Lettres à Anselme

Solution épistolaire au challenge 2015 du SSTIC

Tristan Dubois
<tristan.dubois@etu.univ-lyon1.fr>
<drustan.trybois@me.com>

15 avril 2015

L'auteur de ces lignes tient à remercier M. Philippe-Anselme Foury pour le prêt de son nom, de sa boîte aux lettres et de son temps (pour la relecture et la rédaction de l'épilogue de cette solution). Par ailleurs, il précise qu'Anselme n'a daigné répondre à aucune de ses missives pendant la durée du challenge (c'est-à-dire du samedi 4 au vendredi 10) et n'y a donc participé en aucune manière. Il ne lui tient pas rigueur de son silence.

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le samedi 4 avril

Cher Anselme,

Le silence. Votre exil marseillais nous l'impose depuis de trop longues semaines et, aujourd'hui, il cessera par cette lettre (et ses suites). J'espère que votre stage de fin d'études se déroule paisiblement et que vous êtes bien installé... car je compte débarquer tantôt dans le Vieux-Port. Mais, auparavant, je dois solder quelques affaires et, surtout, finir le challenge 2015 du SSTIC : après le faux-départ de mercredi, figure-toi qu'il est enfin sorti hier soir !

La première étape du challenge consiste à analyser une image disque FAT32 de 128 Mo appelée *sdcard.img*. Cette partition contient apparemment un seul fichier nommé *inject.bin* (la commande *file* ne nous apporte aucune information sur son format et un coup d'œil avec *hexdump* montre simplement que ses octets sont loin de suivre une répartition uniforme).

Vu que les organisateurs nous fournissent une image disque au lieu de nous donner une archive, voire directement le fichier *inject.bin*, je me suis nécessairement dit qu'il y a quelque chose de plus, comme un fichier supprimé dont le contenu n'aurait pas été écrasé...

Sache que mon intuition ne m'a nullement trahi : il y avait bien dans le répertoire racine un fichier *build.sh* ! Ce script comportait comme unique commande `java -jar encoder.jar -i /tmp/duckyscript.txt`. J'en présume que *inject.bin* a été engendré à partir de *duckyscript.txt* à l'aide de ce programme Java *encoder.jar*.

```
0003d200  e5 62 00 75 00 69 00 6c  00 64 00 0f 00 bd 2e 00  |.b.u.i.l.d......|
0003d210  73 00 68 00 00 00 ff ff  ff ff 00 00 ff ff ff ff  |s.h.............|
0003d220  e5 55 49 4c 44 20 20 20  53 48 20 20 00 00 2d 1e  |.UILD   SH  ..-.|
0003d230  7a 46 7a 46 00 00 2d 1e  7a 46 03 00 2d 00 00 00  |zFzF..-.zF..-...|
0003d240  41 69 00 6e 00 6a 00 65  00 63 00 0f 00 e4 74 00  |Ai.n.j.e.c....t.|
0003d250  2e 00 62 00 69 00 6e 00  00 00 00 00 ff ff ff ff  |..b.i.n.........|
0003d260  49 4e 4a 45 43 54 20 20  42 49 4e 20 00 00 3a 1e  |INJECT  BIN ..:.|
0003d270  7a 46 7a 46 00 00 3a 1e  7a 46 04 00 a2 ab 0a 02  |zFzF..:.zF......|
0003d280  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00041a00  6a 61 76 61 20 2d 6a 61  72 20 65 6e 63 6f 64 65  |java -jar encode|
00041a10  72 2e 6a 61 72 20 2d 69  20 2f 74 6d 70 2f 64 75  |r.jar -i /tmp/du|
00041a20  63 6b 79 73 63 72 69 70  74 2e 74 78 74 00 00 00  |ckyscript.txt...|
00041a30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

FIGURE 1 – Le répertoire racine et le contenu du *build.sh* supprimé

En faisant une recherche Google avec ces mots-clef, je tombe sur ce dépôt Github : `https://github.com/hak5darren/USB-Rubber-Ducky`... Il contient le code source du fameux logiciel *encoder.jar* ! Il s'agit d'une sorte de compilateur pour un dispositif appelé *USB Rubber Ducky* ; ce truc simule un clavier USB et exécute une frappe selon un schéma prédéfini – le fameux *inject.bin*.

Pour savoir ce que cet engin de malheur a bien pu saisir, il faut décoder *inject.bin* afin de récupérer *duckyscript.txt*. Avec le code de l'encodeur, c'est plutôt simple. Même si j'en ai un peu honte, je te joins quand même le code source de mon décodeur *decode.py*...

Après décodage, on découvre que, grosso modo, le *Ducky* fait Windows+R (ce qui ouvre la fenêtre *Exécuter*), tape `cmd` (le nom de l'interpréteur de commandes), appuie sur Entrée et entre une série de commandes débutant toutes par `powershell -enc`. Chacune de ces lignes est en fait un script PowerShell écrit en UTF-16 et codé en Base64.

```
DELAY 2000
WINDOWS r
DELAY 500
ENTER
DELAY 1000
STRING cmd
ENTER
DELAY 50
DELAY 10
STRING powershell -enc
ZgB1AG4AYwB0AGkAbwBuACAAdwByAGkAdABlAF8AZgBpAGwAZQBfAGIAeQB0AGUAcwB7AHAAYQByAGEAbQAoAFsAQgB5AHQAZQBbAF0AXQAgACQAZgBpAGw
AZQBfAGIAeQB0AGUAcwAsACAAWwBzAHQAcgBpAG4AZwBdACAAJABmAGkAbABlAF8AcABhAHQAaAAgAD0AIAAiAC4AXABzAHQAYQBnAGUAMgAuAHoAaQBwAC
IAKQA7ACQAZgAgAD0AIABbAGkAbwAuAGYAaQBsAGUAXQA6ADoATwBwAGUAbgBXAHIAaQB0AGUAKAAkAGYAaQBsAGUAXwBwAGEAdABoACkAOwAkAGYALgBTA
```

FIGURE 2 – Le début du script exécuté par le *Ducky*

Chacun de ces scripts (sauf le dernier) suit le même format : ils vérifient l'environnement puis ajoutent des données à la fin d'un fichier nommé *stage2.zip*, éventuellement en le créant. Si l'environnement est invalide, le script écrit `TryHarder` en UTF-16 en lieu et place des données attendues. Le dernier script vérifie que le SHA-1 de *stage2.zip* est bien `EA-9B-8A-6F-5B-52-7E-72-65-20-19-31-3C-25-B5-6A-D2-7C-7E-C6` et indique si le fichier a bien été reconstitué ou non (la reconstitution peut échouer si l'environnement est incorrect).

```
3389  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('KYDyYYpjcrGFgNbHRJvpBgWwB
      rnbxfI2qAaNkiSR0IZa3DnUil1PPAJI24hg7QCr7fd9pifk/wykkxSnoxjJswTNoV1yzYKegpzmC1qd++8GPeEC5Iss5MznqMb
      yFIOTPFLb2w0btstp5qA5zNTQ6T4Vw44jfevsjwMRjWMZDoJDYlu72mCcnMZCJzkTia0nJmCY7i2xZpKFagJ0SB7nBbH2WcfVv
      VhGC4gKVbzJ69Zn4yoMM96khpmqaEs+I1pgiH8IEtaiwhkvAWJja2PeK/8fAWqnhcFrAfxxoDBszKpB8NM/
      UEsBAj8DCgMAAAAANop5Rt9L9d0QpQcAEKUHAAkAAAAAAAAAAAAggKSBAAAAAGVuY3J5cHRlZFBLAQI/
      AxQDAAAIADaKeUbTHuB59QAAAEABAAAIAAAAAAAAAAAAAIICkgTelBwBtZW1vLnR4dFBLAQI/AxQDAAAIAMJSckZ4QV7Va0otAK
      bALQAJAAAAAAAAAAAAIID2gVKmBwBzc3RpYy5wazNQSwUGAAAAAAMAAwCkAAAA5PA0AAAA'));}else{write_file_bytes([
      Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));}
3390  function hash_file{param([string] $filepath);$sha1 = New-Object -TypeName System.Security.
      Cryptography.SHA1CryptoServiceProvider;$h = [System.BitConverter]::ToString($sha1.ComputeHash([
      System.IO.File]::ReadAllBytes($filepath)));$h}$h = hash_file(".\stage2.zip");if($h -eq
      "EA-9B-8A-6F-5B-52-7E-72-65-20-19-31-3C-25-B5-6A-D2-7C-7E-C6"){echo "You WIN";}else{echo "You
      LOSE";}
```

FIGURE 3 – Le code des deux derniers scripts PowerShell

À l'heure où j'écris ces lignes, je viens de construire *stage2.zip*. Il s'agit effectivement d'un fichier PKZIP, comme son extension le suggérait. Cette archive contient trois fichiers : *encrypted* (un fichier chiffré), *memo.txt* (des instructions pour le déchiffrement) et *sstic.pk3* (visiblement une carte personnalisée pour un Quake-like). Mais la nuit est déjà trop sombre et cette lettre déjà trop longue : je te parlerai demain de ce *stage2*.

Amicalement, Tristan.

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le dimanche 5 avril

Cher Anselme,

Si mes courriers te sont arrivés dans le même ordre que je les ai envoyés, tu as déjà dû lire le récit de mes péripéties sur la première étape du challenge du SSTIC. Dans cette nouvelle missive, je te raconterai mes fabuleuses aventures dans le monde merveilleux de la deuxième épreuve (*stage2*) du challenge.

Comme je te l'écrivais hier, l'archive *stage2.zip* comporte trois fichiers : *encrypted* (un fichier chiffré), *memo.txt* (des instructions pour déchiffrer *encrypted*... mais sans l'élément le plus important : la clef) et *sstic.pk3* (une carte *custom* pour un Quake-like et, accessoirement, une archive PKZIP). Le *memo* indique que *encrypted* a été chiffré en AES-OFB avec un certain vecteur d'initialisation fourni. Ni la clef ni sa taille ne sont indiquées.

J'ai pu déterminer que cette carte était pour OpenArena, un FPS libre et multiplateforme. Pour le comprendre, il suffisait de décompresser `sstic.pk3` et de lire la première phrase du *README* qu'elle contient : `Copy the pk3 in your baseoa directory`. Une simple recherche sur Google nous apprend rapidement que `baseoa` est le répertoire de base de OpenArena.

D'après *memo.txt*, l'auteur *found a way to hide* [the key] *into* [his] *favorite game*. Le but du jeu est donc de récupérer dans la carte *sstic.pk3* la clef nécessaire pour déchiffrer le fichier *encrypted*. Depuis le jeu, cette carte peut être chargée, si elle a bien été copiée dans `baseoa`, en tapant `/devmap sstic` dans la console (qui est accessible en faisant Majuscule+Échap). Puis, si l'on est tricheur (et tu sais que je le suis), on peut aussi ajouter la commande `/noclip` qui active le *noclip mode* (`http://en.wikipedia.org/wiki/Noclip_mode`).

Parlons maintenant de la carte. Elle se déroule dans une sorte d'entrepôt avec des caisses en bois, des plateformes métalliques et un super-calculateur grillagé. Premier détail intriguant, il y a, disséminés dans la carte, huit panneaux comportant chacun un symbole et trois inscriptions de huit chiffres hexadécimaux coloriées différemment (une inscription en orange, une en blanc, une en vert). Second détail intéressant, il y a, dans une salle spéciale et cachée, une suite de huit associations (symbole, couleur).

Partant de là, ce qu'il faut faire semble évident : il faut remplacer chaque association (symbole, couleur) de la suite par l'inscription de cette couleur sur le panneau portant ce symbole. On arrive alors à une clef de $8*8*4 = 256$ bits : `9e2f31f7 8153296b 3d9b0ba6 7695dc7c b0daf152 b54cdc34 ffe0d355 26609fac`. Je te fournis la commande permettant de déchiffrer *encrypted* en utilisant cette clef :

```
openssl aes-256-ofb -d -in encrypted -out decrypted -K
    ↪ 9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac
    ↪ -iv 5353544943323031352d537461676532
```

Pour être exhaustif, sache que le fichier *decrypted*, qui est – encore – une archive PKZIP, comporte 16 octets de *padding*, qu'il n'est pas nécessaire de retirer pour le décompresser. Voilà. C'en est fini de cette expédition (et de cette lettre)... mais je t'écrirai dès demain pour monologuer à nouveau sur la troisième étape du challenge !

Amicalement, Tristan.

FIGURE 4 – L'un des huits panneaux (celui avec le symbole *drapeau*)

FIGURE 5 – La suite de huit associations (symbole, couleur)

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le lundi 6 avril

Cher Anselme,

Chose promise, chose dûe : voici donc mon monologue pour la résolution de la troisième étape (*stage3*) du challenge du SSTIC ! Soit dit en passant, si je réussis à arriver jusqu'à la fin du challenge (ce qui est loin d'être certain), tous ces courriers que je t'envoie pourraient très bien faire office de solution écrite : j'espère donc que tu les conserveras précieusement. Bref. Fin de la parenthèse. Nous nous en étions arrêtés à *decrypted*, qui se trouvait être une archive PKZIP.

Cette archive contient trois fichiers : *encrypted* (un fichier chiffré – bis repetita), *memo.txt* (des instructions pour le déchiffrement de *encrypted*, mais sans la clef – bis repetita) et *paint.cap* (une capture *pcap*, lisible avec tcpdump ou Wireshark par exemple). Dans le *memo*, l'auteur prétend, en substance, avoir dessiné en toute sécurité la clef dans Paint. Malheureusement, nous ne disposons pas de l'image qui a résulté de son génie artistique... mais nous avons une capture !

Car, vois-tu, cher ami, *paint.cap* contient en fait le trafic USB échangé entre une machine et une souris, vraisemblablement lors du fameux dessin – soi-disant sécurisé – de la clef. J'ai donc écrit un modeste script, *picasso.py*, qui permet de retrouver ce qui a été dessiné à partir des évènements souris. Tu trouveras son code source en pièce-jointe (pense à installer PIL si tu veux l'essayer). Et le résultat est :

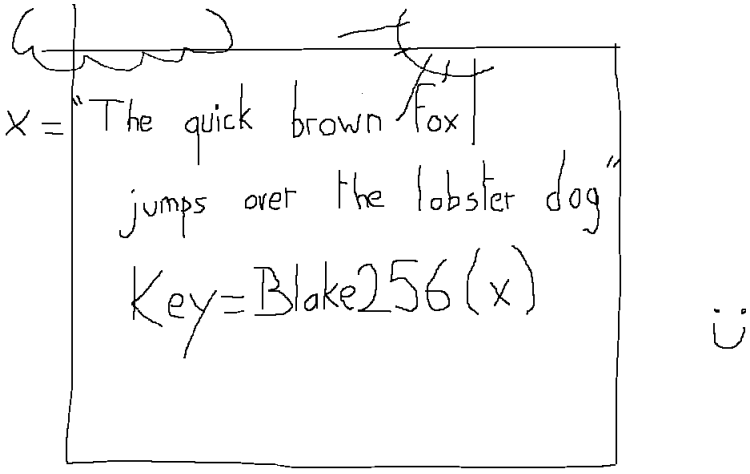

FIGURE 6 – La clef dessinée en toute sécurité dans Paint

Une implémentation en Python de Blake256 peut être trouvée sur ce site : `http://www.seanet.com/~bugbee/crypto/blake/`. Tu pourras toutefois t'épargner la peine de la télécharger puisque je t'informe que le condensat cryptographique de *The quick*

brown fox jumps over the lobster dog par Blake256 vaut `66c1ba5e 8ca29a8a b6c105a9 be9e75fe 0ba07997 a839ffea e9700b00 b7269c8d`. Reste donc à déchiffrer.

D'après le *memo*, le chiffrement utilisé est *Serpent-1-CBC-With-CTS*. Malheureusement, la version de OpenSSL installé sur mon Mac ne le gère pas et je n'ai pas trouvé cette fois-ci d'implémentation en Python. En désespoir de cause, j'ai codé un petit programme en C, *decrypter.c*, qui utilise la bibliothèque *gcrypt* (qui, elle, a tout ce qu'il faut). Comme d'habitude, je te joins le code source de mon programme.

Au final, on obtient – encore – une archive PKZIP. Mais Morphée m'appelle et me pousse à lâcher ma plume. Je te laisse et t'écrirai la suite demain.

Amicalement, Tristan.

PS : Au cas où tu te le demanderais, les évènements souris sont constitués de quatre octets : le premier est un champ de bits indiquant quels boutons ont été pressés, le deuxième est un nombre signé indiquant le déplacement **horizontal** (i.e. axe X), le troisième est un nombre signé indiquant le déplacement **vertical** (i.e. axe Y) et, enfin, le quatrième est un octet de remplissage, toujours nul. Plus d'informations dans la section B.2 de `http://www.usb.org/developers/hidpage/HID1_11.pdf`.

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le mardi 7 avril

Cher Anselme,

À titre liminaire, j'aimerais te demander pourquoi tu as installé Xorg sur Gideon, notre serveur dédié. Cela me semble incongru puisque, d'une part, il n'est connecté à aucun écran et, d'autre part, il traîne dans un datacenter inconnu en Allemagne. En l'absence de réponse d'ici la fin de la semaine, je me permetterai donc de désinstaller ce serveur X11 inutile.

Cela étant dit, parlons de la quatrième étape du challenge du SSTIC (*stage4*). L'archive déchiffrée hier contient un unique fichier *stage4.html*. Cette page HTML comporte principalement un script JavaScript constitué d'une variable `data` (vraisemblablement les données à déchiffrer), d'une variable `hash` (vraisemblablement le hash des données déchiffrées) et de code obscurifié.

La première étape d'obscurification construit un dictionnaire – on parle d'*objet* en vocabulaire JavaScript – nommé `$` et l'utilise pour décoder le reste du programme. Le tableau ci-dessous indique le contenu de ce dictionnaire, tel que je l'ai reconstitué :

clef	**valeur**
`$`	constructeur `Function`
`$$`	`"return"`
`$$$`	`7`
`$$$$`	`"f"`
`$$$_`	`"e"`
`$$_`	`6`
`$$_$`	`"d"`
`$$__`	`"c"`
`$_`	`"constructor"`
`$_$`	`5`
`$_$$`	`"b"`
`$_$_`	`"a"`
`$__`	`4`
`$__$`	`9`
`$___`	`8`
`_`	`"u"`
`_$`	`"o"`
`_$$`	`3`
`_$_`	`2`
`__`	`"t"`
`__$`	`1`
`___`	`0`

Le reste du programme est de la forme `$.$($.$(`*chaine*`)())()` avec *chaine* une chaîne de caractères (formée par concaténation de littéraux et d'éléments du dictionnaire) et `$.$` le constructeur `Function`. La chaîne de caractères *chaine* est elle-même de la forme `"return\"`*sous-chaine*`\""` avec *sous-chaine* une chaîne de caractères incluse dans *chaine*. Elle contient en fait le code qui va être exécuté par le navigateur et est elle-même obscurifiée.

Ci-dessous se trouve une désobscurification à la main (si si, avec un papier et un crayon) de *sous-chaine*, après avoir remplacé `$` par `a` pour des raisons de lisibilité. La première partie est le dictionnaire utilisé par cette étape d'obscurification et la seconde partie correspond au code (dés)obscurifié en lui-même.

