Solution du challenge SSTIC 2014

Anthoine Bourgeois

27 mai 2014

Résumé

Le challenge SSTIC 2014 consiste a trouver une adresse mail en challenge.sstic.org dans une capture de traces USB. Ce document présente la démarche utilisé par son auteur pour parvenir à une solution. Celle-ci se décompose en 4 étapes : l'analyse de l'échange USB, la rétro-conception d'un exécutable AArch64, la rétro-conception d'une machine virtuelle dont les données sont chiffrés et l'analyse d'un firmware pour récupérer l'adresse mail dans une zone protégée de la mémoire. Les outils développés sont présentés dans ce document au fil de la démarche.

Table des matières

1 Analyse des traces USB

Le challenge se présente sous la forme d'un fichier récupérable à l'adresse :
http://static.sstic.org/challenge2014/usbtrace.xz.

Un simple *file* indique le fichier est bien un fichier compressé comme son extension l'indique :

```
$ file usbtrace.xz
usbtrace.xz: XZ compressed data
```

La commande *unxz* ou *xz –decompress* permet de décompresser le fichier téléchargé :

```
$ unxz usbtrace.xz
$ file usbtrace
usbtrace: UTF-8 Unicode text, with very long lines
```

Le fichier est un fichier texte codé en UTF-8. L'ouverture du fichier décompressé avec un éditeur de texte donne le texte suivant :

```
Date: Thu, 17 Apr 2015 00:40:34 +0200
To: <challenge2014@sstic.org>
Subject: Trace USB

Bonjour,

voici une trace USB enregistrée en branchant mon nouveau       \
     téléphone Android sur mon ordinateur personnel air-gapped.
Je suspecte un malware de transiter sur mon téléphone.         \
     Pouvez-vous voir de quoi il en retourne ?

--
```

La suite du fichier est une capture de l'échange USB entre le téléphone et l'ordinateur réalisé avec *usbmon*. Le fichier *D*ocumentation/usb/usbmon.txt contenu dans l'arborescence du noyau Linux décrit le format texte et l'ABI du driver *usbmon*.

Le format texte brut n'est pas exploitable directement sous Wireshark [1], une conversion est nécessaire. Le script de conversion `usbmontext2pcap.sh` qui a été utilisé pour cette solution se trouve en annexe 5. Pour chaque ligne de trace, ce script recrée le paquet correspondant en respectant l'ABI et le converti avec l'outil *text2pcap*. Une fois tous les paquets recréés, ils sont assemblés avec *mergecap*. Ainsi le nouveau fichier généré contient tous les paquets du fichier texte d'origine et peut être exploité par Wireshark.

Avant de lancer le script, la commande *tail* ci-dessous supprime le mail en entête du fichier.

```
$ tail -$(($(wc -l usbtrace | cut -d' ' -f1)-11)) \
> usbtrace > usbmon.txt
$ ./usbmontext2pcap.sh usbmon.txt
```

Wireshark permet une meilleur lisibilité des paquets USB mais ne dissèque pas le protocole ADB qui est au dessus (cf. 1).

Wireshark accepte les extensions Lua pour reconnaître de nouveaux protocoles. En annexe 5, un dissecteur ADB permet d'afficher les commandes ADB clairement dans l'affichage graphique de Wireshark. Pour utiliser Wireshark avec l'extension :

```
$ wireshark -X lua_script:adboverusbdissector.lua
```

En analysant attentivement les traces, plusieurs informations ressortent :

— la commande `id` renvoie : `uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1009(mount), 1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin), 3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0`;

— la commande `uname -a` renvoie `Linux localhost 4.1.0-g4e972ee #1 SMP PREEMPT Mon Feb 24 21:16:40 PST 2015 armv8l GNU/Linux`;

1. https://www.wireshark.org/

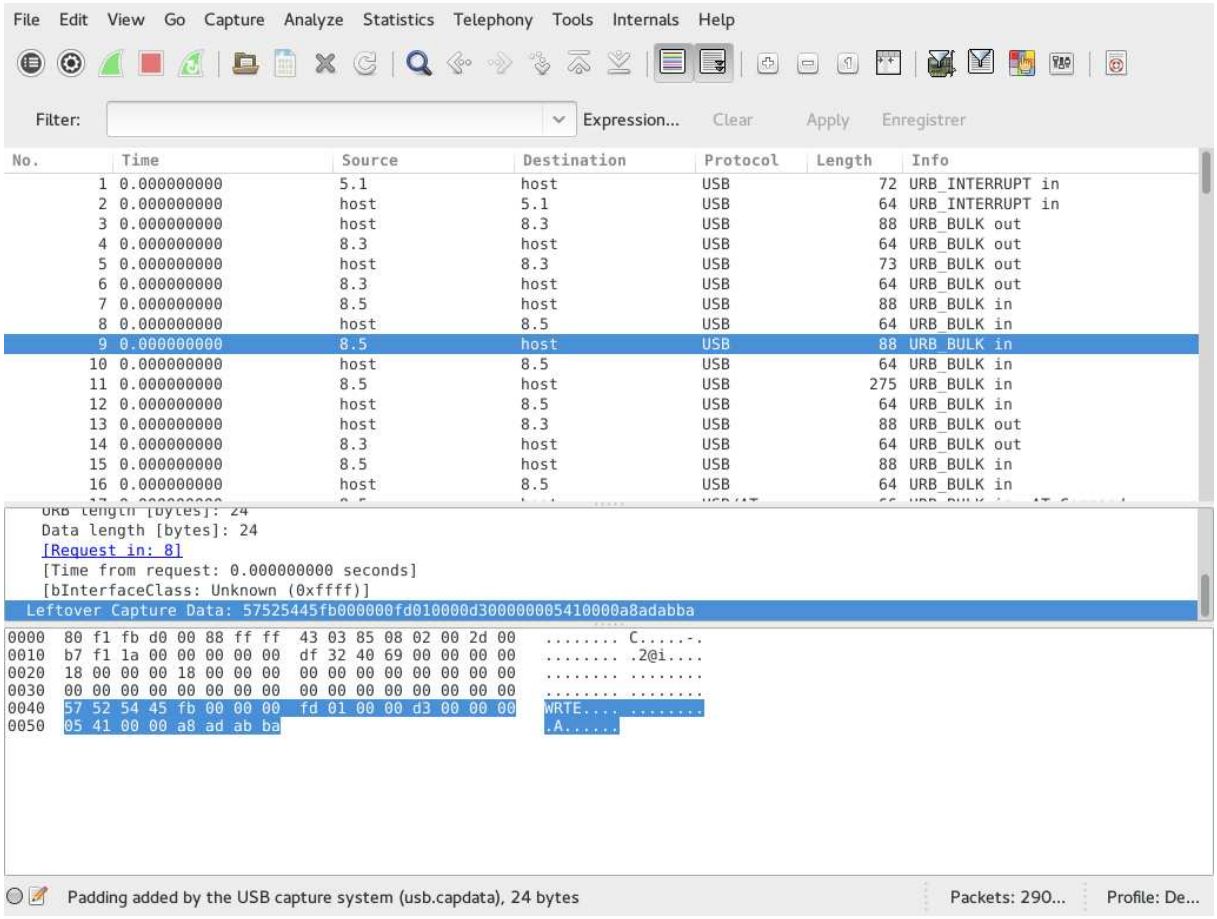

FIGURE 1 – Les trames USB ouvertes avec *Wireshark*

— une commande de listing dans */sdcard/* renvoie les entrées `. , .. , Samsung , Android , face , Music, Podcasts, Ringtones, Alarms, Notifications, Pictures, Movies, Download, DCIM, Documents, SPenSDK30, .enref, Nearby, Playlists, .pla, .estrongs, backups, clockworkmod, CyanogenMod, mmc1` ;

— une commande de listing dans */sdcard/Documents/* renvoie les entrées `. , .. , CSW-2014-Hacking-9.11_uncensored.NATO_Cosmic_Top_Secret.gpg` ;

— une commande de listing dans */data/local/tmp* renvoie les entrées `.` et `..`, le repertoire est vide.

On constate également qu'un seul transfère de fichier est contenu dans l'échange capturer. C'est un fichier binaire nommé *badbios.bin* dont la taille est de 78000 octets et qui est envoyé dans le répertoire /data/local/tmp du téléphone. Les commandes `DATA` du protocole ADB permettent de connaître la taille des morceaux à récupérer.

4 octets	4 octets	*Taille* octets
`DATA`	*Taille*	*Données*

TABLE 1 – Format de la commande `DATA`

Le fichier est découpé en plusieurs commandes `DATA` et les commandes sont découpées en plusieurs paquets USB. Wireshark permet de récupérer le contenu des paquets USB, il ne reste qu'à supprimer le protocole ADB des données et de ré-assembler les données à l'aide de l'outil *hexedit*.

Les paquets ADB composants le fichier *badbios.bin* sont numérotés : 2367, 2375, 2383, 2391, 2399, 2407, 2415, 2423, 2433, 2441, 2449, 2457, 2465, 2475, 2483, 2491, 2499, 2509, 1519 et 2529 dans *Wireshark*. Les paquets 2367 et 2499 contiennent des commandes `DATA`. Le paquet 2529 se termine par une commande de fin au format :

Avec ces données, le fichier peut être reconstruit avec des outils tels que *hexedit* par exemple.

```
$ md5sum badbios.bin
b6097e562cb80a20dfb67a4833b1988a  badbios.bin
```

Une fois le fichier récupéré, la commande *file* apporte de l'information sur le fichier *badbios.bin*.

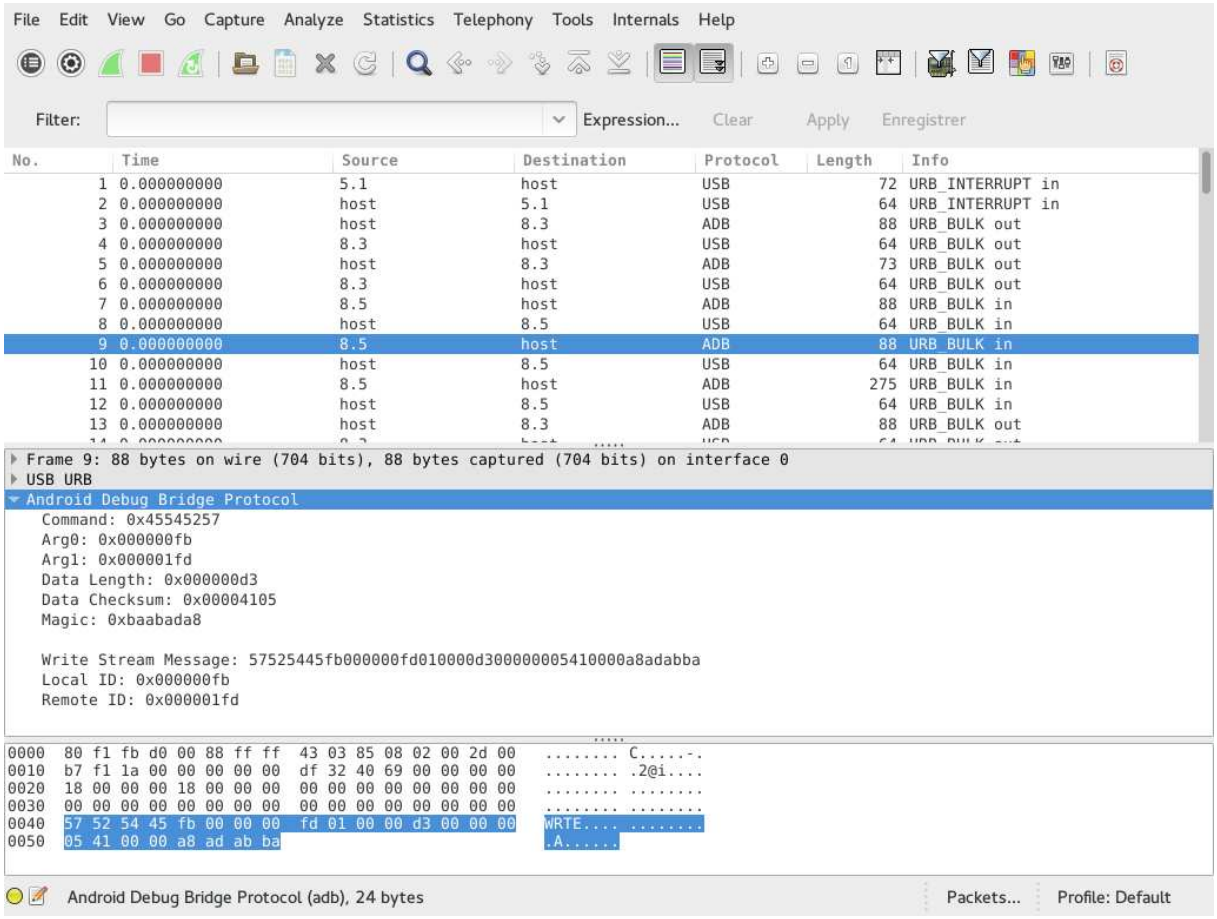

FIGURE 2 – *wireshark* et le dissecteur *ADB*

4 octets	4 octets
DONE	*CRC*

TABLE 2 – Format de la commande `DONE`

```
$ file badbios.bin
badbios.bin: ELF 64-bit LSB executable, ARM aarch64, version 1 \
    (SYSV), statically linked, stripped
```

2 Analyse de l'exécutable AArch64

Avant d'entamer l'analyse de fichier *badbios.bin* il est nécessaire de disposer d'outils pour l'architecture AArch64. Les *binutils*[2] et *gdb*[3] sont les outils minimum à avoir pour manipuler les binaires AArch64. Ces outils doivent avoir le support *multitarget* pour ne pas supporter uniquement l'architecture hôte.

En plus de ces outils, un emulateur est nécessaire pour exécuter le binaire. Qemu[4] supporte l'émulation de l'architecture AArch64.

Le lancement de l'exécutable ci-dessous :

```
$ qemu-aarch64 badbios.bin
:: Please enter the decryption key: ^D   Wrong key format.
$
```

L'exécutable attend une clé de déchiffrement.

2. https://www.gnu.org/software/binutils/
3. https://www.gnu.org/software/gdb/
4. http://wiki.qemu.org/Main_Page

```
$ qemu-aarch64 badbios.bin
:: Please enter the decryption key: 0123456789ABCDEF^D
:: Trying to decrypt payload...
   Invalid padding.
$
```

La clé est au format hexadécimal (Attention les minuscules ne sont pas acceptées). La longueur doit être de 16 caractères hexadécimaux ou 64 bits pour que la tentative de déchiffrement du payload se fasse. Une clé plus courte affiche le message `Wrong key format`, une clé plus long n'est pas consommé au dessus des 16 premiers caractères.

A partir d'ici, il y a deux pistes pour trouver la clé et récupérer le payload :

— la clé est cachée dans un canal caché des traces USB ;

— la clé doit être trouvé par rétro-conception.

Sans indice sur la première possibilité, la seconde a été poursuivie.

2.1 Rétro-conception de l'exécutable *badbios.bin*

Pour effectuer la rétro-conception de l'exécutable AArch64, il est nécessaire de se documenter sur cette architecture encore peu répandue mais si promise à un grand avenir comme ces aînées. ARM fournit une documentation très complète concernant ces processeurs. L'ensemble des instructions ARMv8 se trouve dans le PDF suivant, fournit par la société ARM [5]

La première étape de rétro-conception est de retrouver le code assembleur à partir du binaire. L'utilitaire *objdump* permet de faire cette opération :

```
$ objdump -d badbios.bin > badbios.disas
```

L'utilitaire *readelf* donne aussi des informations intéressantes telles que l'adresse des différentes sections en mémoire :

```
$ readelf -l ../badbios.bin

Type de fichier ELF est EXEC (fichier exécutable)
Point d'entrée 0x102cc
Il y a 3 en-têtes de programme, débutant à l'adresse de décalage64

En-têtes de programme:
  Type Décalage           Adr.virt           Adr.phys.
       Taille fichier     Taille mémoire      Fanion Alignement
  LOAD 0x0000000000000000 0x0000000000010000 0x0000000000010000
       0x00000000000005d8 0x00000000000005d8  R E    10000
  LOAD 0x0000000000001000 0x0000000000021000 0x0000000000021000
       0x0000000000011f50 0x0000000000011f50  RW     10000
  NOTE 0x0000000000000000 0x0000000000000000 0x0000000000000000
       0x0000000000000000 0x0000000000000000  R      8

 Section à la projection de segement:
  Sections de segment...
   00     .text .rodata
   01     .data
   02
```

Ici l'adresse du point d'entrée permet de savoir où commence le programme (`0x102cc`). C'est une première piste pour commencer la rétro-conception. Une autre piste aurait été de trouver dans les données du programme les chaînes de caractères affichées par le programme (Ex : `Please enter the decryption key`) pour connaître leurs adresses et savoir où elles étaient utilisées dans le code. Hélas, une visualisation avec hexedit montre qu'aucune chaîne de caractères n'est présente, l'hypothèse raisonnable est qu'elles sont chiffrées quelque part.

