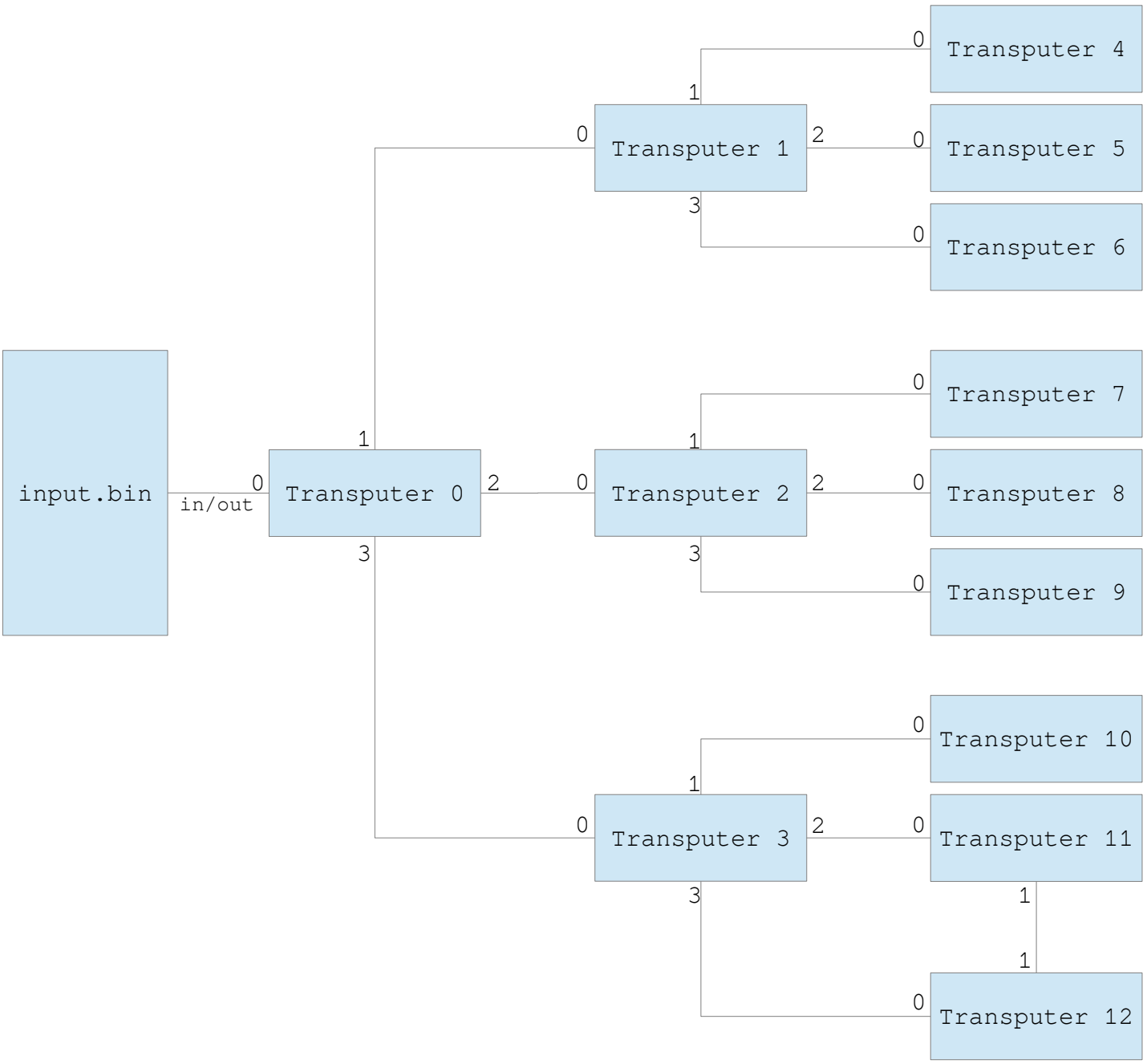
5.1 Découverte des transputers

L'archive zip `stage5.zip` contient deux fichiers, nommés `input.bin` et `schematic.pdf`. Le premier fichier est un fichier binaire d'un format inconnu avec quelques chaînes de caractères intéressantes (repérées en utilisant la commande `strings`) :

- les chaînes de caractères "Boot ok", "Code Ok" et "Decrypt", qui se succèdent de entre les positions 221 et 244 (en hexadécimal 0xdd à 0xf4)
- quatre caractères "KEY:" à la position 2437 (soit 0x985), et
- "congratulations.tar.bz2" à la position 2454 (soit 0x996).

Le second fichier est un document PDF d'une seul page dont le contenu est inclus dans la page suivante. Ce document possède trois parties :

- Un graphe de 14 rectangles bleus nommés `input.bin` et de `Transputer 0` à `Transputer 12`. Des nombres sont placés au début et à la fin de chaque ligne joignant les rectangles, sauf l'extrémité de la ligne connectant `input.bin`, qui est annotée `in/out`.
- Des sommes de contrôle SHA-256 pour deux fichiers, `encrypted` et `decrypted`, qui ne correspondent à aucun fichier extrait jusqu'à maintenant dans le challenge.
- Une section nommée "Test vector" qui contient du code Python 2 (reconnaissable à la structure `"`*`nombre hexadécimal`*`".decode("hex")`) qui exécute une fonction `decrypt` appelée avec une clé et un bloc de données indiqués et compare le résultat obtenu avec la chaîne de caractères "I love ST20 architecture".



SHA256:

a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81 - encrypted
9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321 - decrypted

Test vector:

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

Avant de chercher à savoir ce qu'est un transputer ou l'architecture ST20, il m'a semblé étrange que le document PDF mentionne un fichier `encrypted` sans que celui-ci soit présent dans l'archive `stage5.zip`. En même temps, le fichier `input.bin` contient un nom de fichier, `congratulations.tar.bz2`, suivi par des octets qui semblent être inconnus mais qui pourraient correspondre au contenu chiffré de ce fichier. Cette réflexion m'a amené à exécuter la commande suivante :

```
% dd if=input.bin bs=1 skip=2477 of=encrypted
250606+0 records in
250606+0 records out
250606 bytes (251 kB) copied, 0.886965 s, 283 kB/s

% sha256sum encrypted
a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81  encrypted
```

Ceci signifie que les 250606 octets à la position 2477 du fichier (juste après `congratulations.tar.bz2`) ont une somme de contrôle SHA-256 qui correspond exactement à celle indiquée pour le fichier `encrypted` dans le document PDF. La probabilité d'une collision de somme de contrôle SHA-256 étant suffisamment faible pour être considérée nulle dans le cadre de ce challenge, il est possible d'affirmer que le fichier `encrypted` extrait de `input.bin` par la commande `dd` est précisément le fichier mentionné dans `schematic.pdf`.

Après avoir établi ceci, il reste à comprendre ce qu'est un transputer et ce à quoi correspond l'architecture ST20, à décoder la partie de `input.bin` qui précède le fichier chiffré, à trouver quel algorithme de déchiffrement il faut utiliser pour déchiffrer `encrypted`, et enfin trouver quelle clé utiliser (si elle n'est pas dans `input.bin`).

L'article Wikipedia consacré aux transputers [23] permet de comprendre qu'il s'agit d'une architecture développée dans les années 1980 et que le ST20 est un microcontrôleur implémentant cette architecture. Les transputers utilisent un jeu d'instruction de type RISC (Reduced Instruction Set Computing). Peu d'outils actuels permettent de décoder ou d'exécuter un programme compilé pour l'architecture ST20, et encore moins permettent de comprendre le format utilisé pour `input.bin`. Voici un rapide tour d'horizon qui est loin d'être exhaustif :

- IDA supporte cette architecture [24] mais ne reconnaît pas le format de `input.bin`, et la version gratuite n'intègre pas de décompilateur, qui permettrait de comprendre plus rapidement les instructions.
- Quelques projets de décompilateur ST20 ne fournissent le code source, et le peu dont le code est gratuitement accessible sont peu lisibles (par exemple https://github.com/Simo2553/AndroidPortings/tree/master/MonteDroid/Decompiler/Linux/frontend/machine/st20).
- En ce qui concerne les émulateurs, il existe un projet "ST20 emulator" sur Sourgeforge [25] dont le code est lisible rapidement, et permet au moins de comprendre l'organisation générale de l'architecture ST20.
- J'ai également trouvé un autre émulateur de transputer au code plutôt très lisible sur http://spirit.lboro.ac.uk/emulator.html.
- Ce site dispose d'un lien vers un projet plus récent dont l'auteur semble avoir utilisé une technique innovante pour publier son code source, en utilisant des fichiers PDF soit-disant sécurisés (https://sites.google.com/site/transputeremulator/Home/source-code).

En utilisant toutes les informations recueillies par diverses recherches sur internet ainsi qu'en lisant la documentation abondante disponibles sur http://www.transputer.net/ (en particulier la liste des instructions [26] et une annexe qui définit du vocabulaire utile [27]), j'ai pu décoder le début du fichier `input.bin`.

5.2 Code d'amorçage du transputer 0

La lecture de la page Wikipedia anglaise sur les transputers permet de comprendre comment ceux-ci démarrent. En effet, la section "Booting" [28] indique qu'un transputer peut démarrer en téléchargeant un code d'amorçage sur le réseau. Les données transmises pour cet amorçage consistent simplement en la taille du code, sur un octet, suivi du code proprement dit. Comme sur le graphe des transputers de `schematic.pdf` le fichier `input.bin` est relié à la connexion 0 du transputer 0, il est raisonnable de penser que ce fichier commence par la séquence d'amorçage du premier transputer. Le premier octet du fichier, f8 en hexadécimal, correspond donc a la taille du programme envoyé, soit 248 octets.

Voici les 249 premiers octets de `input.bin`.

```
% head -c 249 input.bin | xxd
00000000: f864 b440 d140 d324 f224 2050 23fc 64b4  .d.@.@.$.$ P#.d.
00000010: 2c49 21fb 24f2 48fb 2419 24f2 544c f724  ,I!.$.H.$.$.TL.$
00000020: 7921 a52c 4d21 fb24 f254 2479 f72c 4321  y!.,M!.$.T$y.,C!
00000030: fb24 7a24 79fb 6100 2419 24f2 514c fb24  .$z$y.a.$.$.QL.$
00000040: 1924 f252 4cfb 2419 24f2 534c fb29 4421  .$.RL.$.$.SL.)D!
```

23. https://en.wikipedia.org/wiki/Transputer
24. https://www.hex-rays.com/products/ida/support/idadoc/618.shtml
25. http://sourceforge.net/projects/st20emu/
26. http://www.transputer.net/iset/pdf/tis-sum.pdf
27. http://www.transputer.net/iset/appendix/appendix.html
28. http://en.wikipedia.org/wiki/Transputer#Booting

```
00000050: fb24 f248 fb12 24f2 5444 f715 24f2 544c  .$.H..$.TD..$.TL
00000060: f728 4821 fb24 f248 fb13 24f2 5441 f719  .(H!.$.H..$.TA..
00000070: 24f2 5413 f1f7 40d4 1124 f254 41f7 1524  $.T...@..$.TA..$
00000080: f251 4cfb 1524 f252 4cfb 1524 f253 4cfb  .QL..$.RL..$.SL.
00000090: 1081 24f2 5541 f710 8224 f256 41f7 1083  ..$.UA...$.VA...
000000a0: 24f2 5741 f710 81f1 1082 f123 f310 83f1  $.WA.......#....
000000b0: 23f3 1081 23fb 11f1 7415 f2f1 742c f123  #...#...t...t,.#
000000c0: f310 23fb 1081 f174 15f2 23fb 7481 25fa  ..#....t..#.t.%.
000000d0: d4cc a380 40d4 1024 f241 fb66 0b42 6f6f  ....@..$.A.f.Boo
000000e0: 7420 6f6b 0043 6f64 6520 4f6b 0044 6563  t ok.Code Ok.Dec
000000f0: 7279 7074 0024 bc22 f0                   rypt.$.".
```

Les données récupérées par le transputer 0 pour démarrer contiennent beaucoup de dollars et de lettres majuscules au début, et sont composées à la fin des trois chaînes de caractères qui ont été précédemment repérées dans `input.bin`.

Il faut donc maintenant décoder le programme qui est exécuté. Pour cela, il est utile de donner quelques précisions sur l'architecture ST20 utilisée ici.

- La mémoire utilisée est adressée sur 32 bits, et certaines instructions utilisent le concept de "mot" ("word" en anglais), qui est un entier de 32 bits.
- Chaque instruction est codée sur 8 bits.
- Le système dispose de 6 registres de 32 bits, nommés `AReg`, `BReg`, `CReg`, `OReg`, `IPtr` et `WPtr`.
- Les trois premiers registres sont utilisés sous forme de pile. Les instructions qui chargent une valeur le font dans le registre `AReg` en déplaçant les valeurs de `AReg` et `BReg` respectivement dans `BReg` et `CReg`.
- Le registre `OReg` permet de construire les opérandes des instructions à l'aide des instructions `PFIX` et `NFIX`.
- Le registre `IPtr` est le pointeur d'instruction courante.
- Le registre `WPtr` est le "Workspace Pointer" (pointeur d'espace de travail). Il est souvent utilisé pour accéder à la mémoire du transputer.

Une instruction est un octet qui peut être lu sous la forme de deux chiffres hexadécimaux. Le chiffre des unités est ajouté à `OReg` avant l'exécution de l'instruction et le second chiffre est utilisé pour déterminer l'instruction. Le jeu d'instruction de base contient donc 16 instructions, présentées dans le tableau ci-dessous avec chaque pseudo-code correspondant. Dans le pseudo-code, `Push` signifie "charger en haut de la pile" et `Pop` "décaler les valeurs de la pile vers le haut".