```
// Dictionnaire
// ============
__=document;
aaa='stage5';
aa_a='load';
a_aa=' ';
_aaaaa='user';
_aaa='div';
aa_aaa='navigator';
aa_aa='preferences';
a_aaa='to';
aaa_a='href';
aaaa_='=';
aaaaa='chrome';
_aaaa='"';
a_aaaa='Agent';
aaa_aa='down';
aaaa_a='import';
a='<b>Failed to load stage5</b>';
___='write';
____='getElementById';
_a_ = "raw";
aa=window;
__a=window.crypto.subtle;
__a_='decrypt';
___a='status';
a____='importKey';
_______=0;
__a__='then';
_a____='digest';
__a___='innerHTML';
___a__={name:'SHA-1'};
____a_=data;
_____a=hash;
_a_____=Blob;
___a___=URL;
____a__='createObjectURL';
______a='type';
a_______='application/octet-stream';
_a______='name';
__a_____='AES-CBC';
___a____='iv';
____a___='<a href="';
_____a__='" download="stage5.zip">download stage5</a>';
______a_='(';
_______a=')';
a________='setTimeout';
_________=parseInt;
__________=window.navigator.userAgent;
_____________='length';
_____________='substr';
______________=1;
_______________=2;
________________=8;
_________________=16;
a_a=1000;
__________________='indexOf';
___________________='charCodeAt';
____________________='push';
_______________________=Uint8Array;
________________________='';
__________________________='byteLength';
___________________________='toString';
```

```
// Code (des)obscurifie
// ====================

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>');

// Transforme une chaine de caracteres en tableau de Uint8.
function f1(arg) {
    res=[];
    for (i=0;i<arg.length;++i)
        res.push(arg.charCodeAt(i));
    return new Uint8Array(res);
}

// Transforme une chaine de chiffres hexadecimaux (comme la variable 'data') en tableau de Uint8.
function f2(arg){
    res=[];
    for (i=0;i<arg.length/2;++i)
        res.push(parseInt(arg.substr(i*2,2),16));
    return new Uint8Array(Uint8Array);
}

// Transforme un tableau de Uint8 en une chaine de chiffres hexadecimaux (comme la variable 'hash').
function f3(arg) {
    res='';
    for (i=0;i<arg.byteLength;++i) {
        tmp=arg[i].toString(16);
        if(tmp.length<2)
            res+='0';
        res+=tmp;
    }
    return res;
}

// Fonction principale : dechiffre 'data', verifie que son condensat est le meme que 'hash'.
function f4() {
    parenthesisStart=f1(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf('(')+1,16));
    parenthesisEnd=f1(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf(')')-16,16));
    cipher_characteristics = {};
    cipher_characteristics.name = 'AES-CBC';
    cipher_characteristics.iv = parenthesisStart;
    cipher_characteristics.length = parenthesisEnd.length*8;
    window.crypto.subtle.importKey('raw', parenthesisEnd, cipher_characteristics, false,
        ↪ ['decrypt']).then(function (arg) {
        window.crypto.subtle.decrypt(cipher_characteristics, arg, f2(data)).then(function (aarg) {
            plaintext = new Uint8Array(aarg);
            window.crypto.subtle.digest({name:'SHA-1'}, plaintext).then(function (aaarg) {
                if (hash == f3(new Uint8Array(aaarg))) {
                    blob_characteristics={};
                    blob_characteristics.type = 'application/octet-stream';
                    blob=new Blob([plaintext],blob_characteristics);
                    url = URL.createObjectURL(blob);
                    document.getElementById('status').innerHTML = '<a href="'+url+'"
                        ↪ download="stage5.zip">download stage5</a>';
                } else {
                    document.getElementById('status').innerHTML = '<b>Failed to load stage5</b>';
                }
            });
        }).catch(function () { document.getElementById('status').innerHTML='<b>Failed to load stage5</b>';
            ↪ });
    }).catch(function () { document.getElementById('status').innerHTML='<b>Failed to load stage5</b>'; });
}

window.setTimeout(f4, 1000);
```

Grosso modo, le code tente de déchiffrer le contenu de la variable `data` avec AES-CBC en utilisant les 16 premiers octets de la première parenthèse de l'User-Agent (UA) du navigateur courant comme vecteur d'initialisation (IV) et les 16 derniers octets de cette même parenthèse comme clef. Puis, il vérifie que le SHA-1 des données déchiffrées correspond bien à la valeur contenue dans la variable `hash`.

Nous devons donc, pour déchiffrer `data`, déterminer quel est l'UA attendu. À cause du lien vers `chrome://browser/content/preferences/preferences.xul`, nous savons qu'il s'agit de Firefox (le XUL étant un langage spécifique à Mozilla). Par ailleurs,

l'API *SubtleCrypto* n'a été implémenté dans ce butineur qu'à partir de la version 33, comme indiqué sur le wiki de Mozilla : `https://developer.mozilla.org/en-US/docs/Web/API/Crypto/subtle`. Enfin, le format de l'UA de Firefox est décrit ici (aussi dans le wiki de Mozilla) : `https://developer.mozilla.org/en-US/docs/Web/HTTP/Gecko_user_agent_string_reference`.

À partir de ces informations, il est possible de construire l'ensemble des premières parenthèses possibles puis de procéder à une *attaque par dictionnaire*. À cet effet, j'ai bien évidemment codé un petit script, *dictattack.py*, que je joins à ce courrier. Au bout de quelques secondes, il trouve que la première parenthèse de l'UA attendu est `Macintosh; Intel Mac OS X 10.6; rv:35.0`. Le fichier déchiffré, *stage5.zip*, se révèle être, comme son extension le suggère, une archive PKZIP.

Voilà, Anselme, où la quatrième étape nous a menés : à la lisière de la cinquième étape (logique, me diras-tu). Toutefois, le ciel est désormais trop sombre pour que j'ose m'y aventurer aujourd'hui. Le soleil reviendra demain et, aux premières lueurs, je jetterai à nouveau l'ancre sur l'île du *stage5.zip* et l'encre sur le papier à lettre.

Amicalement, Tristan.

PS : À bien me relire, tout cela ne me paraît pas clair. En particulier le tout début. En fait, le code de la première partie d'obscurification correspond à quelque chose de la forme `Function(Function("return \"quelque chose\"")())();`. Ce qui se passe, c'est que `Function("return \"quelque chose\"")()` crée une fonction qui fait `return "quelque chose";` et l'exécute (un peu comme un `eval`) : au final, cette expression s'évalue donc à `"quelque chose"`. Et, du coup, il reste `Function("quelque chose")()`, qui exécute le code `"quelque chose"`. Le `"return \"quelque chose\""`, c'est ce que j'appelle *chaine*. Le `"quelque chose"`, c'est ce que j'appelle *sous-chaine*. C'est bien *sous-chaine* qui est finalement exécuté par le navigateur.

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le mercredi 8 avril

Cher Anselme,

Comme promis hier, je suis parti, dès l'aube, à l'abordage de la cinquième étape du challenge du SSTIC, autrement dit de l'archive *stage5.zip*. Celle-ci contient les deux principaux ingrédients d'une bonne histoire de pirate : un butin mystérieux (*input.bin*) et une carte au trésor (*schematic.pdf*). Pour le coup, *input.bin* est réellement énigmatique : *file* ne peut rien en dire de plus que `input.bin: data`... Tu trouveras ci-contre le schéma (une aide bien utile pour s'y retrouver dans le dédale des *transputers*).

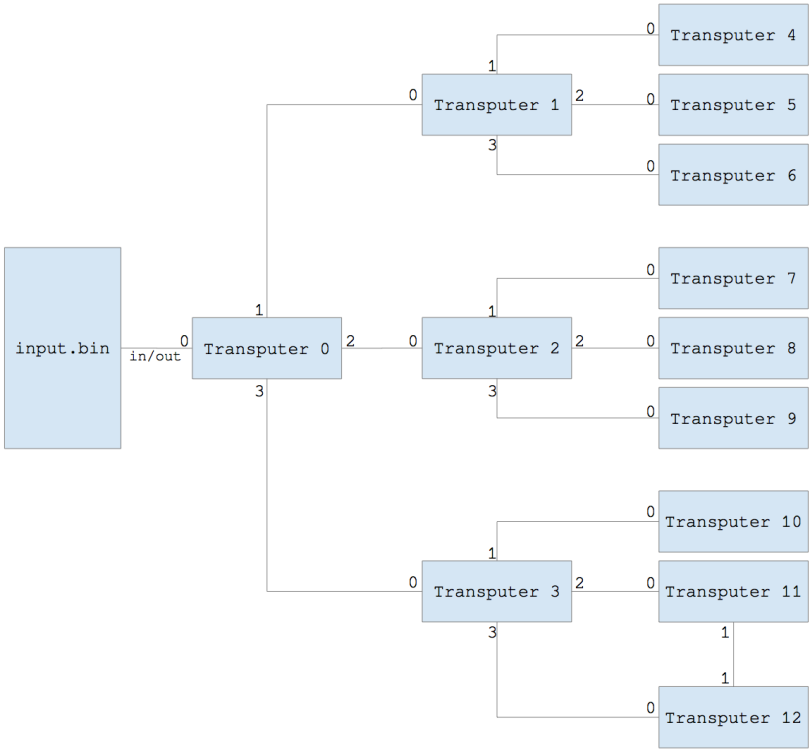

FIGURE 7 – Plan des lieux

Le fichier *input.bin* contient des chaînes de caractères (révélées par *strings*) comme `Boot ok`, `Code Ok`, `Decrypt`, `KEY:` ou `congratulations.tar.bz2`. On peut raisonnablement supposer que cette dernière chaîne indique le nom du fichier qu'il faudra obtenir à la fin de cette étape : il s'agira donc vraisemblablement d'une archive TAR compressée au format *bzip2*. Mon avis ? *input.bin* contient du code exécutable (mais pas que)... sûrement pour une architecture exotique (*transputer* ?). Désassemblons tout ça !

Première étape : trouver le jeu d'instructions. L'auteur, dans sa grande mansuétude, nous glisse un indice : `I love ST20 architecture` (c'est écrit dans *schematic.pdf*). En fait, il existe deux grandes variantes de cette architecture : le ST20-C1 et le ST20-C2/C4. Si les *instructions primaires* sont les mêmes, les *instructions secondaires* changent. Après quelques décodages à la main avec la documentation d'un côté et *hexdump* de l'autre, j'ai acquis la conviction qu'il s'agit de code exécutable pour le ST20-C2/C4. Tu trouveras le manuel de référence ici (entre autres) : `http://pdf.datasheetcatalog.com/datasheet/SGSThomsonMicroelectronics/mXruvtu.pdf`.

Avec ces informations, j'ai pu écrire une sorte de désassembleur, *interpret.py*, dont je joins le code source et qui fonctionne en fait par *interprétation symbolique* (d'où son nom). Tu trouveras sur la figure 9 un exemple de sortie fournie par ce petit module Python. Note que les prochains codes que je t'enverrai seront directement transcrits en (simili-)C car – tu en conviendras – le pseudo-code émis par *interpret.py* est aussi utile qu'illisible.

Seconde étape : trouver comment est encapsulée le programme. Il est rare, en effet, que l'on trouve du code exécutable brute : même les vieux `a.out` des premiers UNIX avaient des octets d'en-tête ! Le schéma nous indique que les données de *input.bin* sont transmises au premier *transputer* par son port 0, la question se résume donc à savoir quel est le protocole attendu par le ST20 lorsqu'il démarre depuis le réseau. J'ai trouvé la réponse dans le datasheet du ST20-GP1. Pour résumer : le premier octet lu correspond à la longueur du programme qui sera chargé ensuite depuis le canal et l'exécution commence au tout début du code reçu (i.e. pas de point d'entrée bizarre).

> **11.2.2 Booting from link**
>
> When booting from a link, the ST20-GP1 will wait for the first bootstrap message to arrive on the link. The first byte received down the link is the control byte. If the control byte is greater than 1 (i.e. 2 to 255), it is taken as the length in bytes of the boot code to be loaded down the link. The bytes following the control byte are then placed in internal memory starting at location MemStart. Following reception of the last byte the ST20-GP1 will start executing code at MemStart. The memory space immediately above the loaded code is used as work space. A byte arriving on the bootstrapping link after the last bootstrap byte, is retained and no acknowledge is sent until a process inputs from the link.

FIGURE 8 – Extrait du manuel du ST20-GP1

On peut désormais extraire de *input.bin* le programme qui sera exécuté sur le premier *transputer* : il mesure 248 octets (valeur du premier byte, cf. *supra*) et correspond au programme C transcrit à la figure 10. À gros traits, il fait ceci :

Première phase il retransmet les messages qui lui sont envoyées au canal indiqué ;

Seconde phase il déchiffre les octets qui lui sont envoyés en les XORant avec le masque $i + 2 * key_i$ (avec i l'indice *module* 12 de l'octet courant) puis met-à-jour le i-ème octet de la clef avec le XOR des octets remontés par chacun des *transputer* après leur avoir envoyés l'état courant de la clef.

```
; == ZONE [0;17[ ==
[W-300] <-- 0
[W-292] <-- 0
send(0x80000000, 220, 8) ; "Boot ok\0"

; == ZONE [17;22[ ==
recv(0x80000010, W+292, 12)
cj to 37 if [tag0:W+292] is null

; == ZONE [22;37[ ==
recv(0x80000010, 243, [tag1:W+292])
send([tag2:W+296], 243, [tag3:W+292])
j to 17

; == ZONE [37;77[ ==
send(0x80000004, W+292, 12)
send(0x80000008, W+292, 12)
send(0x8000000c, W+292, 12)
send(0x80000000, 228, 8) ; "Code Ok\0"
recv(0x80000010, W+8, 4) ; "KEY:"
recv(0x80000010, W+20, 12)
send(0x80000000, 236, 8) ; "Decrypt?"
recv(0x80000010, W+12, 1) ; 23 (i.e. la taille de "congratulations.tar.bz2")
recv(0x80000010, W+36, [[tag4:W+12]]) ; "congratulations.tar.bz2"
[W+16] <-- 0

; == ZONE [77;d2[ ==
recv(0x80000010, W+4, 1)
send(0x80000004, W+20, 12)
send(0x80000008, W+20, 12)
send(0x8000000c, W+20, 12)
recv(0x80000014, W+1, 1)
recv(0x80000018, W+2, 1)
recv(0x8000001c, W+3, 1)
[[W+1]] <-- ([[tag7:W+3]]^[[tag6:W+2]]^[[tag5:W+1]])&255
[[W]] <-- ([tag11:W+16]+2*[[tag10:W+20+[tag9:W+16]]]^[[tag8:W+4]])&255
[[W+20+[tag12:W+16]]] <-- ([[tag7:W+3]]^[[tag6:W+2]]^[[tag5:W+1]])&255
[W+16] <-- [tag13:W+16]+1
cj to d5 if [tag13:W+16]+1==12 is null

; == ZONE [d2;d5[ ==
[W+16] <-- 0

; == ZONE [d5;dc[ ==
send(0x80000000, W, 1)
j to 77
```

FIGURE 9 – Désassemblage par *interpret.py* du code exécuté par le premier *transputer*

```
// Format de l'entête des messages.
struct header {
    uint32_t size; // Taille du message.
    uint32_t channel; // Destinataire du message.
    uint32_t offset; // Point d'entrée du message.
}

// Indique à l'utilisateur que le premier transputer a bien démarré.
send(CHANNEL_0, "Boot␣ok", 8);

// Boucle de transmission des messages vers les transputers fils.
struct header header;
char message[UNKNOW];
while (1) {
    recv(CHANNEL_0, &header, 12);
    if (header.size == 0)
        break;
    recv(CHANNEL_0, message, header.size);
    send(header.channel, message, header.size);
}

// Termine les boucles de transmission des messages des transputers fils.
assert(header.size == 0);
send(CHANNEL_1, &header, 12);
send(CHANNEL_2, &header, 12);
send(CHANNEL_3, &header, 12);

// Indique à l'utilisateur que les transputers ont bien été configurés.
send(CHANNEL_0, "Code␣Ok", 8);

// Reçois la clef de chiffrement.
char keyMarker[4];
char key[12];
recv(CHANNEL_0, keyMarker, 4); // "KEY:"
recv(CHANNEL_0, key, 12); // Par défaut, ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff

// Reçois le nom du fichier chiffré.
uint8_t filenameSize;
char filename[UNKNOW];
send(CHANNEL_0, "Decrypt", 8);
recv(CHANNEL_0, filenameSize, 1);
recv(CHANNEL_0, filename, filenameSize);

// Boucle de (dé)chiffrement.
for (int i = 0; true; i = (i + 1) % 12) {
    // Chiffrement par flot synchrone à addition binaire (i.e. à XOR).
    recv(CHANNEL_0, cipherByte, 1);
    maskByte = i + 2*key[i];
    plainByte = cipherByte ^ maskByte;