La rétro-conception de la fonction *main* est relativement direct car le code ne semble pas obfusqué :

```
1
2 typedef struct section {
3   unsigned char *mmap_addr;   /*  0 */
4   unsigned long  mmap_size;   /*  8 */
5   unsigned char *data_addr;   /* 16 */
```

5. https://silver.arm.com/download/ARM_and_AMBA_Architecture/AR100-DA-70501-r0p0-00eac5/ARMv8_ISA_PRD03-GENC-010197-30-0.pdf

```
  unsigned long  data_size;  /* 24 */
  unsigned int   mprot;      /* 32 */
  unsigned int   chiffre;    /* 36 */
} section; /* 40 octets */

static unsigned long nb_sections = 2;
static struct section *sections = {
  { .mmap_addr = 0x0000000000400000,
    .mmap_size = 0x0000000000002c08,
    .data_addr = &code, .data_size = 0x0000000000001e84,
    .mprot = PROT_READ|PROT_EXEC, .chiffre = 1 },
  { .mmap_addr = 0x0000000000500000,
    .mmap_size = 0x0000000000010040,
    .data_addr = &data, .data_size = 0x0000000000010018,
    .mprot = PROT_READ|PROT_WRITE, .chiffre = 0},
  { 0, }, { 0, } };

static unsigned long enter_program = 0x0000000000400514;

int main(int argc, char **argv)
{
  int idx = 0;
  struct section *s = sections;
  char *mmap_addr;
  unsigned long bsize;
  unsigned long round_size;

  while (idx < nb_sections)
  {
    round_size = s->mmap_size;
    round_size += 0xfff;
    round_size &= 0xfffffffffffff000;
    mmap_addr = mmap(s->mmap_addr, round_size,
                     PROT_READ|PROT_WRITE,
                     MAP_ANONYMOUS, 0/*fd*/, 0/*offset*/);
    if (mmap_addr != s->mmap_addr)
      return 1;

    if (s->chiffre != 0)
    {
      ret = dechiffrement(mmap_addr, s->data_addr,
                          round_size, s->data_size);
      if (ret == 0)
        return 1;
    }
    else
    {
      int i = 0;
      while (s->data_size != i)
      {
        mmap_addr[i] = s->data_addr[i];
        i++;
      }
    }

    ret = mprotect(mmap_addr, round_size, s->mprot);
    if (ret != 0)
      return 1;

    r++;
    s++;
  }
  return enter_program(argc, argv);
}
```

La fonction *main* mappe les deux sections qui sont définies globalement, copie les données dans chacune des sections et définit les protections nécessaires à ces mappings avant de sauter dans l'un d'entre eux. L'adresse de saut est `0x400514`, ce qui la situe dans le premier mapping qui commence en `0x400000`. Ce dernier mapping définit ces protections en lecture et exécution, preuve supplémentaire qu'il contient du code. L'hypothèse pour le second mapping est qu'il correspond aux données du nouveau programme.

Lors de la phase de copie des données dans les deux mappings la fonction emprunt deux chemins différents. Le choix du chemin se fait en fonction d'un champ de la structure `section` (ici nommé *chiffre*). Dans le cas où *chiffre* est nul, le chemin est le plus simple puisque qu'une copie octet par octet est effectuée dans le mapping. Dans le cas

contraire, une fonction est appelée pour déchiffrer les données globales et les placer dans le mapping.

Deux pistes se présente à ce point :

— poursuivre la rétro-conception pour comprendre le déchiffrement et récupérer le code du nouveau programme ;

— utiliser le débuggeur en s'arrêtant juste avant le saut dans le nouveau programme pour faire un duplicata des deux mappings.

Pour gagner du temps, la seconde solution a été utilisée initialement.

```
## dans un premier terminal
$ qemu-aarch64 -g 1234 badbios.bin

...
## dans un second terminal
$ gdb
GNU gdb (Gentoo 7.7 p1) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
...
(gdb) debug aarch64
(gdb) file badbios.bin
Reading symbols from badbios.bin...(no debugging symbols found)...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0x00000000000102cc in ?? ()
(gdb) b *0x102c0
Breakpoint 1 at 0x102c0
(gdb) c
Continuing.

Breakpoint 1, 0x00000000000102c0 in ?? ()
(gdb) dump memory bad.code.bin 0x400000 0x402c08
(gdb) dump memory bad.data.bin 0x500000 0x510040
(gdb)
```

Ici les tailles sont celles données par le programme précédent.

Après avoir récupéré les deux mappings, un *file* sur le code déchiffré donne ceci :

```
$ file bad.code.bin
bad.code.bin: ELF 64-bit LSB executable, ARM aarch64, version 1 \
    (SYSV), statically linked, stripped
```

Le code est précédé d'une entête ELF identique au fichier *badbios.bin*. Il faut reconstruire le fichier ELF complet avec la partie `.data` présente dans le second dump.

Pour commencer, la séparation des l'entête et du code aidera à la reconstruction du binaire. La commande *readelf* informe de la taille de l'entête.

```
$ readelf -h bad.code.bin
readelf: ERREUR: Incapable de lire 0x40 octets de En-têtes de section
En-tête ELF:
  Magique:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Classe:                            ELF64
  Données:                           complément à 2, système à octets      \
                                     de poids faible d'abord (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  Version ABI:                       0
  Type:                              EXEC (fichier exécutable)
  Machine:                           AArch64
  Version:                           0x1
  Adresse du point d'entrée:         0x400514
  Début des en-têtes de programme:   64 (octets dans le fichier)
  Début des en-têtes de section:     131240 (octets dans le fichier)
  Fanions:                           0x0
  Taille de cet en-tête:             64 (bytes)
```

```
  Taille de l'en-tête du programme:  56 (bytes)
  Nombre d'en-tête du programme:     2
  Taille des en-têtes de section:    64 (bytes)
  Nombre d'en-têtes de section:      7
  Table d'indexes des chaînes d'en-tête de section: 6
readelf: ERREUR: Incapable de lire 0x1c0 octets de En-têtes de section
```

L'en-tête fait 64 octets et elle est suivie (offset 64) de deux en-têtes de programme de 56 octets chacune soit 176 octets de structure ELF avant le code lui-même.

```
$ truncate -s 12288 bad.code.bin
$ cat bad.code.bin bad.data.bin > bad.bin
```

En ouvrant *bad.bin*, on peut constater que les deux entêtes de programme nécessaire au chargement de l'exécutable sont vides. Voici leur structure en 64 bits :

```
typedef struct
{
  Elf64_Word    p_type;                 /* Segment type */
  Elf64_Word    p_flags;                /* Segment flags */
  Elf64_Off     p_offset;               /* Segment file offset */
  Elf64_Addr    p_vaddr;                /* Segment virtual address */
  Elf64_Addr    p_paddr;                /* Segment physical address */
  Elf64_Xword   p_filesz;               /* Segment size in file */
  Elf64_Xword   p_memsz;                /* Segment size in memory */
  Elf64_Xword   p_align;                /* Segment alignment */
} Elf64_Phdr;

/* Legal values for p_type (segment type).  */

#define PT_NULL         0               /* Program header table entry unused */
#define PT_LOAD         1               /* Loadable program segment */
#define PT_DYNAMIC      2               /* Dynamic linking information */
#define PT_INTERP       3               /* Program interpreter */
#define PT_NOTE         4               /* Auxiliary information */
#define PT_SHLIB        5               /* Reserved */
#define PT_PHDR         6               /* Entry for header table itself */
#define PT_TLS          7               /* Thread-local storage segment */
#define PT_NUM          8               /* Number of defined types */

/* Legal values for p_flags (segment flags).  */

#define PF_X            (1 << 0)        /* Segment is executable */
#define PF_W            (1 << 1)        /* Segment is writable */
#define PF_R            (1 << 2)        /* Segment is readable */
```

Sans être très lisible, *hexedit* permet d'éditer les 112 octets manquants pour les deux entêtes de programme. Les valeurs utilisées sont encadrées en rouge sur la capture de la figure 3.

La commande *readelf* montre la modification réalisée ci-dessus. Cela est suffit au lanceur de programme pour exécuter le nouveau binaire :

```
$ readelf -l bad.bin
readelf: ERREUR: Incapable de lire 0x40 octets de En-têtes de section
readelf: ERREUR: Incapable de lire 0x1c0 octets de En-têtes de section

Type de fichier ELF est EXEC (fichier exécutable)
Point d'entrée 0x400514
Il y a 2 en-têtes de programme, débutant à l'adresse de décalage64

En-têtes de programme:
  Type Décalage           Adr.virt           Adr.phys.
       Taille fichier     Taille mémoire      Fanion Alignement
  LOAD 0x0000000000000000 0x0000000000400000 0x0000000000400000
       0x00000000000002c08 0x00000000000002c08  R E    10000
  LOAD 0x0000000000003000 0x0000000000500000 0x0000000000500000
       0x0000000000010040 0x0000000000010040  RW     10000
```

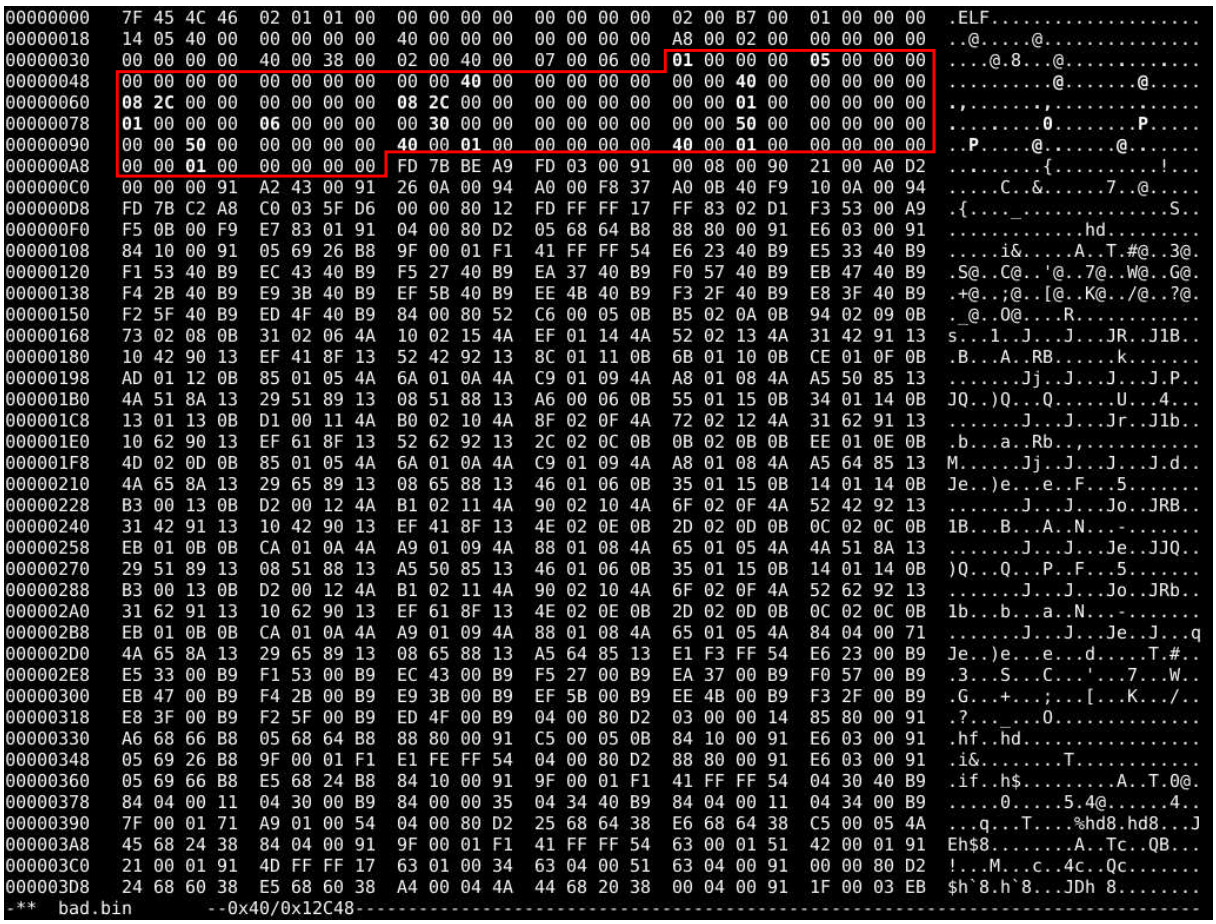

FIGURE 3 – Le fichier *bad.bin* modifié avec `hexedit`

```
$ chmod 755 bad.bin
$ qemu-aarch64 bad.bin
:: Please enter the decryption key:
   Wrong key format.
```

Ainsi, il n'est pas nécessaire de connaître l'algorithme de chiffrement utilisé pour le code du second programme, nous pouvons passer à la section suivante 2.2.

2.2 Rétro-conception de l'exécutable *bad.bin*

Le binaire est toujours de l'architecture AArch64 mais la rétro-conception est plus complexe du fait de l'obfuscation présente ici.

2.2.1 Obfuscation

Le tableau 3 présente quelques exemples trouvés dans le code du programme.

Instructions	Équivalence
`add x0, x0, #0x0`	nop
`sub x0, x0, xzr`	nop
`orr x8, x8, xzr`	nop
`csel x2, x2, x30, al`	nop
`bic x28, x28, xzr`	nop
`eor x8, x8, xzr`	nop
`mov sp, sp`	nop
`madd x6, x1, xzr, x6`	nop
`csneg x6, xzr, x6, le` `csneg x6, xzr, x6, gt`	mov x6, #0x0
`mvn x0, x20` `mvn x0, x0`	mov x0,x20

`sub x3, x3, x3`
`sub x3, x3, #0x3ff`
`add x3, x3, #0x400`
`sub x3, x3, #0x378`
`add x3, x3, #0x379`
`sub x3, x3, #0x2f5`
`add x3, x3, #0x2f6`
`add x3, x3, #0x1`
	mov x3, #0x4

`sub sp, sp, #0x1ad`
`add sp, sp, #0x1b5`
`add sp, sp, #0x18`
`str wzr, [sp]`
`sub sp, sp, #0x18`
`sub sp, sp, #0x8`
	str wzr,[sp,#0x20]

`mov x3, #0x2800000000000000` `clz x3, x3`	mov x3, #0x2

`mov x3, #0x1a0`
`ror x3, x3, #58`
`ror x3, x3, #16`
`clz x3, x3`
	mov x3, #0x1

`eor x0, x0, x20`
`eor x20, x20, x0`
`eor x0, x20, x0`
	mov x0, x20

Table 3 – Exemples d'instructions obfusquées

Ce tableau ne présent qu'un ensemble d'exemples qui peuvent être étendus, allongés et/ou combinés pour varier le degrés d'obfuscation *logique* du code.

Une autre technique d'obfuscation présent dans le programme est de brancher une fonction A au milieu d'une autre fonction B pour revenir en A sans influencer B. Le pseudo-code ci-dessous présente cette situation :

```
debut_fonc_A:
  01:  instA1
  02:  instA2
  03:  jump 07
  04:  instA4
fin_fonc_A:
debut_fonc_B:
  05:  instB1
  06:  jump 09
  07:  instA3
  08:  jump 04
  09:  instB2
fin_func_B:
```

Dans cette exemple, la fonction A saute dans la fonction B à l'adresse 07 pour exécuter sa 3ème instruction puis revient en à l'adresse 04. La fonction B n'exécute jamais les instructions 07 et 08 car elle saute inconditionnellement en à l'adresse 09.

Ce type d'obfuscation *spatiale* rend le code difficile à lire car le fil des instructions est souvent interrompu par des sauts inutiles.

2.2.2 Rétro-conception de la fonction *main*

La fonction *main* reste simple. Le listing ci-dessous présente le code après rétro-conception :

```
1  int main() {
2    void *state;
3    int ret;
4
5    ret = init_vm(blob, blob_size, &state);
6    if (ret < 0) {
7      ret = -1;
8      goto end;
9    }
10   ret = go(state);
11 end:
12   return ret;
13 }
```

Deux parties se dégagent :

1. une fonction d'initialisation du programme étudié en 2.2.3 ;
2. une fonction principale qui demande la clé à l'utilisateur pour déchiffrer le payload (étudié en 2.2.4).

Le paramètre *blob* est un pointeur sur l'adresse `0x500000`, soit l'adresse de la section de données chargée par l'exécutable. Rappelons que la section *.data* était une copie simple dans le premier programme, qu'elle ne contient pas de donnée lisible telle que des chaînes de caractère, l'hypothèse que cette section soit chiffrée reste donc plausible. Le second paramètre *blob_size* contient la valeur `0x10000`, approximativement toute la section de données mais pas totalement. Enfin, *state* est un paramètre en sortie initialisé par la fonction *init_vm*.

2.2.3 Rétro-conception de la fonction d'initialisation

La fonction d'initialisation permet de retrouver et de comprendre la structure de données utilisée par le programme pour l'exécution de la boucle principale. La compréhension de la structure ou de la fonction d'initialisation n'a pas été faite en une seule fois, de nombreux rapprochements avec la boucle principale on du être fait au fur et à mesure de la rétro-conception mais la structure est présenté entièrement ici pour aider à la compréhension plus tard.

```
1  struct vmstate {
2    int active;      // 0
3    int retcode;     // 4
4    long cycles;     // 8
5    char *key[48];   // 16
```

```
  char *iv[16];      // 64
  char *mblob;       // 80
  struct memmap memmap[32]; // 88
  char *mmem;      // 856
  func_t *ops[32];     // 864
} // 1112 octets
```

Les champs `active`, `retcode`, `cycles` et `memmap` sont expliqués dans les parties 2.2.4 et 2.2.5. Les autres champs sont expliqués ci-dessous avec la fonction d'initialisation.

Le commentaire de la fonction est réalisée à la suite :

```
int init_vm(char *blob, int blob_size, void *state) {
  struct vmstate *s;
  char *mblob;
  char *mmem;
  int i;

  s = mmap(0, 0x1000, PROT_READ|PROT_WRITE,
           MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
  if (s == -1)
    goto end;

  mblob = mmap(0, blob_size, PROT_READ|PROT_WRITE,
               MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
  if (mblob == -1)
    goto end;

  /* Copy to blob into the second mapping */
  i = 0;
  while (blob_size != i) {
    mblob[i] = blob[i];
    i++;
  }

  mmem = mmap(0, 0x1000, PROT_READ|PROT_WRITE,
              MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
  if (mmem == -1)
    goto end;

  s->mblob = mblob;
  s->mmem  = mmem;

  /* Init memory mapping entries */
  i = 0;
  while (i != 32) {
    s->memmap[i] = { 0, 0, MFREE};
    i++;
  }

  init_ops(s->ops);
  init_key(s->key, blob + blob_size);
  init_iv(s->iv, blob + blob_size + 32);

  *state = s;

  return 0;
end:
  return -1;
}
```

La fonction d'initialisation crée trois mapping privé. Le premier sert à stocker la structure `struct vmstate` définit plus haut, le second est une copie du blob de donnée et le troisième reste vide pour l'instant. Ensuite la fonction initialise la structure de donnée `vmstate` en plusieurs étapes :

1. elle sauvegarde les pointeurs sur les mapping `mblob` et `mmem`. Ces deux valeurs sont importantes pour l'explication de la section 2.2.5 ;
2. le tableau memmap est initialisé à sa valeur par défaut. Les détails de son fonctionnement sont expliqués dans la partie 2.2.5 ;
3. le tableau de pointeurs sur fonction est initialiser avec la fonction `init_ops`. C'est l'une des fonctions les plus obfusquées du programme, les pointeurs sont additionnés et soustraits de nombreuses fois avant d'être utilisés ou stockés.