Mnémonique	**Opcode**	**Signification**	**Pseudo-code**
`J`	`00`	Jump	`IPtr ← IPtr + OReg`
`LDLP`	`00`	Load Local Pointer	`Push WPtr + 4 × OReg`
`PFIX`	`20`	Prefix	`OReg ← OReg << 4`
`LDNL`	`30`	Load Non-Local	`AReg ← AReg[Oreg]`
`LDC`	`40`	Load Constant	`Push Oreg`
`LDNLP`	`50`	Load Non-Local Pointer	`AReg ← AReg + 4 × Oreg`
`NFIX`	`60`	Negative Prefix	`OReg ← (~OReg) << 4`
`LDL`	`70`	Load Local	`Push WPtr[OReg]`
`ADC`	`80`	Add Constant	`AReg ← AReg + Oreg`
`CALL`	`90`	Subroutine Call	
`CJ`	`a0`	Conditional Jump	`Si AReg = 0, IPtr ← IPtr + OReg, sinon Pop`
`AJW`	`b0`	Adjust Workspace	`WPtr ← WPtr + 4 × Oreg`
`EQC`	`c0`	Equals Constant	`AReg ← (1 si AReg = Optr sinon 0)`
`STL`	`d0`	Store Local	`WPtr[Optr] ← AReg, Pop`
`STNL`	`e0`	Store Non-Local	`AReg[Optr] ← BReg, Pop, Pop`
`OPR`	`f0`	Operate	(dépend de la valeur de `OReg`)

Après chaque instruction sauf `PFIX` et `NFIX` le registre `OReg` est réinitialisé à zéro. Par ailleurs, pour les sauts, le registre `IPtr` est incrémenté avant l'exécution de l'instruction donc contient en pratique l'adresse de l'instruction suivante. Les accès mémoires notés `WPtr[Optr]` et `AReg[Optr]` correspondent à des mots de 32 bits localisés respectivement aux adresses `WPtr` $+ 4 \times$ `OReg` et `AReg` $+ 4 \times$ `Oreg`.

L'instruction `CALL` sauvegarde l'état de la pile ainsi que `IPtr` dans `WPtr` avant de modifier `IPtr` pour matérialiser le saut, et l'instruction `OPR` permet d'exécuter une instruction étendue dont l'identifiant est dans `OReg`.

Le code exécuté par le transputer 0 commence par les octets `64 b4 40 d1` en hexadécimal, ce qui correspond donc aux instructions suivantes :

Code hexadécimal	**Instruction**	**Pseudo-code**
`64`	`NFIX 4`	`OReg ← (~4) << 4 = 0xfffffff0`
`b4`	`AJW 4`	`WPtr ← WPtr + 4 × (0xfffffff0 + 4)`
`40`	`LDC 0`	`Push 0`
`d1`	`STL 1`	`WPtr[1] ← AReg, Pop`

Les deux premières instructions peuvent être combinées en une unique instruction `AJW 0xfffffff4` pour faciliter la lecture. De plus, le nombre hexadécimal de 32 bits fffffff4 correspond au nombre -4c en utilisant la

représentation du complément à 2, qui est pertinente ici car ce nombre est utilisé dans une addition, donc il est également possible d'utiliser la notation `AJW -0x4c` pour représenter l'instruction.

Les deux instructions suivantes on pour effet de charger la valeur 0 dans `WPtr[1]` (à l'adresse `WPtr + 4 × 1`, soit `WPtr + 4`). Il est donc possible de les combiner dans le pseudo-code généré afin de faciliter la compréhension du code.

Après cette analyse, il est possible de décoder les 23 premiers octets du code du transputer 0.

Code	**Instruction**	**Pseudo-code**
`64 b4`	`AJW -0x4c`	`WPtr ← WPtr - 4 × 0x4c`
`40 d1`	`LDC 0; STL 1`	`WPtr[1] ← 0`
`40 d3`	`LDC 0; STL 3`	`WPtr[3] ← 0`
`24 f2`	`OPR 0x42 (MINT)`	`Push 0x80000000`
`24 20 50`	`LDNLP 0x400`	`AReg ← AReg + 4 × 0x400`
`23 fc`	`OPR 0x3c (GAJW)`	`WPtr ↔ AReg`
`64 b4`	`AJW -0x4c`	`WPtr ← WPtr - 4 × 0x4c`
`2c 49`	`LDC 0xc9`	`Push 0xc9`
`21 fb`	`OPR 0x1b (LDPI)`	`AReg ← AReg + IPtr`
`24 f2`	`OPR 0x42 (MINT)`	`Push 0x80000000`
`48`	`LDC 8`	`Push 8`
`fb`	`OPR 0x0b (OUT)`	Envoie `AReg` octets depuis l'adresse `CReg` sur le canal `BReg`

Cette séquence d'instruction utilise quatre instructions étendues :

— `MINT` (Minimum Integer) qui a pour effet de charger la valeur 0x80000000 dans la pile,
— `GAJW` (General Adjust Workspace), qui échange la valeur de `WPtr` avec le contenu du haut de la pile, `AReg`,
— `LDPI` (Load Pointer to Insctruction), qui ajoute la valeur de `IPtr` à `AReg`, et
— `OUT` (Output), qui permet d'envoyer des données sur un lien.

Juste avant d'exécuter l'instruction `OUT`, trois valeurs sont poussées sur la pile :

— La première valeur est la somme de 0xc9 et du pointeur d'instruction après l'instruction `LDPI`. Dans `input.bin`, ceci correspond donc à la position 0xc9 + 0x14 = 0xdd, qui correspond au début de la chaîne de caractères "Boot ok".
— La deuxième valeur est 0x80000000 et est interprétée comme un identifiant de canal. La lecture de documentation au sujet des transputers permet de comprendre que 0x80000000 correspond à l'écriture sur le lien 0, 0x80000004 à celle sur le lien 1, 0x80000008 sur le lien 2, etc.
— La troisième valeur est 8, qui est la longueur de "Boot ok" en prenant en compte le caractère terminateur (zéro ASCII).

Ainsi la première instruction `OUT` a pour effet d'envoyer la chaîne de caractères "Boot ok" sur le lien 0 du transputer 0. Sur le schéma de `schematic.pdf` ce lien est annoté `in/out`. On peut donc imaginer qu'il s'agit de la manière qu'ont les transputers pour transmettre des données "ailleurs" (par exemple dans un fichier, sur un moniteur, etc.), ce qui est communément appelé la *sortie standard* par exemple dans les programmes C.

La suite de l'analyse du code du transputer 0 peut être réalisée à la main, mais j'ai préféré automatiser cette analyse en écrivant un décodeur qui affiche du pseudo-code au lieu des instructions réelles. Cette approche a l'avantage de permettre ensuite une analyse très rapide des codes exécutés par les autres transputers.

Ainsi, j'ai écrit un script Python qui automatise cette analyse. Le code source du décodeur ST20 est disponible à l'annexe A.3.1, celui du script qui analyse le code du transputer 0 à l'annexe A.3.2, et le pseudo-code généré à l'annexe B.2.1.

Voici quelques indications permettant de comprendre le pseudo-code généré.

— Tous les codes ont été décodés à partir de l'adresse 0. Ceci ne correspond pas à l'adresse réellement utilisée en mémoire (qui est nécessairement entre 0x8000000 et 0xffffffff selon les documents que j'ai trouvés), mais une rapide recherche ne m'a pas permis de déterminer quelle adresse était utilisée lors du démarrage de systèmes utilisant l'architecture ST20. Comme le code ne dépend pas fortement de là où il est chargé (grâce à l'instruction `LDPI` qui permet d'accéder aux données en utilisant un décalage relatif au pointeur d'instruction), le plus simple est de considérer que le code d'amorçage est chargé à l'adresse 0, ce qui ne change pas fondamentalement l'analyse.
— Les registres sont nommés `A`, `B`, `C` et `W` au lieu de `AReg`, `AReg`, `BReg`, `CReg` et `WPtr`.
— Au lieu d'utiliser la notation $A \leftarrow B$ pour les affectations j'ai utilisé $A = B$, mais le symbole égal dans "`If A=0`" correspond bien à un test d'égalité.
— L'instruction `IN`, qui permet de recevoir des données d'un lien, utilise la même convention d'appel que l'instruction `OUT`, avec la subtilité que les identifiants des canaux sont différents : 0x80000010 correspond au lien 0, 0x80000014 au lien 1, 0x80000018 au lien 2, etc.
— Les instructions `LB` (Load Byte) et `SB` (Store Byte), qui permettent respectivement de charger et d'enregistrer un octet dans une adresse donnée par `AReg` sont représentées en pseudo code par `LB(`adresse`)` et `SB:`adresse `=` valeur. Ceci est lié au fait que la notation adresse`[`indice`]` a pour signification "le nombre de 4 octets à l'adresse adresse + 4 × indice".

5.3 Décodage du code des transputers 0, 1, 2 et 3

L'analyse du code du transputer 0 (en B.2.1) montre que celui-ci est divisé en plusieurs parties :

— La première partie, dont le code a été étudié manuellement à la section précédente, initialise quelques valeurs en mémoire puis envoie le message "Boot ok" sur le lien 0, qui correspond à la "sortie standard" du réseau de transputers.
— La deuxième, entre les adresse 17 et 35 en hexadécimal, consiste en une boucle dans laquelle le transputer reçoit 3 mots de 4 octets sur le lien 0 (connecté à `input.bin` dans `WPtr[0x49]`, `WPtr[0x4a]` et `WPtr[0x4b]` puis reçoit sur le lien 0 et retransmet `WPtr[0x49]` octets sur lien déterminé par l'identifiant dans `W[0x4a]`. Cette boucle est exécutée jusqu'à ce que `WPtr[0x49]` soit nul.
— Après la boucle le transputer envoie les 12 derniers octets reçus à chaque transputer qui lui est connecté (sur les liens 1, 2 et 3) et envoie "Code Ok" sur la sortie standard
— Il lit ensuite 4 octets dans `WPtr[2]` et 12 octets dans `WPtr[5]`, `WPtr[6]WPtr[7]` (ce qui sera noté `WPtr[5..7]` dans la suite) et envoie "Decrypt" sur la sortie standard.
— Il lit ensuite (dans le code à l'adresse 68) un octet dans `WPtr[3]` puis `WPtr[3]` octets dans `WPtr[9..]`.
— Après cela, le transputer entre dans une boucle infinie (entre les adresses 77 et db), qui sera décrite ensuite.

Cette analyse permet de comprendre l'organisation du fichier `input.bin`. Après le code d'amorçage du transputer 0, celui-ci contient une succession de blocs de données que le transputer 0 transmet aux autres transputers, ces données étant formatées par un entête de 12 octets qui contient la longueur de la charge utile, sur 4 octets, et l'identifiant du lien sur lequel envoyer ces données, sur 4 octets. Cette succession de blocs se termine par un bloc de taille nulle, qui se situe à la position 0x979 de `input.bin`. Juste après (à la position 0x985) se trouvent quatre octets qui forment la chaîne de caractères "KEY:", suivi de 12 octets qui valent 0xff. La mise en correspondance de ce contenu avec l'analyse du code du transputer 0 permet de comprendre que les 4 octets chargés dans `WPtr[2]` sont ceux de "KEY:" et les 12 chargés dans `WPtr[5..7]` sont ceux qui valent 0xff. Après ceci se trouve un octet indiquant une longueur, qui vaut 23 (en décimal) puis la chaîne de caractères "congratulations.tar.bz2", qui occupe 23 octets. Au moment d'arriver dans la boucle infinie, le prochain octet lu de `input.bin` par le transputer 0 correspond donc à l'octet à la position 0x9ad, qui est le premier octet de la partie correspond à ce qui a été précédemment identifié comme étant le fichier `encrypted`.

On peut ainsi en déduire que la boucle infinie du code du transputer 0 a pour but de déchiffrer le fichier `encrypted`, en utilisant les autres transputers pour exécuter l'algorithme de déchiffrement.

L'étape suivante dans l'analyse du réseau des transputers consiste donc à extraire le code d'amorçage des transputers 1, 2 et 3 à partir de `input.bin` et de l'analyser. Comme les transputers 1, 2 et 3 suivent une procédure de démarrage similaire à celle du transputer 0, leurs codes d'amorçage correspond aux premiers blocs de données retransmis par le transputer 0 sur chacun des liens. Il se trouve que les trois premiers blocs de données contiennent exactement le même contenu, mais adressé à chacun des liens. Ainsi, il n'y a en pratique qu'un seul code à analyser. Le script Python qui transforme le code extrait en pseudo-code est disponible à l'annexe A.3.3 et le pseudo-code résultant à l'annexe B.2.2.

Ce code est très court et peut être étudié de manière similaire que pour le code du transputer 0 :

— Les transputers 1, 2 et 3 commencent par initialiser `WPtr`.
— Ensuite ils effectuent la même boucle que le transputer 0 pour relayer des blocs de données reçus par leur liens 0 et retransmis sur leurs autres liens, ce qui a pour effet d'initialiser les transputers 4 à 12.
— Par contre, il est important de noter que lorsqu'ils reçoivent un bloc de données dont l'entête annonce une longueur nulle, ce bloc n'est pas retransmis à tous les liens, contrairement à ce que fait le transputer 0. Ainsi le fait que le transputer 0 transmettre un tel bloc a pour effet de faire sortir les transputers 1 à 3 de leurs boucles de transmission de données, mais les transputers 4 à 12 utilisent certainement un code différent.
— Une fois la phase de "relais de données" terminée, une série d'instructions est exécutée en boucle, dans laquelle chaque transputer :
  — reçoit 12 octets du lien 0 dans `WPtr[1..3]`,
  — retransmet ces 12 octets à chaque lien 1, 2 et 3,
  — reçoit de chaque lien 1, 2 et 3 un octet,
  — combine les 3 octets reçus par un ou exclusif bit-à-bit,
  — envoi le résultat sur le lien 0.

Les transputers 1, 2 et 3 agissent donc en temps qu'intermédiaires, tout d'abord pour initialiser le code des transputers 4 à 12, et ensuite pour combiner les opérations réalisées par ceux-ci. La compréhension du code permet également de comprendre le mode de fonctionnement des opérations réalisées : 12 octets sont envoyés par le transputer 0 aux transputers 4 à 12 par l'intermédiaire de ceux 1 à 3, et le "résultat" consiste en un octet renvoyé au transputer 0 par l'intermédiaire des transputers 1 à 3, qui combinent les résultats intermédiaires par un ou exclusif bit-à-bit.

La reprise de l'analyse du code du transputer 0 (B.2.1) permet de comprendre comment fonctionne la boucle qui effectue le déchiffrement (entre les adresses 77 et db) :

— Le transputer 0 commence une itération de la boucle en lisant dans `WPtr[1]` un octet provenant du lien 0 (dont de `input.bin`).
— Ensuite il envoie le contenu de `WPtr[5..7]` (qui fait 12 octets) sur les liens 1, 2 et 3, c'est-à-dire aux transputers 1, 2 et 3.