    // Modification de l'état interne pour le prochain round.
    send(CHANNEL_1, key, 12);
    send(CHANNEL_2, key, 12);
    send(CHANNEL_3, key, 12);
    recv(CHANNEL_1, &newByte1, 1);
    recv(CHANNEL_2, &newByte2, 1);
    recv(CHANNEL_3, &newByte3, 1);
    key[i] = newByte1 ^ newByte2 ^ newByte3;
}
```

FIGURE 10 – Transcription en (simili-)C du code exécuté par le premier *transputer*

Tu auras remarqué, cher ami, que le chiffrement implémenté dans *input.bin* est susceptible d'une *attaque par texte clair*. En effet, si tu connais – mettons – les 10 premiers octets du fichier à déchiffrer, tu peux en dériver le 10 premiers octets du masque en calculant $mask_i = clair_i \oplus chiffre_i$. Or, $mask_i$ est défini comme étant $mask_i = i + 2 * key_i$, de sorte que $key_i = (mask_i - i)/2 \vee key_i = (mask_i - i)/2 + 128$ (il y a plusieurs valeurs possibles car nous travaillons *modulo* 256 – ce sont des octets – et que 2, le diviseur, n'est pas premier avec 256, le module). Partant de là, il suffit, pour trouver la clef, de deviner les 2 derniers octets (16 bits) et le bit de poids fort des 10 premiers octets (10 bits) : cela fait un espace de recherche de 26 bits – espace raisonnablement parcourable.

Tu te doutes bien que je n'ai pas choisi l'exemple des 10 premiers octets de texte clair au hasard : il s'agit en fait de la longueur du *magic number* des fichiers compressés par *bzip2* avec les réglages par défaut (le signature peut varier si l'on change manuellement la taille des blocs). Pour ton information, en ASCII, ce nombre magique vaut `BZh91AY&SY`. Or, si tu as suivi le fil de mes pensées (exercice difficile), tu te souviens que le fichier chiffré est plausiblement un fichier *bzip2* (`congratulations.tar.bz2`). Dans l'absolu, je pourrai arrêter la cryptanalyse ici et essayer itérativement les différents combinaisons possibles dans un simulateur de ST-20... mais cette approche serait trop lente (et, par ailleurs, il n'existe pas à ma connaissance d'émulateur répondant à mes besoins).

Bref. Après avoir initialisé le premier *transputer*, le fichier *input.bin* s'occupe des *transputers* 1, 2 et 3 en y chargeant à chaque fois le même programme (appellons-le *prog2*) *via* la facilité de transmission de messages fournie par *prog1*. Ce programme pèse 112 octets et fait ceci :

Première phase il retransmet les messages qu'il reçoit au canal indiqué ;

Seconde phase il redecend l'état courant de la clef qu'on lui envoye à ces trois fils puis remonte le XOR des octets que ses mêmes trois fils lui renvoient.

Ensuite, c'est le tour des feuilles de l'arbre (*transputer* 4, 5, 6, 7, 8, 9, 10, 11 et 12) d'être initialisées. Là encore, c'est toujours le même programme (*prog3*) qui est chargé. Il mesure 36 octets et se contente de recevoir un autre programme et de l'exécuter depuis un point d'entrée indiqué (*via* l'instruction `gcall` [1]).

Après, *input.bin* charge 9 programmes distincts dans les feuilles de l'arbre de *transputers* (qui, puisqu'ils exécutent tous *prog3*, n'attendent que ça). Au vu de ce que j'ai analysé jusqu'ici, je sais qu'il s'agit des *primitives* qui serviront à mettre-à-jour, au fur et à mesure du déchiffrement, la clef à partir de son état courant (et des états internes propres aux primitives) en XORant leurs valeurs de retour. C'est donc le noeud du problème.

Enfin, la boucle de transmission est terminée en envoyant un message vide (`size = 0`) au premier *transputer*. Celui-ci le répercute aux *transputers* 1, 2 et 3 qui sortent alors de leurs propres boucles de transmission (i.e. de leur première phase, voir *supra*). La suite de *input.bin* contient le texte `KEY:`, puis les 12 octets de la clef (initialisés à `0xFF`), puis

1. Si tu es curieux et que tu essaye *interpret.py* sur *prog3*, tu verras qu'il lève une exception car il ne sait pas interpréter `gcall`... mais ce n'est pas grave dans la mesure où `gcall` est la dernière instruction du programme : il aura correctement désassemblé le reste.

la longueur du nom du fichier à déchiffrer (23 octets), puis le nom du fichier à déchiffrer (`congratulations.tar.bz2`) et, enfin, les données à déchiffrer (en les récupérant dans un fichier à part, il est possible de vérifier avec *shasum -a 256* que leur SHA-256 est bien celui indiqué dans *schematic.pdf*).

Ci-dessous, je t'ai transcrit le code de *prog2* dans la figure 11 et celui de *prog3* dans la figure 12.

```
// Format de l'entête des messages.
struct header {
    uint32_t size; // Taille du message.
    uint32_t channel; // Destinataire du message.
    uint32_t offset; // Point d'entrée du message.
}

// Boucle de transmission des messages vers les transputers fils.
struct header header;
char message[UNKNOW];
while (1) {
    recv(CHANNEL_0, &header, 12);
    if (header.size == 0)
        break;
    recv(CHANNEL_0, message, header.size);
    send(header.channel, message, header.size);
}

// Boucle de descente de l'état interne vers chaque transputer fils,
// puis de remontée du XOR des octets retournés par les transputers.
char key[12];
while (1) {
  recv(CHANNEL_0, key, 12);
  send(CHANNEL_1, key, 12);
  send(CHANNEL_2, key, 12);
  send(CHANNEL_3, key, 12);
  recv(0x80000014, &newByte1, 1);
  recv(0x80000018, &newByte2, 1);
  recv(0x8000001c, &newByte3, 1);
  newByte = newByte1 ^ newByte2 ^ newByte3;
  send(CHANNEL_0, &newByte, 12);
}
```

FIGURE 11 – Transcription en (simili-)C de *prog2*

```
// Réception et exécution d'un programme personnalisé
struct header header;
char message[UNKNOW];
recv(CHANNEL_0, &header, 12);
recv(CHANNEL_0, message, header.size);
(message+header.offset)();
```

FIGURE 12 – Transcription en (simili-)C de *prog3*

À ce moment du challenge, j'ai accumulé suffisamment de connaissances pour écrire un simulateur capable de déchiffrer le vecteur de test fourni par les organisateurs. Il contient énormément du code repris sur mon désassembleur (oui, je sais, la redondance, c'est mal) et ne passe pas bien à l'échelle... mais il réussit à décoder le vecteur de test ! Je m'en suis servi pour remplacer, petit à petit, les *transputers*-feuilles par les *primitives* qu'ils implémentent (en vérifiant, après chaque substitution, que le vecteur de test était encore déchiffré correctement). Bref, *as usual*, le code source de *simulator.py* est joint avec ce courrier (qui commence à être sacrément long... je crois que je devrai coller un – voire plusieurs ? – timbre fushia).

Le tableau ci-dessous présente dans les grands traits les caractéristiques des neufs *primitives* utilisées. Les transcriptions en C++ sont également disponibles (`ProgA` correspondant au programme exécuté sur le *transputer* 4, `ProgB` à celui exécuté sur le *transputer* 5, et ainsi de suite...).

Transputer	**Primitive**
Transputer 4	Somme des octets de l'ensemble des blocs reçus
Transputer 5	XOR des octets de l'ensemble des blocs reçus
Transputer 6	Registre à décalage à rétroaction linéaire
Transputer 7	XOR des sommes des six premiers octets et des six derniers
Transputer 8	XOR des sommes des octets des quatre derniers blocs
Transputer 9	XOR des octets avec un décalage à gauche variable
Transputer 10	Truc bizarre – regarde le code source
Transputer 11 & 12	Sœurs jumelles – regarde le code source

```cpp
class ProgA {
    // Somme des octets de l'ensemble des blocs reçus

private:
    char sum;

public:
    ProgA() : sum(0) {}

    char operator() (const char* blk) {
        for (int i = 0; i < 12; i++)
            sum = (sum + blk[i]) & 255;
        return sum;
    }

    void reset() {
        sum = 0;
    }
};
```

FIGURE 13 – Transcription en C++ du programme exécuté sur le *transputer* 4

```cpp
class ProgB {
    // XOR des octets de l'ensemble des blocs reçus

private:
    char eor; // xor est un mot réservé du langage

public:
    ProgB() : eor(0) {}

    char operator() (const char* blk) {
        for (int i = 0; i < 12; i++)
            eor = (eor ^ blk[i]) & 255;
        return eor;
    }

    void reset() {
        eor = 0;
    }
};
```

FIGURE 14 – Transcription en C++ du programme exécuté sur le *transputer* 5

```cpp
class ProgC {
    // Registre à décalage à rétroaction linéaire

private:
    int lfsr;
    bool initialized;

public:
    ProgC() : lfsr(0), initialized(0) {}

    char operator() (const char* blk) {
        if (!initialized) {
            for (int i = 0; i < 12; i++)
                lfsr = (lfsr + ((unsigned char)blk[i])) & 65535;
            initialized = true;
        }
        lfsr = ((lfsr << 1) & 65535) ^ ((lfsr >> 14) & 1) ^ ((lfsr >> 15) & 1);
        return lfsr & 255;
    }

    void reset() {
        lfsr = 0;
        initialized = 0;
    }
};
```

FIGURE 15 – Transcription en C++ du programme exécuté sur le *transputer* 6

```
class ProgD {
    // XOR des sommes des six premiers octets et des six derniers

public:
    ProgD() {}

    char operator() (const char* blk) {
        char s1 = 0, s2 = 0;
        for (int i = 0; i < 6; i++)
            s1 = (s1 + blk[i]) & 255, s2 = (s2 + blk[6+i]) & 255;
        return (s1 ^ s2) & 255;
    }

    void reset() {}
};
```

FIGURE 16 – Transcription en C++ du programme exécuté sur le *transputer* 7

```
class ProgE {
    // XOR des sommes des octets des quatre derniers blocs

private:
    char mem[48];
    int curr;

public:
    ProgE() : mem(), curr(0) { }

    char operator() (const char* blk) {
        char eor, sum;

        // Memorize the current block.
        memcpy(&mem[12*curr], blk, 12);
        curr = (curr + 1) & 3;

        // Compute the XOR of the sums.
        eor = 0;
        for (int i = 0; i < 4; i++) {
            sum = 0;
            for (int j = 0; j < 12; j++)
                sum = (sum + mem[12*i + j]) & 255;
            eor = (eor ^ sum) & 255;
        }

        return eor;
    }

    void reset() {
        bzero(mem, 48);
        curr = 0;
    }
};
```

FIGURE 17 – Transcription en C++ du programme exécuté sur le *transputer* 8

```
class ProgF {
    // XOR des octets avec un décalage à gauche variable

public:
    ProgF() {}

    char operator() (const char* blk) {
        char res;
        res = 0;
        for (int i = 0; i < 12; i++)
            res = (res ^ ((blk[i] << (i & 7)) & 255)) & 255;
        return res & 255;
    }

    void reset() {}
};
```

FIGURE 18 – Transcription en C++ du programme exécuté sur le *transputer* 9

```
class ProgG {
    // Truc bizarre

private:
    char mem[48];
    int curr;

public:
    ProgG() : mem(), curr(0) {}

    char operator() (const char* blk) {
        int no, idx;
        char sum;

        // Memorize the current block.
        memcpy(&mem[12*curr], blk, 12);
        curr = (curr + 1) & 3;

        // Sum the first byte of each memorized block.
        sum = 0;
        for (int i = 0; i < 4; i++)
            sum = (sum + mem[12*i]) & 255;

        // Output the right byte of the memory.
        no = ((unsigned char)sum) & 3, idx = (((unsigned char)sum) >> 4) % 12;
        return mem[12*no + idx];
    }

    void reset() {
        bzero(mem, 48);
        curr = 0;
    }
};
```

FIGURE 19 – Transcription en C++ du programme exécuté sur le *transputer* 10

```cpp
class ProgH {
    // Soeurs jumelles
    // Les programmes exécutés sur les deux transputers 11 et 12 ont été
    // ici fusionnés et le XOR qui avait lieu au niveau du transputer 3 est
    // finalement implémenté dans l'expression du 'return'.

private:
    char prev[12];

public:
    ProgH() : prev() {};

    char operator() (const char* blk) {
        char b, c;

        b = prev[1] ^ prev[5] ^ prev[9];
        c = blk[0] ^ blk[3] ^ blk[7];

        // Memorize the current block.
        memcpy(prev, blk, 12);

        return blk[((unsigned char)b)%12] ^ blk[((unsigned char)c)%12];
    }

    void reset() {
        bzero(prev, 12);
    }
};
```

FIGURE 20 – Transcription en C++ du programme exécuté sur le *transputer* 11 et 12

```
MacBook-Pro-de-Tristan:Travail unroot$ python simulator.py
The result is 'I love ST20 architecture'.
MacBook-Pro-de-Tristan:Travail unroot$
```

FIGURE 21 – Exécution de *simulator.py* avec le vecteur de test

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

FIGURE 22 – Le vecteur de test (extrait de *schematic.pdf*)

En partant de ces composants retro-ingénierés en C++ (puis en les remplaçant par des optimisations – à l'exception de `ProgG`), j'ai pu programmer un logiciel réalisant l'*attaque par texte clair* dont je t'ai parlé plus haut. Il s'appelle *testoo* (oui, je sais, c'est un nom un peu pourri, mais c'est comme ça que j'appelle toutes mes expérimentations). Je te fournis le code source en pièce-jointe. J'estime qu'il devrait prendre deux jours pour parcourir l'ensemble des solutions (mais il s'arrêtera probablement avant, sauf si j'ai foiré ma rétro-ingénierie, ce qui est loin d'être improbable) si je le fais tourner sur Gideon (qui – je te le rappelle – est un octo-coeur).

Voilà, j'achève cette lettre ici et je croise les doigts – et j'espère que tu le feras aussi – pour que tout cela aboutisse à quelque chose. Vu le nom de l'archive (`congratulations`), je présume qu'il s'agit en fait de la dernière étape : je touche au but ! Et, si Fortuna est avec moi, peut-être serai-je dans les 10 premiers du classement rapidité ! !

Amicalement, Tristan.

PS : Pour être tout-à-fait complet, je tiens à te préciser, cher Anselme, que j'ai aussi fait joujou avec `st20dis` qui est un désassembleur ST-20 multiplateforme (que tu trouveras à l'adresse `http://digifusion.jeamland.org/st20dis/`) et avec `st20emu` qui est un émulateur ST-20 pour Windows mais parfaitement utilisable sous Wine (que tu pourras télécharger sur SourceForge à l'adresse `http://sourceforge.net/projects/st20emu/`). Malheureusement, ce dernier ne gère pas les instructions `in` et `out` (ni, d'ailleurs, les configurations multi-processeurs comme celles de ce challenge). Au final, j'ai bien utilisé mon propre désassembleur (*interpret.py*) et mon propre simulateur (*simulator.py*).

PPS : Petite note en passant, tu auras sûrement noté que le chiffrement utilisé fait fuiter le dernier bit de chaque octet. Dit autrement, on peut récupérer le bit de poids faible de chaque octet du message clair à partir du message chiffré. Ce n'est pas une propriété très souhaitable... mais, malheureusement, elle ne me permet pas d'optimiser l'*attaque par texte clair*.

M. Philippe-Anselme Foury
[redacted], rue de [redacted]
130[redacted] Marseille

Lyon, le vendredi 10 avril

Cher Anselme,

```
unroot@Gideon:~$ cat results
    0: self-test(0): I love ST20 architecture
  128: self-test(0): I love ST20 architecture
  128: self-test(1): I love ST20 architecture
    0: self-test(1): I love ST20 architecture
  256: self-test(0): I love ST20 architecture
  256: self-test(1): I love ST20 architecture
  374: self-test(0): I love ST20 architecture
  374: self-test(1): I love ST20 architecture
  512: self-test(0): I love ST20 architecture
  512: self-test(1): I love ST20 architecture
  640: self-test(0): I love ST20 architecture
  640: self-test(1): I love ST20 architecture
  768: self-test(0): I love ST20 architecture
  768: self-test(1): I love ST20 architecture
  896: self-test(0): I love ST20 architecture
  896: self-test(1): I love ST20 architecture
> passed the bzip2 test!
>> found an archive containing the file 'congratulations.jpg'!
  358: key: 5e:d4:9b:71 56:fc:e4:7d e9:76:da:c5
```

Mon attaque a terminé ! C'était tellement improbable. Car il était presque arrivé à la fin de l'espace des possibles. Je commençais à penser que je m'étais planté. J'avais lancé l'attaque mercredi soir. Sur Gideon. Tous les coeurs utilisés. `renice -n -20`. Je viens juste de regarder l'état de Gideon. Pas encore fait le déchiffrement sur ma machine. J'avoue que je suis très content que ma tentative ait abouti.

D'après mon programme, l'archive contient une image *congratulations.jpg* (pour vérifier que la clef testé est correct, le brute-force vérifie le nom du premier fichier de l'archive : si c'est un truc bizarre – i.e. pas une chaîne de caractères imprimables complétée par des nuls – alors il considère que ce n'était pas vraiment un TAR et passe au suivant ; effet secondaire : on sait ce que contient l'archive). La clef est donc `5ed49b71 56fce47d e976dac5`. Comme dirait David Tennant – en anglais dans le texte : Allons-y !

```python
import testoo # testoo.py est dans les pièces-jointes à cette lettre.
import bz2, tarfile, StringIO, subprocess
key = "5ed49b71␣56fce47d␣e976dac5".replace("␣", "").decode("hex")
data = open("encrypted", "r").read()
bzip = testoo.decrypt(key, data)
tar = tarfile.open("congratulations.tar.bz2", "r:bz2", StringIO.StringIO(bzip))
tar.extract("congratulations.jpg")
print subprocess.check_output(["file", "congratulations.jpg"]).strip()
# congratulations.jpg: JPEG image data, JFIF standard 1.01
```

FIGURE 23 – Oui, j'ouvre les `.tar.bz2` avec Python

FIGURE 24 – Félicitations !

Las. Il y a encore *un dernier petit effort* à faire. Si l'on en croit le texte écrit dans `congratulation.jpg`. La réponse est dans cette image... (sûrement) Si près du but. Qu'est-ce que c'est frustrant. J'ai vérifié que `congratulations.jpg` ne contient pas de chaînes de caractères compréhensibles avec *strings*. Tout est perdu. Mais ? Qu'est-ce que ? Serait-ce...

```
MacBook-Pro-de-Tristan:Travail unroot$ strings -n 10 -t x congratulations.jpg
1316 4@P5!%"&'13p$27
2d16 gLMRrs1IPP
d7d0 BZh91AY&SY
1c363 #3 ~bc#q_B
25914 q/`k@F'w.{W
36dd0  NR6|zc ^%T8
377d7
      com49L]*a
3be81 `N#)m.B:v^h;[
```

FIGURE 25 – J'ai déjà vu ce *magic number* quelque part...

La signature d'un *bzip2* ! Il y a un fichier *bzip2* coincé à partir de l'offset `d7d0`[2] ! Tirons-le de là, et vite (et en Python) :

```
open("cache.bz2", "w+").write(open("congratulations.jpg", "r").read()[0xd7d0:])
```

FIGURE 26 – Sauvetage d'un *bzip2* noyé au milieu d'un JPEG

2. Noté en hexadécimal bien sûr – mais ça va mieux en le disant...

FIGURE 27 – Félicitations (*bis repetita non placent*) !

C'est un PNG cette fois-ci. Les fichiers PNG sont composés de segments (*chunks*) : `IHDR`, `bKGD`, `pHYs`, `tIME`, `IDAT`, `IEND`, ... Et `sTic`. Ce dernier type de segment est non-standard. Et ressemble étrangement à SSTIC. En supposant que ces segments suivent le même régime que les `IDAT` (c'est-à-dire que leur concaténation forme un flux *zlib*), on peut essayer de récupérer les données qui y sont cachées...