4. la clé est initialisé avec le préfixe `expand 16-byte k` suivit des quatre valeurs hexadécimal `0x0BADB105` (grand boutiste) stocké après le blob (adresses `0x510000-0x510010`). Ces 16 octets sont étendues une seconde fois sur les 16 octets suivants (comme l'indique le préfixe). Le résultat est dans la table 4.

Taille	Valeur
16 octets	`expand 16-byte k`
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
4 octets	0x0BADB105
48 octets	

TABLE 4 – La clé initialisée

5. l'IV sur 16 octets complète la clé. il est initialisé à zéro pour les 8 premiers octets et aux valeurs stockés entre `0x510020-0x510028` (après le blob et la clé) pour les 8 octets suivant. Néanmoins ces 8 derniers octets ne contiennent que des zéros également. L'IV est initialisé à zéro mais la suite de la rétro-conception montrera qu'il n'est pas constant au cours de l'exécution.

Enfin la fonction copie le pointeur de la structure dans le paramètre de retour pour la passer au reste du programme.

2.2.4 Rétro-conception de la boucle principale

La rétro-conception de la boucle principale révèle que le programme est une machine virtuelle. Cette machine virtuelle est en cours d'exécution si le champ `active` de la structure `vmstate` est positionné. Si ce champ est remis à zéro, le champ `retcode` doit en donner la raison.

Ci-dessous le pseudo-code de la boucle principale. Ce code est simplifié pour plus de lisibilité :

```
int go(struct vmstate *s)
{
  int ret;

  s->active = 1;
  ret = interpreter(s);
  if (ret != 0)
    writemsg(err_msg[s->retcode]);

  return ret;
}

int interpreter(struct vmstate *s)
{
  int ip = 0, limit = 0xffff;
  int ret, opcode, inst, inst_size;

  while (ip < limit)
  {
    ip = next_instruction(s);
    if (ip == -1)
      goto end;

    ret = read(s, ip, opcode, 1);
    if (ret != 1)
      goto end;

    if (opcode > 31)
      goto end;

    opsize = opcode > 8 ? 2 : 4;
    ret = read(s, ip, inst, inst_size);
    if (inst_size != ret)
      goto end;

    ip += opsize;

    write_ip(s, ip, 4);
```

```
39
40      s->ops[opcode](s, inst);
41
42      if (s->active == 0)
43        goto end;
44    }
45
46    return 0;
47
48  end:
49    return -1;
50  }
```

La boucle principale lit en boucle les codes d'opération virtuelles avant de lire l'instruction virtuelle complètement (appelé `bytecode` dans la suite du document). La lecture se fait en deux opérations car la longueur des bytecodes est variable.

Avant d'exécuter l'opération, la boucle incrémente le pointeur d'instruction et l'écrit dans le registre virtuel.

Enfin, une vérification est faite pour savoir si l'instruction n'a pas arrêter l'exécution du programme avant de recommencer avec l'instruction suivante.

2.2.5 Modèle mémoire de la machine virtuelle

L'étude des fonctions de lecture et d'écriture de la boucle principale révèle le modèle mémoire utilisé par la machine virtuelle. Celui-ci est décomposé en deux parties :

— le mapping `mblob` est utilisé comme pseudo-disque chiffré et sa taille est de 65536 octets ;

— le mapping `mmem` est utilisé comme pseudo-RAM pour la machine virtuelle. Sa taille est de 2048 octets.

Le déchiffrement se fait lors du transfère du disque vers la RAM.

Comme le disque est trop grand pour être contenu dans la RAM, il existe un mécanisme de pseudo-page. Ce mécanisme est présenté dans la figure 4.

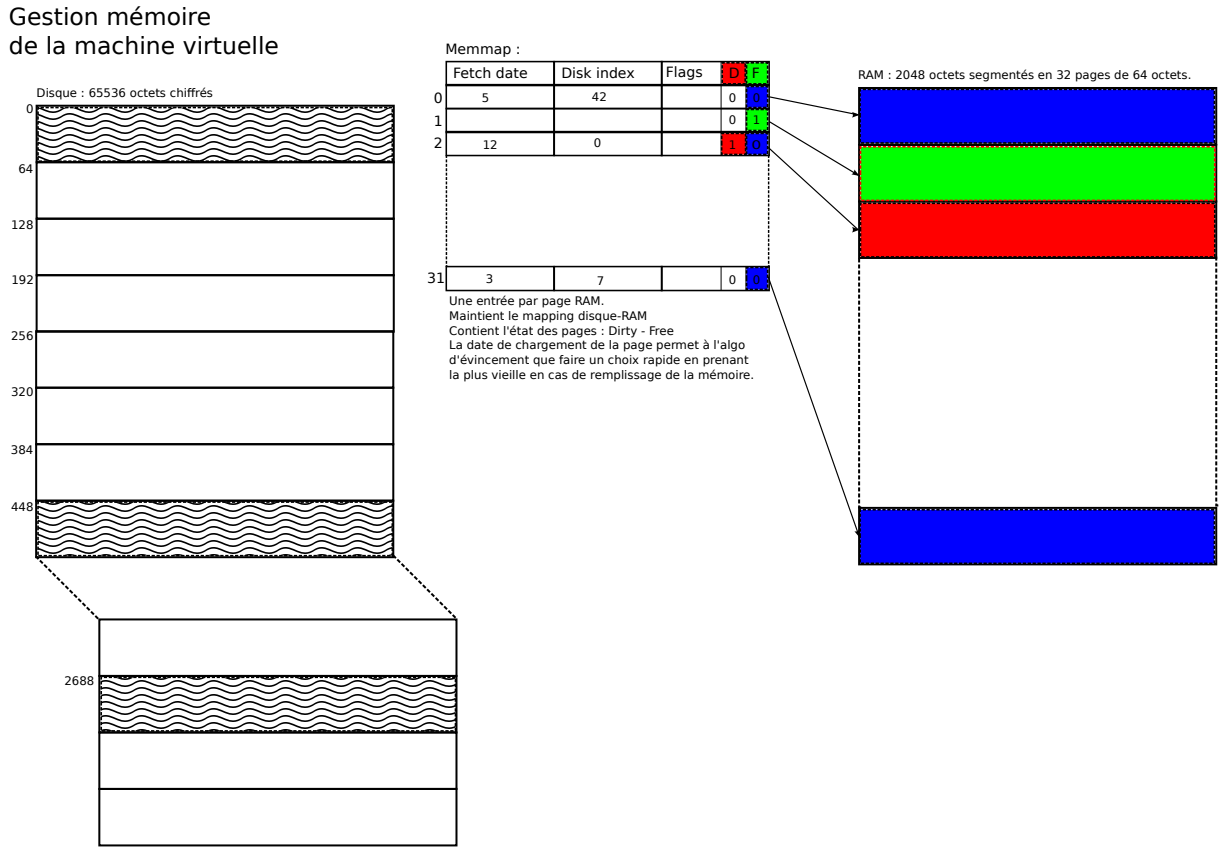

FIGURE 4 – Gestion de la mémoire de la machine virtuelle

Ce mécanisme utilise le tableau `memmap` pour maintenir la correspondance entre les pseudo-page de la RAM et du disque. Les pages ont une taille de 64 octets et sont alignées sur cette valeur aussi bien en RAM que sur disque. Le

tableau de translation à 32 entrées soit `32 * 64 = 2048` octets.

La structure d'une entrée `memmap` est la suivante :

```
struct memmap {
  long fetchdate;
  long index;
  long flags;
}
```

Le champ `index` contient le numéro de la page, celui-ci est calculé avec l'adresse sur disque divisé par 64. Enfin le champ `flags` est un champ de bits qui peut contenir deux valeurs :

— `0x1` pour marquer la page comme portant des modifications (dirty) ;

— `0x2` pour marquer la page comme non utilisé (free).

Lorsqu'une page est écrite le drapeau `dirty` est positionné. Si la RAM est pleine et qu'une nouvelle page doit être lue, l'algorithme choisit la page la plus âgé en RAM avec le champ `fetchdate` et le drapeau est vérifié pour savoir si une écriture sur disque est nécessaire. Si oui, la page est chiffrée pour retourner sur le disque avec le même algorithme que pour le déchiffrement. L'algorithme est symétrique. Le champ `fetchdate` est mis à jour à chaque chargement de la page en RAM avec la valeur de `cycles` (champ de la structure `vmstate`).

Le schéma 4 montre le tableau `memmap` contenant 3 pages utilisées : 0, 2 et 31. Ces 3 pages ont respectivement 42, 0 et 7 comme index sur le disques. Les zones correspondantes sont hachurées dans le disque. Elles sont déchiffrés en bleu lors du chargement en mémoire. La page 0 est passé en rouge car elle est modifiée (le drapeau `dirty` est positionné). Dans le cas où la mémoire serait pleine et que le programme aurait besoin de charger une nouvelle page, la page 31 serait évincée car elle a la date de chargement la plus ancienne (3). Si au moment de l'éviction la page 31 a été modifiée alors elle passera par la phase de chiffrement avant d'être copiée à son index d'origine sur disque soit l'adresse 448 ($7 * 64$).

2.2.6 Extraction du bytecode de la machine virtuelle

Une fois la gestion de la mémoire comprise, la tâche suivante est de récupérer le disque et de le déchiffrer. Le disque est déjà à disposition, c'est le fichier *bad.data.bin* qui a été extrait avec *gdb*. La clé est également a disposition et est la même pour l'intégralité du disque.

La rétro-conception de la fonction de mapping du disque en RAM montre que l'IV est simplement initialisé avec le numéro de page.

Enfin, il reste la rétro-conception de la fonction de chiffrement symétrique. Le listing 2.2.6 présente le code de l'algorithme de chiffrement.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

static unsigned int key[16] = { 0x61707865, 0x3120646e, 0x79622d36, 0x6b206574,
                                0x05b1ad0b, 0x05b1ad0b, 0x05b1ad0b, 0x05b1ad0b,
                                0x05b1ad0b, 0x05b1ad0b, 0x05b1ad0b, 0x05b1ad0b,
                                0, 0, 0, 0};

unsigned int _rotl(const unsigned int value, int shift) {
    if ((shift &= sizeof(value)*8 - 1) == 0)
        return value;
    return (value << shift) | (value >> (sizeof(value)*8 - shift));
}

unsigned int _rotr(const unsigned int value, int shift) {
    if ((shift &= sizeof(value)*8 - 1) == 0)
        return value;
    return (value >> shift) | (value << (sizeof(value)*8 - shift));
}

int main(int argc, char *argv[])
{
    int fdin, fdout, i;
    unsigned int blk[64/sizeof(unsigned int)];
    ssize_t size, wsize;

    if (argc != 3) {
        printf("Usage: %s ifile ofile\n", argv[0]);
        return 0;
    }
```

```
34
35      fdin = open(argv[1], O_RDONLY);
36      if (fdin == -1) {
37          printf("Open failed 1\n");
38          return -1;
39      }
40
41      fdout = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC);
42      if (fdout == -1) {
43          printf("Open failed 2\n");
44          return -1;
45      }
46
47      while ((size = read(fdin, blk, 64)) != 0) {
48          unsigned int tmp[16];
49          unsigned int res[16];
50
51          i = 0;
52          while (i < 16) {
53              tmp[i] = key[i];
54              i++;
55          }
56
57          i = 4;
58          while (i != 0) {
59              tmp[0]  += tmp[4];
60              tmp[1]  += tmp[5];
61              tmp[2]  += tmp[6];
62              tmp[3]  += tmp[7];
63              tmp[12] ^= tmp[0];
64              tmp[13] ^= tmp[1];
65              tmp[14] ^= tmp[2];
66              tmp[15] ^= tmp[3];
67              tmp[12] = _rotr(tmp[12], 16);
68              tmp[13] = _rotr(tmp[13], 16);
69              tmp[14] = _rotr(tmp[14], 16);
70              tmp[15] = _rotr(tmp[15], 16);
71              tmp[8]  += tmp[12];
72              tmp[9]  += tmp[13];
73              tmp[10] += tmp[14];
74              tmp[11] += tmp[15];
75              tmp[4]  ^= tmp[8];
76              tmp[5]  ^= tmp[9];
77              tmp[6]  ^= tmp[10];
78              tmp[7]  ^= tmp[11];
79              tmp[4]  = _rotr(tmp[4], 20);
80              tmp[5]  = _rotr(tmp[5], 20);
81              tmp[6]  = _rotr(tmp[6], 20);
82              tmp[7]  = _rotr(tmp[7], 20);
83              tmp[0]  += tmp[4];
84              tmp[1]  += tmp[5];
85              tmp[2]  += tmp[6];
86              tmp[3]  += tmp[7];
87              tmp[12] ^= tmp[0];
88              tmp[13] ^= tmp[1];
89              tmp[14] ^= tmp[2];
90              tmp[15] ^= tmp[3];
91              tmp[12] = _rotr(tmp[12], 24);
92              tmp[13] = _rotr(tmp[13], 24);
93              tmp[14] = _rotr(tmp[14], 24);
94              tmp[15] = _rotr(tmp[15], 24);
95              tmp[8]  += tmp[12];
96              tmp[9]  += tmp[13];
97              tmp[10] += tmp[14];
98              tmp[11] += tmp[15];
99              tmp[4]  ^= tmp[8];
100             tmp[5]  ^= tmp[9];
101             tmp[6]  ^= tmp[10];
102             tmp[7]  ^= tmp[11];
103             tmp[4]  = _rotr(tmp[4], 25);
104             tmp[5]  = _rotr(tmp[5], 25);
105             tmp[6]  = _rotr(tmp[6], 25);
106             tmp[7]  = _rotr(tmp[7], 25);
107             tmp[0]  += tmp[5];
108             tmp[1]  += tmp[6];
109             tmp[2]  += tmp[7];
```

```
            tmp[3]  += tmp[4];
            tmp[15] ^= tmp[0];
            tmp[12] ^= tmp[1];
            tmp[13] ^= tmp[2];
            tmp[14] ^= tmp[3];
            tmp[15] = _rotr(tmp[15], 16);
            tmp[12] = _rotr(tmp[12], 16);
            tmp[13] = _rotr(tmp[13], 16);
            tmp[14] = _rotr(tmp[14], 16);
            tmp[10] += tmp[15];
            tmp[11] += tmp[12];
            tmp[8]  += tmp[13];
            tmp[9]  += tmp[14];
            tmp[5]  ^= tmp[10];
            tmp[6]  ^= tmp[11];
            tmp[7]  ^= tmp[8];
            tmp[4]  ^= tmp[9];
            tmp[5]  = _rotr(tmp[5], 20);
            tmp[6]  = _rotr(tmp[6], 20);
            tmp[7]  = _rotr(tmp[7], 20);
            tmp[4]  = _rotr(tmp[4], 20);
            tmp[0]  += tmp[5];
            tmp[1]  += tmp[6];
            tmp[2]  += tmp[7];
            tmp[3]  += tmp[4];
            tmp[15] ^= tmp[0];
            tmp[12] ^= tmp[1];
            tmp[13] ^= tmp[2];
            tmp[14] ^= tmp[3];
            tmp[15] = _rotr(tmp[15], 24);
            tmp[12] = _rotr(tmp[12], 24);
            tmp[13] = _rotr(tmp[13], 24);
            tmp[14] = _rotr(tmp[14], 24);
            tmp[10] += tmp[15];
            tmp[11] += tmp[12];
            tmp[8]  += tmp[13];
            tmp[9]  += tmp[14];
            tmp[5]  ^= tmp[10];
            tmp[6]  ^= tmp[11];
            tmp[7]  ^= tmp[8];
            tmp[4]  ^= tmp[9];
            tmp[5]  = _rotr(tmp[5], 25);
            tmp[6]  = _rotr(tmp[6], 25);
            tmp[7]  = _rotr(tmp[7], 25);
            tmp[4]  = _rotr(tmp[4], 25);
            i--;
        }

        i = 0;
        while (i < 16) {
            tmp[i] = key[i] + tmp[i];
            i++;
        }
        i = 0;
        while (i < 16) {
            res[i] = tmp[i];
            i++;
        }
        key[12] += 1;
        if (key[12] == 0)
            key[13] += 1;

        i = 0;
        while (i < (size+3)/4) {
            res[i] = res[i] ^ blk[i];
            i++;
        }
        wsize = write(fdout, res, size);
        if (wsize != size) {
            printf("Write error\n");
            break;
        }
    }
    close(fdin);
    close(fdout);
    return 0;
```

```
186 }
```

dechiffrebad.c

Cette algorithme de chiffrement est un algorithme de chiffrement par flux de la famille *ChaCha*[6] dérivé de l'algorithme *Salsa*[7].

Le programme ci-dessous est utilisé sur le pseudo-disque pour obtenir le bytecode du programme.

```
$ ./dechiffrebad bad.data.bin good.data.bin
```

L'ouverture du fichier *good.data.bin* avec `hexedit` fait enfin apparaître les chaînes de caractères du programme (cf. 5).