— Il reçoit un octet de chaque lien 1, 2 et 3 et les combine par ou exclusif binaire dans le second octet de `WPtr[0]` (à l'adresse `WPtr + 1`, ce qui est noté dans le pseudo-code par `SB:(ptr W[0])+1` et `LB((ptr W[0])+1)`).
— Puis il calcule dans le premier octet de `WPtr[0]` le résultat de `WPtr[1]^(WPtr[4] + 2×`x`)` où x est l'octet à la position `WPtr[4]` dans `WPtr[5..7]`.
— Après ce calcul il met à jour l'octet à la position `WPtr[4]` dans `WPtr[5..7]` avec la valeur du second octet de `WPtr[0]`.
— Chaque itération de la boucle se termine par l'incrémentation modulo 12 de `WPtr[4]`, qui a au passage été initialisé à 0 juste avant la boucle, ainsi que par l'envoi du premier octet de `Wptr[0]` sur la sortie standard.

En résumé, `WPtr[5..7]` est un tableau de 12 octets dont un octet est modifié à chaque itération de la boucle par un octet déterminé en combinant les octets envoyés par les transputers 4 à 12 suite à l'envoi à ces transputers du contenu de `WPtr[5..7]`. La valeur précédente de l'octet qui est ainsi mis à jour dans `WPtr[5..7]` est utilisé avec le compteur qui permet de repérer sa position pour déchiffrer à chaque itération un octet reçu sur le lien 0 et envoyer l'octet déchiffré de nouveau sur le lien 0.

L'algorithme de déchiffrement utilisé (qui par ailleurs a la propriété d'être identique à l'algorithme de chiffrement réciproque grâce au caractère involutif de la fonction utilisée pour calculer l'octet déchiffré) ne peut être étudié qu'après avoir extrait et décodé les codes utilisés par les transputers 4 à 12. Pour cela, j'ai écrit un script Python (annexe A.2) qui lit `input.bin` et crée les fichiers contenant le code de chaque transputer, en affichant les positions du premier et dernier octet correspondant. Le résultat affiché par ce script est disponible à l'annexe B.1.

5.4 Transputers 4 à 12

Chacun des transputers 4 à 12 commence par recevoir un programme d'amorçage qui est transmis par les transputers 1, 2 et 3. Tous ces programmes d'amorçage sont identiques et sont relativement courts. Une rapide analyse (effectuée par le script dont le code source est à l'annexe A.3.4 et le pseudo-code généré à l'annexe B.2.3) permet de comprendre que ce programme :

— reçoit sur le lien 0 trois entiers de 32-bits, enregistrés dans `WPtr[0]`, `WPtr[1]` et `WPtr[2]` ;
— reçoit `WPtr[0]` octets sur le lien 0, enregistrés à partir de la position 0x1f relative par rapport à l'adresse de la première instruction du programme d'amorçage ;
— exécute le code nouvellement téléchargé en plaçant la position 0x1f + `WPtr[2]` dans `IPtr`.

L'entête de trois entiers qui précède le code réellement exécuté contient donc la longueur de ce code ainsi que la position de la première instruction à exécutée, qui en pratique est toujours `0xc` (cf. B.1).

Il est donc maintenant possible d'extraire le code exécuté sur chacun des transputers 4 à 12 et de l'analyser pour reconstruire l'algorithme de déchiffrement qui a été implémenté. Pour cela, j'ai écrit un script Python dont le code est situé à l'annexe A.3.5 et le pseudo-code extrait à l'annexe B.2.4.

L'exécution de chaque code commence à l'adresse 0xc, après la définition de deux fonctions que j'ai nommées `recv(chan=B, mem=C, len=W[0])` et `send(chan=B, mem=C, len=W[0])` et qui concrètement utilisent les instructions `IN` et `OUT` pour recevoir ou envoyer des données.

La lecture et la compréhension des pseudo-codes permet alors d'écrire en C une fonction qui déchiffre `size` octets contenus à l'emplacement `src` dans un bloc mémoire alloué en `dst` en utilisant la clé `key` :

```c
#include <stdint.h>
#include <string.h>

void decrypt(const uint8_t argkey[12], const uint8_t *src, uint8_t *dst, uint32_t size)
{
    /* Local state of each transputer which uses one */
    uint8_t trans4w1 = 0, trans5w1 = 0;
    uint16_t trans6w1 = 0;
    uint8_t trans8w4 = 0, trans8w5sums[4] = {};
    uint8_t trans10w2 = 0, trans10w4buff0[4] = {}, trans10w4buffers[4 * 12] = {};
    uint8_t trans11next = 0;

    uint32_t byteidx;
    uint8_t key[12], i;

    /* Copy the key in a local buffer which is modified */
    memcpy(key, argkey, 12);

    /* Initialize transputer 6 state with the initial key buffer */
    for (i = 0; i < 12; i++)
        trans6w1 += key[i];
```

```
    for (byteidx = 0; byteidx < size; byteidx++) {
        const uint32_t keyidx = byteidx % 12;
        uint8_t transres, s0_5 = 0, s6_11 = 0, s0_11, s_t10 = 0;

        /* Decrypt one byte (code in transputer 0) */
        *(dst++) = *(src++) ^ (keyidx + 2 * key[keyidx]);

        /* Compute sums of key[0] to key[5], key[6] to key[11]... */
        for (i = 0; i < 6; i++)
            s0_5 += key[i];
        for (i = 6; i < 12; i++)
            s6_11 += key[i];
        /* ... and from key[0] to key[11] */
        s0_11 = s0_5 + s6_11;

        trans6w1 =
            ((trans6w1 & 0x8000) >> 15) ^
            ((trans6w1 & 0x4000) >> 14) ^
            ((trans6w1 << 1) & 0xffff);

        trans8w5sums[trans8w4] = s0_11;
        trans8w4 = (trans8w4 + 1) % 4;

        trans10w4buff0[trans10w2] = key[0];
        memcpy(trans10w4buffers + 12 * trans10w2, key, 12);
        trans10w2 = (trans10w2 + 1) % 4;

        transres = (trans6w1 & 0xff) ^ s0_5 ^ s6_11 ^ key[trans11next];
        transres ^= key[(key[0] ^ key[3] ^ key[7]) % 12];
        trans11next = (key[1] ^ key[5] ^ key[9]) % 12;

        for (i = 0; i < 12; i++) {
            uint8_t k = key[i];
            trans4w1 += k;
            trans5w1 ^= k;
            transres ^= k << (i & 7);
        }
        transres ^= trans4w1 ^ trans5w1;

        for (i = 0; i < 4; i++)
            transres ^= trans8w5sums[i];

        /* Transputer 10 sum */
        for (i = 0; i < 4; i++)
            s_t10 += trans10w4buff0[i];
        transres ^= trans10w4buffers[12 * (s_t10 & 3) + ((s_t10 >> 4) % 12)];

        /* Transputer 0 combines the results for all transputers into one byte of the key */
        key[keyidx] = transres;
    }
}
```

Comme il est très facile de faire des erreurs lors de l'écriture de ce code C, il est utile d'avoir un moyen réaliste de tester l'implémentation. Heureusement, le fichier `schematic.pdf` contient un *Test vector* permettant de réaliser un tel test : le déchiffrement des données indiquées avec la clé "*SSTIC-2015*" devrait produire le texte "I love ST20 architecture", ce qui est le cas.

L'étude des transputers a ainsi permis de déterminer l'algorithme de déchiffrement utilisé et de le réimplémenter en C. L'analyse des transputers est donc maintenant terminer et l'étape suivante consiste maintenant à trouver la clé permettant de déchiffrer le fichier chiffré.

5.5 Attaque par force brute de la clé

Dans le code du transputer 0, la clé de 12 octets utilisée pour le déchiffrement des données est initialisée en lisant 12 octets de valeur 0xff dans `input.bin`. Néanmoins cette clé ne permet pas de déchiffrer correctement le fichier chiffré extrait à la section 5.1 (la somme de contrôle SHA-256 est donnée dans `schematic.pdf`). Il faut donc trouver la clé à utiliser.

Pour cela, voici les informations qui ont été recueillies précédemment.

— Le fichier déchiffré s'appelle probablement `congratulations.tar.bz2` et est donc une archive compressée avec le format BZip2.
— Le fichier déchiffré fait la même taille que le fichier chiffré, 250606 octets.
— Dans l'algorithme de déchiffrement, la clé de déchiffrement est utilisée quasiment directement pour déchiffrer les 12 premiers octets, son contenu n'ayant pas été modifié par les transputers.

L'article Wikipedia consacré à BZip2 permet d'apprendre que l'entête d'un fichier dans ce format contient presque toujours les même 10 premiers octets, qui sont en hexadécimal `42 5a 68 39 31 41 59 26 53 59`. De plus, comme le fichier est relativement petit, il est possible de supposer que le champ `origPtr` de l'entête est inférieur à 2^{17}, ce qui signifie, en supposant également que le champ `randomised` est nul, que l'octet à la position 14 (en décimal) est probablement nul. Cette hypothèse est discutable mais permet toutefois de considérablement accélérer le bruteforce de la clé de déchiffrement et donc de se rendre compte relativement rapidement si elle est juste ou fausse.

Pour les 12 premiers octets, le déchiffrement des données consiste à effectuer cette opération pour tout i entre 0 et 11 (l'opérateur ˆ correspondant au ou exclusif bit-à-bit sur 8 bits et *mod* à l'opération modulo) :

$$\text{déchiffré}[i] \leftarrow \text{chiffré}[i] \text{ ˆ } ((i + 2 \times \text{clé}[i]) \mod 256)$$

Comme déchiffré[i] et chiffré[i] sont connus pour i entre 0 et 9 (grâce à l'entête BZip2), il est possible de vérifier que déchiffré[i] ˆ chiffré[i] $- i$ est pair et ensuite d'en déduire que pour tout i entre 0 et 9,

$$\text{clé}[i] \mod 128 = \frac{(\text{déchiffré}[i] \text{ ˆ } \text{chiffré}[i] - i) \mod 256}{2}$$

Cette équation donne donc directement les 7 bits de poids faible des 10 premiers octets de la clé. Il reste à trouver le bit fort de chacun de ces 10 octets ainsi que les deux derniers octets de la clé, c'est-à-dire au total $10+2\times8 = 26$ bits. Pour tester de manière automatique si chaque possibilité est valide, il est possible de comparer la somme de contrôle SHA-256 des données déchiffrées à la somme de contrôle indiquée dans `schematic.pdf`. Au moment du challenge je n'ai pas utilisé cette méthode car dans les étapes précédentes, la présence d'un rembourrage à la fin des données déchiffrées a fait qu'il est arrivé que la somme de contrôle fournie ne corresponde pas directement au résultat déchiffré (il fallait enlever le rembourrage). À la place, j'ai utilisé la bibliothèque bzlib pour tenter de décompresser les données déchiffrées, en sachant que le format BZip2 contient une somme de contrôle CRC32 des données décompressées, qui assure l'intégrité du fichier. Ceci est certes légèrement plus complexe mais en pratique a mené au bon résultat.

Par ailleurs, l'hypothèse qui a été faite que l'octet à la position 14 soit nul permet d'interrompre rapidement le déchiffrement des données après 3 itérations de la boucle de déchiffrement. En effet, après 3 itérations, la clé contient à la position 2 la valeur qui sera utilisée ensuite pour déchiffrer l'octet à la position 14 et il est donc possible de déchiffrer spécifiquement cet octet sans attendre 12 autres itérations.

Cette astuce permet de trouver en 12 minutes 24 secondes la clé de déchiffrement en utilisant un seul thread sur un processeur Intel Core i7 cadencé à 2,2 GHz. Le code utilisé pour réaliser le bruteforce est présenté à l'annexe A.3.6.

La clé de déchiffrement trouvée est alors, en hexadécimal :

```
5e d4 9b 71 56 fc e4 7d e9 76 da c5
```

Ceci permet de finalement déchiffrer le fichier `encrypted` puis de vérifier que la somme de contrôle SHA-256 du fichier obtenu correspond bien à celle attendue dans `schematic.pdf` :

```
% sha256sum congratulations.tar.bz2
9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321  congratulations.tar.bz2
```

6 Quelques derniers petits efforts en image

6.1 Une première image

L'archive `congratulations.tar.bz2` déchiffrée par le réseau de transputers contient un unique fichier nommé `congratulations.jpg`, qui est une image JPEG contenant le message "Félicitations... un dernier petit effort ?", avec le logo du SSTIC et des diablotins qui ont déjà été rencontrés précédemment dans la carte OpenArena.

FIGURE 9 – `congratulations.jpg`

L'utilisation de `binwalk` permet de trouver rapidement la suite du challenge, qui a été simplement concaténée à l'image. Voici les commandes que j'ai effectuées avec ce qu'elles affichent dans le terminal.

```
% binwalk congratulations.jpg

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

% dd if=congratulations.jpg of=hidden1.tar.bz2 bs=1 skip=55248
197321+0 records in
197321+0 records out
197321 bytes (197 kB) copied, 0.728172 s, 271 kB/s

% tar tjf hidden1.tar.bz2
congratulations.png
```

6.2 Une deuxième image

L'archive au format tar.bz2 cachée dans l'image `congratulations.jpg` contient un unique fichier, nommé `congratulations.png`, qui contient le message "Félicitations... deux derniers petit efforts ?" (figure 10).

Une analyse avec Wireshark de l'image révèle la présence de blocs du type non-standard `sTic` au sein de l'image PNG (capture d'écran en figure 11). Il semble donc pertinent d'extraire ces blocs afin d'en étudier le contenu. Il apparaît alors que le contenu extrait a été compressé avec la bibliothèque zlib.