```
import struct, zlib
png = open("congratulations.png", "r")
magic = png.read(8)
chunks = []
while True:
    chunkSize = png.read(4)
    if not chunkSize:
        break
    chunkSize = struct.unpack(">I", chunkSize)[0]
    chunkType = png.read(4)
    chunkData = png.read(chunkSize)
    chunkCode = png.read(4)
    chunks.append((chunkType, chunkData))
print set(x[0] for x in chunks)
# set(['bKGD', 'IHDR', 'pHYs', 'sTic', 'tIME', 'IDAT', 'IEND'])
sTic = ''.join(x[1] for x in chunks if x[0] == 'sTic')
print len(sTic)
# 132851
decomp = zlib.decompress(sTic)
print decomp[:10]
# BZh91AY&SY
open("cache2.bz2", "w+").write(decomp)
```

FIGURE 28 – Dis quels sont tes *chunks* et je te dirai qui tu es...

FIGURE 29 – Félicitations (*ter repetita non placent*) !

Un TIFF cette fois. Intéressant : le noir n'est pas uniforme. Il varie entre le vrai noir (0, 0, 0) et d'autres noirs : (0, 1, 0), (1, 0, 0), (1, 1, 0), ... Un message en binaire ? J'en sais trop rien. Sûrement une technique qui code les données cachés dans les bits de poids faible (LSB) des canaux RGB... Testons cette hypothèse :

```
from PIL import Image
ori = Image.open("congratulations.tiff")
lsb = Image.new("RGB", (636, 474))
oriMap, lsbMap = ori.load(), lsb.load()
for x in range(636):
  for y in range(474):
    oriPix = oriMap[x,y]
    lsbPix = tuple(255*(x&1) for x in oriPix)
    lsbMap[x,y] = lsbPix
lsb.save("lsb.bmp")
```

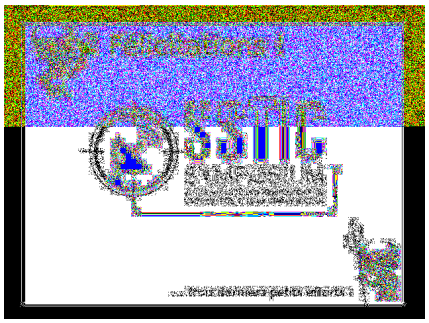

FIGURE 30 – Isolation des bits de poids faible (LSB)

En isolant les LSB, nous constatons deux choses. Premièrement, on voit qu'il y a bien de la stéganographie dans la partie haute de l'image. Secondement, on peut encore reconnaître l'image support dans cette zone, ce qui signifie qu'elle en dépend encore partiellement. En fait, seuls les LSB des canaux rouge et vert semblent utilisés : les LSB du canal bleu conservent leurs valeurs initiales (c'est-à-dire 0 pour la bordure noire et 1 pour le fond blanc).

```python
from PIL import Image
ori = Image.open("congratulations.tiff")
lsb = Image.new("RGB", (636, 474))
oriMap, lsbMap = ori.load(), lsb.load()
for x in range(636):
  for y in range(474):
    oriPix = oriMap[x,y]
    lsbR = 255*(oriPix[0]&1)
    lsbG = 255*(oriPix[1]&1)
    lsbB = int((lsbR+lsbG)/2.)
    lsbPix = (lsbR, lsbG, lsbB)
    lsbMap[x,y] = lsbPix
lsb.save("lsb2.bmp")
```

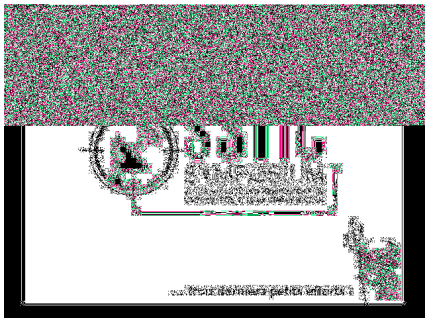

FIGURE 31 – Isolation des bits de poids faible (LSB) – canaux rouge et vert uniquement

Le fantôme de l'image support n'apparaît plus dans la partie stéganographiée : tentons désormais d'extraire les données cachées ! Je sais déjà (grâce à l'analyse graphique) que les informations sont codées dans les pixels de gauche à droite puis de haut en bas. Je suppose alors que, dans chaque pixel, le canal rouge porte le premier bit puis le canal vert le second – c'est l'ordre RGB. Alors on peut donner un petit coup de Python :

```python
from PIL import Image
ori = Image.open("congratulations.tiff")
oriMap = ori.load()
bin = []
for y in range(474):
    for x in range(636):
        oriPix = oriMap[x,y]
        lsbR = oriPix[0]&1
        lsbG = oriPix[1]&1
        bin.append(str(lsbR)+str(lsbG))
dat = []
for i in range(len(bin)/4):
    byte = int(bin[4*i]+bin[4*i+1]+bin[4*i+2]+bin[4*i+3], 2)
    char = chr(byte)
    dat.append(char)
res = ''.join(dat)
print res[:10]
# BZh91AY&SY
open("cache3.bz2", "w+").write(res)
```

FIGURE 32 – Extraction de données cachées dans les tréfonds des LSB

Cette fois, il s'agit d'un GIF. Il ne contient pas de segments superflus et utilise une palette de 256 couleurs – le maximum. Une intuition me mène à essayer d'associer à chaque indice de la palette une nuance arbitrairement prise arbitrairement dans le cercle chromatique, sans considération de la couleur stockée dans la palette. Et...

```
from PIL import Image
from colorsys import hsv_to_rgb
ori = Image.open("congratulations.gif")
pal = Image.new("RGB", (636, 474))
oriMap = ori.load()
palMap = pal.load()
for x in range(636):
  for y in range(474):
    oriPix = oriMap[x,y]
    palPix = tuple(int(255*z) for z in hsv_to_rgb(oriPix/256., 1, 1))
    palMap[x,y] = palPix
pal.save("pal.bmp")
```

FIGURE 33 – Coloriage psychédélique

Enfin : `1713e7c1d0b750ccd4e002bb957aa799@challenge.sstic.org` ! Ce challenge est désormais fini, du moins dès que j'aurai envoyé le courriel. Je finis donc cette dernière lettre ici avec la félicité d'être arrivé au bout et l'espérence de bientôt lire de tes nouvelles. Vite, vite, dépêche-toi, plume, de poser mes pensées sur le papier : je ne dois pas me faire voler cette septième place !

Amicalement, Tristan.

De : Tristan Dubois
À : 1713e7c1d0b750ccd4e002bb957aa799@challenge.sstic.org
Date : le 10 avril 2015 à 12h56
Sujet : Coucou

Bonsoir,

J'aimerais juste savoir si je frappe bien à la bonne adresse.

Cordialement,
Tristan Dubois

De : Pierre [redacted]
À : Tristan Dubois
Date : le 10 avril 2015 à 13h30
Sujet : Rép : Coucou

Oui !

Bien joué :)

De : Ben
À : Tristan Dubois
Date : le 10 avril 2015 à 23h01
Sujet : Rép : Coucou

En effet.

Félicitations :-)

–
Ben

M. Tristan DUBOIS
[redacted], rue [redacted]
690[redacted] Lyon

Marseille, le mardi 14 avril

Très cher ami,

La dernière clé de mon trousseau est donc celle qui ouvre ma boîte aux lettres...

Quelle heureuse surprise lorsque j'ai découvert dans l'amas de prospectus pour pizzas, clubs de sport et autres marabouts, ces lettres cachetées de votre sceau. Je vous conseille vivement d'ouvrir un compte professionnel à la Poste si vous décidez de continuer à m'envoyer des documents, vous ferez certainement d'intéressantes économies.

Bien que très exhaustif, j'ai lu tout votre récit et je l'ai presque entièrement compris ! Joli travail de vulgarisation.

Viendrez-vous à la soirée au Manoir des Montgomery de ce weekend ? Ce sera l'occasion d'approfondir certains aspects techniques de ce challenge, et je vous expliquerai pourquoi j'avais installé un serveur X11 sur Gideon !

A très bientôt je l'espère,

Anselme.

Pièces-jointes

decode.py

```
#!/usr/bin/python

def byteToChar(b, m = False):
    """Convert a scan-code to the corresponding character."""
    if b >= 0x04 and b <= 0x1d:
        return chr(b + 0x3d) if m else chr(b + 0x5d)
    elif b >= 0x1e and b <= 0x26 and not m:
        return chr(b + 0x13)
    elif b == 0x2c:
        return ' '
    elif b == 0x1e:
        return '!'
    elif b == 0x1f:
        return chr(64)
    elif b == 0x20:
        return '#'
    elif b == 0x21:
        return '$'
    elif b == 0x22:
        return '%'
    elif b == 0x23:
        return '^'
    elif b == 0x24:
        return '&'
    elif b == 0x25:
        return '*'
    elif b == 0x26:
        return '('
    elif b == 0x27:
        return ')' if m else '0'
    elif b == 0x2D:
        return '_' if m else '-'
    elif b == 0x2E:
        return '+' if m else '='
    elif b == 0x2F:
        return '{' if m else '['
    elif b == 0x30:
        return '}' if m else ']'
    elif b == 0x31:
        return '|' if m else '\\'
    elif b == 0x33:
        return ':' if m else ';'
    elif b == 0x34:
        return '"' if m else '\''
    elif b == 0x35:
        return '~' if m else '`'
    elif b == 0x36:
        return '<' if m else ','
    elif b == 0x37:
        return '>' if m else '.'
    elif b == 0x38:
        return '?' if m else '/'
    else:
        return False

def byteToSpecial(b):
    """Convert a scan-code to a description of the corresponding key."""
    if b == 0x2c:
        return 'SPACE'
    elif byteToChar(b):
        return '%s' % (byteToChar(b), )
    elif b == 0x00:
        return 'NULL'
    elif b == 0x3a:
        return 'F1'
    elif b == 0x3b:
        return 'F2'
    elif b == 0x3c:
        return 'F3'
    elif b == 0x3d:
        return 'F4'
    elif b == 0x3e:
        return 'F5'
    elif b == 0x3f:
        return 'F6'
    elif b == 0x40:
        return 'F7'
    elif b == 0x41:
        return 'F8'
    elif b == 0x42:
        return 'F9'
```

```
    elif b == 0x43:
        return 'F10'
    elif b == 0x44:
        return 'F11'
    elif b == 0x45:
        return 'F12'
    elif b == 0x29:
        return 'ESCAPE'
    elif b == 0x48:
        return 'PAUSE'
    elif b == 0x2b:
        return 'TAB'
    elif b == 0x28:
        return 'ENTER'
    elif b == 0x99:
        return 'UNKNOW'
    elif b == 0x4a:
        return 'HOME'
    elif b == 0xe3:
        return 'WINDOWS'
    elif b == 0x49:
        return 'INSERT'
    elif b == 0x4b:
        return 'PAGEUP'
    elif b == 0x4e:
        return 'PAGEDOWN'
    elif b == 0x4c:
        return 'DELETE'
    elif b == 0x4d:
        return 'END'
    elif b == 0x52:
        return 'UPARROW'
    elif b == 0x51:
        return 'DOWNARROW'
    elif b == 0x50:
        return 'LEFTARROW'
    elif b == 0x4f:
        return 'RIGHTARROW'
    elif b == 0xe1:
        return 'SHIFT'
    elif b == 0xe2:
        return 'MUTE'
    elif b == 0xe9:
        return 'VOLUMEUP'
    elif b == 0xea:
        return 'VOLUMEDOWN'
    elif b == 0xcd:
        return 'PLAY'
    elif b == 0xb5:
        return 'STOP'
    elif b == 0x53:
        return 'NUMLOCK'
    elif b == 0x47:
        return 'SCROLLLOCK'
    elif b == 0x46:
        return 'PRINTSCREEN'
    elif b == 0x39:
        return 'CAPSLOCK'
    elif b == 0x83:
        return 'SYSTEMWAKE'
    elif b == 0x82:
        return 'SYSTEMSLEEP'
    elif b == 0x81:
        return 'SYSTEMPOWER'
    else:
        return '[0x%02x]' % b

inject = open("inject.bin").read()
res = open("duckyscript.txt", "w+")
delay = 0
string = ""

for i in range(len(inject)/2):
    injectToken = tuple(inject[2*i:2*(i+1)])

    # DELAY command
    if injectToken[0] == chr(0x00):
        delay += ord(injectToken[1])
        continue
    if delay:
        res.write("DELAY␣%d\n" % (delay, ))
        delay = 0
```

```
# STRING command
if injectToken[1] in (chr(0x00), chr(0x02)) and byteToChar(ord(injectToken[0])):
    string += byteToChar(ord(injectToken[0]), injectToken[1] == chr(0x02))
    continue
if string:
    res.write("STRING␣%s\n" % (string, ))
    string = ""

# CONTROL command
if injectToken[1] == chr(0x01):
    res.write("COMMAND␣%s\n" % (byteToSpecial(ord(injectToken[0])), ))
    continue

# ALT command
if injectToken[1] == chr(0xe2):
    res.write("ALT␣%s\n" % (byteToSpecial(ord(injectToken[0])), ))
    continue

# SHIFT command
if injectToken[1] == chr(0xe1):
    res.write("SHIFT␣%s\n" % (byteToSpecial(ord(injectToken[0])), ))
    continue

# WINDOWS command
if injectToken[1] == chr(0x08):
    res.write("WINDOWS␣%s\n" % (byteToSpecial(ord(injectToken[0])), ))
    continue

# Other commands
if injectToken[1] == chr(0x00):
    res.write("%s\n" % (byteToSpecial(ord(injectToken[0])), ))
    continue

raise ValueError("Unknow␣inject␣token:␣%s" % (injectToken.encode('hex'), ))
```

```python
# picasso.py

from dpkt.pcap import Reader
from itertools import izip, count
from struct import unpack
from PIL import Image, ImageDraw

curX, curY = 0, 0
maxX, maxY = 0, 0
minX, minY = 0, 0
mouseEvents = []
for no, data in izip(count(1), (x[1] for x in Reader(open("paint.cap", "rb")))):
    if no < 7 or no % 2 != 1:
        continue
    mouseEvent = unpack("?bbx", data[-4:])
    curX, curY = curX + mouseEvent[1], curY + mouseEvent[2]
    maxX, maxY = max(maxX, curX), max(maxY, curY)
    minX, minY = min(minX, curX), min(minY, curY)
    mouseEvents.append(mouseEvent)

curX, curY = -minX, -minY
nxtX, nxtY = None, None
lenX, lenY = maxX - minX, maxY - minY
image = Image.new("RGB", (lenX, lenY), "white")
draw = ImageDraw.Draw(image)
for mouseEvent in mouseEvents:
    nxtX, nxtY = curX + mouseEvent[1], curY + mouseEvent[2]
    if mouseEvent[0]:
        draw.line([(curX, curY), (nxtX, nxtY)], fill="black", width=2)
    curX, curY = nxtX, nxtY

del draw
image.save("trace.bmp")
```

```c
/* decrypter.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <fcntl.h>
#include <assert.h>
#include <gcrypt.h>

#define LENGTH 296798

// # Pour dériver la clef de chiffrement.
// import blake
// Blake256 = blake.BLAKE(256)
// Blake256.update("The quick brown fox jumps over the lobster dog")
// print Blake256.final().encode('hex')
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 };

char iv[] = "SSTIC2015-Stage3"; // C'est mieux en ASCII :-)

char in[LENGTH];
char out[LENGTH];

int main(void){
    int fd, cnt;

    fd = open("encrypted", O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    cnt = read(fd, in, LENGTH);
    if (cnt != LENGTH) {
        if (cnt == -1)
            perror("read");
        else
            fprintf(stderr, "read:␣got␣%d,␣expected␣%d\n", cnt, LENGTH);
        exit(EXIT_FAILURE);
    }

    close(fd);

    gcry_cipher_hd_t hd;
    assert(gcry_cipher_open(&hd, GCRY_CIPHER_SERPENT256, GCRY_CIPHER_MODE_CBC, GCRY_CIPHER_CBC_CTS) == 0);
    assert(gcry_cipher_setkey(hd, key, 32) == 0);
    assert(gcry_cipher_setiv(hd, iv, 16) == 0);
    assert(gcry_cipher_decrypt(hd, out, LENGTH, in, LENGTH) == 0);
    gcry_cipher_close(hd);

    fd = creat("decrypted.zip", 0666);
    if (fd == -1) {
        perror("creat");
        exit(EXIT_FAILURE);
    }

    cnt = write(fd, out, LENGTH);
    if (cnt != LENGTH) {
        if (cnt == -1)
            perror("write");
        else
            fprintf(stderr, "write:␣got␣%d,␣expected␣%d\n", cnt, LENGTH);
        exit(EXIT_FAILURE);
    }

    fsync(fd);

    close(fd);

    return 0;
}
```

dictattack.py

```
from Crypto.Cipher import AES
from hashlib import sha1
from itertools import product
from subprocess import check_output
from tempfile import mkstemp
from os import fdopen, unlink

DATA = filter(lambda x: x.find("var␣data") != -1,
     ↪ open("stage4.html").readlines())[0].split("\"")[1].decode("hex")
HASH = "08c3be636f7dffd91971f65be4cec3c6d162cb1c".decode('hex')

def guess(data):
    fildes, path = mkstemp();
    filobj = fdopen(fildes, "w+")
    filobj.write(data)
    filobj.flush()
    educatedGuess = check_output(["file", path]).split(":␣")[1].strip()
    filobj.close()
    unlink(path)
    return educatedGuess

def attempt(parenthese):
    if len(parenthese) < 16:
        return None
    IV, key = parenthese[:16], parenthese[-16:]
    plaintext = AES.new(key=key, mode=AES.MODE_CBC, IV=IV).decrypt(DATA)
    p = ord(plaintext[-1])
    if p == 0 or p > 16:
        return None
    if len(set(plaintext[-p:])) != 1:
        return None
    plaintext = plaintext[:-p]
    if sha1(plaintext).digest() != HASH:
        return None
    return plaintext

platforms = [
    # Windows
    "Windows␣x86",
    "Windows␣NT␣5.0",
    "Windows␣NT␣5.1",
    "Windows␣NT␣5.2",
    "Windows␣NT␣5.2;␣Win64;␣x64",
    "Windows␣NT␣5.2;␣WOW64",
    "Windows␣NT␣6.0",
    "Windows␣NT␣6.0;␣Win64;␣x64",
    "Windows␣NT␣6.0;␣WOW64",
    "Windows␣NT␣6.1",
    "Windows␣NT␣6.1;␣Win64;␣x64",
    "Windows␣NT␣6.1;␣WOW64",
    "Windows␣NT␣6.2",
    "Windows␣NT␣6.2;␣Win64;␣x64",
    "Windows␣NT␣6.2;␣WOW64",
    "Windows␣NT␣6.3",
    "Windows␣NT␣6.3;␣Win64;␣x64",
    "Windows␣NT␣6.3;␣WOW64",