```
00000030   00 00 00 00  00 20 00 00  00 00 00 00  40 00 00 00  00 01 00 00  01 21 00 00  ..... ......@........!..
00000048   00 02 00 00  01 12 00 00  00 03 00 00  01 E3 32 00  00 04 00 00  01 44 02 00  ..............2......D..
00000060   1D 00 00 01  00 00 01 11  00 00 0A 22  00 03 00 00  01 C3 3F 00  00 04 00 00  ...........".......?.....
00000078   01 04 01 00  1D 00 02 05  00 00 00 03  00 00 01 03  01 00 13 35  08 6A B4 02  ...................5.j..
00000090   00 0F 00 00  01 0F 01 00  00 0E 00 00  01 CE 3F 00  00 0D 00 00  01 6D 32 00  ..............?......m2.
000000A8   17 0D 00 02  00 00 01 02  03 00 00 03  00 00 01 93  03 00 00 04  00 00 01 14  ........................
000000C0   04 00 00 05  00 00 01 65  04 00 04 EC  00 00 02 01  2C 00 13 21  08 82 B4 02  .......e........,..!....
000000D8   02 01 2C 00  13 31 08 C2  06 01 02 01  2C 00 13 41  08 82 B4 02  02 01 2C 00  ..,..1......,..A......,.
000000F0   13 51 08 A2  B4 02 13 4C  00 01 00 00  01 A1 00 00  12 1C 08 00  08 01 13 2C  .Q.....L..............,
00000108   00 07 00 00  01 07 01 00  13 F7 00 01  00 00 01 11  00 00 0C 71  08 62 2C 01  ...................q.b,.
00000120   00 07 00 00  01 47 00 00  0D 7C 16 0D  04 D1 00 00  0B C1 07 D1  00 00 16 0E  .....G...|..............
00000138   17 0F 08 7E  CA 00 00 01  00 00 01 21  00 00 00 02  00 00 01 12  00 00 00 03  ...~.......!............
00000150   00 00 01 43  35 00 00 04  00 00 01 04  02 00 1D 00  00 01 00 00  01 61 32 00  ...C5..............a2.
00000168   02 1A 00 00  02 1B 04 00  0A 11 00 02  00 00 01 02  00 08 00 03  00 00 01 83  ........................
00000180   00 00 0A 44  00 0C 00 0B  01 0C 00 00  00 0D 00 00  01 1D 00 00  02 08 24 00  ...D..................$.
00000198   02 09 28 00  0C C8 0C D9  0A 98 1E 89  00 08 00 00  01 18 00 00  00 07 00 00  ..(.....................
000001B0   01 F7 01 00  02 06 24 00  0C 86 0D 76  0E 8B 0B 6B  0E 8A 0D 79  0B 9A 17 03  ......$....v...k...y....
000001C8   02 07 28 00  0C 87 0D 37  0B 74 08 66  F6 01 00 07  00 00 01 07  00 08 12 17  ..(....7.t.f............
000001E0   04 78 00 00  0A 48 07 78  00 00 00 03  00 00 01 83  00 00 16 01  0A 44 00 08  .x...H.x.............D..
000001F8   00 00 01 08  00 02 13 18  08 B0 94 01  00 0D 00 00  01 0D 00 08  00 0C 00 00  ........................
00000210   01 0C 00 02  00 0B 00 00  01 0B 08 00  0A AA 00 09  00 00 01 89  00 00 16 0A  ........................
00000228   17 0C 08 D8  DA 02 02 0A  30 00 12 CA  04 A1 00 00  08 42 26 02  13 B1 08 62  ........0........B&....b
00000240   DA 02 13 9A  08 D4 DA 02  0C 01 00 02  00 00 01 02  3F 00 00 03  00 00 01 13  ................?.......
00000258   24 00 00 04  00 00 01 64  1B 00 1D 00  08 82 B2 02  02 02 00 00  00 01 00 00  $.......d...............
00000270   01 21 00 00  00 03 00 00  01 03 00 08  02 04 2C 00  1D 00 00 01  00 00 01 31  .!..........,........1
00000288   00 00 1D 00  00 01 00 00  01 21 00 00  00 02 00 00  01 12 00 00  00 03 00 00  .........!..............
000002A0   01 23 3C 00  00 04 00 00  01 D4 02 00  1D 00 08 00  B2 02 1C 00  00 01 00 00  .#<.....................
000002B8   01 21 00 00  00 02 00 00  01 22 00 00  00 03 00 00  01 43 37 00  00 04 00 00  .!.......".......C7.....
000002D0   01 54 01 00  1D 00 08 00  B2 02 00 01  00 00 01 21  00 00 00 02  00 00 01 22  .T.............!.......".
000002E8   00 00 00 03  00 00 01 A3  38 00 00 04  00 00 01 44  01 00 1D 00  08 00 B2 02  ........8......D........
00000300   00 01 00 00  01 21 00 00  00 02 00 00  01 22 00 00  00 03 00 00  01 03 3A 00  .....!.......".........:.
00000318   00 04 00 00  01 14 02 00  1D 00 08 00  B2 02 00 00  00 00 00 00  00 00 3A 3A  ......................::
00000330   20 50 6C 65  61 73 65 20  65 6E 74 65  72 20 74 68  65 20 64 65  63 72 79 70   Please enter the decryp
00000348   74 69 6F 6E  20 6B 65 79  3A 20 00 00  3A 3A 20 54  72 79 69 6E  67 20 74 6F  tion key: ..:: Trying to
00000360   20 64 65 63  72 79 70 74  20 70 61 79  6C 6F 61 64  2E 2E 2E 0A  20 20 20 57   decrypt payload....   W
00000378   72 6F 6E 67  20 6B 65 79  20 66 6F 72  6D 61 74 2E  0A 00 20 20  20 49 6E 76  rong key format...   Inv
00000390   61 6C 69 64  20 70 61 64  64 69 6E 67  2E 0A 00 00  20 20 20 43  61 6E 6E 6F  alid padding....   Canno
000003A8   74 20 6F 70  65 6E 20 66  69 6C 65 20  70 61 79 6C  6F 61 64 2E  62 69 6E 2E  t open file payload.bin.
000003C0   0A 00 3A 3A  20 44 65 63  72 79 70 74  65 64 20 70  61 79 6C 6F  61 64 20 77  ..:: Decrypted payload w
000003D8   72 69 74 74  65 6E 20 74  6F 20 70 61  79 6C 6F 61  64 2E 62 69  6E 2E 0A 00  ritten to payload.bin...
000003F0   70 61 79 6C  6F 61 64 2E  62 69 6E 00  58 58 58 58  58 58 58 58  58 58 58 58  payload.bin.XXXXXXXXXXXX
00000408   58 58 58 58  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  XXXX....................
---  good.data.bin       --0x408/0x10018-----------------------------------------------------------------------------
```

FIGURE 5 – Le fichier *good.data.bin* ouvert avec `hexedit`

La table 5 présente l'agencement du pseudo-disque de la machine virtuelle.

Offset	Taille	Description
0	64	Registres : `r1` à `r14` + `sp` et `ip`
64	742	Bytecode
806	230	Données
1036	31732	Padding 1
32768	8192	Payload chiffré
40960	24575	Padding 2
65535	0	Fin de la mémoire

TABLE 5 – Agencement du pseudo-disque de la machine virtuelle

6. http://cr.yp.to/chacha.html
7. https://en.wikipedia.org/wiki/Salsa20

3 Analyse du bytecode de la machine virtuelle

L'étape suivante de résolution du challenge est de comprendre chaque bytecode dans la machine virtuelle. Pour cela, il faut comprendre les opérations réalisés dans la boucle principale de la machine virtuelle lorsque les fonctions au format :

```
1 void (*func_t)(struct vmstate *s, unsigned int instruction);
```

sont exécuter à partir du bytecode. Trente et une fonctions sont stockés dans le tableau `ops` de la *VM*. La rétro-conception de ces fonctions nous indique que la pseudo-architecture comprend 17 registres : un registre appelé `rz` toujours égal à 0, 14 registres généraux appelés `r1` à `r14`, un pointeur de pile `sp` et un pointeur d'instruction `ip`. Dans le codage binaire des bytecodes, les registres sont composé de 4 bits soit 16 registres possibles car le pointeur d'instruction n'est pas accessible directement pour les opérations.

Enfin, l'étude des registres indique que ceux-ci sont stockés au début de la mémoire. Le registre zéro n'a pas besoin d'être stocké car sa valeur est toujours la même (zéro), il peut être écrit mais la valeur écrite est alors perdue. Il reste 14 registres généraux, le pointeur de pile et le pointeur d'instruction chacun faisant 4 octets soit une zone registre de 64 octets. Le bytecode commence après cette zone à partir du décalage 64. De plus la valeur initiale du pointeur d'instruction est 64. Le début du bytecode est aussi sont point d'entrée.

Le tableau 6 liste les opérations désassemblées.

Opcode	Binaire	Instruction	Description
0	0000iiiiiiiiiiiiiiiidddd00000000	movimm rd, #imm«16	remplir `rd` avec la valeur immédiate 0x*imm*0000
1	0000iiiiiiiiiiiiiiiidddd00000001	orimm rd, #imm	ou avec immédiat
2	iiiiiiiiiiiiiiiissssdddd00000010	lddw rd, [rs,#imm]	charger un double-mot
3	iiiiiiiiiiiiiiiissssdddd00000011	ldw rd, [rs,#imm]	charger un mot
4	iiiiiiiiiiiiiiiissssdddd00000100	ldb rd, [rs,#imm]	charger un octet
5	iiiiiiiiiiiiiiiissssdddd00000101	stdw rd, [rs,#imm]	stocker un double-mot
6	iiiiiiiiiiiiiiiissssdddd00000110	stw rd, [rs,#imm]	stocker un mot
7	iiiiiiiiiiiiiiiissssdddd00000111	stb rd, [rs,#imm]	stocker un octet

8	iiiiiiiiiiiiiiiicccssssk00001000	if k == 1 then
		cc.cond rs, #imm
		else
		cj.cond rs, #imm
		fi
		cond = 0 = toujours
		cond = 1 = jamais
		cond = 2 = égal à zéro
		cond = 3 = différent de zéro
		cond = 4 = inférieur à zéro
		cond = 5 = supérieur à zéro
		cond = 6 = inférieur ou égal à zéro
		cond = 7 = supérieur ou égal à zéro
			appel ou saut conditionnel

9	0000dddd00001001	not rd	non logique
10	ssssdddd00001010	xor rd, rs	ou exclusif
11	ssssdddd00001011	or rd, rs	ou avec registre
12	ssssdddd00001100	and rd, rs	et logique
13	ssssdddd00001101	lls rd, rs	décalage logique à gauche
14	ssssdddd00001110	lrs rd, rs	décalage logique à droite
15	ssssdddd00001111	ars rd, rs	décalage arithmétique à droit
16	ssssdddd00010000	lrot rd, rs	rotation à gauche
17	ssssdddd00010001	rrot rd, rs	rotation à droite
18	ssssdddd00010010	add rd, rs	addition
19	ssssdddd00010011	sub rd, rs	soustraction
20	ssssdddd00010100	mul rd, rs	multiplication
21	ssssdddd00010101	udiv rd, rs	division
22	0000dddd00010110	inc rd	incrementation
23	0000dddd00010111	dec rd	décrementation
24	0000ssss00011000	push rs	pousser sur la pile
25	0000dddd00011001	pop rd	récupérer sur la pile
26	0000000000011010	ret	retour de fonction
27	0000000000011011	nop	opération blanche
28	0000000000011100	stop	arrêt de la machine virtuelle
29	0000000000011101	vmcall (see 7)	appelle aux fonctions spéciales de la machine virtuelle
30	ssssdddd00011110	xorb rd, rs	ou exclusif bit à bit de `rs`

TABLE 6 – Bytecodes

Tous les octets sont des instructions classiques mise à part la vingt neuvième qui agit comme un appel système pour lequel il faut positionner les registres avant appel. Le registre `r1` définit la fonction à appeler et stocke la valeur de retour. Les registres suivant `r2, r3, ...` contiennent les paramètres. Le tableau 7 récapitulatif des différents fonctions et leurs paramètres.

Valeur de r1	Registres paramètre utilisés	Opération effectuée	Description
0	r2,r3,r4	`int open(char *path, int flags unsigned int mode)`	Ouvre le fichier `path`. Ouvre le fichier `path`.
1	r2,r3,r4	`int read(int fd, int offset, int size)`	Lit `size` octets à partir du descripteur de fichier `fd` et stocke cette lecture à `offset` en mémoire.
2	r2,r3,r4	`int write(int fd, int offset, int size)`	Écrit `size` octets stocké à la position `offset` en mémoire dans le fichier `fd`.
3	r2	`int close(int fd)`	Ferme le fichier `fd`.

TABLE 7 – Bytecode 29 : vmcall

Avec ces informations, un désassembleur de bytecode est simple. L'annexe 5 fournit le désassembleur utilisé pour la rétro-conception du programme.

Ci-dessous le résultat, en pseudo-code, de la rétro-conception du bytecode. Plusieurs informations supplémentaires ici : la zone de mémoire nommé `buf` dans le pseudo-code ci-dessous est la zone marqué par des `X` dans la capture d'écran 5 (seize grand `X` pour les seize caractères *ASCII* entrés par l'utilisateur), la zone de mémoire nommé `lfsr` dans le pseudo-code est une zone de 8 octets à l'adresse `0x326`, cette zone est coincé entre la fin du bytecode (en `0x325`) et la zone de stockage des chaînes de caractères qui commence en `0x32E`. La zone mémoire nommé `payload` commence à l'adresse `0x8000` est à une taille de `0x2000` octets, elle stocke les données du fichier *payload.bin* chiffré.

```
entry_point:
  size = write(stdout, ":: Please enter the decryption key: ", 36);

  size = read(stdin, buf, 16);

  if (size != 16)
    goto wrong_format;

  idx = 0;
  symbol = 0;

  // loop convert hexa key to binary.
  while (size != 0) {
    if (buf[idx] >= 'A' && buf[idx] <= 'F') {
      buf[idx] -= 'A';
      buf[idx] += 10;
    } else if (buf[idx] >= '0' && buf[idx] <= '9') {
      buf[idx] -= '0';
    } else {
      goto wrong_format;
    }

    if ((size % 2) == 0) {
      buf[idx] = buf[idx]<<4;
      symbol++;
    }

    lfsr[symbol] |= buf[idx];
    size--;
    idx++;
  }

  write(stdout, ":: Trying to decrypt payload...\n", 32);

  idx = 0;
  count = 8;
  mask = 0;
  size = 0x2000;

  // compute LFSR and XOR with payload to decrypt
```

```
41    do {
42      newbit = xorb(lfsr & 0xb000000000000001);
43
44      lfsr = (lfsr >> 1) | (newbit << 63);
45
46      bit = lfsr & 1;
47
48      mask = bit << count;
49      count--;
50
51      if (count == 0) {
52        payload[idx] ^= mask;
53        count = 8;
54        idx++;
55        mask = 0;
56      }
57
58    } while (size - idx > 0);
59
60    size--;
61    // set 'size' on the last non-null character of payload
62    while (payload[size] == 0) {
63      if (size <= 0)
64        goto invalid_padding;
65      size--;
66    }
67
68    // payload must end with the byte 0x80
69    if (data[size] - 0x80 != 0)
70      goto invalid_padding;
71
72    // invalid condition ??? data = 0x8000, size > 0 => condition always true ?
73    // Shouldn't it be (0x2000 - size <= 8) ?
74    // payload must end with at least 8 bytes set to 0 ?
75    if (data + size - 8 <= 0)
76      goto invalid_padding;
77
78    fd = open("payload.bin", O_TRUNC|O_CREAT|O_WRONLY, 0666);
79    if (fd < 0)
80      goto ok;
81
82    write(fd, payload, size);
83    close(fd);
84
85    write(stdout, ":: Decrypted payload written to payload.bin.\n", 45);
86  ok:
87    stop;
88  wrong_format:
89    write(stderr, "   Wrong key format.\n", 21);
90    stop;
91  invalid_padding:
92    write(stderr, "   Invalid padding.\n", 20);
93    stop;
94  cannot_open:
95    write(stderr, "   Cannot open file payload.bin.\n", 33);
96    stop;
```

Note : la ligne 80 du listing semble fausse car elle ne pointe pas sur le label qui affiche le message d'erreur `Cannot open file payload.bin` *alors que la condition correspond pourtant à ce message. Le message n'est donc jamais affiché par le programme.*

Les lignes 41-58 du listing ci-dessus montre la boucle qui calcul le registre à décalage à rétroaction linéaire (souvent noté par l'acronyme anglophone LFSR venant de *Linear Feedback Shift Register*. La fonction `xorb` représente l'instruction `xorb` vu dans le tableau des bytecodes 6. Cette fonction est *étendue* en 64 bits dans le listing pour pour avoir une meilleur lisibilité mais cela ne change rien à la logique du calcul. Le schéma 6 représente le même calcul du LFSR.

Le schéma montre que le bit utilisé pour le masque de chiffrement n'est pas celui qui sort du registre mais la nouvelle valeur en tête du registre contrairement à ce qui se fait habituellement (le bit est mis en valeur en vert sur le schéma). C'est à dire qu'à t_0 le registre est d'abord décalé, le bit de tête est perdu, et c'est seulement à t_1 que le nouveau bit de tête est récupéré pour le masque de chiffrement. Ce détail est important pour l'implémentation de l'algorithme.

Le LFSR a comme polynôme de rétroaction : $T(x) = 1 + x + x^{60} + x^{61} + x^{63}$.

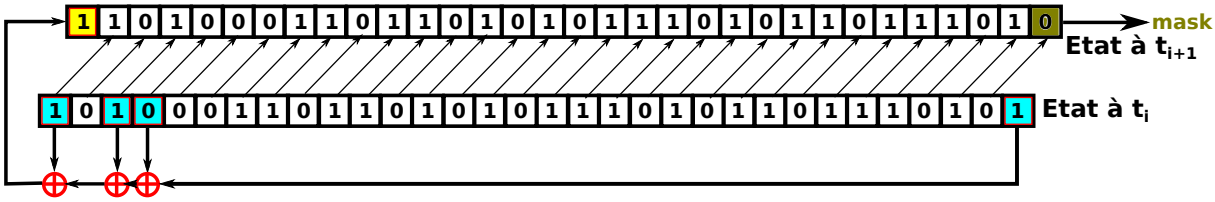

FIGURE 6 – Le LFSR calculé dans le bytecode

Les lignes 62-70 indiquent que la zone mémoire du payload devrait se terminer par les octets $0x80, 0, 0, 0, 0, 0, \ldots$. La ligne 74 n'est pas très clair, il est probable que la ligne contienne une erreur de programmation comme la ligne 80, mais la supposition qui a été faite par l'auteur ici est que la chaîne de zéro à la fin du payload doit avoir une longueur d'au moins 8 octets soit la largeur du LFSR. Comme la valeur 0 est l'élément neutre de l'opérateur ou exclusif ($x^0 = x$), les 8 derniers octets du payload chiffré sont identiques à la valeur du LFSR à la fin du chiffrement soit `0x6AB654C3CA8F5302`. Avec la valeur finale du LFSR est son polynôme de rétroaction, il est possible de remonter la valeur du LFSR pour déchiffrer le payload et retrouver la valeur initiale de la clé.

Le programme `revert_lfsr.c` en annexe 5 a été utilisé pour cette opération.