FIGURE 10 – `congratulations.png`

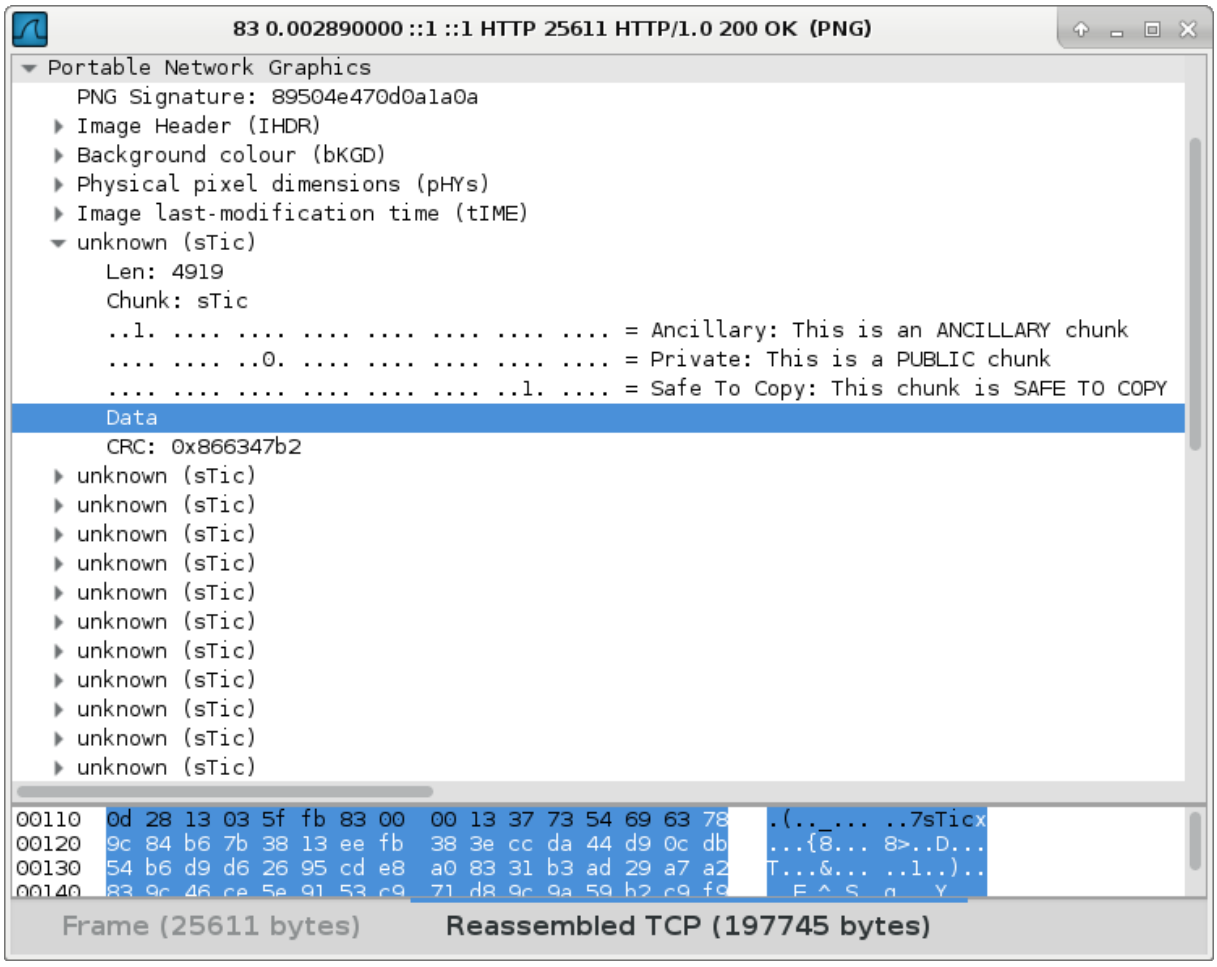


FIGURE 11 – Analyse dans Wireshark de `congratulations.png`

Voici le script Python que j'ai utilisé pour extraire et décompresser les blocs de données de l'image.

```
#!/usr/bin/env python3
"""Extract the data from the sTic sections of congratulations.png"""
import struct
import zlib

with open('congratulations.png', 'rb') as f:
    pngdata = f.read()

offset = 0x5b
hiddendata = b''
while True:
    length, chunktype = struct.unpack('>II', pngdata[offset:offset + 8])

    # Stop when the chunk type is no longer "sTic"
    if chunktype != 0x73546963:
        break
    hiddendata += pngdata[offset + 8:offset + 8 + length]
    offset += 12 + length

with open('hidden2.tar.bz2', 'wb') as f:
    f.write(zlib.decompress(hiddendata))
```

Ceci permet d'obtenir une nouvelle archive, au format tar.bz2.

6.3 Jamais deux sans trois

La troisième archive contient une image nommée `congratulations.tiff` similaire aux précédentes, et cette fois au format TIFF. En la comparant pixel à pixel avec les images précédentes, il apparait que l'image est quasiment identique à la première (`congratulations.jpg`, au texte près). Plus précisément, dans la moitié supérieure de chaque image, une grande partie des pixels ont le bit de poids faible des composantes rouge et verte qui diffèrent entre les deux images, le reste des bits étant toujours égaux. Ceci indique probablement qu'un algorithme de stéganographie a été utilisé pour cacher des données dans les bits de poids faible des composantes rouge et verte de certains pixels de l'image TIFF.

En combinant ces bits, il est possible d'arriver à une archive tar.bz2 valide. Voici un script Python qui extrait cette archive de l'image :

```
#!/usr/bin/env python3
"""Extract the hidden data from congratulations.tiff pixels"""
from PIL import Image

imtiff = Image.open('congratulations.tiff')
(width, height) = imtiff.size
imdata = imtiff.getdata()

# A byte is represented by 4 pixels, and there is no data past half the height of the image
databytes = bytearray(width * height // 8)

for pxlpos, pxlvalues in enumerate(imdata):
    if pxlpos >= width * height // 2:
        break
    bytepos = pxlpos // 4
    bitpos_in_byte = 6 - 2 * (pxlpos - 4 * bytepos)
    databytes[bytepos] |= (pxlvalues[0] & 1) << (bitpos_in_byte + 1)
    databytes[bytepos] |= (pxlvalues[1] & 1) << bitpos_in_byte

with open('hidden3.tar.bz2', 'wb') as f:
    f.write(databytes)
```

6.4 La dernière image

Une quatrième fois, l'archive obtenue contient une image de félicitations, `congratulations.gif`. Cette image GIF utilise une palette de 256 couleurs qui contient beaucoup de cases de couleur noire sans raison logique apparente.

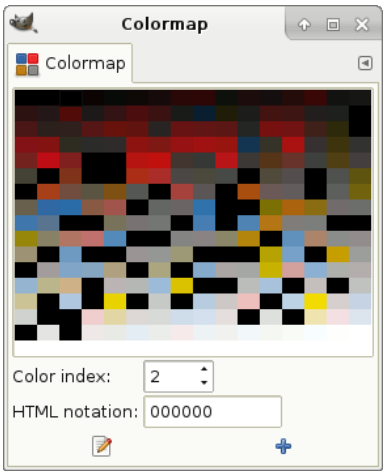


FIGURE 12 – Palette des couleurs de `congratulations.gif` dans GIMP

En modifiant la couleur d'indice 2 de la palette (initialement noir), on se rend compte que cet indice est utilisé par les pixels du cadre noir de l'image, sauf certains qui forment une adresse e-mail. Il s'agit de l'adresse e-mail qu'il faut trouver pour résoudre le challenge, et l'image GIF est donc l'image qui marque la fin du challenge.

FIGURE 13 – Remplacement de la couleur 2 par une couleur de notation HTML b5e6ff dans `congratulations.gif`

7 Conclusion

Le challenge 2015 du SSTIC est constitué de six étapes très différentes les unes des autres. Les résoudre toutes nécessite de mettre en œuvre des compétences variées : décodage de format peu répandu (pour le script Rubber Ducky, la capture des mouvements de souris...), compréhension d'un jeu d'instruction ancien mais documenté (pour les transputers), résolution d'une énigme dans une carte OpenArena, recherche d'astuces pour accélérer une attaque par force brute, etc. Ce document a présenté comment j'ai résolu ce challenge, en détaillant les divers outils que j'ai utilisés et les déductions logiques que j'ai réalisées.

Tous les programmes utilisés pour résoudre le challenge sont des logiciels libres. L'ensemble du code que j'ai écrit pour le challenge (scripts Python et programmes C/C++) ainsi que les fichiers ayant servis à générer ce document LaTeX seront disponibles à l'adresse https://github.com/fishilico/sstic-2015 une fois le challenge terminé. Tout ceci est publié sous la licence Beerware [29] [30].

8 Remerciements

Je souhaite tout d'abord remercier les concepteurs du challenge, qui ont conçu un challenge intéressant même si le scénario est très linéaire. J'ai apprécié découvrir OpenArena et le monde des transputers, ainsi que l'API Javascript permettant d'utiliser des routines de chiffrement dans un navigateur.

Je souhaite ensuite remercier ceux qui ont organisé le challenge du premier avril, qui a l'avantage d'être une entrée en matière simple pour des personnes qui s'intéressent à l'informatique au sens large et à qui j'ai pu donner un rapide aperçu des concepts fondamentaux de la résolution des challenges SSTIC, à savoir l'utilisation d'outils modernes pour résoudre des énigmes difficiles.

Je remercie ensuite tout ceux qui m'ont permis de participer au challenge en me permettant d'avoir un cadre de vie adapté à Singapour, où je vis depuis quelques mois, ainsi que tous ceux qui m'ont motivé malgré le fait que je participais "pour le plaisir, uniquement sur mon temps libre et quand je n'avais rien de plus important à faire, comme aller me balader en Malaisie (ce que j'ai fait le jour de la sortie du challenge) ou participer aux célébrations de Pâques (ce que j'ai fait le lendemain)".

29. https://en.wikipedia.org/wiki/Beerware
30. https://tldrlegal.com/license/beerware-license

Annexes

A Scripts utilisés

Cette section regroupe les divers scripts que j'ai écrits pour la résolution du challenge et qui n'ont pas été mis dans les parties précédentes.

A.1 Partie 1, décodeur des invocations PowerShell

```
#!/usr/bin/env python3
"""Decode the Powershell command lines which were encoded in inject.bin"""
import base64
import codecs

# Common prefix
PREFIX = (
    '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(\'')
SUFFIX = (
    '\'));" +
    "}else{" +
    "write_file_bytes([Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));" +
    "}")

with open('decoded.out.txt', 'r') as fin:
    with open('stage2.zip', 'wb') as fout:
        for line in fin:
            if line.startswith('powershell -enc '):
                cmd = base64.b64decode(line[len('powershell -enc '):])
                cmd = codecs.decode(cmd, 'utf16')
                if cmd.startswith(PREFIX) and cmd.endswith(SUFFIX):
                    fout.write(base64.b64decode(cmd[len(PREFIX):-len(SUFFIX)]))
                else:
                    print(cmd)
```

A.2 Partie 5, extraction des parties du fichier `input.bin`

```
#!/usr/bin/env python3
"""Split input.bin into the several payloads which are sent to transputers"""
import struct

with open('input.bin', 'rb') as f:
    inputdata = f.read()

# Input data starts with the booted firmware
bootlen = inputdata[0]
print("[{:04x}-{:04x}] Transputer 0 code ({} bytes)".format(
```

```
    1, bootlen, bootlen))
with open('transputer0.bin', 'wb') as f:
    f.write(inputdata[1:1 + bootlen])

# Then there is a zero-terminated list of packets forwarded to sub-links
offset = 1 + bootlen
level1pkts = {1: [], 2: [], 3: []}
while True:
    # Decode the 12-byte header
    length, destination, _ = struct.unpack('<III', inputdata[offset:offset + 12])
    offset += 12
    if length == 0:
        break
    # destination 0x80000004 is link 1, 0x80000008 is link 2, 0x8000000c link 3
    lnkdst = (destination - 0x80000000) // 4
    assert destination == 0x80000000 + lnkdst * 4
    level1pkts[lnkdst].append((offset, length))
    offset += length

final_offset = offset
print("[{:04x}-{:04x}] Transputer 0 forwarded packets:".format(
    1 + bootlen, final_offset - 1))

for lnkdst in sorted(level1pkts.keys()):
    off_len_list = level1pkts[lnkdst]

    # First payload is main code for transputers 1, 2 and 3
    (offset, length) = off_len_list[0]
    codelen = inputdata[offset]
    assert codelen == length - 1
    print("[{:04x}-{:04x}] * Transputer {} code ({} bytes)".format(
        offset + 1, offset + length - 1, lnkdst, codelen))
    with open('transputer{}.bin'.format(lnkdst), 'wb') as f:
        f.write(inputdata[offset + 1:offset + length])

    # Then forwarded packets
    level2pkts = {1: [], 2: [], 3: []}
    for offset, length in off_len_list[1:]:
        length2, destination, _ = struct.unpack('<III', inputdata[offset:offset + 12])
        assert length2 + 12 == length
        lnkdst2 = (destination - 0x80000000) // 4
        assert destination == 0x80000000 + lnkdst2 * 4
        level2pkts[lnkdst2].append((offset + 12, length2))

    for lnkdst2 in sorted(level2pkts.keys()):
        off_len_list = level2pkts[lnkdst2]

        # First payload is loader code for transputers 4-12
        transputer_id = 3 * lnkdst + lnkdst2
        (offset, length) = off_len_list[0]
        codelen = inputdata[offset]
        assert codelen == length - 1
        print("[{:04x}-{:04x}]   * Transputer {} loader code ({} bytes)".format(
            offset + 1, offset + length - 1, transputer_id, codelen))
        with open('transputer{}_loader.bin'.format(transputer_id), 'wb') as f:
            f.write(inputdata[offset + 1:offset + length])

        # Second payload is main code
        assert len(off_len_list) == 2
        (offset, length) = off_len_list[1]
        length2, _, entry = struct.unpack('<III', inputdata[offset:offset + 12])
        assert length2 + 12 == length
        print("[{:04x}-{:04x}]     * Transputer {} main code ({} bytes), entry {:#x}".format(
            offset + 12, offset + length - 1, transputer_id, length2, entry))
```

```
        with open('transputer{}_main.bin'.format(transputer_id), 'wb') as f:
            f.write(inputdata[offset + 12:offset + length])

offset = final_offset

# Key
print("[{:04x}-{:04x}] '{}'".format(
    offset, offset + 3, inputdata[offset:offset + 4].decode('ascii')))
print("[{:04x}-{:04x}] {}".format(
    offset + 4, offset + 15, repr(inputdata[offset + 4:offset + 16])))
offset += 16

# Name
namelen = inputdata[offset]
name = inputdata[offset + 1:offset + 1 + namelen].decode('ascii')
print("[{:04x}-{:04x}] Name: '{}'".format(offset, offset + namelen, name))
offset += 1 + namelen

# Encrypted data
print("[{:04x}-{:04x}] Encrypted data ({} bytes)".format(
    offset, len(inputdata) - 1, len(inputdata) - offset))

with open('encrypted', 'wb') as f:
    f.write(inputdata[offset:])
```