    # Linux
    "X11;␣OpenBSD;␣amd64",
    "X11;␣OpenBSD;␣x86_64",
    "X11;␣OpenBSD;␣i686",
    "X11;␣OpenBSD;␣i686␣on␣x86_64",
    "X11;␣OpenBSD;␣i686␣on␣amd64",
    "X11;␣OpenBSD;␣i586",
    "X11;␣OpenBSD;␣i386",
    "X11;␣OpenBSD;␣arm",
    "X11;␣OpenBSD;␣arm64",
    "X11;␣OpenBSD;␣armel",
    "X11;␣OpenBSD;␣armhf",
    "X11;␣OpenBSD;␣hppa",
    "X11;␣OpenBSD;␣ia64",
    "X11;␣OpenBSD;␣mips",
    "X11;␣OpenBSD;␣mipsel",
    "X11;␣OpenBSD;␣powerpc",
    "X11;␣OpenBSD;␣ppc",
    "X11;␣OpenBSD;␣powerpcspe",
    "X11;␣OpenBSD;␣ppc64",
    "X11;␣OpenBSD;␣ppc64el",
    "X11;␣OpenBSD;␣s390x",
    "X11;␣OpenBSD;␣sparc",
    "X11;␣OpenBSD;␣sparc64",
```

```
"X11;␣FreeBSD;␣amd64",
"X11;␣FreeBSD;␣x86_64",
"X11;␣FreeBSD;␣i686",
"X11;␣FreeBSD;␣i686␣on␣x86_64",
"X11;␣FreeBSD;␣i686␣on␣amd64",
"X11;␣FreeBSD;␣i586",
"X11;␣FreeBSD;␣i386",
"X11;␣FreeBSD;␣arm",
"X11;␣FreeBSD;␣arm64",
"X11;␣FreeBSD;␣armel",
"X11;␣FreeBSD;␣armhf",
"X11;␣FreeBSD;␣hppa",
"X11;␣FreeBSD;␣ia64",
"X11;␣FreeBSD;␣mips",
"X11;␣FreeBSD;␣mipsel",
"X11;␣FreeBSD;␣powerpc",
"X11;␣FreeBSD;␣ppc",
"X11;␣FreeBSD;␣powerpcspe",
"X11;␣FreeBSD;␣ppc64",
"X11;␣FreeBSD;␣ppc64el",
"X11;␣FreeBSD;␣s390x",
"X11;␣FreeBSD;␣sparc",
"X11;␣FreeBSD;␣sparc64",
"X11;␣NetBSD;␣amd64",
"X11;␣NetBSD;␣x86_64",
"X11;␣NetBSD;␣i686",
"X11;␣NetBSD;␣i686␣on␣x86_64",
"X11;␣NetBSD;␣i686␣on␣amd64",
"X11;␣NetBSD;␣i586",
"X11;␣NetBSD;␣i386",
"X11;␣NetBSD;␣arm",
"X11;␣NetBSD;␣arm64",
"X11;␣NetBSD;␣armel",
"X11;␣NetBSD;␣armhf",
"X11;␣NetBSD;␣hppa",
"X11;␣NetBSD;␣ia64",
"X11;␣NetBSD;␣mips",
"X11;␣NetBSD;␣mipsel",
"X11;␣NetBSD;␣powerpc",
"X11;␣NetBSD;␣ppc",
"X11;␣NetBSD;␣powerpcspe",
"X11;␣NetBSD;␣ppc64",
"X11;␣NetBSD;␣ppc64el",
"X11;␣NetBSD;␣s390x",
"X11;␣NetBSD;␣sparc",
"X11;␣NetBSD;␣sparc64",
"X11;␣Linux;␣amd64",
"X11;␣Linux;␣x86_64",
"X11;␣Linux;␣i686",
"X11;␣Linux;␣i686␣on␣x86_64",
"X11;␣Linux;␣i686␣on␣amd64",
"X11;␣Linux;␣i586",
"X11;␣Linux;␣i486",
"X11;␣Linux;␣i386",
"X11;␣Linux;␣arm",
"X11;␣Linux;␣arm64",
"X11;␣Linux;␣armel",
"X11;␣Linux;␣armhf",
"X11;␣Linux;␣hppa",
"X11;␣Linux;␣ia64",
"X11;␣Linux;␣mips",
"X11;␣Linux;␣mipsel",
"X11;␣Linux;␣powerpc",
"X11;␣Linux;␣ppc",
"X11;␣Linux;␣powerpcspe",
"X11;␣Linux;␣ppc64",
"X11;␣Linux;␣ppc64el",
"X11;␣Linux;␣s390x",
"X11;␣Linux;␣sparc",
"X11;␣Linux;␣sparc64",
"X11;␣Ubuntu;␣Linux;␣amd64",
"X11;␣Ubuntu;␣Linux;␣x86_64",
"X11;␣Ubuntu;␣Linux;␣i686",
"X11;␣Ubuntu;␣Linux;␣i686␣on␣x86_64",
"X11;␣Ubuntu;␣Linux;␣i686␣on␣amd64",
"X11;␣Ubuntu;␣Linux;␣i586",
"X11;␣Ubuntu;␣Linux;␣i386",
"X11;␣Ubuntu;␣Linux;␣arm",
"X11;␣Ubuntu;␣Linux;␣arm64",
"X11;␣Ubuntu;␣Linux;␣armel",
"X11;␣Ubuntu;␣Linux;␣armhf",
"X11;␣Ubuntu;␣Linux;␣hppa",
"X11;␣Ubuntu;␣Linux;␣ia64",
```

```
    "X11; Ubuntu; Linux; mips",
    "X11; Ubuntu; Linux; mipsel",
    "X11; Ubuntu; Linux; powerpc",
    "X11; Ubuntu; Linux; ppc",
    "X11; Ubuntu; Linux; powerpcspe",
    "X11; Ubuntu; Linux; ppc64",
    "X11; Ubuntu; Linux; ppc64el",
    "X11; Ubuntu; Linux; s390x",
    "X11; Ubuntu; Linux; sparc",
    "X11; Ubuntu; Linux; sparc64",

    # Macintosh
    "Macintosh; Intel Mac OS X 10.4",
    "Macintosh; PPC Mac OS X 10.4",
    "Macintosh; Intel Mac OS X 10.5",
    "Macintosh; PPC Mac OS X 10.5",
    "Macintosh; Intel Mac OS X 10.6",
    "Macintosh; PPC Mac OS X 10.6",
    "Macintosh; Intel Mac OS X 10.7",
    "Macintosh; PPC Mac OS X 10.7",
    "Macintosh; Intel Mac OS X 10.8",
    "Macintosh; PPC Mac OS X 10.8",
    "Macintosh; Intel Mac OS X 10.9",
    "Macintosh; PPC Mac OS X 10.9",
    "Macintosh; Intel Mac OS X 10.10",
    "Macintosh; PPC Mac OS X 10.10",
    "Macintosh; Intel Mac OS X 10_10",
]

revisions = [
    "rv:33.0a1",
    "rv:33.0a2",
    "rv:33.0b1",
    "rv:33.0",
    "rv:33.0.1",
    "rv:33.0.2",
    "rv:33.0.3",
    "rv:33.1",
    "rv:33.1.1",
    "rv:34.0a1",
    "rv:34.0a2",
    "rv:34.0b1",
    "rv:34.0",
    "rv:34.0.1",
    "rv:34.0.2",
    "rv:34.0.3",
    "rv:34.0.4",
    "rv:34.0.5",
    "rv:35.0a1",
    "rv:35.0a2",
    "rv:35.0b1",
    "rv:35.0",
    "rv:35.0.1",
    "rv:36.0a1",
    "rv:36.0a2",
    "rv:36.0b1",
    "rv:36.0",
    "rv:36.0.1",
    "rv:36.0.2",
    "rv:36.0.3",
    "rv:36.0.4",
    "rv:37.0a1",
    "rv:37.0a2",
    "rv:37.0b1",
    "rv:37.0",
    "rv:37.0.1",
    "rv:38.0a1",
    "rv:38.0a2",
    "rv:38.0b1",
    "rv:38.0",
    "rv:39.0a1",
    "rv:39.0a2",
    "rv:39.0b1",
    "rv:39.0",
    "rv:40.0a1",
    "rv:40.0a2",
    "rv:40.0b1",
    "rv:41.0a1",
    "rv:41.0a2",
    "rv:42.0a1"
]

parentheses = ["%s; %s" % x for x in product(platforms, revisions)] + platforms + revisions
```

```
hasResult = False
for parenthese in parentheses:
    result = attempt(parenthese)
    if result:
        hasResult = True
        print "{parenthese: %r, guess: %r}" % (parenthese, guess(result))
        open("stage5.zip", "w+").write(result)
        break
if not hasResult:
    print "No result found :("
```

interpret.py

```python
# -*- coding: utf8 -*-

from bisect import bisect, insort
from collections import namedtuple
from itertools import islice

Inf = float("+inf")

Instruction = namedtuple("Instruction", ["code", "data", "length", "offset"])
Zone = namedtuple("Zone", ["start", "end", "final"])
Edge = namedtuple("Edge", ["source", "sink", "fallthrough"])

# Décodage des instructions
# ========================

def decode(bytestream, offset = None):
    icode, idata, ileng = None, 0, 0
    for byte in bytestream:
        if isinstance(byte, str):
            byte = ord(byte)
        ileng += 1
        bcode, bdata = byte >> 4, byte & 0xf
        idata = (idata | bdata) % 0x100000000
        if bcode in (0x2, 0x6):
            if bcode == 0x6:
                idata = (~idata) % 0x100000000
            idata = (idata << 4) % 0x100000000
        else:
            icode = bcode
            break
    if icode != None:
        if idata > 0x7fffffff:
            idata = idata - 0x100000000
        return Instruction(icode, idata, ileng, offset)
    return None

def target(ins):
    if not ins.code in (0x0, 0x9, 0xa):
        raise ValueError("not␣a␣jump/call␣instruction")
    if ins.offset != None:
        return ins.offset + ins.length + ins.data
    return None


# Construction du CFG
# ===================

def nextjump(program, offset = 0):
    stream = islice(program, offset, None)
    ins = decode(stream, offset)
    while ins and ins.code not in (0x0, 0xa):
        offset = offset + ins.length
        ins = decode(stream, offset)
    return (offset, ins)

def contains(list, elem):
    idx = bisect(list, elem) - 1
    if idx == -1:
        return False
    return list[idx] == elem

def merge(S, E):
    S, E = iter(S), iter(E)
    s, e = next(S, None), next(E, None)
    while s != None and e != None:
        i, s_, e_ = 0, s, e
        if s <= e:
            i, s_ = i + 1, next(S, None)
            while s_ == s:
                i, s_ = i + 1, next(S, None)
        if e <= s:
            i, e_ = i - 1, next(E, None)
            while e_ == e:
                i, e_ = i - 1, next(E, None)
        yield (min(s, e), i)
        s, e = s_, e_
    while s != None:
        i, s_ = 1, next(S, None)
        while s_ == s:
            i, s_ = i + 1, next(S, None)
```

```
        yield (s, i)
        s = s_
    while e != None:
        i, e_ = -1, next(E, None)
        while e_ == e:
            i, e_ = i - 1, next(E, None)
        yield (e, i)
        e = e_

def walk(program, offset = 0):
    starts, ends = [], []
    stack = [offset]
    while stack:
        pos = stack.pop()
        if contains(starts, pos):
            continue
        insort(starts, pos)
        pos, ins = nextjump(program, pos)
        if ins:
            pos = pos + ins.length
            if ins.code == 0xa:
                stack.append(pos)
            stack.append(target(ins))
        insort(ends, pos)

    zones = []
    cnt, ppos = 0, None
    for pos, incr in merge(starts, ends):
        if cnt != 0:
            lpos = pos - 1
            while lpos > ppos and (ord(program[lpos - 1]) >> 4) in (0x2, 0x6):
                lpos = lpos - 1
            zones.append(Zone(ppos, pos, decode(program[lpos:pos], lpos)))
        cnt, ppos = cnt + incr, pos
        if cnt < 0:
            raise RuntimeError("zone␣count␣negative")
    if cnt != 0:
        raise RuntimeError("runaway␣zone")

    edges = []
    for zone in zones:
        if zone.final.code in (0x0, 0xa):
            edges.append(Edge(zone.start, target(zone.final), False))
        if zone.final.code != 0x0 and contains(starts, zone.end):
            edges.append(Edge(zone.start, zone.end, True))

    return (zones, edges)

def graphviz(path, edges):
    f = open(path, "w+")
    f.write("digraph␣{\n");
    for edge in edges:
        f.write("␣␣␣␣\"%x\"␣->␣\"%x\";\n" % (edge.source, edge.sink))
    f.write("}\n")
    f.close()

# Conversion de nombres en entier 32-bits signé ou non-signé
# ==========================================================

def signed(num):
    num = num % 0x100000000
    return num if num < 0x80000000 else num - 0x100000000

def unsigned(num):
    return num % 0x100000000



# Identification du type d'une valeur
# ==================================

def isint(val):
    """Is␣this␣value␣an␣integer?"""
    return isinstance(val, int) or isinstance(val, long)

def isstr(val):
    """Is␣this␣value␣a␣string?"""
    return isinstance(val, str)

def isadd(val):
    """Is␣this␣value␣an␣addition?"""
    return isinstance(val, Add)
```

```
def ismul(val):
    """Is␣this␣value␣a␣multiplication?"""
    return isinstance(val, Mul)

def isneg(val):
    """Is␣this␣value␣a␣negation?"""
    return isinstance(val, Neg)

def ismod(val):
    """Is␣this␣value␣a␣modulus?"""
    return isinstance(val, Mod)

def isand(val):
    """Is␣this␣value␣a␣bitwise␣and?"""
    return isinstance(val, And)

def isor(val):
    """Is␣this␣value␣a␣bitwise␣or?"""
    return isinstance(val, Or)

def isxor(val):
    """Is␣this␣value␣a␣xor?"""
    return isinstance(val, Xor)

def isnot(val):
    """Is␣this␣value␣a␣bitwise␣not?"""
    return isinstance(val, Not)

def isshr(val):
    """Is␣this␣value␣a␣right␣shift?"""
    return isinstance(val, Shr)

def isshl(val):
    """Is␣this␣value␣a␣left␣shift?"""
    return isinstance(val, Shl)

def isequ(val):
    """Is␣this␣value␣a␣equal␣comparison?"""
    return isinstance(val, Equ)

def isgt(val):
    """Is␣this␣value␣a␣greater-than␣comparison?"""
    return isinstance(val, Gt)

def ismemword(val):
    """Is␣this␣value␣a␣word␣from␣memory?"""
    return isinstance(val, MemWord)

def ismembyte(val):
    """Is␣this␣value␣a␣byte␣from␣memory?"""
    return isinstance(val, MemByte)


# Différentes classes de valeur
# ============================

class Add(namedtuple("Add", ["left", "right"])):
    def __new__(cls, left, right):
        if isint(left) and not isint(right):
            # Ensure that the number is on the right
            left, right = right, left
        return super(Add, cls).__new__(cls, left, right)

    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 4, '+'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Mul(namedtuple("Mul", ["left", "right"])):
    def __new__(cls, left, right):
        if isint(right) and not isint(left):
            # Ensure that the number is on the left
            left, right = right, left
        return super(Mul, cls).__new__(cls, left, right)

    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 3, '*'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
```

```
        return r

class Neg(namedtuple("Neg", ["value"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 2, '-'
        r = "%s%s" % (symb, rep(s.value, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Mod(namedtuple("Mod", ["dividend", "divisor"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 3, '%'
        r = "%s%s%d" % (rep(s.dividend, prec, symb), symb, s.divisor)
        if pprec <= prec:
            r = "(%s)" % (r,)
        return r

class And(namedtuple("And", ["left", "right"])):
    def __new__(cls, left, right):
        if isint(left) and not isint(right):
            # Ensure that the number is on the right
            left, right = right, left
        return super(And, cls).__new__(cls, left, right)

    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 8, '&'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Or(namedtuple("Or", ["left", "right"])):
    def __new__(cls, left, right):
        if isint(left) and not isint(right):
            # Ensure that the number is on the right
            left, right = right, left
        return super(Or, cls).__new__(cls, left, right)

    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 10, '|'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Xor(namedtuple("Xor", ["left", "right"])):
    def __new__(cls, left, right):
        if isint(left) and not isint(right):
            # Ensure that the number is on the right
            left, right = right, left
        return super(Xor, cls).__new__(cls, left, right)

    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 9, '^'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Not(namedtuple("Not", ["value"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 2, '~'
        r = "%s%s" % (symb, rep(s.value, prec, symb))
        if pprec < prec or (pprec == prec and psymb != symb):
            r = "(%s)" % (r,)
        return r

class Shr(namedtuple("Shr", ["value", "shift"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 5, '>>'
        r = "%s%s%d" % (rep(s.value, prec, symb), symb, s.shift)
        if pprec <= prec:
            r = "(%s)" % (r,)
        return r

class Shl(namedtuple("Shl", ["value", "shift"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 5, '<<'
        r = "%s%s%d" % (rep(s.value, prec, symb), symb, s.shift)
        if pprec <= prec:
            r = "(%s)" % (r,)
        return r
```

```
class Equ(namedtuple("Equ", ["left", "right"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 7, '=='
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec <= prec:
            r = "(%s)" % (r,)
        return r

class Gt(namedtuple("Gt", ["left", "right"])):
    def rep(s, pprec = Inf, psymb = None):
        prec, symb = 6, '>'
        r = "%s%s%s" % (rep(s.left, prec, symb), symb, rep(s.right, prec, symb))
        if pprec <= prec:
            r = "(%s)" % (r,)
        return r

class MemWord(namedtuple("MemWord", ["tag", "addr"])):
    def rep(s, pprec = Inf, psymb = None):
        r = "[%s:%s]" % (s.tag, rep(s.addr))
        return r

class MemByte(namedtuple("MemByte", ["tag", "addr"])):
    def rep(s, pprec = Inf, psymb = None):
        r = "[[%s:%s]]" % (s.tag, rep(s.addr))
        return r