```
$ ./revert_lfsr
key = 0x0BADB10515DEAD11
$ qemu-aarch64 badbios.bin
:: Please enter the decryption key: 0BADB10515DEAD11
:: Trying to decrypt payload...
:: Decrypted payload written to payload.bin.
$ file payload.bin
payload.bin: Zip archive data, at least v2.0 to extract
$ mv payload.bin payload.zip
```

La clé trouvée a la valeur hexadécimale `0BADB10515DEAD11`. Cette valeur est de l'Hexspeak [8] qui confirme que ce programme est terminé car *badbios is dead!!*. Le payload déchiffré est un fichier de type ZIP.

4 Analyse du firmware

La première chose à faire est de décompresser l'archive ZIP.

```
$ unzip payload.zip
Archive:  ../payload.zip
  inflating: mcu/upload.py
  inflating: mcu/fw.hex
$ file mcu/upload.py
mcu/upload.py: Python script, ASCII text executable
$ file file mcu/fw.hex
mcu/fw.hex: ASCII text
```

L'archive contient un script python et un fichier ASCII *fw.hex*. Ces deux fichiers sont dans un répertoire nommé `mcu` qui est la notation abrégée en anglais de *MicroController Unit* [9]. Voici le listing du fichier *upload.py* :

```
1  #!/usr/bin/env python
2
3  import socket, select
4
5  #
6  # Microcontroller architecture appears to be undocumented.
7  # No disassembler is available.
8  #
9  # The datasheet only gives us the following information:
10 #
11 #   == MEMORY MAP ==
12 #
13 #   [0000-07FF] - Firmware                    \
```

8. https://fr.wikipedia.org/wiki/Hexspeak
9. https://fr.wikipedia.org/wiki/Microcontrôleur

```
#    [0800-0FFF] - Unmapped                  | User
#    [1000-F7FF] - RAM                       /
#    [F000-FBFF] - Secret memory area        \
#    [FC00-FCFF] - HW Registers              | Privileged
#    [FD00-FFFF] - ROM (kernel)              /
#

FIRMWARE = "fw.hex"

print("-----------------------------------------------")
print("----- Microcontroller firmware uploader -----")
print("-----------------------------------------------")
print()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('178.33.105.197', 10101))

print(":: Serial port connected.")
print(":: Uploading firmware... ", end='')

[ s.send(line) for line in open(FIRMWARE, 'rb') ]

print("done.")
print()

resp = b''
while True:
    ready, _, _ = select.select([s], [], [], 10)
    if ready:
        try:
            data = s.recv(32)
        except:
            break
        if not data:
            break
        resp += data
    else:
        break

print(resp.decode("utf-8"))
s.close()
```

mcu/upload.py

Maintenant que le script Python a été lu, une tentative d'exécution s'impose :

```
$ ./upload.py
-----------------------------------------------
----- Microcontroller firmware uploader -----
-----------------------------------------------

:: Serial port connected.
:: Uploading firmware... done.

System reset.
Firmware v1.33.7 starting.
Execution completed in  8339 CPU cycles.
Halting.
```

Comme son nom l'indique, le fichier *upload.py* envoie le firmware qui l'accompagne à une adresse sur le réseau. Le commentaire en début de fichier indique que l'architecture du microcontrolleur n'est pas documenté mais le mapping mémoire est fourni. L'information intéressante ici semble être la *Secret memory area* dans la mémoire du microcontrolleur mais avant de pouvoir la récupérer il faut trouver un moyen de récupérer le jeu d'instruction du microcontrolleur pour en faire la rétro-conception.

Le fichier *fw.hex* ouvert dans un éditeur texte montre que le firmware est au format HEX [10]. Ce format est utilisé depuis les années 1970 pour prendre en charge de l'information binaire pour des composants programmables comme les microcontrolleurs.

Le format HEX est un ensemble de lignes de texte. Chaque ligne respecte la syntaxe suivante :

10. https://fr.wikipedia.org/wiki/HEX_(Intel)

```
:SSAAAATTHHHHHH.....HHHHCC
```

Tous les champs sont en hexadécimal. Voici la description de chaque champ :

SS est la taille en octets des données de la ligne ;

AAAA est l'adresse absolue où sont stocké les données de la ligne en mémoire ;

TT est le champ spécifiant le type. Dans le cas du firmware seules les valeurs `0x00` : *Données* et `0x01` : *Fin de fichier* sont utilisées ;

HH...HHHH est le champ des données ;

CC est l'octet de somme de contrôle. C'est le complément à deux de la somme des valeurs binaires des octets de toute la ligne, entêtes comprises.

L'image 7 montre la version binaire du firmware convertie avec la commande :

```
$ cut -c 10-41 fw.hex | head -n -1 | xxd -p -r > fw.bin
```

FIGURE 7 – Affichage du firmware avec *hexedit*

Plusieurs chaînes de caractères sont visibles :

— `"YeahRiscIsGood!"` à l'adresse `0x17C`, longueur 15 ;

— `"Firmware v1.33.7 starting."` à l'adresse `0x18C`, longueur 27 (avec le retour chariot) ;

— `"Halting."` à l'adresse `0x1A8`, longueur 9 (avec le retour chariot).

Les deux dernières chaînes sont visibles lors de l'exécution du firmware. La chaîne `Firmware v1.33.7 starting` est la première a être affichée, le code l'affichant devrait probablement être au début du firmware. Les 8 premiers octets du firmware sont :

```
21 00 11 1B 20 01 10 8C
```

L'octet `1B` correspond à 27, la longueur de la chaîne. Le dernier octet est `8C` et l'antépénultième `01`, ils vont probablement ensemble pour former l'adresse `18C` de la chaîne de caractères. On retrouve `21 11 20 10` dans les octets de point fort de chaque mot pour ce qui est probablement l'opcode de l'instruction. Comme le mapping mémoire l'indique, le

microcontrolleur à un espace d'adressage de 16 bits, il est probable que les instructions soient aussi codé sur 16 bits et que leurs longueur soit fixe pour garder étage de décodage simple dans les circuits logiques du microcontrolleur.

Ces quatre premières instructions sont suivies de `C0 D2`, probablement une instruction de saut dans la fonction d'affichage des chaînes de caractères. Changer cette instruction par `C0 D1` pour dérouter le programme donne le résultat suivant.

```
-------------------------------------------
----- Microcontroller firmware uploader -----
-------------------------------------------

:: Serial port connected.
:: Uploading firmware... done.

System reset.
-- Exception occurred at 00DB: Unaligned instruction.
   r0:018C     r1:001B    r2:0000    r3:0000
   r4:0000     r5:0000    r6:0000    r7:0000
   r8:0000     r9:0000   r10:0000   r11:0000
  r12:0000    r13:EFFE   r14:0000   r15:000A
   pc:00DB fault_addr:0000 [S:0 Z:0] Mode:user
CLOSING: Unaligned instruction.
```

Une exception est lancée et la trace retournée donne plusieurs informations :

- Le message de l'exception est *Unaligned instruction*. Cela semble normal au vu des suppositions exprimées ci-dessus sur l'adressage 16 bits. Les instructions sont fixes et de longueur 16 bits et alignés sur 16 bits, il n'est donc pas possible de sauter à une adresse impaire comme c'est le cas ici avec `pc = 0x00DB`.
- `r0` est chargé avec l'adresse de la chaîne *Firmware v1.33.7 starting.* ;
- `r1` est chargé avec la taille de la chaîne soit 27 en décimal ;
- `pc` contient `0xDB` soit `0xD1`, l'adresse immédiate stocké dans l'instruction que nous avons modifié, plus `0xA` l'adresse du pointeur d'instruction lors de l'exécution de l'instruction. L'instruction `C0 XX` serait donc un saut relatif de `pc + XX`.
- `r15` contient `0xA` qui contient l'adresse suivante du saut modifié ce qui correspond à l'adresse de retour pour un appel de fonction. `r15` correspondrait au *link register* en ARM qui sauvegarde l'adresse suivante séquentiellement pour faire un branchement sur celle-ci à la prochaine instruction `ret`.
- `r13` contient une valeur élevée en fin de RAM. Cette valeur correspond probablement au pointeur de pile.
- Deux drapeaux `S` et `Z` sont présents pour contrôler les conditions du microcontrolleur. `S` correspond probablement au signe (`Sign`) du résultat de l'opération et `Z` si l'opération donne pour résultat zéro (`Zero`).
- Le microcontrolleur est en mode *user*, il contient très certainement un mode *kernel* dans lequel le noyau dans la mémoire la ROM s'exécute. De plus, le mapping mémoire indique que les adresses `0xF000-0xFFFF` contenant la zone secrète, les registres et la ROM sont privilégiées, le mode *kernel* permettra probablement d'accéder à ces plages d'adresses.

La trace est très utile et il est très simple de la déclencher avec une instruction de saut non alignée comme réalisé dans le cas présent. En expérimentant à partir de les quelques instructions que nous connaissons déjà (`mov rX, #imm` ou `call #imm`) il est possible de découvrir les autres instructions au nombre de 16.

Opcode	Binaire	Instruction	Description
0			Instruction invalide
1	`0001ddddiiiiiiii`	movlo rd, #imm	remplir la valeur basse de `rd` avec la valeur immédiate `0x`*imm*
2	`0010ddddiiiiiiii`	movhi rd, #imm	remplir la valeur haute de `rd` avec la valeur immédiate `0x`*imm*
3	`0011ddddffffssss`	xor rd, rf, rs	`rd` reçoit le résultat du ou exclusif entre `rf` et `rs`
4	`0100ddddffffssss`	or rd, rf, rs	`rd` reçoit le résultat du ou inclusif entre `rf` et `rs`
5	`0101ddddffffssss`	and rd, rf, rs	`rd` reçoit le résultat du et logique entre `rf` et `rs`
6	`0110ddddffffssss`	add rd, rf, rs	`rd` reçoit le résultat de l'addition entre `rf` et `rs`
7	`0111ddddffffssss`	sub rd, rf, rs	`rd` reçoit le résultat de la soustraction entre `rf` et `rs`
8	`1000ddddffffssss`	mul rd, rf, rs	`rd` reçoit le résultat de la multiplication entre `rf` et `rs`
9	`1001ddddffffssss`	div rd, rf, rs	`rd` reçoit le résultat de la division entre `rf` et `rs`
10	`1010csiiiiiiiiii`	cj.*cs* #imm - `cs = 00` : égal à zéro (`S=? && Z=1`) - `cs = 01` : différent de zéro (`S=? && Z=0`) - `cs = 10` : inférieur à zéro (`S=1 && Z=?`) - `cs = 11` : supérieur ou égal à zéro (`S=0 && Z=?`)	saut conditionnel relatif de #imm octets (#imm est signé)
11	`101100iiiiiiiiii`	jmp #imm	saut inconditionnel relatif de #imm octets (#imm est signé)
12	`110000iiiiiiiiii`	call #imm	appel inconditionnel relatif de #imm octets (#imm est signé)
12	`1100100000000ii`	syscall #imm	appel système numéro #imm. Passe en mode *kernel*. Le numéro ne peut aller que de 1 à 3 : - `0 = reset_MCU` : réinitialisation du microcontrolleur - `1 = exit` : arrêt de l'exécution de firmware - `2 = print` : affichage de `r1` caractères de la chaîne stockée à l'adresse contenu dans `r0` - `3 = get_cycles` : écrit à l'adresse `r0` le nombre de cycles écoulé depuis la ré-initialisation du microcontrolleur
13	`1101000000000ssss`	ret rs	retour à l'adresse contenu dans `rs`
13	`1101100000000000`	iret	retour en mode `user`
14	`1110ddddffffssss`	ld rd, [rf,rs]	chargement de la mémoire à l'adresse `rf + rs` dans `rd`
15	`1111ddddffffssss`	st rd, [rf,rs]	écriture du registre `rd` dans l'adresse mémoire `rf + rs`

TABLE 8 – Instructions du microcontrolleur

Les informations contenues dans le tableau 8 permettent d'écrire un désassembleur. Un désassembleur est donné en annexe 5.

Le firmware en lui-même fait très peu de chose. Il écrit une séquence d'octets en RAM avant d'effectuer une série de permutation grâce à la clé `YeahRiscIsGood!` présente dans le firmware. Ces permutations sont ensuite utilisé comme flux de déchiffrement pour la phrase `Execution completed in $$$$$ CPU cycles.` stockée à l'adresse `0x01B2`. Une fonction se charge enfin de remplacer le pattern `$$$$$` par le nombre de cycle récupérer par l'appel système *get_ cycles* au format décimal. A noter que cette dernière fonction a une variante hexadécimale dans le code du firmware mais la variante hexadécimale n'est jamais appelée, elle est en code mort.

Le firmware utilise une grande variété d'instruction, cela permet d'apprendre le jeu d'instruction plus rapidement mais cela ne semble pas directement utile à la poursuite du challenge.

La rétro-conception du firmware explique aussi le fonctionnement des appels systèmes et l'information intéressante ici est la présence un l'appel système *print*. Un appel système s'exécute en mode *kernel*. Ce mode privilégié devrait permettre d'accéder à la zone secrète et de l'afficher. Un petit firmware tel que celui présenté ci-dessous devrait permettre d'exécuter l'opération d'affichage de la zone secrète :

```
  movhi r1, #0x0C
  movlo r1, #0x00
  movhi r0, #0xF0
  movlo r0, #0x00
  syscall #0x2
  syscall #0x1
```

Le registre `r1` stocke la taille de la zone secrète `0xC00` et `r0` stocke son adresse `0xF000`. Il ne reste plus qu'à appeler l'appel système `print`.

Le programme *bin2hex.c* en annexe fournit un convertisseur de fichier binaire vers format HEX (annexe 5).

Après assemblage et empaquetage dans le format HEX, le programme ressemble à ceci :

```
$ cat fw.hex
:0C000000210c110020f01000c802c80103
:00000001FF
```

Un appel à *upload.py* affiche le résultat suivant :

```
$ ./upload.py
-------------------------------------------
----- Microcontroller firmware uploader -----
-------------------------------------------

:: Serial port connected.
:: Uploading firmware... done.

System reset.
[ERROR] Printing at unallowed address. CPU halted.
```

Le noyau semble empêcher l'affichage de cette adresse. Pour en avoir la confirmation est-il possible de faire la rétro-conception de noyau ? Pour cela il faut tenter d'afficher la zone noyau avec le petit firmware ci-dessus modifié.

```
  movhi r1, #0x03
  movlo r1, #0x00
  movhi r0, #0xFD
  movlo r0, #0x00
  syscall #0x2
  syscall #0x1
```

Ici, l'adresse est `0xFD00` et la taille `0x0300`.

L'assemblage et l'empaquetage donne ceci :

```
$ cat fw.hex
:0C0000002103110020fd1000c802c801FF
:00000001FF
```

Cette fois le résultat de l'appel à *upload.py* est :

```
$ ./upload.py
-------------------------------------------
----- Microcontroller firmware uploader -----
-------------------------------------------

:: Serial port connected.
:: Uploading firmware... done.

Traceback (most recent call last):
  File "./upload.py", line 53, in <module>
    print(resp.decode("utf-8"))
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xa0 in \
                    position 16: invalid start byte
```

Le noyau est binaire est le script python tente de le convertir en UTF-8. Ce n'est pas une bonne idée. La ligne 53 du script sera remplacé par une écriture dans un fichier.

```
#print(resp.decode("utf-8"))
outFile = open("resp.fw.bin", "wb")
outFile.write(resp)
```

Ça fonctionne, le fichier binaire *resp.fw.bin* contient bien le noyau du microcontrolleur. La capture d'écran de la figure 8 montre le noyau dans *hexedit* où la chaîne de caractères `[ERROR] Printing at unallowed address.` est présente notamment.