A.3 Partie 5, décodeur des instructions ST20

A.3.1 Générateur de pseudo-code ST20

Ce fichier s'appelle `decode_st20.py`

```
#!/usr/bin/env python3
"""Decode an ST20 firmware containing code and data"""

def to_signed32(value):
    """Make an unsigned 32-bit number signed in Python"""
    if value & 0x80000000:
        return -((-value) & 0xffffffff)
    return value & 0xffffffff


def to_decorhex(value):
    """Represent a number in decimal or hexadecimal depending on its value"""
    return str(value) if 0 <= value < 16 else hex(to_signed32(value))


class PushExpr(object):
    """Represent an expression which is pushed on the stack with maybe several
    instructions.

    If the value is an integer, it is a constant which can be combined with
    other push instructions.
    Otherwise, the value is a string describing the expression which is
    computed in pseudo-code.
    """
    def __init__(self, filemeta, iaddr, isize, value, previous=None):
        """Define a new pushed expression in the given FileMeta object.

        previous is another PushExpr object which was previosly pushed
        """
        self.filemeta = filemeta
        self.iaddr = iaddr
```

```
        self.isize = isize
        self.value = value
        self.previous = previous
        self.addparens = False  # Add parentheses around the value

    def __str__(self):
        if self.addparens:
            return '(' + self.value + ')'
        elif not isinstance(self.value, int):
            # the value is already a string
            return self.value
        elif 0 <= self.value < 16:
            return str(self.value)
        return hex(self.value)

    def show(self):
        """Show the pushed value as a pseudo-code instruction"""
        if self.previous:
            self.previous.show()
        desc = 'Push ' + str(self)
        self.filemeta.show_instruction(self.iaddr, self.isize, desc)

    def add(self, isize, value):
        """Merge an instruction of size isize which adds the given value to the
        top of the stack"""
        self.isize += isize
        if isinstance(self.value, int):
            self.value = (self.value + value) & 0xffffffff
        else:
            self.value = '({})+{}'.format(self.value, value)

    def merge1(self, isize, opformat, addparens=True):
        """Merge an instruction which operates on the stack top"""
        self.isize += isize
        self.value = opformat.format(str(self))
        self.addparens = addparens

    def merge2(self, isize, opformat, addparens=True):
        """Merge an instruction of size isize which has 2 operands"""
        if self.previous is None:
            self.isize += isize
            self.value = opformat.format(A=str(self), B='Pop')
            self.addparens = addparens
        else:
            self.value = opformat.format(A=str(self), B=str(self.previous))
            self.iaddr = self.previous.iaddr
            self.isize += self.previous.isize + isize
            self.previous = self.previous.previous
            self.addparens = addparens


class FileMeta(object):
    """Store the specific meta-information which help decoding the file.

    This information consists in code labels, data labels and comments
    """
    def __init__(self, mem, first_addr, codelab, datalab, comments):
        """
        Initialize a file which content is in mem, loaded at address
        first_addr in virtual memory, with the given meta-information
        """
        self.mem = mem
        self.first_addr = first_addr
        self.codelab = codelab
```

```
        self.datalab = datalab
        self.comments = comments

    def show_instruction(self, iaddr, isize, desc=None):
        """Show instruction at virtual address iaddr, size isize

        desc is an additional description. "None" means displaying the ASCII
        representation of the bytes, if available
        """
        ioffset = iaddr - self.first_addr
        assert 0 <= ioffset < ioffset + isize <= len(self.mem)
        idata = self.mem[ioffset:ioffset + isize]

        if desc is None:
            # Show ASCII characters like an hexadecimal dump
            desc = ''.join(chr(b) if 32 <= b < 127 else '\\x{:02x}'.format(b) for b in idata)
            desc = '"{}"'.format(desc)
        if isize > 8:
            # Show description on the next line if it is too long
            desc = '\n                                    ' + desc

        comment = self.comments.get(iaddr, '')
        if comment:
            comment = ' # ' + comment
        line = '  {:04x}: '.format(iaddr)
        while len(idata):
            line += '{:25s}'.format(' '.join('{:02x}'.format(b) for b in idata[:16]))
            idata = idata[16:]
            if len(idata):
                line += '\n        '
        print(line + desc + comment)

    def show_all(self):
        """Show all instructions in the file"""
        in_code = False
        cur_offset = 0  # Offset in self.mem
        cur_pushexpr = None  # Combine constants together using PushExpr

        while cur_offset < len(self.mem):
            iaddr = self.first_addr + cur_offset

            # Label management, to jump into code or data
            label = self.codelab.get(iaddr)
            if label is not None:
                in_code = True
            else:
                label = self.datalab.get(iaddr)
                if label is not None:
                    in_code = False
            if label is not None:
                if cur_pushexpr is not None:
                    cur_pushexpr.show()
                    cur_pushexpr = None
                print("\n{}:".format(label))

            isize = 1
            if in_code:
                # Decode instruction
                desc = 'Unknown instruction'
                instr = self.mem[cur_offset]
                icode = instr & 0xf0
                oreg = instr & 0x0f

                # Read prefix
```

```
while icode in (0x20, 0x60) and cur_offset + isize < len(self.mem):
    if icode == 0x20:  # PFIX, Prefix
        oreg = (oreg << 4) & 0xffffffff
    else:
        assert icode == 0x60  # NFIX, Negative Prefix
        oreg = ((~oreg) << 4) & 0xffffffff
    icode = self.mem[cur_offset + isize] & 0xf0
    oreg |= self.mem[cur_offset + isize] & 0x0f
    isize += 1

# Decode instruction
if icode == 0x00:  # J, Jump
    destaddr = iaddr + isize + to_signed32(oreg)
    label = self.codelab.get(destaddr, '??')
    desc = 'Jump {:#04x} <{}>'.format(destaddr, label)

elif icode == 0x10:  # LDLP, Load Local Pointer
    expr = 'ptr W[{}]'.format(to_decorhex(oreg))
    cur_pushexpr = PushExpr(self, iaddr, isize, expr, cur_pushexpr)
    cur_offset += isize
    continue

elif icode == 0x30:  # LDNL, Load Non-Local
    if cur_pushexpr is None:
        desc = 'A = A[{:#x}]'.format(to_signed32(oreg))
    else:
        cur_pushexpr = PushExpr(
            self,
            cur_pushexpr.iaddr,
            cur_pushexpr.isize + isize,
            '({})[{:#x}]'.format(str(cur_pushexpr), to_signed32(oreg)))
        cur_offset += isize
        continue

elif icode == 0x40:  # LDC, Load Constant
    cur_pushexpr = PushExpr(self, iaddr, isize, oreg, cur_pushexpr)
    cur_offset += isize
    continue

elif icode == 0x50:  # LDNLP, Load Non-Local Pointer
    if cur_pushexpr is None:
        desc = 'A = ptr A[{:#x}]'.format(to_signed32(oreg))
    else:  # Merge with a previous push
        cur_pushexpr.add(isize, 4 * to_signed32(oreg))
        cur_offset += isize
        continue

elif icode == 0x70:  # LDL, Load Local
    expr = 'W[{}]'.format(to_decorhex(oreg))
    cur_pushexpr = PushExpr(self, iaddr, isize, expr, cur_pushexpr)
    cur_offset += isize
    continue

elif icode == 0x80:  # ADC, Add Constant
    if cur_pushexpr is None:
        desc = 'A += {:#x}'.format(to_signed32(oreg))
    else:
        cur_pushexpr.add(isize, to_signed32(oreg))
        cur_offset += isize
        continue

elif icode == 0x90:  # CALL, Subroutine Call
    destaddr = iaddr + isize + to_signed32(oreg)
    label = self.codelab.get(destaddr, '??')
```

```
        desc = 'Call {:#04x} <{}>'.format(destaddr, label)

    elif icode == 0xa0:  # CJ, Conditional Jump
        destaddr = iaddr + isize + to_signed32(oreg)
        label = self.codelab.get(destaddr, '??')
        desc = 'If A=0 Jump {:#04x} <{}> else Pop'.format(destaddr, label)

    elif icode == 0xb0:  # AJW, Adjust Workspace
        desc = 'W += 4 * {:#x}'.format(to_signed32(oreg))

    elif icode == 0xc0:  # EQC, Equals Constant
        desc = 'A = (A == {:#x}) ? 1 : 0'.format(oreg)

    elif icode == 0xd0:  # STL, Store Local
        soreg = to_signed32(oreg)
        if cur_pushexpr is None:
            desc = 'W[{}] = Pop'.format(to_decorhex(oreg))
        else:
            desc = 'W[{}] = {}'.format(to_decorhex(oreg), str(cur_pushexpr))
            cur_offset -= cur_pushexpr.isize
            isize += cur_pushexpr.isize
            cur_pushexpr = cur_pushexpr.previous

    elif icode == 0xe0:  # STNL, Store Non-Local
        desc = 'A[{:#x}] = B, Pop, Pop'.format(to_signed32(oreg))

    elif icode == 0xf0:  # OPR, Operate (extended instruction set)
        desc = '\033[31;1mUnknown OPR {:#x}\033[m'.format(oreg)

        if oreg == 0x01:  # LB, Load Byte
            if cur_pushexpr is None:
                desc = 'A = loadbyte(A)'
            else:
                cur_pushexpr.addparens = False
                cur_pushexpr.merge1(isize, 'LB({})', False)
                cur_offset += isize
                continue

        elif oreg == 0x02:  # BSUB, Byte Subscript
            if cur_pushexpr is None:
                desc = 'BSUB (A += B, pop)'
            else:
                cur_pushexpr.merge2(isize, '{A}+{B}')
                cur_offset += isize
                continue

        elif oreg == 0x06:  # GCALL, General Call
            desc = 'GCALL (swap A, IPtr)'

        elif oreg == 0x07:  # IN(address=C, channel=B, length=A)
            lendesc = 'A'
            chandesc = 'B'
            addrdesc = 'C'
            if cur_pushexpr is not None:
                lendesc = str(cur_pushexpr)
                cur_offset -= cur_pushexpr.isize
                isize += cur_pushexpr.isize
                cur_pushexpr = cur_pushexpr.previous
            if cur_pushexpr is not None:
                chandesc = str(cur_pushexpr)
                cur_offset -= cur_pushexpr.isize
                isize += cur_pushexpr.isize
                cur_pushexpr = cur_pushexpr.previous
            if cur_pushexpr is not None:
```

```
            addrdesc = str(cur_pushexpr)
            cur_offset -= cur_pushexpr.isize
            isize += cur_pushexpr.isize
            cur_pushexpr = cur_pushexpr.previous
        desc = 'IN(addr={}, chan={}, len={})'.format(
            addrdesc, chandesc, lendesc)

    elif oreg == 0x08:  # PROD, Product
        if cur_pushexpr is None:
            desc = 'PROD (1 *= B, pop)'
        else:
            cur_pushexpr.merge2(isize, '{A}*{B}')
            cur_offset += isize
            continue

    elif oreg == 0x09:  # GT, Greater Than
        if cur_pushexpr is None:
            desc = 'GT (A = (B > A? 1 : 0), pop)'
        else:
            cur_pushexpr.merge2(isize, '({B} > {A})? 1 : 0')
            cur_offset += isize
            continue

    elif oreg == 0x0a:  # WSUB, Word Subscript
        if cur_pushexpr is None:
            desc = 'WSUB (A += 4*B, pop)'
        else:
            cur_pushexpr.merge2(isize, '{A}+4*{B}')
            cur_offset += isize
            continue

    elif oreg == 0x0b:  # OUT(address=C, channel=B, length=A)
        lendesc = 'A'
        chandesc = 'B'
        addrdesc = 'C'
        if cur_pushexpr is not None:
            lendesc = str(cur_pushexpr)
            cur_offset -= cur_pushexpr.isize
            isize += cur_pushexpr.isize
            cur_pushexpr = cur_pushexpr.previous
        if cur_pushexpr is not None:
            chandesc = str(cur_pushexpr)
            cur_offset -= cur_pushexpr.isize
            isize += cur_pushexpr.isize
            cur_pushexpr = cur_pushexpr.previous
        if cur_pushexpr is not None:
            addrdesc = str(cur_pushexpr)
            addrval = cur_pushexpr.value
            if isinstance(addrval, int) and addrval in self.datalab:
                addrdesc += ' <{}>'.format(self.datalab[addrval])
            cur_offset -= cur_pushexpr.isize
            isize += cur_pushexpr.isize
            cur_pushexpr = cur_pushexpr.previous
        desc = 'OUT(addr={}, chan={}, len={})'.format(
            addrdesc, chandesc, lendesc)

    elif oreg == 0x1b:  # LDPI, Load Pointer to Instruction
        if cur_pushexpr is None:
            desc = 'A += {:#x}'.format(iaddr + isize)
        else:
            cur_pushexpr.add(isize, iaddr + isize)
            cur_offset += isize
            continue
```

```
elif oreg == 0x1f:  # REM, Remainder
    if cur_pushexpr is None:
        desc = 'REM (A = B % A, pop)'
    else:
        cur_pushexpr.merge2(isize, '{B}%{A}')
        cur_offset += isize
        continue

elif oreg == 0x20:  # RET, Return from a function call
    desc = 'Return'

elif oreg == 0x33:  # XOR
    if cur_pushexpr is None:
        desc = 'XOR (A = B ^ A, pop)'
    else:
        cur_pushexpr.merge2(isize, '{B} ^ {A}')
        cur_offset += isize
        continue

elif oreg == 0x3b:  # SB, Store Byte
    if cur_pushexpr is None:
        desc = 'storebyte(A, B), pop2'
    else:
        cur_pushexpr.merge2(0, 'SB:{A} = {B}')
        desc = cur_pushexpr.value
        cur_offset -= cur_pushexpr.isize
        isize += cur_pushexpr.isize
        cur_pushexpr = cur_pushexpr.previous

elif oreg == 0x3c:  # GAJW, General Adjust Workspace, swap A and W
    if cur_pushexpr is None:
        desc = 'SWAP(A, W)'
    else:
        desc = 'Push W; W = {}'.format(str(cur_pushexpr))
        cur_offset -= cur_pushexpr.isize
        isize += cur_pushexpr.isize
        cur_pushexpr = cur_pushexpr.previous

elif oreg == 0x40:  # SHR, Shift Right
    if cur_pushexpr is None:
        desc = 'SHR (A = B >> A, pop)'
    else:
        cur_pushexpr.merge2(isize, '{B}>>{A}')
        cur_offset += isize
        continue

elif oreg == 0x41:  # SHL, Shift Left
    if cur_pushexpr is None:
        desc = 'SHL (A = B << A, pop)'
    else:
        cur_pushexpr.merge2(isize, '{B}<<{A}')
        cur_offset += isize
        continue

elif oreg == 0x42:  # MINT, push 0x80000000
    cur_pushexpr = PushExpr(self, iaddr, isize, 0x80000000, cur_pushexpr)
    cur_offset += isize
    continue

elif oreg == 0x46:  # AND
    if cur_pushexpr is None:
        desc = 'AND (A = B & A, pop)'
    else:
        cur_pushexpr.merge2(isize, '{B}&{A}')
```

```
                        cur_offset += isize
                        continue

                elif oreg == 0x5a:  # DUP
                    desc = 'DUP (Push A)'

                elif oreg == 0xc1:  # SSUB, 16-bit word subscript
                    if cur_pushexpr is None:
                        desc = 'SSUB (A += 2*B, pop)'
                    else:
                        cur_pushexpr.merge2(isize, '{A}+2*{B}')
                        cur_offset += isize
                        continue
            else:
                # Show data
                desc = None
                while isize < 16 and cur_offset + isize < len(self.mem):
                    if iaddr + isize in self.codelab:
                        break
                    if iaddr + isize in self.datalab:
                        break
                    isize += 1

            # Show all previous push instructions
            if cur_pushexpr is not None:
                cur_pushexpr.show()
                cur_pushexpr = None
            # Show current instruction and go to the next one
            self.show_instruction(self.first_addr + cur_offset, isize, desc)
            cur_offset += isize
```