# Fonctions pour faire des calculs sur les valeurs
# ================================================

def rep(val, pprec = Inf, psymb = None):
    """Get␣the␣string␣representation␣of␣a␣value."""
    try:
        return val.rep(pprec, psymb).replace("+-", "-")
    except AttributeError:
        if isint(val):
            s, u = signed(val), unsigned(val)
            if -65536 < s < 0:
                return "%d" % (s,)
            else:
                if u < 65536:
                    return "%d" % (u,)
                else:
                    return "0x%x" % (u,)
        elif isstr(val):
            return val
        elif val == None:
            return "Unknow"
        else:
            raise TypeError("invalid␣value")

def add(left, right):
    """Add␣two␣values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return (left + right) % 0x100000000
    if isint(left) and isadd(right) and isint(right.right):
        left, right = right.left, (left + right.right) % 0x100000000
    if isadd(left) and isint(left.right) and isint(right):
        left, right = left.left, (left.right + right) % 0x100000000
    if right == 0:
        return left
    if left == 0:
        return right
    return Add(left, right)

def mul(left, right):
    """Multiply␣two␣values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return (left * right) % 0x100000000
    if isint(left) and ismul(right) and isint(right.left):
        left, right = (left * right.left) % 0x100000000, right.right
    if ismul(left) and isint(left.left) and isint(right):
        left, right = (left.left * right) % 0x100000000, left.right
    if right == 1:
        return left
    if left == 1:
        return right
```

```
    if left == 0 or right == 0:
        return 0
    return Mul(left, right)

def neg(value):
    """Negate␣a␣value."""
    if value == None:
        return None
    if isint(value):
        return (-value) % 0x100000000
    if isneg(value):
        return value.value
    return Neg(value)

def sub(left, right):
    """Subtract␣two␣values."""
    return add(left, neg(right))

def mod_(dividend, divisor):
    """Compute␣remainder␣of␣the␣Euclidian␣division␣of␣a␣value␣by␣a␣constant."""
    if dividend == None or divisor == None:
        return None
    if not isint(divisor):
        raise RuntimeError("divisor␣not␣constant")
    divisor = signed(divisor)
    if divisor <= 0:
        raise RuntimeError("divisor␣non-positive")
    if divisor == 1:
        return 0
    if isint(dividend):
        dividend = signed(dividend)
        if dividend < 0:
            raise RuntimeError("dividend␣negative")
        return (dividend % divisor) % 0x100000000
    if ismod(dividend):
        if dividend.divisor % divisor == 0:
            dividend = dividend.dividend
        elif divisor % dividend.divisor == 0:
            return dividend
    return Mod(dividend, divisor)

def and_(left, right):
    """Do␣the␣bitwise␣and␣of␣two␣values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return (unsigned(left) & unsigned(right)) % 0x100000000
    if isint(left) and isand(right) and isint(right.right):
        left, right = right.left, (unsigned(left) & unsigned(right.right)) % 0x100000000
    if isand(left) and isint(left.right) and isint(right):
        left, right = left.left, (unsigned(left.right) & unsigned(right)) % 0x100000000
    if right == 0xffffffff:
        return left
    if left == 0xffffffff:
        return right
    if left == 0 or right == 0:
        return 0
    return And(left, right)

def or_(left, right):
    """Do␣the␣bitwise␣or␣of␣two␣values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return (unsigned(left) | unsigned(right)) % 0x100000000
    if isint(left) and isor(right) and isint(right.right):
        left, right = right.left, (unsigned(left) | unsigned(right.right)) % 0x100000000
    if isor(left) and isint(left.right) and isint(right):
        left, right = left.left, (unsigned(left.right) | unsigned(right)) % 0x100000000
    if right == 0:
        return left
    if left == 0:
        return right
    if left == 0xffffffff or right == 0xffffffff:
        return 0
    return Or(left, right)

def xor_(left, right):
    """Do␣the␣xor␣of␣two␣values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return (unsigned(left) ^ unsigned(right)) % 0x100000000
```

```
    if isint(left) and isxor(right) and isint(right.right):
        left, right = right.left, (unsigned(left) ^ unsigned(right.right)) % 0x100000000
    if isxor(left) and isint(left.right) and isint(right):
        left, right = left.left, (unsigned(left.right) ^ unsigned(right)) % 0x100000000
    if right == 0:
        return left
    if left == 0:
        return right
    if right == 0xffffffff:
        return not_(left)
    if left == 0xffffffff:
        return not_(right)
    return Xor(left, right)

def not_(value):
    """Do the bitwise not of a value."""
    if value == None:
        return None
    if isint(value):
        return (~unsigned(value)) % 0x100000000
    if isnot(value):
        return value.value
    return Not(value)

def shr_(value, shift):
    """Shift to the right a value by a constant amount."""
    if value == None or shift == None:
        return None
    if not isint(shift):
        raise RuntimeError("shift not constant")
    if isint(value):
        return (unsigned(value) >> shift) % 0x100000000
    if shift == 0:
        return value
    if isshr(value) and isint(value.shift):
        value, shift = value.value, (value.shift + shift)
    if shift >= 32:
        return 0
    return Shr(value, shift)

def shl_(value, shift):
    """Shift to the left a value by a constant amount."""
    if value == None or shift == None:
        return None
    if not isint(shift):
        raise RuntimeError("shift not constant")
    if isint(value):
        return (unsigned(value) << shift) % 0x100000000
    if shift == 0:
        return value
    if isshl(value) and isint(value.shift):
        value, shift = value.value, (value.shift + shift)
    if shift >= 32:
        return 0
    return Shl(value, shift)

def equ(left, right):
    """Compare for equlity two values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return int(signed(left) == signed(right))
    return Equ(left, right)

def gt(left, right):
    """Compare for greater-than two values."""
    if left == None or right == None:
        return None
    if isint(left) and isint(right):
        return int(signed(left) > signed(right))
    return Gt(left, right)

tagCnt = 0
def memword(addr):
    """Get a word from memory."""
    global tagCnt
    tag = "tag%d" % (tagCnt,)
    tagCnt += 1
    print "; %s = [%s]" % (tag, rep(addr))
    return MemWord(tag, addr)

def membyte(addr):
    """Get a byte from memory."""
```

```
    global tagCnt
    tag = "tag%d" % (tagCnt,)
    tagCnt += 1
    print "; %s = [[%s]]" % (tag, rep(addr))
    return MemByte(tag, addr)


# Gestion de la mémoire...
# ========================

class Memory:
    def __init__(s):
        s.words = {}
        s.bytes = {}
        s.cycle = 0

    def checkaddr(s, addr):
        if isint(addr):
            return addr
        if isstr(addr):
            return addr
        if isadd(addr) and isstr(addr.left) and isint(addr.right):
            return add(addr.left, unsigned(addr.right))
        raise ValueError("bad address")

    def setword(s, addr, val):
        addr = s.checkaddr(addr)
        for i in range(0, 4):
            addr2 = add(addr, i)
            if addr2 in s.bytes:
                del s.bytes[addr2]
        if val == None and addr in s.words:
            del s.words[addr]
            return
        s.words[addr] = (val, s.cycle)
        s.cycle += 1

    def setbyte(s, addr, val):
        addr = s.checkaddr(addr)
        for i in range(-3, 4):
            addr2 = add(addr, i)
            if addr2 in s.words:
                del s.words[addr2]
        if val == None and addr in s.bytes:
            del s.bytes[addr]
            return
        s.bytes[addr] = (val, s.cycle)
        s.cycle += 1

    def getword(s, addr):
        addr = s.checkaddr(addr)
        if addr in s.words:
            return s.words[addr][0]
        return None

    def getbyte(s, addr):
        addr = s.checkaddr(addr)
        if addr in s.bytes:
            return s.bytes[addr][0]
        if addr in s.words:
            return and_(s.words[addr][0], 255)
        return None

    def invalidate(s, start, count = 256):
        try:
            start = s.checkaddr(start)
            if not isint(count):
                print "; Using default count value for memory invalidation!"
                count = 256
            for i in range(count):
                addr = add(start, i)
                if addr in s.bytes:
                    del s.bytes[addr]
            for  i in range(-3, count):
                addr = add(start, i)
                if addr in s.words:
                    del s.words[addr]
        except ValueError:
            pass # s.invalidateall() - disabled for prog8

    def invalidateall(s):
        print "; Whole memory invalidated!"
```

```
        s.words = {}
        s.bytes = {}

def setword(addr, val, mem = None):
    print "[%s]␣<--␣%s" % (rep(addr), rep(val))
    if mem:
        try:
            mem.setword(addr, val)
        except ValueError:
            pass

def setbyte(addr, val, mem = None):
    print "[[%s]]␣<--␣%s" % (rep(addr), rep(val))
    if mem:
        try:
            mem.setbyte(addr, val)
        except ValueError:
            pass

def getword(addr, mem = None):
    if mem:
        try:
            w = mem.getword(addr)
            if w != None:
                return w
        except ValueError:
            pass
    return memword(addr)

def getbyte(addr, mem = None):
    if mem:
        try:
            b = mem.getbyte(addr)
            if b != None:
                return b
        except ValueError:
            pass
    return membyte(addr)

def invalidate(start, count, mem = None):
    if mem:
        mem.invalidate(start, count)

def invalidateall(mem = None):
    if mem:
        mem.invalidateall()


# Exécution symbolique
# ====================

PrimaryInstructions = [
    "j",
    "ldlp",
    "pfix",
    "ldnl",
    "ldc",
    "ldnlp",
    "nfix",
    "ldl",
    "adc",
    "call",
    "cj",
    "ajw",
    "eqc",
    "stl",
    "stnl",
    "opr"
]

SecondaryInstructions = {
    0x01: "lb",
    0x02: "bsub",
    0x06: "gcall",
    0x07: "in",
    0x08: "prod",
    0x09: "gt",
    0x0a: "wsub",
    0x0b: "out",
    0x1b: "ldpi",
    0x1f: "rem",
    0x20: "ret",
    0x33: "xor",
```

```
    0x3b: "sb",
    0x3c: "gajw",
    0x40: "shr",
    0x41: "shl",
    0x42: "mint",
    0x46: "and",
    0x5a: "dup",
    0xc1: "ssub"
}

class JumpError(Exception):
    pass

class SymbolicMachine:
    def __init__(s, prog, verb = False, mem = None, Iptr = 0, Wptr = 'W', Areg = None, Breg = None, Creg =
         ↪ None):
        s.prog = prog
        s.verb = verb
        s.mem = mem
        s.Iptr = Iptr
        s.Wptr = Wptr
        s.Areg = Areg
        s.Breg = Breg
        s.Creg = Creg
        if s.verb:
            print ";␣Areg␣=␣%s,␣Breg␣=␣%s,␣Creg␣=␣%s,␣Wptr␣=␣%s,␣Iptr␣=␣%s" % (rep(s.Areg), rep(s.Breg),
                 ↪ rep(s.Creg), rep(s.Wptr), rep(s.Iptr))

    def rununtiljump(s, end):
        """Execute␣le␣programme␣jusqu'a␣atteindre␣un␣saut␣ou␣une␣certaine␣valeur␣du␣compteur␣ordinal."""
        while s.Iptr != end and decode(islice(s.prog, s.Iptr, None), s.Iptr).code not in (0x0, 0xa):
            s.do()

    def do(s):
        """Execute␣une␣instruction␣du␣programme."""
        ins = decode(islice(s.prog, s.Iptr, None), s.Iptr)
        s.Iptr = s.Iptr + ins.length
        handler = getattr(s, 'do_' + PrimaryInstructions[ins.code])(ins)
        if not isint(s.Iptr):
            raise RuntimeError("Iptr␣not␣a␣int")
        if s.verb:
            print ";␣Areg␣=␣%s,␣Breg␣=␣%s,␣Creg␣=␣%s,␣Wptr␣=␣%s,␣Iptr␣=␣%s" % (rep(s.Areg), rep(s.Breg),
                 ↪ rep(s.Creg), rep(s.Wptr), rep(s.Iptr))

    def do_j(s, ins):
        """Jump"""
        if s.verb:
            print ";␣j␣%s␣(target␣=␣0x%x)" % (rep(ins.data), target(ins))
        s.Iptr, s.Areg, s.Breg, s.Creg = target(ins), None, None, None

    def do_ldlp(s, ins):
        """LoaD␣Local␣Pointer"""
        if s.verb:
            print ";␣ldlp␣%s" % (rep(ins.data),)
        s.Areg, s.Breg, s.Creg = add(s.Wptr, 4 * ins.data), s.Areg, s.Breg

    def do_pfix(s, ins):
        """Positive␣preFIX"""
        raise RuntimeError("pfix␣instruction")

    def do_ldnl(s, ins):
        """LoaD␣Non-Local"""
        if s.verb:
            print ";␣ldnl␣%s" % (rep(ins.data),)
        s.Areg = getword(add(s.Areg, 4 * ins.data), s.mem)

    def do_ldc(s, ins):
        """LoaD␣Constant"""
        if s.verb:
            print ";␣ldc␣%s" % (rep(ins.data),)
        s.Areg, s.Breg, s.Creg = ins.data, s.Areg, s.Breg

    def do_ldnlp(s, ins):
        """LoaD␣Non-Local␣Pointer"""
        if s.verb:
            print ";␣ldnlp␣%s" % (rep(ins.data),)
        s.Areg = add(s.Areg, 4 * ins.data)

    def do_nfix(s, ins):
        """Negative␣preFIX"""
        raise RuntimeError("nfix␣instruction")

    def do_ldl(s, ins):
```

```
    """LoaD␣Local"""
    if s.verb:
        print ";␣ldl␣%s" % (rep(ins.data),)
    s.Areg, s.Breg, s.Creg = getword(add(s.Wptr, 4 * ins.data), s.mem), s.Areg, s.Breg

def do_adc(s, ins):
    """ADd␣Constant"""
    if s.verb:
        print ";␣adc␣%s" % (rep(ins.data),)
    s.Areg = add(s.Areg, ins.data)

def do_call(s, ins):
    """CALL"""
    if s.verb:
        print ";␣call␣%s␣(target␣=␣0x%x)" % (rep(ins.data), target(ins))
    s.Wptr = sub(s.Wptr, 16)
    setword(add(s.Wptr, 0), s.Iptr, s.mem)
    setword(add(s.Wptr, 4), s.Areg, s.mem)
    setword(add(s.Wptr, 8), s.Breg, s.mem)
    setword(add(s.Wptr, 12), s.Creg, s.mem)
    s.Iptr, s.Areg, s.Breg, s.Creg = ins.offset + ins.length + ins.data, s.Iptr, None, None

def do_cj(s, ins):
    """Conditional␣Jump"""
    if s.verb:
        print ";␣cj␣%s␣(target␣=␣0x%x)" % (rep(ins.data), target(ins))
    if isint(s.Areg):
        if s.Areg == 0: # branch taken
            s.Iptr, s.Areg, s.Breg, s.Creg = target(ins), None, None, None
        else:           # branch not taken
            s.Areg, s.Breg, s.Creg = s.Breg, s.Creg, None
    else:
        raise JumpError("cj␣to␣%x␣if␣%s␣is␣null" % (target(ins), rep(s.Areg)))

def do_ajw(s, ins):
    """AdJust␣Workspace"""
    if s.verb:
        print ";␣ajw␣%s" % (rep(ins.data),)
    s.Wptr = add(s.Wptr, 4 * ins.data)

def do_eqc(s, ins):
    """EQuals␣Constant"""
    if s.verb:
        print ";␣eqc␣%s" % (rep(ins.data),)
    s.Areg = equ(s.Areg, ins.data)

def do_stl(s, ins):
    """STore␣Local"""
    if s.verb:
        print ";␣stl␣%s" % (rep(ins.data),)
    setword(add(s.Wptr, 4 * ins.data), s.Areg, s.mem)
    s.Areg, s.Breg, s.Creg = s.Breg, s.Creg, None

def do_stnl(s, ins):
    """STore␣Non-Local"""
    if s.verb:
        print ";␣stnl␣%s" % (rep(ins.data),)
    setword(add(s.Areg, 4 * ins.data), s.Breg, s.mem)
    s.Areg, s.Breg, s.Creg = s.Creg, None, None

def do_opr(s, ins):
    """OPeRate"""
    if ins.data not in SecondaryInstructions:
        raise RuntimeError("unknown␣opr␣(opcode:␣%02x)" % (ins.data,))
    if not hasattr(s, 'do_' + SecondaryInstructions[ins.data]):
        raise RuntimeError("unimplemented␣opr␣(opname:␣%s)" % (SecondaryInstructions[ins.data],))
    getattr(s, 'do_' + SecondaryInstructions[ins.data])(ins)

def do_lb(s, ins):
    """Load␣Byte"""
    if s.verb:
        print ";␣lb"
    s.Areg = getbyte(s.Areg, s.mem)

def do_bsub(s, ins):
    """Byte␣SUBscript"""
    if s.verb:
        print ";␣bsub"
    s.Areg, s.Breg, s.Creg = add(s.Areg, s.Breg), s.Creg, None

def do_gcall(s, ins):
    """General␣CALL"""
    raise RuntimeError("gcall␣to␣%s␣--␣not␣implemented␣yet" % (rep(s.Areg),))
```