```
00000000   53 79 73 74  65 6D 20 72  65 73 65 74  2E 0A 50 00  A0 6C 21 00  11 03 72 10  System reset..P..l!...r.
00000018   A8 12 22 00  12 02 81 02  71 12 20 F0  10 00 60 01  C0 94 D0 00  21 00 11 2B  .."".....q. ...`.....!..+
00000030   20 FE 10 5A  C0 BE 30 00  21 FC 11 10  22 00 12 01  F2 10 B3 F2  20 FC 10 22   ..Z..0.!..."....... .."
00000048   C0 74 55 00  20 FC 10 20  C0 6C 51 55  C0 9E D8 00  20 FC 10 20  C0 60 26 FC  .tU. .. .lQU.... .. .`&.
00000060   16 12 21 00  11 01 34 44  E5 61 E2 64  E3 64 73 32  A7 F6 23 01  13 00 82 23  ..!...4D.a.d.ds2..#....#
00000078   41 25 C0 56  D8 00 21 00  11 0E 20 FE  10 86 C0 6C  24 00 14 02  21 FD 11 28  A%.V..!... ....l$...!..(
00000090   20 F0 10 00  C0 3C 60 04  21 FD 11 36  C0 34 60 04  21 FD 11 4A  C0 2C 20 FC   ....<`.!..6.4`.!..J., .
000000A8   10 20 31 11  22 00 12 36  C0 32 20 FC  10 3A 21 EF  11 FE C0 16  D8 00 21 00  . 1."..6.2 ..:!.......!.
000000C0   11 01 22 01  12 00 E3 01  71 11 E4 01  84 42 40 34  D0 0F 22 00  12 01 23 01  .."".....q....B@4.."...#.
000000D8   13 00 F1 02  72 22 91 13  F1 02 D0 0F  23 00 13 01  52 22 A0 06  72 23 F1 02  ....r"......#...R"..r#..
000000F0   B3 F2 D0 0F  5E 00 2D FC  1D 00 2C F0  1C 00 38 88  59 88 2A 00  1A 01 3B BB  ....^.-...,...8.Y.*...;.
00000108   51 11 A0 1A  69 E8 79 9C  A8 08 69 E8  79 9D AC 02  B0 0E 39 99  E9 E8 F9 DB  Q...i.y...i.y.....9.....
00000120   68 8A 71 1A  B3 E2 D0 0F  21 00 11 33  20 FE 10 26  C3 C2 B3 02  5B 45 52 52  h.q.....!..3 ..&....[ERR
00000138   4F 52 5D 20  50 72 69 6E  74 69 6E 67  20 61 74 20  75 6E 61 6C  6C 6F 77 65  OR] Printing at unallowe
00000150   64 20 61 64  64 72 65 73  73 2E 20 43  50 55 20 68  61 6C 74 65  64 2E 0A 00  d address. CPU halted...
00000168   5B 45 52 52  4F 52 5D 20  55 6E 64 65  66 69 6E 65  64 20 73 79  73 74 65 6D  [ERROR] Undefined system
00000180   20 63 61 6C  6C 2E 20 43  50 55 20 68  61 6C 74 65  64 2E 0A 00  53 79 73 74   call. CPU halted...Syst
00000198   65 6D 20 72  65 73 65 74  2E 0A 00 00  00 00 00 00  00 00 00 00  00 00 00 00  em reset.............
000001B0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000001C8   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000001E0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000001F8   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000210   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000228   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000240   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000258   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000270   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000288   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000002A0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000002B8   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000002D0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
000002E8   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ....................
00000300   00 00 00 00  00 00 00 00  00 00 00 00  00 00                                 ..............
00000318
00000330
00000348
00000360
00000378
00000390
000003A8
000003C0
000003D8
---  resp.fw.bin       --0x0/0x30E-----------------------------------------------------------------------------
```

FIGURE 8 – Affichage de la ROM contenant le noyau du microcontrolleur

La rétro-conception du noyau confirme que la zone secrète est protégée contre l'affichage. Il faudrait trouver un contournement à cette protection. Sans indice, il y a nécessité de collecter plus d'information sur le noyau pour avancer.

La fonction d'affichage du noyau écrit caractère par caractère dans un registre hardware à l'adresse `0xFC00`. Ce registre sert à écrire sur le port série qui nous est renvoyé.

La fonction *exit* écrit la valeur 1 dans le registre hardware à l'adresse `0xFC10` pour l'arrêter l'exécution du firmware.

La fonction *get_ cycles* lit les registres hardware aux l'adresses `0xFC12` et `0xFC13` pour connaître le nombre de cycles écoulés depuis la ré-initialisation du microcontrolleur (voir récapitulatif tableau 9).

La rétro-conception permet également de découvrir que la zone secrète commence par un tableau de trois pointeurs de fonction. Lors d'un appel système, ce tableau est indexé par le numéro de l'appel système en cours et le pointeur de fonction récupéré est alors exécuté avant le retour en mode utilisateur.

Adresse pointeur	Adresse fonction	Fonction	Registres hardware utilisés
0xF000	0xFD28	exit	0xFC10
0xF002	0xFD36	print	0xFC00
0xF004	0xFD4A	get_cycles	0xFC12 et 0xFC13

TABLE 9 – Contenu du tableau de pointeur rempli lors de l'initialisation du microcontrolleur et registre hardware utilisé par les appels systèmes

En étudiant attentivement l'appel système *get_ cycles* on peut voir qu'il peut écrire le nombre de cycles n'importe où en mémoire. C'est une information importante car cette écriture peut potentiellement aider à écraser un pointeur de fonction de la table des appels systèmes. Cet écrasement de pointeur pourrait amener à une élévation de privilège ce qui permettrait le contournement nécessaire à l'afficher de la zone secrète. Le problème est que le nombre de cycle n'est pas une information maîtriser par l'utilisateur.

Le petit programme de teste ci-dessous va permettre de mesurer le nombre de cycles le plus tôt possible lors du démarrage du firmware :

```
movhi r0, 0x10
movlo r0, 0x00
syscall #3
movhi r1, 0x00
movlo r1, 0x02
movhi r0, 0x10
movlo r0, 0x00
syscall #2
syscall #1
```

Le nombre de cycles sera stocké au début de la zone de RAM (`0x1000`). Ensuite le nombre de cycles sera affiché et le firmware quittera.

Ci-dessous le programme de teste assemblé et formaté :

```
$ cat fw.hex
:10000000020101000c80321001102201010000c802A7
:02001000c80125
:00000001FF
```

En exécutant ce programme de test, l'affichage récupérer (en binaire) est :

```
$ hexdump -C resp.fw.bin
00000000  53 79 73 74 65 6d 20 72  65 73 65 74 2e 0a 07 c0  |System reset....|
00000010
```

La sortie hexadécimal des deux octets est `0x07c0` (deux derniers octets du dump). Cette sortie est très intéressante car le nombre de cycle est compris dans la plage d'adresse `0x0000-0x07FF` du firmware et est aligné sur un multiple de 2. Il suffit donc d'écrire ce nombre de cycle au début de la zone secrète pour écraser un des pointeurs de fonction appelé lors d'un appel système ce qui détournera l'exécution du noyau vers une adresse que l'utilisateur contrôle.

Voici les étapes nécessaires à la création de l'exploit :

1. Faire un appel au syscall *get_ cycles* le plus tôt possible comme pour le programme de teste ci-dessus mais avec l'adresse d'un pointeur de fonction dans la zone secrète.
2. Au retour de l'appel système *get_ cycles*, appeler le syscall écrasé. Le noyau appel le pointeur `0x07C0`.
3. A l'adresse `0x07C0`, le firmware sera chargé avec une fonction d'affichage modifiée qui affichera toute la zone secrète.
4. La fonction d'affichage modifiée retourne en mode utilisateur et le firmware peut appeler l'appel système *exit* pour quitter proprement.

```
# étape 1: écrasement du pointeur de fonction de "print"
 0000   movhi r0, 0xF0
 0002   movlo r0, 0x02
 0004   syscall #3
# étape 2: appel au pointeur écrasé
 0006   syscall #2
 0008   syscall #1
 ...
# étape 3: affichage de la zone secrète
 07C0   movhi r1, 0x0C
 07C2   movlo r1, 0x00
 07C4   movhi r0, 0xF0
 07C6   movlo r0, 0x00
 07C8   movhi r13, 0xFC
 07CA   movlo r13, 0x00
 07CC   movhi r10, 0x00
 07CE   movlo r10, 0x01
 07D0   xor r8, r8, r8
 07D2   xor r11, r11, r11
 07D4   xor r9, r9, r9
 07D6   ld r9, [r0,r8]
 07D8   st r9, [r13,r11]
 07DA   add r8, r8, r10
 07DC   sub r1, r1, r10
 07DE   cj.eq 0x7D4
# étape 4: retour et arrêt du firmware
 07E0   iret
```

La réponse semble contenir beaucoup de caractères ASCII à partir de l'adresse `0xF61A`. Une petite modification du firmware permet de n'avoir que les caractères imprimables.

```
 07C0   movhi r1, 0x05
 07C2   movlo r1, 0xE6
 07C4   movhi r0, 0xF6
 07C6   movlo r0, 0x1A
```

Il faut changer de nouveau la sortie du script python pour avoir la conversion UTF-8 originale.
Un jolie ASCII art, contient l'adresse mail attendue (voir la figure 9). Le fichier *fw.hex* final est en annexe 5.

5 Conclusion

Le challenge du SSTIC 2014 est un voyage en rétro-conception. On début avec un programme en assembleur AArch64 assez simple, dont les spécifications sont connues, puis un autre programme AArch64 obfusqué cette fois. Ce dernier nous mène vers une machine virtuelle dont l'assembleur inconnu nous pousse à faire la rétro-conception avec moins d'information. Par la suite, on obtient un dernier assembleur, inconnu également, sur lequel il faudra se poser des questions plus profonde concernant le format des instructions, leur taille, le nombres de registres, leur utilité, ...pour enfin comprendre ces routines et réaliser une élévation des privilèges.

Tous mes remerciements aux concepteurs de ce challenge intéressant, original et varié.

Annexes

usbmontext2pcap.sh

```
#!/bin/bash

FILE=${1}
OUT=out

rm -rf ${OUT}
mkdir -p ${OUT}
MERGFILE="${OUT}/usbtrace.pcap"

```

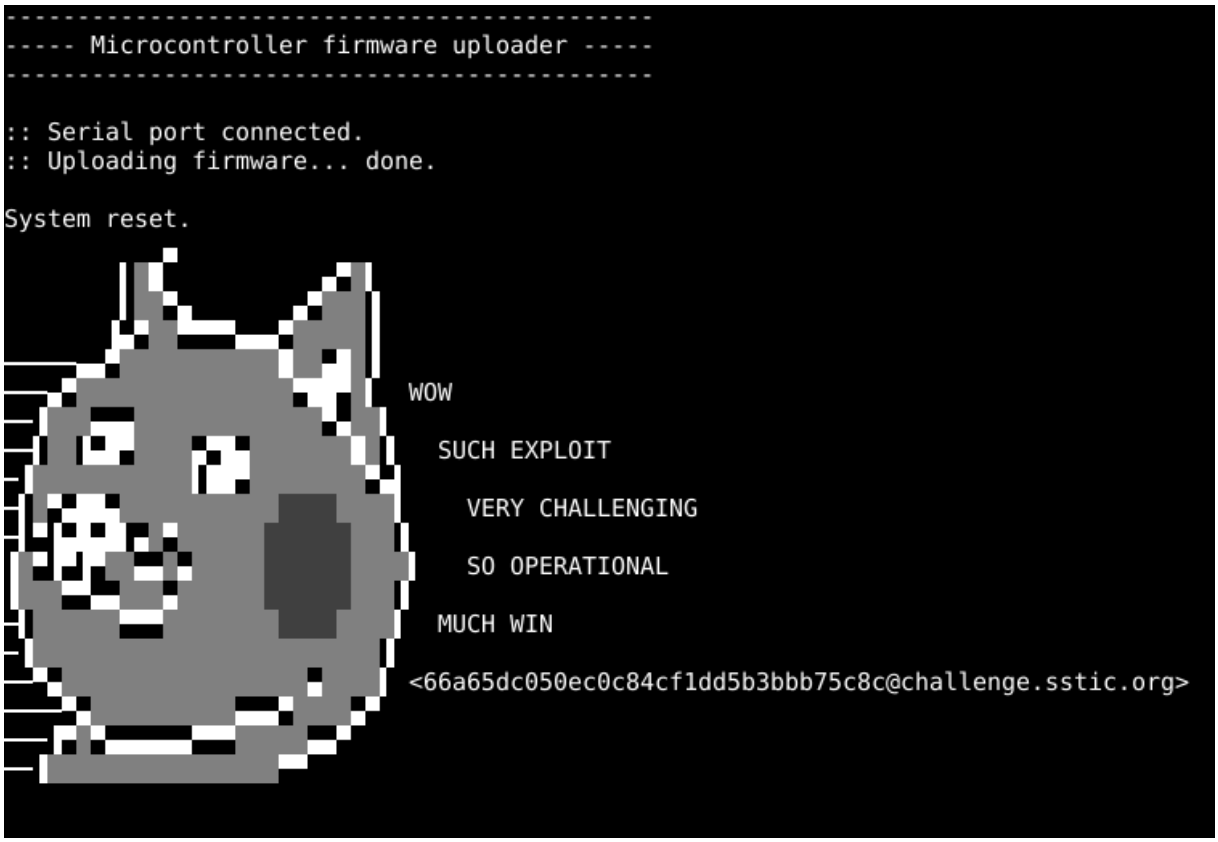

FIGURE 9 – ASCII art

```
endian8() {
    printf "%02x%02x%02x%02x%02x%02x%02x%02x" ${0:14:2} ${0:12:2} ${0:10:2} ${0:8:2} ${0:6:2} $
    {0:4:2} ${0:2:2} ${0:0:2}
}

i=0
while read l; do
    declare -a tab=(${l})
    BASEOUT="${OUT}/usb-$(printf "%04d" ${i})"
    OUTFILE="${BASEOUT}.hex"
    BINFILE="${BASEOUT}.bin"
    CAPFILE="${BASEOUT}.pcap"
    DATFILE="${BASEOUT}.data"

    #u64 id;          /*  0: URB ID - from submission to callback */
    printf "%s%s%s%s%s%s%s%s" ${tab[0]:14:2} ${tab[0]:12:2} ${tab[0]:10:2} ${tab[0]:8:2} ${tab
    [0]:6:2} ${tab[0]:4:2} ${tab[0]:2:2} ${tab[0]:0:2} > ${OUTFILE}
    #unsigned char type;/*  8: Same as text; extensible. */
    if [[ ${tab[2]} == "S" ]]; then
  printf "%02x" 83 >> ${OUTFILE}
    elif [[ ${tab[2]} == "C" ]]; then
  printf "%02x" 67 >> ${OUTFILE}
    else
  printf "%02x" 69 >> ${OUTFILE}
    fi
    #unsigned char xfer_type; /*    ISO (0), Intr, Control, Bulk (3) */
    if [[ ${tab[3]:0:1} == "I" ]]; then
  printf "%02x" 1 >> ${OUTFILE}
    elif [[ ${tab[3]:0:1} == "C" ]]; then
  printf "%02x" 2 >> ${OUTFILE}
    elif [[ ${tab[3]:0:1} == "B" ]]; then
  printf "%02x" 3 >> ${OUTFILE}
    else
  printf "%02x" 0 >> ${OUTFILE}
    fi
    #unsigned char epnum; /*     Endpoint number and transfer direction */
    epnum=$((${tab[3]:9:1}))
    if [[ ${tab[3]:1:1} == "i" ]]; then
```

```
  epnum=$((${epnum}+128))
    fi
    printf "%02x" ${epnum} >> ${OUTFILE}
    #unsigned char devnum;  /*     Device address */
    printf "%02x" $((${tab[3]:5:1}*100+${tab[3]:6:1}*10+${tab[3]:7:1})) >> ${OUTFILE}
    #u16 busnum;  /* 12: Bus number */
    printf "%02x%02x" ${tab[3]:3:1} 0 >> ${OUTFILE}
    #char flag_setup; /* 14: Same as text */
    if [[ ${tab[2]} == "S" && ${tab[3]:0:1} == "C" && ${tab[4]} == "s" ]]; then
  printf "%02x" 0 >> ${OUTFILE}
    else
  printf "%02x" 45 >> ${OUTFILE}
    fi
    #char flag_data;  /* 15: Same as text; Binary zero is OK. */
    if [[ ${tab[6]} == "=" ]]; then
  printf "%02x" 0 >> ${OUTFILE}
    elif [[ ${tab[6]} == "<" ]]; then
  printf "%02x" 60 >> ${OUTFILE}
    elif [[ ${tab[6]} == ">" ]]; then
  printf "%02x" 62 >> ${OUTFILE}
    else
  printf "%02x" 0 >> ${OUTFILE}
    fi
    #s64 ts_sec;  /* 16: gettimeofday */
    sec="$(printf "%016x" $((${tab[1]}/1000)))"
    printf "%s%s%s%s%s%s%s%s" ${sec[@]:14:2} ${sec[@]:12:2} ${sec[@]:10:2} ${sec[@]:8:2} ${sec[@
    ]:6:2} ${sec[@]:4:2} ${sec[@]:2:2} ${sec[@]:0:2} >> ${OUTFILE}
    #s32 ts_usec; /* 24: gettimeofday */
    msec="$(printf "%08x" ${tab[1]})"
    printf "%s%s%s%s" ${msec[@]:6:2} ${msec[@]:4:2} ${msec[@]:2:2} ${msec[@]:0:2} >> ${OUTFILE}
    #int status;  /* 28: */
    if [[ ${tab[2]} == "S" && ${tab[3]:0:1} == "C" ]]; then
  printf "%08x" 0 >> ${OUTFILE}
    else
  if [[ ${tab[3]:0:1} == "I" ]]; then
      len=$(echo -n ${tab[4]} | cut -d: -f1 | wc -c)
      stat="$(printf "%016x" ${tab[4]:0:$((len-1))})"
      printf "%s%s%s%s" ${stat[@]:14:2} ${stat[@]:12:2} ${stat[@]:10:2} ${stat[@]:8:2} >> ${OUTFILE}
  else
      stat="$(printf "%016x" ${tab[4]})"
      printf "%s%s%s%s" ${stat[@]:14:2} ${stat[@]:12:2} ${stat[@]:10:2} ${stat[@]:8:2} >> ${OUTFILE}
  fi
    fi
    #unsigned int length;   /* 32: Length of data (submitted or actual) */
    if [[ ${tab[2]} != "S" || ${tab[3]:0:1} != "C" ]]; then
  length="$(printf "%08x" ${tab[5]})"
  printf "%s%s%s%s" ${length[@]:6:2} ${length[@]:4:2} ${length[@]:2:2} ${length[@]:0:2} >> ${OUTFILE
    }
    else
  length="$(printf "%08x" ${tab[10]})"
  printf "%s%s%s%s" ${length[@]:6:2} ${length[@]:4:2} ${length[@]:2:2} ${length[@]:0:2} >> ${OUTFILE
    }
    fi
    #unsigned int len_cap;  /* 36: Delivered length */
    if [[ ${tab[6]} == "=" ]]; then
  if [[ ${tab[2]} != "S" || ${tab[3]:0:1} != "C" ]]; then
      length="$(printf "%08x" ${tab[5]})"
      printf "%s%s%s%s" ${length[@]:6:2} ${length[@]:4:2} ${length[@]:2:2} ${length[@]:0:2} >> ${
    OUTFILE}
  else
      length="$(printf "%08x" ${tab[10]})"
      printf "%s%s%s%s" ${length[@]:6:2} ${length[@]:4:2} ${length[@]:2:2} ${length[@]:0:2} >> ${
    OUTFILE}
  fi
    else
  printf "%08x" 0 >> ${OUTFILE}
    fi
    #unsigned char setup[SETUP_LEN];  /* 40: Only for Control S-type */
    if [[ ${tab[2]} == "S" && ${tab[3]:0:1} == "C" && ${tab[4]} == "s" ]]; then
  setup="$(echo -n ${l} | cut -ds -f2- | tr -d ' ')"
  printf "%s%s%s%s%s%s%s%s" ${setup[@]:0:2} ${setup[@]:2:2} ${setup[@]:6:2} ${setup[@]:4:2} ${setup[
    @]:10:2} ${setup[@]:8:2} ${setup[@]:14:2} ${setup[@]:12:2} >> ${OUTFILE}
  setupsize=$(echo -n ${setup[@]:0:16} | wc -c)
  if [[ ${setupsize} != 16 ]]; then
      echo "Warning: Setup is different than 16 (${setupsize})"
  fi
```