A.3.2 Décodeur du code du transputer 0

```
#!/usr/bin/env python3
"""Decode the firmware of transputer 0"""
import sys
sys.path.insert(0, '.')
from decode_st20 import FileMeta

CODE_LABELS = {
    0x00: 'boot',
    0x17: 'initialize_other_transputers',
    0x37: 'initialization_done',
    0x77: 'mainloop',
    0xf4: 'end',
}

DATA_LABELS = {
    0xdc: '"Boot ok"',
    0xe4: '"Code Ok"',
    0xec: '"Decrypt"',
}

COMMENTS = {
    0x54: "read 'KEY:'",
    0xcf: "W[4] is incremented modulo 12",
}

with open('input.bin', 'rb') as f:
    inputdata = f.read()
```

```
# First byte if the size of the firmware
fwsize = inputdata[0]
fw = inputdata[1:1 + fwsize]
FileMeta(fw, 0, CODE_LABELS, DATA_LABELS, COMMENTS).show_all()
```

A.3.3 Décodeur du code des transputers 1, 2 et 3

```
#!/usr/bin/env python3
"""Decode the firmware shared between transputers 1, 2 and 3"""
import sys
sys.path.insert(0, '.')
from decode_st20 import FileMeta

CODE_LABELS = {
    0x00: 'boot',
    0x0b: 'relay_code',
    0x26: 'mainloop',
    0x6d: 'exit',
}

with open('transputer1.bin', 'rb') as f:
    fw = f.read().rstrip(b' ')

FileMeta(fw, 0, CODE_LABELS, {}, {}).show_all()
```

A.3.4 Décodeur du code d'amorçage des transputers 4 à 12

```
#!/usr/bin/env python3
"""Decode the firmware of transputer 4-12 loader"""
import sys
sys.path.insert(0, '.')
from decode_st20 import FileMeta

CODE_LABELS = {
    0x00: 'boot',
    0x20: 'exit',
}

COMMENTS = {
    0x19: "Download the next code in 0x1f",
    0x1e: "Run the main code",
}

with open('transputer4_loader.bin', 'rb') as f:
    fw = f.read().rstrip(b' ')

FileMeta(fw, 0, CODE_LABELS, {}, COMMENTS).show_all()
```

A.3.5 Décodeur du code principal des transputers 4 à 12

```
#!/usr/bin/env python3
"""Decode the main program which is run on transputers 4 to 12"""
import sys
```

```
sys.path.insert(0, '.')
from decode_st20 import FileMeta

CODE_LABELS = {
    4: {
        0x33: 'mainloop',
        0x3e: 'forloop',
        0x55: 'forloop_end',
        },
    5: {
        0x33: 'mainloop',
        0x3e: 'forloop_W[0]=0..11',
        0x55: 'forloop_end',
    },
    6: {
        0x33: 'mainloop',
        0x42: 'forloop_W[0]=0..11',
        0x59: 'forloop_end',
        0x5b: 'endif',
    },
    7: {
        0x2f: 'mainloop',
        0x3e: 'forloop_W[0]=0..6',
        0x5e: 'forloop_end',
    },
    8: {
        0x33: 'init_forloop1_W[2]=0..3',
        0x35: 'init_forloop2_W[0]=0..11',
        0x48: 'init_forloop2_end',
        0x51: 'mainloop',
        0x69: '_0x80001069',
        0x6f: 'forloop1_W[2]=0..3',
        0x73: 'forloop2_W[0]=0..11',
        0x8c: 'forloop2_end',
        0xa1: 'forloop1_end',
    },
    9: {
        0x2f: 'mainloop',
        0x3e: 'forloop_W[0]=0..11',
        0x5b: 'forloop_end',
    },
    10: {
        0x33: 'init_forloop1_W[0]=0..3',
        0x35: 'init_forloop2_W[1]=0..11',
        0x48: 'init_forloop2_end',
        0x51: 'mainloop',
        0x69: 'endif',
        0x6d: 'forloop_W[0]=0..4',
        0x84: 'forloop_end',
    },
    11: {
        0x31: 'mainloop',
    },
    12: {
        0x33: 'fill_W[3,4,5]_with_zeros',
        0x42: 'mainloop',
    }
}

for i in range(4, 13):
    print("\n--------- [ TRANSPUTER {}] ---------".format(i))
    with open('transputer{}_main.bin'.format(i), 'rb') as f:
        fw = f.read().rstrip(b' ')
```

```
    code_labels = CODE_LABELS.get(i, {})
    code_labels[0x1f] = 'recv(chan=B, mem=C, len=W[0])'
    code_labels[0x25] = 'send(chan=B, mem=C, len=W[0])'
    code_labels[0x2b] = 'entry'
    comments = {}

    FileMeta(fw, 0x1f, code_labels, {}, comments).show_all()
```

A.3.6 Attaque par force brute de la clé de déchiffrement

```
/**
 * Decrypt congratulations.tar.bz2.enc by bruteforcing the key
 * Compile with: gcc -Wall -Wextra -O3 decrypt.c -o decrypt -lbz2
 */
#include <assert.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <bzlib.h>

/**
 * Transputers code: decrypt size bytes in src to dst using the specified key
 */
static int decrypt(const uint8_t argkey[12], const uint8_t *src, uint8_t *dst, uint32_t size)
{
    uint8_t trans4w1 = 0, trans5w1 = 0;
    uint16_t trans6w1 = 0;
    uint8_t trans8w4 = 0, trans8w5sums[4] = {};
    uint8_t trans10w2 = 0, trans10w4buff0[4] = {}, trans10w4buffers[4 * 12] = {};
    uint8_t trans11next = 0;
    uint32_t byteidx;
    uint8_t key[12], i;

    memcpy(key, argkey, 12);

    for (i = 0; i < 12; i++)
        trans6w1 += key[i];

    for (byteidx = 0; byteidx < size; byteidx++) {
        const uint32_t keyidx = byteidx % 12;
        uint8_t transres, s0_5 = 0, s6_11 = 0, s0_11, s_t10 = 0;

        *(dst++) = *(src++) ^ (keyidx + 2 * key[keyidx]);

        for (i = 0; i < 6; i++)
            s0_5 += key[i];
        for (i = 6; i < 12; i++)
            s6_11 += key[i];
        s0_11 = s0_5 + s6_11;

        trans6w1 =
            ((trans6w1 & 0x8000) >> 15) ^
            ((trans6w1 & 0x4000) >> 14) ^
            ((trans6w1 << 1) & 0xffff);

        trans8w5sums[trans8w4] = s0_11;
        trans8w4 = (trans8w4 + 1) % 4;

        trans10w4buff0[trans10w2] = key[0];
```

```
        memcpy(trans10w4buffers + 12 * trans10w2, key, 12);
        trans10w2 = (trans10w2 + 1) % 4;

        transres = (trans6w1 & 0xff) ^ s0_5 ^ s6_11 ^ key[trans11next];
        transres ^= key[(key[0] ^ key[3] ^ key[7]) % 12];
        trans11next = (key[1] ^ key[5] ^ key[9]) % 12;

        for (i = 0; i < 12; i++) {
            uint8_t k = key[i];
            trans4w1 += k;
            trans5w1 ^= k;
            transres ^= k << (i & 7);
        }
        transres ^= trans4w1 ^ trans5w1;

        for (i = 0; i < 4; i++)
            transres ^= trans8w5sums[i];
        for (i = 0; i < 4; i++)
            s_t10 += trans10w4buff0[i];
        transres ^= trans10w4buffers[12 * (s_t10 & 3) + ((s_t10 >> 4) % 12)];

        key[keyidx] = transres;

        /* Destination[14] will be written using key[2] later.
         * Abort early if this byte will be non null for the BZip2 header.
         * Don't abort when decrypting the test vector.
         */
        if (size > 100 && byteidx == 2) {
            const uint8_t *origsrc = src - byteidx - 1;
            uint8_t decrypt14 = (origsrc[14] ^ (2 + 2 * key[2])) & 0xff;
            if (decrypt14 != 0)
                return 0;
        }
    }
    return 1;
}

/**
 * Test decrypt() function with the provided test vector
 */
static int testvector(void)
{
    const uint8_t encrypted[] = {
        0x1d, 0x87, 0xc4, 0xc4, 0xe0, 0xee, 0x40, 0x38, 0x3c, 0x59, 0x44, 0x7f,
        0x23, 0x79, 0x8d, 0x9f, 0xef, 0xe7, 0x4f, 0xb8, 0x24, 0x80, 0x76, 0x6e
    };
    const uint8_t key[] = "*SSTIC-2015*";
    uint8_t decrypted[25];

    decrypt(key, encrypted, decrypted, 24);
    decrypted[24] = '\0';
    printf("%s\n", (char *)decrypted);
    return !strcmp((char *)decrypted, "I love ST20 architecture");
}

int main(int argc, char **argv)
{
#define INPUT_SIZE 250606
    static uint8_t encrypted[INPUT_SIZE];
    static uint8_t decrypted[INPUT_SIZE];
    static uint8_t decompressed[INPUT_SIZE];
    uint8_t key[12];
    const uint8_t bz2head[10] = {
        0x42, 0x5a, 0x68, 0x39, 0x31, 0x41, 0x59, 0x26, 0x53, 0x59
```

```
};
unsigned int i;
ssize_t ret_size;
FILE *f;
bz_stream bst;
int bzret, j;

assert(testvector());

f = fopen("encrypted", "rb");
assert(f);
ret_size = fread(encrypted, 1, INPUT_SIZE, f);
assert(ret_size == INPUT_SIZE);
fclose(f);

/* Initialize key constants from BZip2 header magic */
for (i = 0; i < sizeof(bz2head); i++) {
    uint8_t wanted = encrypted[i] ^ bz2head[i];
    assert((wanted - i) % 2 == 0);
    key[i] = ((wanted - i) / 2) & 0x7f;
}

/* Bruteforce keys, starting with the given integer, if any */
i = (argc >= 2) ? (unsigned int)atoi(argv[1]) : 0;
for (; i < 256 * 256 * 1024; i++) {
    /* Choose the most significant bits of 10 first bytes according to i */
    for (j = 0; j < 10; j++) {
        if ((i >> (16 + j)) & 1) {
            key[j] |= 0x80;
        } else {
            key[j] &= 0x7f;
        }
    }
    /* Bytes 10 and 11 can't be known */
    key[10] = (i >> 8) & 0xff;
    key[11] = i & 0xff;

    if (!decrypt(key, encrypted, decrypted, INPUT_SIZE)) {
        /* Decrypt aborted early, so the key is invalid */
        continue;
    }

    /* Test BZ2 data */
    memset(&bst, 0, sizeof(bst));
    bzret = BZ2_bzDecompressInit(&bst, 0, 0);
    assert(bzret == BZ_OK);
    bst.next_in = (char *)decrypted;
    bst.avail_in = sizeof(decrypted);
    bst.next_out = (char *)decompressed;
    bst.avail_out = sizeof(decompressed);
    bzret = BZ2_bzDecompress(&bst);
    BZ2_bzDecompressEnd(&bst);
    if (bzret < 0) {
        /* Invalid data */
        continue;
    }

    /* Key found */
    printf("Found %#04x\n", i);
    printf("Key:");
    for (j = 0; j < 12; j++) {
        printf(" %02x", key[j]);
    }
    printf("\n");
```

```
        /* Save the file */
        f = fopen("congratulations.tar.bz2", "wb");
        assert(f != NULL);
        ret_size = fwrite(decrypted, 1, sizeof(decrypted), f);
        assert(ret_size == (ssize_t)sizeof(decrypted));
        fclose(f);
        return 0;
    }
    fprintf(stderr, "Bruteforce failed!\n");
    return 1;
}
```

B Sortie produite par les scripts de la partie 5

B.1 Organisation du fichier `input.bin`

Voici la sortie du script dont le code a été donné dans la partie A.2 et qui présente l'organisation du contenu du fichier `input.bin`.

```
[0001-00f8] Transputer 0 code (248 bytes)
[00f9-0984] Transputer 0 forwarded packets:
[0106-0175] * Transputer 1 code (112 bytes)
[0289-02ac]   * Transputer 4 loader code (36 bytes)
[04b9-04fc]     * Transputer 4 main code (68 bytes), entry 0xc
[02c6-02e9]   * Transputer 5 loader code (36 bytes)
[0521-0564]     * Transputer 5 main code (68 bytes), entry 0xc
[0303-0326]   * Transputer 6 loader code (36 bytes)
[0589-0608]     * Transputer 6 main code (128 bytes), entry 0xc
[0183-01f2] * Transputer 2 code (112 bytes)
[0340-0363]   * Transputer 7 loader code (36 bytes)
[062d-0684]     * Transputer 7 main code (88 bytes), entry 0xc
[037d-03a0]   * Transputer 8 loader code (36 bytes)
[06a9-0738]     * Transputer 8 main code (144 bytes), entry 0xc
[03ba-03dd]   * Transputer 9 loader code (36 bytes)
[075d-07a4]     * Transputer 9 main code (72 bytes), entry 0xc
[0200-026f] * Transputer 3 code (112 bytes)
[03f7-041a]   * Transputer 10 loader code (36 bytes)
[07c9-0854]     * Transputer 10 main code (140 bytes), entry 0xc
[0434-0457]   * Transputer 11 loader code (36 bytes)
[0879-08dc]     * Transputer 11 main code (100 bytes), entry 0xc
[0471-0494]   * Transputer 12 loader code (36 bytes)
[0901-0978]     * Transputer 12 main code (120 bytes), entry 0xc
[0985-0988] 'KEY:'
[0989-0994] b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff'
[0995-09ac] Name: 'congratulations.tar.bz2'
[09ad-3dc9a] Encrypted data (250606 bytes)
```

B.2 Pseudo-code généré par l'analyse des codes des transputers

Voici la sortie des scripts de la partie A.3.1.