```
def do_in(s, ins):
    """INput␣message"""
    if s.verb:
        print ";␣in"
    print "recv(%s,␣%s,␣%s)" % (rep(s.Breg), rep(s.Creg), rep(s.Areg))
    invalidate(s.Creg, s.Areg, s.mem)
    s.Areg, s.Breg, s.Creg = None, None, None

def do_prod(s, ins):
    """PRODuct"""
    if s.verb:
        print ";␣prod"
    s.Areg, s.Breg, s.Creg = mul(s.Areg, s.Breg), s.Creg, None

def do_gt(s, ins):
    """Greater-Than"""
    if s.verb:
        print ";␣gt"
    s.Areg, s.Breg, s.Creg = gt(s.Breg, s.Areg), s.Creg, None

def do_wsub(s, ins):
    """Word␣SUBscript"""
    s.Areg, s.Breg, s.Creg = add(s.Areg, mul(s.Breg, 4)), s.Creg, None

def do_out(s, ins):
    """OUTput␣message"""
    if s.verb:
        print ";␣out"
    print "send(%s,␣%s,␣%s)" % (rep(s.Breg), rep(s.Creg), rep(s.Areg))
    s.Areg, s.Breg, s.Creg = None, None, None

def do_ldpi(s, ins):
    """LoaD␣Pointer␣to␣Instruction"""
    if s.verb:
        print ";␣ldpi"
    s.Areg = add(s.Areg, s.Iptr)

def do_rem(s, ins):
    """REMainder"""
    if s.verb:
        print ";␣rem"
    s.Areg, s.Breg, s.Creg = mod_(s.Breg, s.Areg), s.Creg, None

def do_ret(s, ins):
    """RETurn"""
    if s.verb:
        print ";␣ret"
    s.Iptr = getword(s.Wptr, s.mem)
    s.Wptr = add(s.Wptr, 16)
    if not isint(s.Iptr):
        raise JumpError("return␣address␣left␣unresolved")

def do_xor(s, ins):
    """XOR"""
    if s.verb:
        print ";␣xor"
    s.Areg, s.Breg, s.Creg = xor_(s.Areg, s.Breg), s.Creg, None

def do_sb(s, ins):
    """Store␣Byte"""
    if s.verb:
        print ";␣sb"
    setbyte(s.Areg, and_(s.Breg, 255), s.mem)
    s.Areg, s.Breg, s.Creg = s.Creg, None, None

def do_gajw(s, ins):
    """General␣AdJust␣Workspace"""
    if s.verb:
        print ";␣gajw"
    s.Wptr, s.Areg = s.Areg, s.Wptr

def do_shr(s, ins):
    """SHift␣Right"""
    if s.verb:
        print ";␣shr"
    s.Areg, s.Breg, s.Creg = shr_(s.Breg, s.Areg), s.Creg, None

def do_shl(s, ins):
    """SHift␣Left"""
    if s.verb:
        print ";␣shl"
    s.Areg, s.Breg, s.Creg = shl_(s.Breg, s.Areg), s.Creg, None
```

```
    def do_mint(s, ins):
        """Minimum␣INT"""
        if s.verb:
            print ";␣mint"
        s.Areg, s.Breg, s.Creg = 0x80000000, s.Areg, s.Breg

    def do_and(s, ins):
        """AND"""
        s.Areg, s.Breg, s.Creg = and_(s.Areg, s.Breg), s.Creg, None

    def do_dup(s, ins):
        """DUPplicate␣top␣of␣stack"""
        s.Areg, s.Breg, s.Creg = s.Areg, s.Areg, s.Breg

    def do_ssub(s, ins):
        """Sixteen␣SUBscript"""
        s.Areg, s.Breg, s.Creg = add(s.Areg, mul(s.Breg, 2)), s.Creg, None

def disassemble(prog, offset = 0, verbose = False):
    zones, edges = walk(prog, offset)
    for zone in zones:
        print ""
        print ";␣==␣ZONE␣[%x;%x[␣==" % (zone.start, zone.end)
        sm = SymbolicMachine(prog, mem=Memory(), Iptr=zone.start, verb=verbose)
        sm.rununtiljump(zone.end)
        if sm.Iptr != zone.end: # There is a jump remaining!
            if zone.final.code == 0x0:
                print "j␣to␣%x" % (target(zone.final),)
            elif zone.final.code == 0xa:
                print "cj␣to␣%x␣if␣%s␣is␣null" % (target(zone.final), rep(sm.Areg))
            else:
                raise RuntimeError("where␣is␣my␣jump?!")
```

simulator.py

```
# -*- coding: utf8 -*-

from bisect import bisect, insort
from collections import namedtuple
from itertools import islice
from collections import defaultdict
from multiprocessing import Queue, Process
from struct import unpack
from time import sleep

Instruction = namedtuple("Instruction", ["code", "data", "length", "offset"])

def sword(num):
    num = num % 0x100000000
    return num if num < 0x80000000 else num - 0x100000000

def ssixt(num):
    num = num % 0x10000
    return num if num < 0x8000 else num - 0x10000

def sbyte(num):
    num = num % 0x100
    return num if num < 0x80 else num - 0x100

def uword(num):
    return num % 0x100000000

def usixt(num):
    return num % 0x10000

def ubyte(num):
    return num % 0x100

PrimaryInstructions = [
    "j",
    "ldlp",
    "pfix",
    "ldnl",
    "ldc",
    "ldnlp",
    "nfix",
    "ldl",
    "adc",
    "call",
    "cj",
    "ajw",
    "eqc",
    "stl",
    "stnl",
    "opr"
]

SecondaryInstructions = {
    0x01: "lb",
    0x02: "bsub",
    0x06: "gcall",
    0x07: "in",
    0x08: "prod",
    0x09: "gt",
    0x0a: "wsub",
    0x0b: "out",
    0x1b: "ldpi",
    0x1f: "rem",
    0x20: "ret",
    0x33: "xor",
    0x3b: "sb",
    0x3c: "gajw",
    0x40: "shr",
    0x41: "shl",
    0x42: "mint",
    0x46: "and",
    0x5a: "dup",
    0xc1: "ssub"
}

class Machine:
    def __init__(s, name):
        s.Name = name
        s.Mem = defaultdict(lambda: 0x55)
        s.Areg = 0x00000000
        s.Breg = 0x00000000
```

```
    s.Creg = 0x00000000
    s.Wptr = 0x20000000
    s.Iptr = 0x00000000
    s.Chan = {}

# Ecriture dans la mémoire
# ========================
def setbuff(s, addr, buff):
    """Ecrit␣une␣chaine␣de␣caracteres␣dans␣la␣memoire."""
    for posi, char in enumerate(buff):
        s.Mem[uword(addr+posi)] = ubyte(ord(char))

def setword(s, addr, word):
    """Ecrit␣un␣mot␣(32␣bits)␣dans␣la␣memoire."""
    addr, word = uword(addr), uword(word)
    s.Mem[addr] = word & 0xff
    s.Mem[addr+1] = (word >> 8) & 0xff
    s.Mem[addr+2] = (word >> 16) & 0xff
    s.Mem[addr+3] = word >> 24

def setsixt(s, addr, sixt):
    """Ecrit␣un␣demi-mot␣(16␣bits)␣dans␣la␣memoire."""
    addr, sixt = uword(addr), usixt(sixt)
    s.Mem[addr] = sixt & 0xff
    s.Mem[addr+1] = sixt >> 8

def setbyte(s, addr, byte):
    """Ecrit␣un␣octet␣(8␣bits)␣dans␣la␣memoire."""
    addr, byte = uword(addr), ubyte(byte)
    s.Mem[addr] = byte

# Lecture depuis la mémoire
# =========================
def getbuff(s, addr, leng):
    """Lit␣une␣chaine␣de␣caracteres␣depuis␣la␣memoire."""
    data = ""
    for i in range(leng):
        data = data + chr(s.Mem[uword(addr + i)])
    return data

def getword(s, addr):
    """Lit␣un␣mot␣(32␣bits)␣depuis␣la␣memoire."""
    addr = uword(addr)
    return s.Mem[addr] | (s.Mem[addr+1] << 8) | (s.Mem[addr+2] << 16) | (s.Mem[addr+3] << 24)

def getsixt(s, addr):
    """Lit␣un␣demi-mot␣(16␣bits)␣depuis␣la␣memoire."""
    addr = uword(addr)
    return s.Mem[addr] | (s.Mem[addr+1] << 8)

def getbyte(s, addr):
    """Lit␣un␣octet␣(8␣bits)␣depuis␣la␣memoire."""
    addr = uword(addr)
    return s.Mem[addr]

# Décodage d'une instruction
# ==========================
def decode(s, offset = None):
    """Decode␣l'instruction␣presente␣a␣la␣position␣indiquee."""
    if offset == None:
        offset = s.Iptr
    icode, idata, ileng, pos = None, 0, 0, offset
    while True:
        byte = s.getbyte(pos)
        ileng += 1
        bcode, bdata = byte >> 4, byte & 0xf
        idata = uword(idata | bdata)
        if bcode in (0x2, 0x6):
            if bcode == 0x6:
                idata = uword(~idata)
            idata = uword(idata << 4)
        else:
            icode = bcode
            break
        pos += 1
    if icode != None:
        return Instruction(icode, idata, ileng, offset)
    return None

def target(s, ins):
    """Indique␣la␣cible␣d'un␣saut␣ou␣d'un␣appel␣de␣procedure."""
    if not ins.code in (0x0, 0x9, 0xa):
        raise ValueError("not␣a␣jump/call␣instruction")
```

```
    if ins.offset != None:
        return ins.offset + ins.length + ins.data
    return None

# Routine de démarrage
# ====================
def start(s):
    chan = s.Chan[0x80000010]
    leng = ord(chan.get())
    data = ""
    for i in range(leng):
        data += chan.get()
    s.setbuff(s.Iptr, data)
    s.run()

# Routines principales d'exécution
# ================================
def run(s):
    """Execute␣le␣programme."""
    while True:
        s.runonce()

def runonce(s):
    """Execute␣une␣instruction␣du␣programme."""
    ins = s.decode()
    s.Iptr = s.Iptr + ins.length
    getattr(s, 'do_' + PrimaryInstructions[ins.code])(ins)

# Routine d'exécution des instructions principales
# ================================================
def do_j(s, ins):
    """Jump"""
    s.Iptr, s.Areg, s.Breg, s.Creg = s.target(ins), 0, 0, 0

def do_ldlp(s, ins):
    """LoaD␣Local␣Pointer"""
    s.Areg, s.Breg, s.Creg = uword(s.Wptr + 4*ins.data), s.Areg, s.Breg

def do_pfix(s, ins):
    """Positive␣preFIX"""
    raise RuntimeError("pfix␣instruction")

def do_ldnl(s, ins):
    """LoaD␣Non-Local"""
    s.Areg = s.getword(s.Areg + 4*ins.data)

def do_ldc(s, ins):
    """LoaD␣Constant"""
    s.Areg, s.Breg, s.Creg = ins.data, s.Areg, s.Breg

def do_ldnlp(s, ins):
    """LoaD␣Non-Local␣Pointer"""
    s.Areg = s.Areg + 4*ins.data

def do_nfix(s, ins):
    """Negative␣preFIX"""
    raise RuntimeError("nfix␣instruction")

def do_ldl(s, ins):
    """LoaD␣Local"""
    s.Areg, s.Breg, s.Creg = s.getword(s.Wptr + 4*ins.data), s.Areg, s.Breg

def do_adc(s, ins):
    """ADd␣Constant"""
    s.Areg = uword(s.Areg + ins.data)

def do_call(s, ins):
    """CALL"""
    s.Wptr = uword(s.Wptr - 16)
    s.setword(s.Wptr + 0, s.Iptr)
    s.setword(s.Wptr + 4, s.Areg)
    s.setword(s.Wptr + 8, s.Breg)
    s.setword(s.Wptr + 12, s.Creg)
    s.Iptr, s.Areg, s.Breg, s.Creg = s.target(ins), s.Iptr, 0, 0

def do_cj(s, ins):
    """Conditional␣Jump"""
    if s.Areg == 0: # branch taken
        s.Iptr = s.target(ins)
    else:           # branch not taken
        s.Areg, s.Breg, s.Creg = s.Breg, s.Creg, 0

def do_ajw(s, ins):
```

```
    """AdJust␣Workspace"""
    s.Wptr = uword(s.Wptr + 4*ins.data)

def do_eqc(s, ins):
    """EQuals␣Constant"""
    s.Areg = int(uword(s.Areg) == uword(ins.data))

def do_stl(s, ins):
    """STore␣Local"""
    s.setword(s.Wptr + 4*ins.data, s.Areg)
    s.Areg, s.Breg, s.Creg = s.Breg, s.Creg, 0

def do_stnl(s, ins):
    """STore␣Non-Local"""
    s.setword(s.Areg + 4*ins.data, s.Breg)
    s.Areg, s.Breg, s.Creg = s.Creg, 0, 0

def do_opr(s, ins):
    """OPeRate"""
    if ins.data not in SecondaryInstructions:
        raise RuntimeError("unknown␣opr␣(opcode:␣%02x)" % (ins.data,))
    if not hasattr(s, 'do_' + SecondaryInstructions[ins.data]):
        raise RuntimeError("unimplemented␣opr␣(opname:␣%s)" % (SecondaryInstructions[ins.data],))
    getattr(s, 'do_' + SecondaryInstructions[ins.data])(ins)

# Routines d'exécution des instructions secondaires
# ================================================
def do_lb(s, ins):
    """Load␣Byte"""
    s.Areg = s.getbyte(s.Areg)

def do_bsub(s, ins):
    """Byte␣SUBscript"""
    s.Areg, s.Breg, s.Creg = uword(s.Areg + s.Breg), s.Creg, 0

def do_gcall(s, ins):
    """General␣CALL"""
    s.Iptr, s.Areg = s.Areg, s.Iptr

def do_in(s, ins):
    """INput␣message"""
    data = ""
    leng, chan, addr = s.Areg, s.Chan[s.Breg], s.Creg
    for i in range(leng):
        data += chan.get()
    s.setbuff(addr, data)
    s.Areg, s.Breg, s.Creg = 0, 0, 0

def do_prod(s, ins):
    """PRODuct"""
    s.Areg, s.Breg, s.Creg = uword(s.Areg * s.Breg), s.Creg, 0

def do_gt(s, ins):
    """Greater-Than"""
    s.Areg, s.Breg, s.Creg = int(s.Breg > s.Areg), s.Creg, 0

def do_wsub(s, ins):
    """Word␣SUBscript"""
    s.Areg, s.Breg, s.Creg = uword(s.Areg + 4*s.Breg), s.Creg, 0

def do_out(s, ins):
    """OUTput␣message"""
    leng, chan, addr = s.Areg, s.Chan[s.Breg], s.Creg
    data = s.getbuff(addr, leng)
    for i in range(leng):
        chan.put(data[i])
    s.Areg, s.Breg, s.Creg = 0, 0, 0

def do_ldpi(s, ins):
    """LoaD␣Pointer␣to␣Instruction"""
    s.Areg = uword(s.Areg + s.Iptr)

def do_rem(s, ins):
    """REMainder"""
    s.Areg, s.Breg, s.Creg = uword(s.Breg % s.Areg), s.Creg, 0

def do_ret(s, ins):
    """RETurn"""
    s.Iptr = s.getword(s.Wptr,)
    s.Wptr = s.Wptr + 16

def do_xor(s, ins):
    """XOR"""
```

```
        s.Areg, s.Breg, s.Creg = uword(s.Areg ^ s.Breg), s.Creg, 0

    def do_sb(s, ins):
        """Store␣Byte"""
        s.setbyte(s.Areg, s.Breg)
        s.Areg, s.Breg, s.Creg = s.Creg, 0, 0

    def do_gajw(s, ins):
        """General␣AdJust␣Workspace"""
        s.Wptr, s.Areg = s.Areg, s.Wptr

    def do_shr(s, ins):
        """SHift␣Right"""
        s.Areg, s.Breg, s.Creg = uword(s.Breg >> s.Areg), s.Creg, 0

    def do_shl(s, ins):
        """SHift␣Left"""
        s.Areg, s.Breg, s.Creg = uword(s.Breg << s.Areg), s.Creg, 0

    def do_mint(s, ins):
        """Minimum␣INT"""
        s.Areg, s.Breg, s.Creg = 0x80000000, s.Areg, s.Breg

    def do_and(s, ins):
        """AND"""
        s.Areg, s.Breg, s.Creg = uword(s.Areg & s.Breg), s.Creg, 0

    def do_dup(s, ins):
        """DUPplicate␣top␣of␣stack"""
        s.Areg, s.Breg, s.Creg = s.Areg, s.Areg, s.Breg

    def do_ssub(s, ins):
        """Sixteen␣SUBscript"""
        s.Areg, s.Breg, s.Creg = uword(s.Areg + 2*s.Breg), s.Creg, 0

class Program:
    def __init__(s, prog):
        s.Chan = {}
        s.Prog = prog

    def start(s):
        inChan, outChan = s.Chan[0x80000010], s.Chan[0x80000000]

        prog = ""
        for i in range(ord(inChan.get())):
            prog += inChan.get()

        hdr = ""
        for i in range(12):
            hdr += inChan.get()

        msg = ""
        for i in range(unpack('<I', hdr[:4])[0]):
            msg += inChan.get()

        while True:
            data = ""
            for i in range(12):
                data += inChan.get()
            outChan.put(s.Prog(data))

class ProgA:
    """Somme"""
    def __init__(s):
        s.foo = 0

    def __call__(s, blk):
        for i in range(12):
            s.foo = (s.foo + ord(blk[i])) % 256
        return chr(s.foo)

class ProgB:
    """XOR"""
    def __init__(s):
        s.foo = 0

    def __call__(s, blk):
        for i in range(12):
            s.foo = (s.foo ^ ord(blk[i])) % 256
        return chr(s.foo)

class ProgC:
    """LSFR"""
```

```
    def __init__(s):
        s.state = 0
        s.initialized = False

    def __call__(s, blk):
        if not s.initialized:
            for i in range(12):
                s.state = (s.state + ord(blk[i])) % 65536
            s.initialized = True
        s.state = ((s.state << 1) & 65535) ^ ((s.state >> 14) & 1) ^ ((s.state >> 15) & 1)
        return chr(s.state % 256)

class ProgD:
    """XOR␣de␣la␣somme␣des␣six␣premiers␣octets␣et␣des␣six␣derniers␣octets"""
    def __call__(s, blk):
        s1 = chr(sum(ord(x) for x in blk[:6]) % 256)
        s2 = chr(sum(ord(x) for x in blk[6:]) % 256)
        return chr(ord(s1) ^ ord(s2))

class ProgE:
    """XOR␣des␣sommes␣des␣octets␣des␣quatre␣derniers␣blocs"""
    def __init__(s):
        s.state = 4*[12*chr(0)]

    def __call__(s, blk):
        s.state.pop(0)
        s.state.append(blk)
        r = 0
        for b in s.state:
            r = r ^ (sum(ord(x) for x in b) % 256)
        return chr(r)

class ProgF:
    """Decalage␣et␣XOR"""
    def __call__(s, blk):
        r = 0
        for i in range(12):
            r = r ^ ((ord(blk[i]) << (i % 8)) % 256)
        return chr(r)

class ProgG:
    """Truc␣bizarre"""
    def __init__(s):
        s.curr = 0
        s.blks = 4*[12*chr(0)]

    def __call__(s, blk):
        s.blks[s.curr] = blk
        s.curr = s.curr + 1 if s.curr < 3 else 0
        foo = 0
        for i in range(4):
            foo = (foo + ord(s.blks[i][0])) % 256
        return s.blks[foo % 4][(foo >> 4) % 12]

class ProgH:
    """Soeurs␣jumelles"""
    def __init__(s):
        s.prev = 12*chr(0)

    def __call__(s, blk):
        b = ord(s.prev[1]) ^ ord(s.prev[5]) ^ ord(s.prev[9])
        s.prev = blk
        c = ord(blk[0]) ^ ord(blk[3]) ^ ord(blk[7])
        return chr(ord(blk[b % 12]) ^ ord(blk[c % 12]))

class ProgI:
    """Renvoie␣toujours␣0␣(le␣XOR␣entre␣le␣resultat␣des␣deux␣jumelles␣se␣fait␣dans␣ProgH)."""
    def __call__(s, blk):
        return chr(0)