```
    else
  printf "%016x" 0 >> ${OUTFILE}
    fi
    #int interval;      /* 48: Only for Interrupt and ISO */
    if [[ ${tab[3]:0:1} == "I" ]]; then
  len=$(echo -n ${tab[4]} | cut -d: -f1 | wc -c)
  len2=$(echo -n ${tab[4]} | cut -d: -f2 | wc -c)
  stat="$(printf "%016x" ${tab[4]:${len}:$((${len2}-1))})"
  printf "%s%s%s%s" ${stat[@]:14:2} ${stat[@]:12:2} ${stat[@]:10:2} ${stat[@]:8:2} >> ${OUTFILE}
    else
  printf "%08x" 0 >> ${OUTFILE}
    fi
    #int start_frame;   /* 52: For ISO */
    printf "%08x" 0 >> ${OUTFILE}
    #unsigned int xfer_flags; /* 56: copy of URB's transfer_flags */
    length="$(printf "%08x" 0)"
    printf "%s%s%s%s" ${length[@]:6:2} ${length[@]:4:2} ${length[@]:2:2} ${length[@]:0:2} >> ${
    OUTFILE}
    #unsigned int ndesc; /* 60: Actual number of ISO descriptors */
    printf "%08x" 0 >> ${OUTFILE}
    pad=0
    if [[ ${tab[6]} == "=" ]]; then
  echo -n ${l} | cut -d= -f2- | tr -d ' ' > ${DATFILE}
  pad=$(wc -c ${DATFILE} | cut -d' ' -f1)
  pad=$((${pad}-1))
  pad=$((${pad}/2))
  cat ${DATFILE} >> ${OUTFILE}
  if [[ ${pad} != ${tab[5]} ]]; then
      echo "Warning: data length is different ${pad} != ${tab[5]}"
  fi
    fi
    cat ${OUTFILE} | xxd -r -p - > ${BINFILE}
    od -Ax -tx1 -v ${BINFILE} | text2pcap -q -l 220 -m$((64+${pad})) - ${CAPFILE}
    if [[ $i == 0 ]]; then
  prev=${CAPFILE}
    elif [[ $i == 1 ]]; then
  mergecap -w ${MERGFILE} ${CAPFILE} ${prev}
    else
  mv ${MERGFILE} ${MERGFILE}.tmp
  mergecap -w ${MERGFILE} ${CAPFILE} ${MERGFILE}.tmp
    fi
    i=$((i+1))
done < ${FILE}
```

usbmontext2pcap.sh

adboverusbdissector.lua

```
-- ADB 1.0 Protocol Dissector For Wireshark
--
-- Cameron Gutman (aicommander@gmail.com)
-- Anthoine Bourgeois (anthoine.bourgeois@gmail.com)
-- Licensed under GPLv3
--

-- Standard ADB Header
local pf_header_command = ProtoField.uint32("adb.command", "Command", base.HEX)
local pf_header_arg0 = ProtoField.uint32("adb.arg0", "Arg0", base.HEX)
local pf_header_arg1 = ProtoField.uint32("adb.arg1", "Arg1", base.HEX)
local pf_header_datalen = ProtoField.uint32("adb.datalen", "Data Length", base.HEX)
local pf_header_datachk = ProtoField.uint32("adb.datachk", "Data Checksum", base.HEX)
local pf_header_magic = ProtoField.uint32("adb.magic", "Magic", base.HEX)
local pf_header_payload = ProtoField.bytes("adb.payload", "Payload")

-- Connect message
local pf_cnxn = ProtoField.bytes("adb.cnxn", "Connect Message")
local pf_cnxn_version = ProtoField.uint32("adb.cnxn.version", "Version", base.HEX)
local pf_cnxn_maxdata = ProtoField.uint32("adb.cnxn.maxdata", "Max Data", base.HEX)
local pf_cnxn_sysident = ProtoField.string("adb.cnxn.sysident", "System Identifier")

-- Open message
local pf_open = ProtoField.bytes("adb.open", "Open Stream Message")
local pf_open_localid = ProtoField.uint32("adb.open.localid", "Local ID", base.HEX)
local pf_open_dest = ProtoField.string("adb.open.dest", "Destination")

```

```
-- Okay message
local pf_okay = ProtoField.bytes("adb.okay", "Stream Ready Message")
local pf_okay_localid = ProtoField.uint32("adb.okay.localid", "Local ID", base.HEX)
local pf_okay_remoteid = ProtoField.uint32("adb.okay.remoteid", "Remote ID", base.HEX)

-- Write message
local pf_write = ProtoField.bytes("adb.write", "Write Stream Message")
local pf_write_localid = ProtoField.uint32("adb.write.localid", "Local ID", base.HEX)
local pf_write_remoteid = ProtoField.uint32("adb.write.remoteid", "Remote ID", base.HEX)
local pf_write_data = ProtoField.bytes("adb.write.databytes", "Data (Bytes)")
local pf_write_string = ProtoField.string("adb.write.datastring", "Data (String)")

-- Close message
local pf_close = ProtoField.bytes("adb.close", "Close Stream Message")
local pf_close_localid = ProtoField.uint32("adb.close.localid", "Local ID", base.HEX)
local pf_close_remoteid = ProtoField.uint32("adb.close.remoteid", "Remote ID", base.HEX)

-- Auth messages
local pf_auth_token = ProtoField.bytes("adb.auth.token", "Auth Message (Token Signing Request)")
local pf_auth_sig = ProtoField.bytes("adb.auth.signature", "Auth Message (Signed Token Reply)")
local pf_auth_pubkey = ProtoField.bytes("adb.auth.pubkey", "Auth Message (Public Key)")
local pf_auth_payload = ProtoField.bytes("adb.auth.payload", "Auth Payload")

p_adb = Proto ("adb", "Android Debug Bridge Protocol")
p_adb.fields = {
    pf_header_command,
    pf_header_arg0,
    pf_header_arg1,
    pf_header_datalen,
    pf_header_datachk,
    pf_header_magic,
    pf_header_payload,
    pf_cnxn,
    pf_cnxn_version,
    pf_cnxn_maxdata,
    pf_cnxn_sysident,
    pf_open,
    pf_open_localid,
    pf_open_dest,
    pf_okay,
    pf_okay_localid,
    pf_okay_remoteid,
    pf_write,
    pf_write_localid,
    pf_write_remoteid,
    pf_write_data,
    pf_write_string,
    pf_close,
    pf_close_localid,
    pf_close_remoteid,
    pf_auth_token,
    pf_auth_sig,
    pf_auth_pubkey,
    pf_auth_payload
    }

-- Helper XOR function shamelessly scraped from the interwebs
function bxor (a,b)
  local r = 0
  for i = 0, 31 do
    local x = a / 2 + b / 2
    if x ~= math.floor (x) then
      r = r + 2^i
    end
    a = math.floor (a / 2)
    b = math.floor (b / 2)
  end
  return r
end

function p_adb.dissector(buf, pkt, root)
    pkt.cols.protocol = p_adb.name

    subtree = root:add(p_adb, buf(0))

    -- Process all the ADB packets within the TCP packet
```

```
    i = 0;

    if buf:len() ~= 24 then
  -- We need another segment
  payloadentry = subtree:add(pf_header_payload, buf(i, buf:len()))
  i = i + buf:len()
    else
  -- This marks the start of the current packet
  pktst = i

  command = buf(i, 4):le_uint()
  cmdentry = subtree:add(pf_header_command, buf(i, 4), command)
  i = i + 4

  arg0 = buf(i, 4):le_uint()
  arg0entry = subtree:add(pf_header_arg0, buf(i, 4), arg0)
  i = i + 4

  arg1 = buf(i, 4):le_uint()
  arg1entry = subtree:add(pf_header_arg1, buf(i, 4), arg1)
  i = i + 4

  datalen = buf(i, 4):le_uint()
  datalenentry = subtree:add(pf_header_datalen, buf(i, 4), datalen)
  i = i + 4

  datachk = buf(i, 4):le_uint()
  datachkentry = subtree:add(pf_header_datachk, buf(i, 4), datachk)
  i = i + 4

  magic = buf(i, 4):le_uint()
  magicentry = subtree:add(pf_header_magic, buf(i, 4), magic)
  i = i + 4

  -- Validate the magic
  if bxor(command, 0xFFFFFFFF) ~= magic then
      -- Invalid magic
      magicentry:add_expert_info(PI_MALFORMED, PI_WARN)
      return
  end

  -- Process the specific commands
  subtree:add("")
  if command == 0x4e584e43 then
      -- CNXN
      subtree:add(pf_cnxn, buf(pktst, 24))

      versionentry = subtree:add(pf_cnxn_version, buf(pktst+4, 4), arg0)
      if arg0 ~= 0x01000000 then
    -- Invalid version
    versionentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      maxdataentry = subtree:add(pf_cnxn_maxdata, buf(pktst+8, 4), arg1)
      if arg1 == 0 then
    -- Invalid max data
    maxdataentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if datalen == 0 then
    -- No sysident string
    datalenentry:add_expert_info(PI_MALFORMED, PI_ERROR)
    return
      end
  elseif command == 0x4e45504f then
      -- OPEN
      open = 1
      subtree:add(pf_open, buf(pktst, 24))

      localidentry = subtree:add(pf_open_localid, buf(pktst+4, 4), arg0)
      if arg0 == 0 then
    -- Invalid local ID
    localidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if arg1 ~= 0 then
```

```
    -- Remote ID must be zero
    arg0entry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if datalen == 0 then
    -- We must have a destination to open
    datalenentry:add_expert_info(PI_PROTOCOL, PI_ERROR)
    return
      end
  elseif command == 0x59414b4f then
      -- OKAY
      subtree:add(pf_okay, buf(pktst, 24))

      localidentry = subtree:add(pf_okay_localid, buf(pktst+4, 4), arg0)
      if arg0 == 0 then
    -- Invalid local ID
    localidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      remoteidentry = subtree:add(pf_okay_remoteid, buf(pktst+8, 4), arg1)
      if arg1 == 0 then
    -- Invalid remote ID
    remoteidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if datalen ~= 0 then
    -- We shouldn't have a payload here
    datalenentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end
  elseif command == 0x45534c43 then
      -- CLOSE
      subtree:add(pf_close, buf(pktst, 24))

      localidentry = subtree:add(pf_close_localid, buf(pktst+4, 4), arg0)

      remoteidentry = subtree:add(pf_close_remoteid, buf(pktst+8, 4), arg1)
      if arg1 == 0 then
    -- Invalid remote ID
    remoteidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if datalen ~= 0 then
    -- We shouldn't have a payload here
    datalenentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end
  elseif command == 0x45545257 then
      -- WRITE
      subtree:add(pf_write, buf(pktst, 24))

      localidentry = subtree:add(pf_write_localid, buf(pktst+4, 4), arg0)
      if arg0 == 0 then
    -- Invalid local ID
    localidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      remoteidentry = subtree:add(pf_write_remoteid, buf(pktst+8, 4), arg1)
      if arg1 == 0 then
    -- Invalid remote ID
    remoteidentry:add_expert_info(PI_PROTOCOL, PI_WARN)
      end

      if datalen == 0 then
    -- We should have a payload here
    datalenentry:add_expert_info(PI_PROTOCOL, PI_ERROR)
    return
      end
  elseif command == 0x48545541 then
      -- AUTH
      if arg0 == 1 then
    -- Token request
    subtree:add(pf_auth_token, buf(pktst, 24))
      elseif arg0 == 2 then
    -- Signature reply
    subtree:add(pf_auth_sig, buf(pktst, 24))

    if datalen ~= 0x100 then
```

```
        -- Signature length is wrong
        datalenentry:add_expert_info(PI_PROTOCOL, PI_WARN)
        return
    end
      elseif arg0 == 3 then
    -- Public key
    subtree:add(pf_auth_pubkey, buf(pktst, 24))
      else
    -- Unrecognized type
    arg0entry:add_expert_info(PI_PROTOCOL, PI_ERROR)
    return
      end
  else
      -- Unrecognized message
      cmdentry:add_expert_info(PI_PROTOCOL, PI_ERROR)
      return
  end
    end
end

function p_adb.init()
end

-- Devices listen on TCP port 5555 for remote ADB connections
local usb_dissector_table = DissectorTable.get("usb.bulk")
usb_dissector_table:add(0xffff, p_adb)
```

adboverusbdissector.lua

bytecode_disassembler.c

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

enum opcodes {
    MOVIMM,
    ORIMM,
    LOAD_DWORD,
    LOAD_WORD,
    LOAD_BYTE,
    STORE_DWORD,
    STORE_WORD,
    STORE_BYTE,
    COND_JMP_CALL,
    NOT,
    XOR,
    OR,
    AND,
    LLEFT_SHIFT,
    LRIGHT_SHIFT,
    ARIGHT_SHIFT,
    LEFT_ROTATION,
    RIGHT_ROTATION,
    ADD,
    SUB,
    MUL,
    UDIV,
    INC,
    DEC,
    PUSH_DWORD,
    POP_DWORD,
    RET,
    NOP,
    STOP,
    VMCALL,
    XORB
};

const char * condstr(unsigned int cond)
{
```

```
45      switch (cond) {
46          case 0:
47              return "al";
48          case 1:
49              return "nv";
50          case 2:
51              return "ez";
52          case 3:
53              return "nz";
54          case 4:
55              return "ng";
56          case 5:
57              return "po";
58          case 6:
59              return "le";
60          case 7:
61              return "ge";
62          default:
63              return "??";
64      }
65  }
66
67  const char * regstr(unsigned int reg)
68  {
69      switch (reg) {
70          case 0:
71              return "rz";
72          case 1:
73              return "r1";
74          case 2:
75              return "r2";
76          case 3:
77              return "r3";
78          case 4:
79              return "r4";
80          case 5:
81              return "r5";
82          case 6:
83              return "r6";
84          case 7:
85              return "r7";
86          case 8:
87              return "r8";
88          case 9:
89              return "r9";
90          case 10:
91              return "r10";
92          case 11:
93              return "r11";
94          case 12:
95              return "r12";
96          case 13:
97              return "r13";
98          case 14:
99              return "r14";
100         case 15:
101             return "sp";
102         default:
103             return "r?";
104     }
105 }
106
107
108 int main(int argc, char *argv[])
109 {
110     FILE *in, *out;
111     long entry = 64;
112     unsigned int end = 806, i;
113     size_t size, wsize;
114     unsigned char opcode, opsize;
115     unsigned int inst, rout, rin, imm, offset, cond, jmp, nbinst;
116     int ret;
117
118     if (argc != 3) {
119         printf("Usage: %s binaryfile disassemblefile\n", argv[0]);
120         return 0;
```

```
121     }
122
123     in = fopen(argv[1], "r");
124     if (in == NULL) {
125         printf("Open failed (%d): '%s'\n", __LINE__, strerror(errno));
126         return -1;
127     }
128
129     out = fopen(argv[2], "w+");
130     if (out == NULL) {
131         printf("Open failed (%d): '%s'\n", __LINE__, strerror(errno));
132         return -1;
133     }
134
135     fseek(in, entry, SEEK_SET);
136
137     i = entry;
138     nbinst = 0;
139     while (i < end) {
140         size = fread(&opcode, 1, 1, in);
141         if (size != 1) {
142             printf("Opcode read error : '%s'\n", strerror(errno));
143             break;
144         }
145
146         ret = fseek(in, -1, SEEK_CUR);
147         if (ret == -1) {
148             printf("Seek error : '%s'\n", strerror(errno));
149             break;
150         }
151
152         opsize = opcode > 8 ? 2 : 4;
153
154         inst = 0;
155         size = fread(&inst, opsize, 1, in);
156         if (size != 1) {
157             printf("Instruction read error : '%s'\n", strerror(errno));
158             break;
159         }
160
161
162         if (opcode > 30) {
163             printf("Invalid opcode %d\n", opcode);
164             break;
165         }
166
167         fprintf(out, "  %08x:\t %08x:%d\t", i, inst, opsize);
168         switch ((enum opcodes)opcode) {
169             case MOVIMM:
170                 imm = (inst & 0xffff000)<<4;
171                 rout = (inst & 0xf00)>>8;
172                 fprintf(out, "mov\t %s, #0x%x\n", regstr(rout), imm);
173                 break;
174             case ORIMM:
175                 imm = (inst & 0xffff000)>>12;
176                 rout = (inst & 0xf00)>>8;
177                 fprintf(out, "or\t %s, #0x%x\n", regstr(rout), imm);
178                 break;
179             case LOAD_DWORD:
180                 rin = (inst & 0xf000)>>12;
181                 offset = (inst & 0xffff0000)>>16;
182                 rout = (inst & 0xf00)>>8;
183                 fprintf(out, "ldw\t %s, [%s,#0x%x]\n",
184                         regstr(rout), regstr(rin), offset);
185                 break;
186             case LOAD_WORD:
187                 rin = (inst & 0xf000)>>12;
188                 offset = (inst & 0xffff0000)>>16;
189                 rout = (inst & 0xf00)>>8;
190                 fprintf(out, "lw\t %s, [%s,#0x%x]\n",
191                         regstr(rout), regstr(rin), offset);
192                 break;
193             case LOAD_BYTE:
194                 rin = (inst & 0xf000)>>12;
195                 offset = (inst & 0xffff0000)>>16;
196                 rout = (inst & 0xf00)>>8;
```

```
                fprintf(out, "lb\t %s, [%s,#0x%x]\n",
                        regstr(rout), regstr(rin), offset);
                break;
            case STORE_DWORD:
                rin = (inst & 0xf000)>>12;
                offset = (inst & 0xffff0000)>>16;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "sdw\t %s, [%s,#0x%x]\n",
                        regstr(rout), regstr(rin), offset);
                break;
            case STORE_WORD:
                rin = (inst & 0xf000)>>12;
                offset = (inst & 0xffff0000)>>16;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "sw\t %s, [%s,#0x%x]\n",
                        regstr(rout), regstr(rin), offset);
                break;
            case STORE_BYTE:
                rin = (inst & 0xf000)>>12;
                offset = (inst & 0xffff0000)>>16;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "sb\t %s, [%s,#0x%x]\n",
                        regstr(rout), regstr(rin), offset);
                break;
            case COND_JMP_CALL:
                rin = (inst & 0x1e00)>>9;
                offset = (inst & 0xffff0000)>>16;
                cond = (inst & 0xe000)>>13;
                jmp = (inst & 0x100)>>8;
                if (jmp) {
                    fprintf(out, "cc.%s\t %s, #0x%x\n",
                            condstr(cond), regstr(rin), offset);
                } else {
                    fprintf(out, "cj.%s\t %s, #0x%x\n",
                            condstr(cond), regstr(rin), offset);
                }
                break;
            case NOT:
                rout = (inst & 0xf00)>>8;
                fprintf(out, "not\t %s\n", regstr(rout));
                break;
            case XOR:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "xor\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case OR:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "or\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case AND:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "and\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case LLEFT_SHIFT:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "lls\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case LRIGHT_SHIFT:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "lrs\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case ARIGHT_SHIFT:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "ars\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case LEFT_ROTATION:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "lrot\t %s, %s\n", regstr(rout), regstr(rin));
                break;
```

```
            case RIGHT_ROTATION:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "rrot\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case ADD:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "add\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case SUB:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "sub\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case MUL:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "mul\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case UDIV:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "udiv\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            case INC:
                rout = (inst & 0xf00)>>8;
                fprintf(out, "inc\t %s\n", regstr(rout));
                break;
            case DEC:
                rout = (inst & 0xf00)>>8;
                fprintf(out, "dec\t %s\n", regstr(rout));
                break;
            case PUSH_DWORD:
                rout = (inst & 0xf00)>>8;
                fprintf(out, "pushd\t %s\n", regstr(rout));
                break;
            case POP_DWORD:
                rout = (inst & 0xf00)>>8;
                fprintf(out, "popd\t %s\n", regstr(rout));
                break;
            case RET:
                fprintf(out, "ret\n");
                break;
            case NOP:
                fprintf(out, "nop\n");
                break;
            case STOP:
                fprintf(out, "stop\n");
                break;
            case VMCALL:
                fprintf(out, "vmcall\n");
                break;
            case XORB:
                rin = (inst & 0xf000)>>12;
                rout = (inst & 0xf00)>>8;
                fprintf(out, "xorb\t %s, %s\n", regstr(rout), regstr(rin));
                break;
            default:
                printf("Unknown opcode number %d\n", opcode);
                break;
        }
        i += opsize;
        nbinst++;
    }
    fclose(in);
    fclose(out);
}
```