B.2.1 Pseudo-code du transputer 0

```
boot:
  0000: 64 b4                   W += 4 * -0x4c
  0002: 40 d1                   W[1] = 0
  0004: 40 d3                   W[3] = 0
  0006: 24 f2 24 20 50 23 fc    Push W; W = 0x80001000
  000d: 64 b4                   W += 4 * -0x4c
  000f: 2c 49 21 fb 24 f2 48 fb  OUT(addr=0xdc <"Boot ok">, chan=0x80000000, len=8)

initialize_other_transputers:
  0017: 24 19 24 f2 54 4c f7    IN(addr=ptr W[0x49], chan=0x80000010, len=12)
  001e: 24 79                   Push W[0x49]
  0020: 21 a5                   If A=0 Jump 0x37 <initialization_done> else Pop
  0022: 2c 4d 21 fb 24 f2 54 24 79 f7
                                IN(addr=0xf3, chan=0x80000010, len=W[0x49])
  002c: 2c 43 21 fb 24 7a 24 79 fb
                                OUT(addr=0xf3, chan=W[0x4a], len=W[0x49])
  0035: 61 00                   Jump 0x17 <initialize_other_transputers>

initialization_done:
  0037: 24 19 24 f2 51 4c fb    OUT(addr=ptr W[0x49], chan=0x80000004, len=12)
  003e: 24 19 24 f2 52 4c fb    OUT(addr=ptr W[0x49], chan=0x80000008, len=12)
  0045: 24 19 24 f2 53 4c fb    OUT(addr=ptr W[0x49], chan=0x8000000c, len=12)
  004c: 29 44 21 fb 24 f2 48 fb  OUT(addr=0xe4 <"Code Ok">, chan=0x80000000, len=8)
  0054: 12 24 f2 54 44 f7       IN(addr=ptr W[2], chan=0x80000010, len=4) # read 'KEY:'
```

```
  005a: 15 24 f2 54 4c f7        IN(addr=ptr W[5], chan=0x80000010, len=12)
  0060: 28 48 21 fb 24 f2 48 fb  OUT(addr=0xec <"Decrypt">, chan=0x80000000, len=8)
  0068: 13 24 f2 54 41 f7        IN(addr=ptr W[3], chan=0x80000010, len=1)
  006e: 19 24 f2 54 13 f1 f7     IN(addr=ptr W[9], chan=0x80000010, len=LB(ptr W[3]))
  0075: 40 d4                    W[4] = 0

mainloop:
  0077: 11 24 f2 54 41 f7        IN(addr=ptr W[1], chan=0x80000010, len=1)
  007d: 15 24 f2 51 4c fb        OUT(addr=ptr W[5], chan=0x80000004, len=12)
  0083: 15 24 f2 52 4c fb        OUT(addr=ptr W[5], chan=0x80000008, len=12)
  0089: 15 24 f2 53 4c fb        OUT(addr=ptr W[5], chan=0x8000000c, len=12)
  008f: 10 81 24 f2 55 41 f7     IN(addr=(ptr W[0])+1, chan=0x80000014, len=1)
  0096: 10 82 24 f2 56 41 f7     IN(addr=(ptr W[0])+2, chan=0x80000018, len=1)
  009d: 10 83 24 f2 57 41 f7     IN(addr=(ptr W[0])+3, chan=0x8000001c, len=1)
  00a4: 10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3 10 81 23
        fb
                                 SB:(ptr W[0])+1 = ((LB((ptr W[0])+1) ^ LB((ptr W[0])+2)) ^
                                 LB((ptr W[0])+3))
  00b5: 11 f1 74 15 f2 f1 74 2c f1 23 f3 10 23 fb
                                 SB:ptr W[0] = (LB(ptr W[1]) ^ (W[4]+2*LB(ptr W[5]+W[4])))
  00c3: 10 81 f1 74 15 f2 23 fb  SB:(ptr W[5]+W[4]) = LB((ptr W[0])+1)
  00cb: 74 81                    Push (W[4])+1
  00cd: 25 fa                    DUP (Push A)
  00cf: d4                       W[4] = Pop # W[4] is incremented modulo 12
  00d0: cc                       A = (A == 0xc) ? 1 : 0
  00d1: a3                       If A=0 Jump 0xd5 <??> else Pop
  00d2: 80                       A += 0x0
  00d3: 40 d4                    W[4] = 0
  00d5: 10 24 f2 41 fb           OUT(addr=ptr W[0], chan=0x80000000, len=1)
  00da: 66 0b                    Jump 0x77 <mainloop>

"Boot ok":
  00dc: 42 6f 6f 74 20 6f 6b 00  "Boot ok\x00"

"Code Ok":
  00e4: 43 6f 64 65 20 4f 6b 00  "Code Ok\x00"

"Decrypt":
  00ec: 44 65 63 72 79 70 74 00  "Decrypt\x00"

end:
  00f4: 24 bc                    W += 4 * 0x4c
  00f6: 22 f0                    Return
```

B.2.2 Pseudo-code des transputers 1, 2 et 3

```
boot:
  0000: 60 b8                    W += 4 * -0x8
  0002: 24 f2 24 20 50 23 fc     Push W; W = 0x80001000
  0009: 60 b8                    W += 4 * -0x8

relay_code:
  000b: 15 24 f2 54 4c f7        IN(addr=ptr W[5], chan=0x80000010, len=12)
  0011: 75                       Push W[5]
  0012: 21 a2                    If A=0 Jump 0x26 <mainloop> else Pop
  0014: 25 44 21 fb 24 f2 54 75 f7
                                 IN(addr=0x6c, chan=0x80000010, len=W[5])
  001d: 24 4b 21 fb 76 75 fb     OUT(addr=0x6c, chan=W[6], len=W[5])
  0024: 61 05                    Jump 0x0b <relay_code>

mainloop:
  0026: 11 24 f2 54 4c f7        IN(addr=ptr W[1], chan=0x80000010, len=12)
  002c: 11 24 f2 51 4c fb        OUT(addr=ptr W[1], chan=0x80000004, len=12)
  0032: 11 24 f2 52 4c fb        OUT(addr=ptr W[1], chan=0x80000008, len=12)
  0038: 11 24 f2 53 4c fb        OUT(addr=ptr W[1], chan=0x8000000c, len=12)
```

```
  003e: 10 81 24 f2 55 41 f7     IN(addr=(ptr W[0])+1, chan=0x80000014, len=1)
  0045: 10 82 24 f2 56 41 f7     IN(addr=(ptr W[0])+2, chan=0x80000018, len=1)
  004c: 10 83 24 f2 57 41 f7     IN(addr=(ptr W[0])+3, chan=0x8000001c, len=1)
  0053: 10 81 f1 10 82 f1 23 f3 10 83 f1 23 f3
                                 Push ((LB((ptr W[0])+1) ^ LB((ptr W[0])+2)) ^ LB((ptr W[0])+3))
  0060: 25 fa                    DUP (Push A)
  0062: 10 23 fb                 SB:ptr W[0] = Pop
  0065: 10 24 f2 41 fb           OUT(addr=ptr W[0], chan=0x80000000, len=1)
  006a: 64 0a                    Jump 0x26 <mainloop>
  006c: 00                       Jump 0x6d <exit>

exit:
  006d: b8                       W += 4 * 0x8
  006e: 22 f0                    Return
```

B.2.3 Pseudo-code d'amorçage des transputers 4 à 12

```
boot:
  0000: 60 bd                    W += 4 * -0x3
  0002: 24 f2 24 20 50 23 fc     Push W; W = 0x80001000
  0009: 60 bd                    W += 4 * -0x3
  000b: 10 24 f2 54 4c f7        IN(addr=ptr W[0], chan=0x80000010, len=12)
  0011: 4b 21 fb 24 f2 54 70 f7  IN(addr=0x1f, chan=0x80000010, len=W[0])
  0019: 43 21 fb 72 f2           Push (W[2]+0x1f) # Download the next code in 0x1f
  001e: f6                       GCALL (swap A, IPtr) # Run the main code
  001f: 00                       Jump 0x20 <exit>

exit:
  0020: b3                       W += 4 * 0x3
  0021: 22 f0                    Return
```