# Création des liens de communication
qI0, q0I = Queue(), Queue()
q01, q10 = Queue(), Queue()
q02, q20 = Queue(), Queue()
q03, q30 = Queue(), Queue()
q14, q41 = Queue(), Queue()
q15, q51 = Queue(), Queue()
q16, q61 = Queue(), Queue()
q27, q72 = Queue(), Queue()
q28, q82 = Queue(), Queue()
q29, q92 = Queue(), Queue()
q3A, qA3 = Queue(), Queue()
q3B, qB3 = Queue(), Queue()
```

```
q3C, qC3 = Queue(), Queue()
qBC, qCB = Queue(), Queue()

# Création du Transputer-0
t0 = Machine("Transputer-0")
t0.Chan[0x80000000] = q0I
t0.Chan[0x80000004] = q01
t0.Chan[0x80000008] = q02
t0.Chan[0x8000000c] = q03
t0.Chan[0x80000010] = qI0
t0.Chan[0x80000014] = q10
t0.Chan[0x80000018] = q20
t0.Chan[0x8000001c] = q30
p0 = Process(target=t0.start)

# Création du Transputer-1
t1 = Machine("Transputer-1")
t1.Chan[0x80000000] = q10
t1.Chan[0x80000004] = q14
t1.Chan[0x80000008] = q15
t1.Chan[0x8000000c] = q16
t1.Chan[0x80000010] = q01
t1.Chan[0x80000014] = q41
t1.Chan[0x80000018] = q51
t1.Chan[0x8000001c] = q61
p1 = Process(target=t1.start)

# Création du Transputer-2
t2 = Machine("Transputer-2")
t2.Chan[0x80000000] = q20
t2.Chan[0x80000004] = q27
t2.Chan[0x80000008] = q28
t2.Chan[0x8000000c] = q29
t2.Chan[0x80000010] = q02
t2.Chan[0x80000014] = q72
t2.Chan[0x80000018] = q82
t2.Chan[0x8000001c] = q92
p2 = Process(target=t2.start)

# Création du Transputer-3
t3 = Machine("Transputer-3")
t3.Chan[0x80000000] = q30
t3.Chan[0x80000004] = q3A
t3.Chan[0x80000008] = q3B
t3.Chan[0x8000000c] = q3C
t3.Chan[0x80000010] = q03
t3.Chan[0x80000014] = qA3
t3.Chan[0x80000018] = qB3
t3.Chan[0x8000001c] = qC3
p3 = Process(target=t3.start)

# Création du Transputer-4
#t4 = Machine("Transputer-4")
t4 = Program(ProgA())
t4.Chan[0x80000000] = q41
t4.Chan[0x80000010] = q14
p4 = Process(target=t4.start)

# Création du Transputer-5
#t5 = Machine("Transputer-5")
t5 = Program(ProgB())
t5.Chan[0x80000000] = q51
t5.Chan[0x80000010] = q15
p5 = Process(target=t5.start)

# Création du Transputer-6
#t6 = Machine("Transputer-6")
t6 = Program(ProgC())
t6.Chan[0x80000000] = q61
t6.Chan[0x80000010] = q16
p6 = Process(target=t6.start)

# Création du Transputer-7
#t7 = Machine("Transputer-7")
t7 = Program(ProgD())
t7.Chan[0x80000000] = q72
t7.Chan[0x80000010] = q27
p7 = Process(target=t7.start)

# Création du Transputer-8
#t8 = Machine("Transputer-8")
t8 = Program(ProgE())
t8.Chan[0x80000000] = q82
```

```
t8.Chan[0x80000010] = q28
p8 = Process(target=t8.start)

# Création du Transputer-9
#t9 = Machine("Transputer-9")
t9 = Program(ProgF())
t9.Chan[0x80000000] = q92
t9.Chan[0x80000010] = q29
p9 = Process(target=t9.start)

# Création du Transputer-A
#tA = Machine("Transputer-A")
tA = Program(ProgG())
tA.Chan[0x80000000] = qA3
tA.Chan[0x80000010] = q3A
pA = Process(target=tA.start)

# Création du Transputer-B
#tB = Machine("Transputer-B")
tB = Program(ProgH())
tB.Chan[0x80000000] = qB3
tB.Chan[0x80000004] = qBC
tB.Chan[0x80000010] = q3B
tB.Chan[0x80000014] = qCB
pB = Process(target=tB.start)

# Création du Transputer-C
#tC = Machine("Transputer-C")
tC = Program(ProgI())
tC.Chan[0x80000000] = qC3
tC.Chan[0x80000004] = qCB
tC.Chan[0x80000010] = q3C
tC.Chan[0x80000014] = qBC
pC = Process(target=tC.start)

# Démarrage des Transputers
p0.start()
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()
p7.start()
p8.start()
p9.start()
pA.start()
pB.start()
pC.start()

# Configuration de la machine
input_bin = open("input.bin", "r").read()
for c in input_bin[:input_bin.find("KEY:")]:
    qI0.put(c)

# Envoi du texte "KEY:"
for c in "KEY:":
    qI0.put(c)

# Envoi de la clef
key = "*SSTIC-2015*"
for c in key:
    qI0.put(c)

# Envoi du nom du fichier
nam = "congratulations.tar.bz2"
qI0.put(chr(len(nam)))
for c in nam:
    qI0.put(c)

# Envoi des données
dat = "1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e".decode("hex")
for c in dat:
    qI0.put(c)

sleep(1)

# Affichage du résultat
try:
    res = ""
    while True:
        res += qOI.get(block=False)
except:
```

```
    pass
res = res[24:]

# Meurtres des Transputer
p0.terminate()
p1.terminate()
p2.terminate()
p3.terminate()
p4.terminate()
p5.terminate()
p6.terminate()
p7.terminate()
p8.terminate()
p9.terminate()
pA.terminate()
pB.terminate()
pC.terminate()

print "The␣result␣is␣%r." % (res,)
```

```cpp
// testoo.cpp

#include <cstdlib>
#include <cstdio>
#include <cstring>
extern "C" {
#include <bzlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <time.h>
}

class ProgG {
private:
    char mem[48];
    int curr;

public:
    ProgG() : mem(), curr(0) {}

    char operator() (const char* blk) {
        int no, idx;
        char sum;

        // Memorize the current block.
        memcpy(&mem[12*curr], blk, 12);
        curr = (curr + 1) & 3;

        // Sum the first byte of each memorized block.
        sum = 0;
        for (int i = 0; i < 4; i++)
            sum = (sum + mem[12*i]) & 255;

        // Output the right byte of the memory.
        no = ((unsigned char)sum) & 3, idx = (((unsigned char)sum) >> 4) % 12;
        return mem[12*no + idx];
    }

    void reset() {
        bzero(mem, 48);
        curr = 0;
    }
};

class SSTIC {
private:
    char key[12];
    int i;

    // Transputer-4
    char blksum;
    char sum;

    // Transputer-5
    char blkeor;
    char eor;

    // Transputer-6
    unsigned short lfsr;

    // Transputer-7
    char s1;
    char s2;

    // Transputer-8
    char blksum2;
    char blksum3;
    char blksum4;
    char eorsum;

    // Transputer-9
    char foobar;

    // Transputer-B et Transputer-C
    unsigned char prev;

// ProgA A;
// ProgB B;
// ProgC C;
// ProgD D;
// ProgE E;
```

```
// ProgF F;
    ProgG G;
// ProgH H;

public:
    SSTIC() : key(), i(0) {};

    char next() {
        char new_, old;

        i = (i + 1) % 12;

        // Transputer-B et Transputer-C (first part)
        unsigned char curr = key[0] ^ key[3] ^ key[7];

        // r = (A(key) ⊕ B(key) ⊕ C(key) ⊕ D(key) ⊕ E(key) ⊕ F(key) ⊕ G(key) ⊕ H(key)) & 255;
        new_ = sum ^ eor ^ lfsr ^ (s1 ^ s2) ^ eorsum ^ foobar ^ G(key) ^ (key[prev % 12] ^ key[curr % 12]);
        old = key[i];

        // Transputer-8 (first part, needs the old value of blksum)
        eorsum = eorsum ^ blksum4;
        blksum4 = blksum3, blksum3 = blksum2, blksum2 = blksum;

        // Transputer-4
        blksum = blksum + (new_ - old);
        sum = sum + blksum;

        // Transputer-8 (second part, needs the new value of blksum)
        eorsum = eorsum ^ blksum;

        // Transputer-5
        blkeor = blkeor ^ (new_ ^ old);
        eor = eor ^ blkeor;

        // Transputer-6
        lfsr = ((lfsr << 1) & 0xffff) ^ ((lfsr >> 14) & 1) ^ ((lfsr >> 15) & 1);

        // Transputer-7
        if (i < 6)
            s1 = s1 + (new_ - old);
        else
            s2 = s2 + (new_ - old);

        // Transputer-9
        foobar = foobar ^ (((new_ ^ old) << (i & 7)) & 255);

        // Transputer-B et Transputer-C (second part)
        prev = key[1] ^ key[5] ^ key[9];

        key[i] = new_;

        return i + 2*old;
    }

    void reset(char* k) {
        for (int j = 0; j < 12; j++)
            key[j] = k[j];
        i = 11;

        // Transputer-4
        blksum = 0;
        for (int i = 0; i < 12; i++)
            blksum = blksum + key[i];
        sum = blksum;

        // Transputer-5
        blkeor = 0;
        for (int i = 0; i < 12; i++)
            blkeor = blkeor ^ key[i];
        eor = blkeor;

        // Transputer-6
        lfsr = 0;
        for (int i = 0; i < 12; i++)
            lfsr = (lfsr + (unsigned char)key[i]) & 65535;
        lfsr = ((lfsr << 1) & 0xffff) ^ ((lfsr >> 14) & 1) ^ ((lfsr >> 15) & 1);

        // Transputer-7
        s1 = 0;
        for (int i = 0; i < 6; i++)
            s1 = s1 + key[i];
        s2 = 0;
        for (int i = 6; i < 12; i++)
```

```
            s2 = s2 + key[i];

        // Transputer-8
        blksum2 = 0, blksum3 = 0, blksum4 = 0, eorsum = blksum;

        // Transputer-9
        foobar = 0;
        for (int i = 0; i < 12; i++)
            foobar = foobar ^ ((key[i] << (i & 7)) & 255);

        // Transputer-B et Transputer-C
        prev = 0;

// A.reset();
// B.reset();
// C.reset();
// D.reset();
// E.reset();
// F.reset();
        G.reset();
// H.reset();
    }
};

// decrypt - Déchiffre un message en utilisant l'algorithme de la cinquième étape du challenge SSTIC 2015.
void decrypt(char* key, char* input, char* output, int leng) {
    static SSTIC S;
    S.reset(key);
    for (int i = 0; i < leng; i++)
        output[i] = input[i] ^ S.next();
    return;
}

// printable - Indique si le caractère fourni est imprimable.
inline bool printable(char c) {
    return (c >= 32 && c <= 126);
}

// test - Teste si un buffer contient une archive TAR bzippé.
bool test(char* src, int srcLen) {
    char start[101];
    unsigned int startLen;
    int res;

    startLen = 100;
    res = BZ2_bzBuffToBuffDecompress(start, &startLen, src, srcLen, false, false);
    if (res != BZ_OUTBUFF_FULL && res != BZ_OK)
        return false;

    printf(">␣passed␣the␣bzip2␣test!\n");

    int i = 0;
    for (i = 0; i < 100 && printable(start[i]); i++);
    while (i < 100 && start[i] == 0) i++;
    if (i != 100)
        return false;

    start[100] = 0;
    printf(">>␣found␣an␣archive␣containing␣the␣file␣'%s'!\n", start);

    return true;
}

#define PAYLOAD_LENGTH 250606
char encrypted[PAYLOAD_LENGTH];
char decrypted[PAYLOAD_LENGTH];

// load - Charge le texte chiffré dans la mémoire.
void load(void) {
    int fd, n;

    fd = open("encrypted", O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(1);
    }

    n = read(fd, encrypted, PAYLOAD_LENGTH);
    if (n == -1) {
        perror("read");
        exit(1);
    } else if (n < PAYLOAD_LENGTH) {
        fprintf(stderr, "read:␣premature␣end-of-file␣(expected␣%d␣bytes,␣got␣%d␣bytes)\n", PAYLOAD_LENGTH,
```

```
            ↪ n);
        exit(1);
    }
}

// For the test vector decoding
char testvec[] = { 0x1d, 0x87, 0xc4, 0xc4, 0xe0, 0xee, 0x40, 0x38, 0x3c, 0x59, 0x44, 0x7f, 0x23, 0x79, 0x8d,
     ↪ 0x9f, 0xef, 0xe7, 0x4f, 0xb8, 0x24, 0x80, 0x76, 0x6e };
char ilovest20[sizeof(testvec)+1];

// The already known 7-bits of the key.
char basekey[] = { 0x5e, 0x54, 0x1b, 0x71, 0x56, 0x7c, 0x64, 0x7d, 0x69, 0x76 };

int main(int argc, char** argv) {
    unsigned short me;
    char key[12];

    if (argc == 1)
        me = 0;
    else
        me = atoi(argv[1]);

    // Self-test: can I decode the test vector two times?
    for (int i = 0; i < 2; i++) {
        decrypt("*SSTIC-2015*", testvec, ilovest20, sizeof(testvec));
        ilovest20[sizeof(testvec)] = 0;
        printf(">>␣%s\n", ilovest20);
    }

    load();

    while (me < 1024) {
        for (int i = 0, j = me; i < 10; i = i + 1, j = j >> 1) {
            if ((j & 1) == 0)
                key[i] = basekey[i];
            else
                key[i] = basekey[i] + 128;
        }

        for (int k = 0; k < 65536; k++) {
            key[10] = (k >> 8) & 0xff;
            key[11] = k & 0xff;

            if (k % 64 == 0)
                printf("%␣5d:%␣5d/1024\n", me, k/64);

            decrypt(key, encrypted, decrypted, PAYLOAD_LENGTH);

            bool isok = test(decrypted, PAYLOAD_LENGTH);
            if (isok) {
                printf("%␣5d:␣key:␣%02x:%02x:%02x:%02x␣%02x:%02x:%02x:%02x␣%02x:%02x:%02x:%02x\n",
                    me,
                    (unsigned char)key[0], (unsigned char)key[1],
                    (unsigned char)key[2], (unsigned char)key[3],
                    (unsigned char)key[4], (unsigned char)key[5],
                    (unsigned char)key[6], (unsigned char)key[7],
                    (unsigned char)key[8], (unsigned char)key[9],
                    (unsigned char)key[10], (unsigned char)key[11]);
            }
        }

        me++;
    }

    return 0;
}
```

testoo.py

```
# -*- coding: utf8 -*-

from itertools import izip

class ProgA:
    """Somme"""
    def __init__(s):
        s.foo = 0

    def __call__(s, blk):
        for i in range(12):
            s.foo = (s.foo + ord(blk[i])) % 256
        return chr(s.foo)

class ProgB:
    """XOR"""
    def __init__(s):
        s.foo = 0

    def __call__(s, blk):
        for i in range(12):
            s.foo = (s.foo ^ ord(blk[i])) % 256
        return chr(s.foo)

class ProgC:
    """LSFR"""
    def __init__(s):
        s.state = 0
        s.initialized = False

    def __call__(s, blk):
        if not s.initialized:
            for i in range(12):
                s.state = (s.state + ord(blk[i])) % 65536
            s.initialized = True
        s.state = ((s.state << 1) & 65535) ^ ((s.state >> 14) & 1) ^ ((s.state >> 15) & 1)
        return chr(s.state % 256)

class ProgD:
    """XOR␣de␣la␣somme␣des␣six␣premiers␣octets␣et␣des␣six␣derniers␣octets␣du␣bloc"""
    def __call__(s, blk):
        s1 = chr(sum(ord(x) for x in blk[:6]) % 256)
        s2 = chr(sum(ord(x) for x in blk[6:]) % 256)
        return chr(ord(s1) ^ ord(s2))

class ProgE:
    """XOR␣des␣sommes␣des␣octets␣des␣quatre␣derniers␣blocs"""
    def __init__(s):
        s.state = 4*[12*chr(0)]

    def __call__(s, blk):
        s.state.pop(0)
        s.state.append(blk)
        r = 0
        for b in s.state:
            r = r ^ (sum(ord(x) for x in b) % 256)
        return chr(r)

class ProgF:
    """Decalage␣et␣XOR"""
    def __call__(s, blk):
        r = 0
        for i in range(12):
            r = r ^ ((ord(blk[i]) << (i % 8)) % 256)
        return chr(r)

class ProgG:
    """Truc␣bizarre"""
    def __init__(s):
        s.curr = 0
        s.blks = 4*[12*chr(0)]

    def __call__(s, blk):
        s.blks[s.curr] = blk
        s.curr = s.curr + 1 if s.curr < 3 else 0
        foo = 0
        for i in range(4):
            foo = (foo + ord(s.blks[i][0])) % 256
        return s.blks[foo % 4][(foo >> 4) % 12]

class ProgH:
```

```
    """Soeurs␣jumelles"""
    def __init__(s):
        s.prev = 12*chr(0)

    def __call__(s, blk):
        b = ord(s.prev[1]) ^ ord(s.prev[5]) ^ ord(s.prev[9])
        s.prev = blk
        c = ord(blk[0]) ^ ord(blk[3]) ^ ord(blk[7])
        return chr(ord(blk[b % 12]) ^ ord(blk[c % 12]))

def xor(it):
    """Fait␣le␣XOR␣des␣elements␣de␣l'iterable."""
    r = 0;
    for i in it:
        r = r ^ i
    return r

def SSTIC(key):
    """Construit␣un␣flux␣de␣clef␣en␣utilisant␣l'algorithme␣du␣challenge␣SSTIC."""
    i = 0
    progs = (ProgA(), ProgB(), ProgC(), ProgD(), ProgE(), ProgF(), ProgG(), ProgH())

    while True:
        for i in range(12):
            r = xor(ord(p(key)) for p in progs)
            yield chr((i + 2 * ord(key[i])) % 256)
            key = key[:i] + chr(r) + key[i+1:] # key[i] = chr(r)

def decrypt(key, data):
    res = []
    for i, j in izip(SSTIC(key), data):
        res.append(chr(ord(i)^ord(j)))
    return ''.join(res)
```