B.2.4 Pseudo-code principal des transputers 4 à 12

```
---------------- [ TRANSPUTER 4] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7              IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                    Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb              OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                    Return

entry:
  002b: 60 bb                    W += 4 * -0x5
  002d: 40 d1                    W[1] = 0
  002f: 40 11 23 fb              SB:ptr W[1] = 0

mainloop:
  0033: 4c d0                    W[0] = 12
  0035: 12                       Push ptr W[2]
  0036: 24 f2 54                 Push 0x80000010
  0039: 76                       Push W[6]
  003a: 61 93                    Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  003c: 40 d0                    W[0] = 0

forloop:
  003e: 70 12 f2 f1 11 f1 f2 2f 4f 24 f6 11 23 fb
                                 SB:ptr W[1] = ((LB(ptr W[1])+LB(ptr W[2]+W[0]))&0xff)
  004c: 70 81 d0                 W[0] = (W[0])+1
  004f: 4c 70 f9                 Push ((12 > W[0]) ? 1 : 0)
  0052: a2                       If A=0 Jump 0x55 <forloop_end> else Pop
  0053: 61 09                    Jump 0x3e <forloop>

forloop_end:
```

```
  0055: 41 d0                   W[0] = 1
  0057: 11                      Push ptr W[1]
  0058: 24 f2                   Push 0x80000000
  005a: 76                      Push W[6]
  005b: 63 98                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  005d: 20 62 03                Jump 0x33 <mainloop>

---------------- [ TRANSPUTER 5] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 60 bb                   W += 4 * -0x5
  002d: 40 d1                   W[1] = 0
  002f: 40 11 23 fb             SB:ptr W[1] = 0

mainloop:
  0033: 4c d0                   W[0] = 12
  0035: 12                      Push ptr W[2]
  0036: 24 f2 54                Push 0x80000010
  0039: 76                      Push W[6]
  003a: 61 93                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  003c: 40 d0                   W[0] = 0

forloop_W[0]=0..11:
  003e: 70 12 f2 f1 71 23 f3 2f 4f 24 f6 11 23 fb
                                SB:ptr W[1] = ((LB(ptr W[2]+W[0]) ^ W[1])&0xff)
  004c: 70 81 d0                W[0] = (W[0])+1
  004f: 4c 70 f9                Push ((12 > W[0]) ? 1 : 0)
  0052: a2                      If A=0 Jump 0x55 <forloop_end> else Pop
  0053: 61 09                   Jump 0x3e <forloop_W[0]=0..11>

forloop_end:
  0055: 41 d0                   W[0] = 1
  0057: 11                      Push ptr W[1]
  0058: 24 f2                   Push 0x80000000
  005a: 76                      Push W[6]
  005b: 63 98                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  005d: 20 62 03                Jump 0x33 <mainloop>

---------------- [ TRANSPUTER 6] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 60 b9                   W += 4 * -0x7
  002d: 40 d2                   W[2] = 0
  002f: 40 d1                   W[1] = 0
  0031: 40 d3                   W[3] = 0

mainloop:
  0033: 4c d0                   W[0] = 12
```

```
  0035: 14                        Push ptr W[4]
  0036: 24 f2 54                  Push 0x80000010
  0039: 78                        Push W[8]
  003a: 61 93                     Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  003c: 73                        Push W[3]
  003d: c0                        A = (A == 0x0) ? 1 : 0
  003e: 21 ab                     If A=0 Jump 0x5b <endif> else Pop
  0040: 40 d0                     W[0] = 0

forloop_W[0]=0..11:
  0042: 70 14 f2 f1 71 f2 2f 2f 2f 4f 24 f6 d1
                                  W[1] = ((W[1]+LB(ptr W[4]+W[0]))&0xffff)
  004f: 70 81 d0                  W[0] = (W[0])+1
  0052: 4c 70 f9                  Push ((12 > W[0]) ? 1 : 0)
  0055: a3                        If A=0 Jump 0x59 <forloop_end> else Pop
  0056: 80                        A += 0x0
  0057: 61 09                     Jump 0x42 <forloop_W[0]=0..11>

forloop_end:
  0059: 41 d3                     W[3] = 1

endif:
  005b: 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
                                  Push ((((((W[1]&0x8000)>>15) ^
                                  ((W[1]&0x4000)>>14))&0xffff) ^
                                  ((W[1]<<1)&0xffff))&0xffff)
  0089: 25 fa                     DUP (Push A)
  008b: d1                        W[1] = Pop
  008c: 2f 4f 24 f6 12 23 fb      SB:ptr W[2] = (Pop&0xff)
  0093: 41 d0                     W[0] = 1
  0095: 12                        Push ptr W[2]
  0096: 24 f2                     Push 0x80000000
  0098: 78                        Push W[8]
  0099: 67 9a                     Call 0x25 <send(chan=B, mem=C, len=W[0])>
  009b: 20 66 05                  Jump 0x33 <mainloop>

---------------- [ TRANSPUTER 7] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7               IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                     Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb               OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                     Return

entry:
  002b: 60 b9                     W += 4 * -0x7
  002d: 40 d3                     W[3] = 0

mainloop:
  002f: 4c d0                     W[0] = 12
  0031: 14                        Push ptr W[4]
  0032: 24 f2 54                  Push 0x80000010
  0035: 78                        Push W[8]
  0036: 61 97                     Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  0038: 40 d1                     W[1] = 0
  003a: 40 d2                     W[2] = 0
  003c: 40 d0                     W[0] = 0

forloop_W[0]=0..6:
  003e: 70 14 f2 f1 71 f2 2f 4f 24 f6 d1
```

```
                                W[1] = ((W[1]+LB(ptr W[4]+W[0]))&0xff)
  0049: 14 70 86 f2 f1 72 f2 2f 4f 24 f6 d2
                                W[2] = ((W[2]+LB((W[0])+6+ptr W[4]))&0xff)
  0055: 70 81 d0                W[0] = (W[0])+1
  0058: 46 70 f9                Push ((6 > W[0]) ? 1 : 0)
  005b: a2                      If A=0 Jump 0x5e <forloop_end> else Pop
  005c: 61 00                   Jump 0x3e <forloop_W[0]=0..6>

forloop_end:
  005e: 72 71 23 f3 2f 4f 24 f6 13 23 fb
                                SB:ptr W[3] = ((W[2] ^ W[1])&0xff)
  0069: 41 d0                   W[0] = 1
  006b: 13                      Push ptr W[3]
  006c: 24 f2                   Push 0x80000000
  006e: 78                      Push W[8]
  006f: 64 94                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  0071: 20 64 0b                Jump 0x2f <mainloop>

---------------- [ TRANSPUTER 8] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 61 bf                   W += 4 * -0x11
  002d: 40 d3                   W[3] = 0
  002f: 40 d4                   W[4] = 0
  0031: 40 d2                   W[2] = 0

init_forloop1_W[2]=0..3:
  0033: 40 d0                   W[0] = 0

init_forloop2_W[0]=0..11:
  0035: 40 72 43 f8 15 fa 70 f2 23 fb
                                SB:(W[0]+(ptr W[5]+4*(3*W[2]))) = 0
  003f: 70 81 d0                W[0] = (W[0])+1
  0042: 4c 70 f9                Push ((12 > W[0]) ? 1 : 0)
  0045: a2                      If A=0 Jump 0x48 <init_forloop2_end> else Pop
  0046: 61 0d                   Jump 0x35 <init_forloop2_W[0]=0..11>

init_forloop2_end:
  0048: 72 81 d2                W[2] = (W[2])+1
  004b: 44 72 f9                Push ((4 > W[2]) ? 1 : 0)
  004e: a2                      If A=0 Jump 0x51 <mainloop> else Pop
  004f: 61 02                   Jump 0x33 <init_forloop1_W[2]=0..3>

mainloop:
  0051: 4c d0                   W[0] = 12
  0053: 74 43 f8 15 fa          Push (ptr W[5]+4*(3*W[4]))
  0058: 24 f2 54                Push 0x80000010
  005b: 21 72                   Push W[0x12]
  005d: 63 90                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  005f: 74 81                   Push (W[4])+1
  0061: 25 fa                   DUP (Push A)
  0063: d4                      W[4] = Pop
  0064: c4                      A = (A == 0x4) ? 1 : 0
  0065: a3                      If A=0 Jump 0x69 <_0x80001069> else Pop
  0066: 80                      A += 0x0
  0067: 40 d4                   W[4] = 0
```

```
_0x80001069:
  0069: 40 13 23 fb              SB:ptr W[3] = 0
  006d: 40 d2                    W[2] = 0

forloop1_W[2]=0..3:
  006f: 40 d1                    W[1] = 0
  0071: 40 d0                    W[0] = 0

forloop2_W[0]=0..11:
  0073: 72 43 f8 15 fa 70 f2 f1 71 f2 2f 4f 24 f6 d1
                                 W[1] = ((W[1]+LB(W[0]+(ptr W[5]+4*(3*W[2]))))&0xff)
  0082: 70 81 d0                 W[0] = (W[0])+1
  0085: 4c 70 f9                 Push ((12 > W[0]) ? 1 : 0)
  0088: a3                       If A=0 Jump 0x8c <forloop2_end> else Pop
  0089: 80                       A += 0x0
  008a: 61 07                    Jump 0x73 <forloop2_W[0]=0..11>

forloop2_end:
  008c: 73 71 23 f3 2f 4f 24 f6 13 23 fb
                                 SB:ptr W[3] = ((W[3] ^ W[1])&0xff)
  0097: 72 81 d2                 W[2] = (W[2])+1
  009a: 44 72 f9                 Push ((4 > W[2]) ? 1 : 0)
  009d: a3                       If A=0 Jump 0xa1 <forloop1_end> else Pop
  009e: 80                       A += 0x0
  009f: 63 0e                    Jump 0x6f <forloop1_W[2]=0..3>

forloop1_end:
  00a1: 41 d0                    W[0] = 1
  00a3: 13                       Push ptr W[3]
  00a4: 24 f2                    Push 0x80000000
  00a6: 21 72                    Push W[0x12]
  00a8: 68 9b                    Call 0x25 <send(chan=B, mem=C, len=W[0])>
  00aa: 20 65 04                 Jump 0x51 <mainloop>

---------------- [ TRANSPUTER 9] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7              IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                    Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb              OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                    Return

entry:
  002b: 60 bb                    W += 4 * -0x5
  002d: 40 d1                    W[1] = 0

mainloop:
  002f: 4c d0                    W[0] = 12
  0031: 12                       Push ptr W[2]
  0032: 24 f2 54                 Push 0x80000010
  0035: 76                       Push W[6]
  0036: 61 97                    Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  0038: 40 11 23 fb              SB:ptr W[1] = 0
  003c: 40 d0                    W[0] = 0

forloop_W[0]=0..11:
  003e: 70 12 f2 f1 70 47 24 f6 24 f1 71 23 f3 2f 4f 24
        f6 11 23 fb
                                 SB:ptr W[1] = (((LB(ptr W[2]+W[0])<<(W[0]&7)) ^
                                 W[1])&0xff)
  0052: 70 81 d0                 W[0] = (W[0])+1
```

```
  0055: 4c 70 f9                Push ((12 > W[0]) ? 1 : 0)
  0058: a2                      If A=0 Jump 0x5b <forloop_end> else Pop
  0059: 61 03                   Jump 0x3e <forloop_W[0]=0..11>

forloop_end:
  005b: 41 d0                   W[0] = 1
  005d: 11                      Push ptr W[1]
  005e: 24 f2                   Push 0x80000000
  0060: 76                      Push W[6]
  0061: 63 92                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  0063: 20 63 09                Jump 0x2f <mainloop>

---------------- [ TRANSPUTER 10] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 60 b0                   W += 4 * -0x10
  002d: 40 d3                   W[3] = 0
  002f: 40 d2                   W[2] = 0
  0031: 40 d0                   W[0] = 0

init_forloop1_W[0]=0..3:
  0033: 40 d1                   W[1] = 0

init_forloop2_W[1]=0..11:
  0035: 40 70 43 f8 14 fa 71 f2 23 fb
                                SB:(W[1]+(ptr W[4]+4*(3*W[0]))) = 0
  003f: 71 81 d1                W[1] = (W[1])+1
  0042: 4c 71 f9                Push ((12 > W[1]) ? 1 : 0)
  0045: a2                      If A=0 Jump 0x48 <init_forloop2_end> else Pop
  0046: 61 0d                   Jump 0x35 <init_forloop2_W[1]=0..11>

init_forloop2_end:
  0048: 70 81 d0                W[0] = (W[0])+1
  004b: 44 70 f9                Push ((4 > W[0]) ? 1 : 0)
  004e: a2                      If A=0 Jump 0x51 <mainloop> else Pop
  004f: 61 02                   Jump 0x33 <init_forloop1_W[0]=0..3>

mainloop:
  0051: 4c d0                   W[0] = 12
  0053: 72 43 f8 14 fa          Push (ptr W[4]+4*(3*W[2]))
  0058: 24 f2 54                Push 0x80000010
  005b: 21 71                   Push W[0x11]
  005d: 63 90                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  005f: 72 81                   Push (W[2])+1
  0061: 25 fa                   DUP (Push A)
  0063: d2                      W[2] = Pop
  0064: c4                      A = (A == 0x4) ? 1 : 0
  0065: a3                      If A=0 Jump 0x69 <endif> else Pop
  0066: 80                      A += 0x0
  0067: 40 d2                   W[2] = 0

endif:
  0069: 40 d1                   W[1] = 0
  006b: 40 d0                   W[0] = 0

forloop_W[0]=0..4:
```

```
  006d: 70 43 f8 14 fa f1 71 f2 2f 4f 24 f6 d1
                                W[1] = ((W[1]+LB(ptr W[4]+4*(3*W[0])))&0xff)
  007a: 70 81 d0                W[0] = (W[0])+1
  007d: 44 70 f9                Push ((4 > W[0]) ? 1 : 0)
  0080: a3                      If A=0 Jump 0x84 <forloop_end> else Pop
  0081: 80                      A += 0x0
  0082: 61 09                   Jump 0x6d <forloop_W[0]=0..4>

forloop_end:
  0084: 71 43 24 f6 43 f8 14 fa 71 44 24 f0 4c 21 ff 2f
        4f 24 f6 f2 f1 13 23 fb
                                SB:ptr W[3] = LB((((W[1]>>4)%12)&0xff)+(ptr
                                W[4]+4*(3*(W[1]&3))))
  009c: 41 d0                   W[0] = 1
  009e: 13                      Push ptr W[3]
  009f: 24 f2                   Push 0x80000000
  00a1: 21 71                   Push W[0x11]
  00a3: 67 90                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  00a5: 20 65 09                Jump 0x51 <mainloop>

---------------- [ TRANSPUTER 11] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 60 ba                   W += 4 * -0x6
  002d: 40 d1                   W[1] = 0
  002f: 40 d2                   W[2] = 0

mainloop:
  0031: 4c d0                   W[0] = 12
  0033: 13                      Push ptr W[3]
  0034: 24 f2 54                Push 0x80000010
  0037: 77                      Push W[7]
  0038: 61 95                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  003a: 40 11 23 fb             SB:ptr W[1] = 0
  003e: 13 f1 13 83 f1 23 f3 13 87 f1 23 f3 2f 4f 24 f6
        11 23 fb
                                SB:ptr W[1] = (((LB(ptr W[3]) ^ LB((ptr W[3])+3)) ^
                                LB((ptr W[3])+7))&0xff)
  0051: 41 d0                   W[0] = 1
  0053: 11                      Push ptr W[1]
  0054: 24 f2 51                Push 0x80000004
  0057: 77                      Push W[7]
  0058: 63 9b                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  005a: 41 d0                   W[0] = 1
  005c: 11                      Push ptr W[1]
  005d: 24 f2 55                Push 0x80000014
  0060: 77                      Push W[7]
  0061: 64 9c                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  0063: 11 f1 4c 21 ff 2f 4f 24 f6 11 23 fb
                                SB:ptr W[1] = ((LB(ptr W[1])%12)&0xff)
  006f: 11 f1 13 f2 f1 12 23 fb  SB:ptr W[2] = LB(ptr W[3]+LB(ptr W[1]))
  0077: 41 d0                   W[0] = 1
  0079: 12                      Push ptr W[2]
  007a: 24 f2                   Push 0x80000000
  007c: 77                      Push W[7]
  007d: 65 96                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
```

```
  007f: 20 65 0f                Jump 0x31 <mainloop>

---------------- [ TRANSPUTER 12] ----------------

recv(chan=B, mem=C, len=W[0]):
  001f: 73 72 74 f7             IN(addr=W[3], chan=W[2], len=W[4])
  0023: 22 f0                   Return

send(chan=B, mem=C, len=W[0]):
  0025: 73 72 74 fb             OUT(addr=W[3], chan=W[2], len=W[4])
  0029: 22 f0                   Return

entry:
  002b: 60 ba                   W += 4 * -0x6
  002d: 40 d2                   W[2] = 0
  002f: 40 d1                   W[1] = 0
  0031: 40 d0                   W[0] = 0

fill_W[3,4,5]_with_zeros:
  0033: 40 70 13 f2 23 fb       SB:(ptr W[3]+W[0]) = 0
  0039: 70 81 d0                W[0] = (W[0])+1
  003c: 4c 70 f9                Push ((12 > W[0]) ? 1 : 0)
  003f: a2                      If A=0 Jump 0x42 <mainloop> else Pop
  0040: 60 01                   Jump 0x33 <fill_W[3,4,5]_with_zeros>

mainloop:
  0042: 40 11 23 fb             SB:ptr W[1] = 0
  0046: 13 81 f1 13 85 f1 23 f3 13 89 f1 23 f3 2f 4f 24
        f6 11 23 fb
                                SB:ptr W[1] = (((LB((ptr W[3])+1) ^ LB((ptr W[3])+5))
                                ^ LB((ptr W[3])+9))&0xff)
  005a: 4c d0                   W[0] = 12
  005c: 13                      Push ptr W[3]
  005d: 24 f2 54                Push 0x80000010
  0060: 77                      Push W[7]
  0061: 64 9c                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  0063: 41 d0                   W[0] = 1
  0065: 12                      Push ptr W[2]
  0066: 24 f2 55                Push 0x80000014
  0069: 77                      Push W[7]
  006a: 64 93                   Call 0x1f <recv(chan=B, mem=C, len=W[0])>
  006c: 41 d0                   W[0] = 1
  006e: 11                      Push ptr W[1]
  006f: 24 f2 51                Push 0x80000004
  0072: 77                      Push W[7]
  0073: 64 90                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  0075: 12 f1 4c 21 ff 2f 4f 24 f6 12 23 fb
                                SB:ptr W[2] = ((LB(ptr W[2])%12)&0xff)
  0081: 12 f1 13 f2 f1 11 23 fb  SB:ptr W[1] = LB(ptr W[3]+LB(ptr W[2]))
  0089: 41 d0                   W[0] = 1
  008b: 11                      Push ptr W[1]
  008c: 24 f2                   Push 0x80000000
  008e: 77                      Push W[7]
  008f: 66 94                   Call 0x25 <send(chan=B, mem=C, len=W[0])>
  0091: 20 65 0e                Jump 0x42 <mainloop>
```