bytecode_disassembler.c

revert_lfsr.c

```
#include <unistd.h>
#include <fcntl.h>
```

```c
#include <stdio.h>
#include <stdlib.h>


/* From "hexdump -e '16/1 "0x%02x, " "\n"' -n 8192 -s 32768 good.data.bin" */
unsigned char data[] = {
    0x00, 0xbc, 0x68, 0x15, 0xb5, 0x6b, 0x1b, 0x41, 0xa2, 0x19, 0xc4, 0x57, 0xe0, 0x01, 0xf6, 0xaf,
    0x4b, 0x35, 0x98, 0xb9, 0x38, 0x94, 0x3a, 0x6f, 0x8c, 0x86, 0x6a, 0xd7, 0x2a, 0x23, 0x4f, 0x6f,
    0xee, 0xa5, 0x93, 0x20, 0x4c, 0x55, 0xf0, 0xaa, 0xe5, 0xf3, 0x59, 0x38, 0xda, 0x18, 0x39, 0xbf,
    0x6a, 0xbb, 0x4e, 0x12, 0xa6, 0x80, 0x30, 0xa5, 0x0f, 0xc4, 0x7a, 0xb7, 0x2f, 0x02, 0x6c, 0x23,
    0x58, 0x2d, 0xe6, 0x63, 0x5c, 0xfb, 0x89, 0x89, 0x41, 0x14, 0xe2, 0xc2, 0x72, 0xe3, 0x9c, 0x92,
    0xc1, 0xda, 0x73, 0x24, 0x9f, 0x0d, 0xbb, 0x1c, 0x75, 0x9e, 0x5c, 0x2f, 0x9f, 0xde, 0xca, 0x97,
    0x61, 0xe9, 0xe0, 0x24, 0xa8, 0xa7, 0x65, 0xae, 0xe8, 0x02, 0xfe, 0x07, 0xa5, 0x2f, 0xac, 0xa2,
    0x4a, 0xad, 0x93, 0x6f, 0xe4, 0xd2, 0x52, 0x5e, 0x45, 0xac, 0x15, 0xa3, 0x56, 0x60, 0xd9, 0x43,
    0x8e, 0x06, 0xef, 0x6b, 0x19, 0xe8, 0x37, 0x34, 0x70, 0x0d, 0x7b, 0x86, 0x71, 0x7a, 0x09, 0x79,
    0x13, 0x7e, 0x97, 0xed, 0xb3, 0x64, 0x9a, 0x99, 0xea, 0xc5, 0xbf, 0x69, 0x30, 0x92, 0x3c, 0x6f,
    0x88, 0x14, 0x6f, 0xb6, 0xcf, 0xc9, 0x14, 0x2f, 0x4b, 0x24, 0x10, 0x4e, 0x82, 0x03, 0x4a, 0x4a,
    0x2d, 0x09, 0xb7, 0x54, 0xa8, 0xff, 0xb7, 0x80, 0x03, 0x61, 0x3b, 0x76, 0x20, 0x64, 0x11, 0xa7,
    0x3f, 0x18, 0x68, 0xb4, 0xdc, 0xfe, 0x11, 0x6f, 0x48, 0xbd, 0x59, 0x46, 0xa2, 0x46, 0xa6, 0x71,
    0x09, 0x4e, 0xac, 0x40, 0xb1, 0x59, 0xee, 0xc2, 0x46, 0x1f, 0x45, 0xbb, 0x01, 0x4d, 0x53, 0x92,
    0xda, 0x17, 0x4b, 0x78, 0x72, 0x37, 0x44, 0xc5, 0x5d, 0x34, 0xdd, 0x35, 0xe4, 0x9d, 0x07, 0xbe,
    0xdd, 0xd4, 0xfb, 0x23, 0xbc, 0x91, 0xf2, 0xa1, 0xcb, 0xb1, 0x45, 0xba, 0x31, 0xe2, 0xff, 0x5e,
    0x35, 0xc0, 0xb1, 0x54, 0xcc, 0xdc, 0x4d, 0xea, 0xa3, 0x81, 0xc7, 0x15, 0x2f, 0x93, 0x57, 0x7a,
    0xc6, 0x6c, 0xae, 0x1f, 0x6b, 0xd8, 0x89, 0xc3, 0xba, 0x09, 0x9c, 0x0a, 0x6b, 0x0f, 0x3b, 0x9c,
    0x1e, 0xd0, 0x7b, 0xca, 0x15, 0xae, 0x30, 0x2c, 0xfe, 0x23, 0x9b, 0xa8, 0xb7, 0x41, 0x19, 0x50,
    0x60, 0xae, 0x1e, 0x26, 0xaa, 0x5c, 0x5f, 0xf7, 0x6d, 0x11, 0x38, 0xd5, 0xf7, 0xa9, 0xf5, 0x02,
    0x15, 0xea, 0x0c, 0xb2, 0xd4, 0xdf, 0x27, 0x7e, 0xfc, 0x81, 0x08, 0xff, 0x3f, 0xba, 0xa0, 0x21,
    0xac, 0x97, 0xb1, 0x2f, 0x1b, 0x6e, 0xdb, 0xd6, 0xd6, 0x76, 0x88, 0x1a, 0xb5, 0x81, 0xf9, 0x39,
    0x45, 0x41, 0x2d, 0x6a, 0x56, 0x76, 0x3f, 0x29, 0x34, 0xf7, 0x29, 0x03, 0x8f, 0x82, 0xd0, 0x66,
    0x5f, 0x1e, 0x87, 0xff, 0x06, 0xc1, 0x9f, 0x60, 0x59, 0x14, 0xac, 0x07, 0xa5, 0xd3, 0x99, 0x5d,
    0xb1, 0x51, 0x7b, 0x39, 0x29, 0x77, 0xd8, 0x14, 0x65, 0x8f, 0xe9, 0x62, 0xc2, 0xf6, 0x86, 0x4e,
    0x79, 0x43, 0x7f, 0x5b, 0x44, 0xaa, 0x21, 0x31, 0xed, 0x79, 0x5d, 0x1a, 0x2e, 0x61, 0x01, 0x5a,
    0x3e, 0x2c, 0x18, 0xcc, 0x69, 0x6c, 0x78, 0xaa, 0xef, 0x84, 0xdb, 0xc6, 0x64, 0x4c, 0x95, 0x10,
    0xb2, 0xa1, 0x06, 0xcf, 0x27, 0xcd, 0xf3, 0x46, 0x9f, 0x51, 0x5b, 0x90, 0xc0, 0xbe, 0x19, 0xa8,
    0x2f, 0x5d, 0xd6, 0x57, 0x3c, 0x6b, 0x84, 0xc6, 0x02, 0x78, 0x95, 0x20, 0x9e, 0x9f, 0xca, 0x92,
    0x31, 0x0e, 0xd8, 0x9b, 0xc6, 0x8e, 0xfd, 0xd6, 0x58, 0xbb, 0x37, 0x28, 0xe9, 0x75, 0xc7, 0x84,
    0x52, 0x07, 0x0b, 0xbe, 0x6b, 0x1c, 0xee, 0x26, 0xf2, 0xba, 0x5c, 0x45, 0x31, 0xcd, 0x77, 0xca,
    0xf1, 0x94, 0x07, 0xab, 0x1e, 0x96, 0xad, 0xae, 0xb0, 0xd5, 0xb1, 0x33, 0x87, 0xa9, 0x31, 0x96,
    0xff, 0xbc, 0xcb, 0x02, 0xd9, 0x8f, 0x95, 0xae, 0x43, 0x19, 0x34, 0xfd, 0x3d, 0xbe, 0x28, 0xd2,
    0x2d, 0xe7, 0x7d, 0x96, 0x76, 0x7a, 0xaf, 0xf1, 0xd9, 0x36, 0xf7, 0xf0, 0xf1, 0xb9, 0x22, 0xad,
    0x56, 0x33, 0x6a, 0x09, 0xec, 0xde, 0xdf, 0xbc, 0xff, 0x41, 0x4a, 0xed, 0xbd, 0xb6, 0xe6, 0x4a,
    0x41, 0x39, 0xe5, 0x3b, 0x37, 0x8f, 0x11, 0xca, 0x67, 0xea, 0x59, 0x43, 0xee, 0xa8, 0xa3, 0x65,
    0x53, 0x2b, 0xbf, 0x7c, 0x69, 0x28, 0x35, 0x31, 0x1e, 0x99, 0x8f, 0xc4, 0x54, 0x5f, 0x0f, 0x17,
    0x4b, 0x72, 0x79, 0xf9, 0x6f, 0x4a, 0x55, 0xe7, 0xc7, 0x81, 0x76, 0x39, 0x6a, 0xb8, 0xf1, 0xe4,
    0xf2, 0x75, 0xbf, 0x53, 0x7b, 0x7a, 0x51, 0x5e, 0xec, 0xe6, 0x3c, 0x58, 0xc3, 0x90, 0x7f, 0x63,
    0x9f, 0x58, 0x3c, 0x2e, 0xda, 0x51, 0xae, 0x28, 0x41, 0xf0, 0xbb, 0xcc, 0xc1, 0x6f, 0x1f, 0x26,
    0xc5, 0xb4, 0xde, 0x69, 0x76, 0x27, 0x16, 0x31, 0x27, 0x9f, 0xcd, 0x3a, 0x7c, 0xc0, 0xac, 0xb8,
    0x4f, 0x48, 0x30, 0x4a, 0x7e, 0x1a, 0xc5, 0x39, 0x8f, 0x3c, 0x47, 0xac, 0x24, 0xc5, 0x64, 0xf7,
    0x83, 0xea, 0xcb, 0x00, 0x26, 0xad, 0x98, 0x06, 0x64, 0x04, 0x4d, 0x2a, 0xa1, 0xc2, 0x9c, 0x17,
    0x0c, 0x7f, 0x9e, 0xa2, 0xa0, 0xf4, 0xfa, 0xa8, 0xa1, 0xf0, 0x76, 0x42, 0x3e, 0x3c, 0xef, 0x16,
    0x7c, 0x70, 0x27, 0xc9, 0x59, 0xe9, 0xc5, 0x20, 0xf5, 0x1c, 0x3d, 0x72, 0xf8, 0xbd, 0x58, 0x4f,
    0xe3, 0xbc, 0x0d, 0x27, 0xc2, 0x63, 0xdf, 0x53, 0x52, 0xb6, 0x7b, 0x6c, 0x66, 0x57, 0x27, 0x78,
    0xfa, 0xc7, 0xf6, 0x76, 0x04, 0xf8, 0xbf, 0xb6, 0x60, 0x18, 0x8c, 0x7b, 0x62, 0x8b, 0x40, 0x4a,
    0x47, 0x59, 0x0f, 0x6d, 0x5d, 0x10, 0xd9, 0xb7, 0x5d, 0xd2, 0xa2, 0x42, 0x84, 0x9d, 0xbf, 0xb7,
    0x01, 0x42, 0x88, 0x6e, 0xa4, 0x77, 0xb3, 0x63, 0x5e, 0xcb, 0x50, 0x5f, 0x4b, 0x22, 0x75, 0x83,
    0x50, 0x88, 0x99, 0xbb, 0xeb, 0x48, 0xdf, 0xf3, 0xa7, 0xd6, 0x67, 0x69, 0xd9, 0xc4, 0xac, 0x9f,
    0x7b, 0x36, 0x5a, 0x81, 0xc3, 0x1b, 0x01, 0x54, 0xe3, 0x65, 0x2c, 0x6f, 0x7b, 0x90, 0xb7, 0x7e,
    0x88, 0x30, 0x60, 0x3e, 0x9a, 0xb5, 0x37, 0x4d, 0xd9, 0x62, 0x09, 0x54, 0xc0, 0xa5, 0x14, 0xa0,
    0xfe, 0xf1, 0x99, 0xe3, 0xba, 0x86, 0xde, 0xda, 0x4f, 0x42, 0x2a, 0xbb, 0x5e, 0x6c, 0xad, 0x1a,
    0x1f, 0x65, 0x00, 0x90, 0x15, 0xe3, 0x52, 0x29, 0x73, 0x5c, 0xdf, 0x54, 0x4f, 0x0e, 0xdc, 0xfe,
    0x26, 0x23, 0xbb, 0xb4, 0x79, 0x2d, 0x42, 0x82, 0x05, 0x71, 0x2d, 0xbc, 0x72, 0x74, 0x88, 0x17,
    0x7b, 0xa2, 0x6a, 0x9b, 0x27, 0xd8, 0xdf, 0x47, 0x11, 0xe4, 0x77, 0x3a, 0x2a, 0x6e, 0xb8, 0x46,
    0xe8, 0xc4, 0xb9, 0xf4, 0x38, 0x1c, 0x08, 0xa9, 0xe1, 0x08, 0xf4, 0xe9, 0x4d, 0xdf, 0x3d, 0xe0,
    0x37, 0x10, 0xdd, 0xaa, 0x6a, 0x3a, 0x24, 0x4e, 0xb6, 0xf1, 0x0f, 0xb5, 0xb4, 0x69, 0x58, 0x81,
    0x49, 0x99, 0xfa, 0x65, 0x3f, 0x0e, 0x0f, 0x64, 0xa6, 0x7a, 0x3b, 0xdf, 0x7b, 0x82, 0xcf, 0xb5,
    0xec, 0xaf, 0x8a, 0x27, 0xac, 0x24, 0xf0, 0x4c, 0x80, 0xf5, 0xb0, 0x12, 0x99, 0xcd, 0xe2, 0xfb,
    0x80, 0x34, 0x1b, 0xf4, 0x05, 0x42, 0xc3, 0x1a, 0x4a, 0x22, 0x26, 0x2a, 0x61, 0xc0, 0x2d, 0xfc,
    0xfe, 0xb6, 0x58, 0xea, 0xcf, 0x9a, 0x44, 0x98, 0x57, 0x33, 0x1a, 0xac, 0xac, 0x3d, 0x91, 0x92,
    0x74, 0xe4, 0xb3, 0xa0, 0x03, 0xe6, 0x4b, 0x04, 0xe2, 0x20, 0xab, 0x4e, 0xb4, 0x6e, 0x06, 0x27,
    0x7f, 0x10, 0xef, 0xeb, 0x29, 0x81, 0xf8, 0xfb, 0x61, 0x7a, 0xd8, 0xfb, 0x2c, 0x28, 0x36, 0xf8,
    0xd5, 0x48, 0x6a, 0x59, 0xb0, 0x14, 0xc7, 0x51, 0x49, 0x56, 0xd1, 0x47, 0xc9, 0x8c, 0xdc, 0x92,
    0x12, 0x13, 0x3c, 0x5c, 0xd3, 0x70, 0x91, 0x82, 0x77, 0x51, 0xc8, 0x24, 0x96, 0x03, 0x73, 0x04,
    0x79, 0x99, 0x31, 0xcc, 0x7c, 0x10, 0x47, 0x18, 0xf1, 0x54, 0x82, 0xbd, 0x49, 0xfc, 0xb9, 0xde,
    0xe6, 0x63, 0xd6, 0xe3, 0xc2, 0x9a, 0x03, 0xdb, 0x9e, 0x14, 0x9b, 0xaf, 0xa8, 0x58, 0x68, 0x2a,
    0xa6, 0x51, 0x99, 0xa3, 0xb4, 0xf7, 0x98, 0x26, 0xb3, 0x2f, 0x9e, 0x9c, 0xdd, 0x7b, 0xb9, 0x19,
    0x6d, 0xde, 0xf6, 0x7f, 0x52, 0xd7, 0x0f, 0xac, 0xc5, 0x44, 0x41, 0x12, 0xd6, 0x33, 0xd0, 0x86,
```

```
    0x53, 0x1e, 0x2d, 0x8e, 0x84, 0x89, 0x15, 0x32, 0x2c, 0xcf, 0xa8, 0x57, 0xb7, 0x31, 0xe5, 0xd4,
    0x0b, 0x41, 0xea, 0xe6, 0xc1, 0x5a, 0xe5, 0x32, 0xdf, 0x95, 0x69, 0x5b, 0x2d, 0x98, 0xbb, 0x7f,
    0xea, 0xc1, 0x8e, 0xa6, 0xa5, 0xaf, 0x79, 0xbe, 0x1f, 0x42, 0x9c, 0xf4, 0xdb, 0x98, 0x8f, 0x29,
    0x95, 0x75, 0xcd, 0xa4, 0x37, 0x3a, 0x3c, 0xd8, 0x21, 0xf6, 0x8c, 0xad, 0x5f, 0x66, 0xaf, 0x5b,
    0x91, 0x7f, 0x50, 0xb3, 0x4e, 0x88, 0xc6, 0x58, 0x40, 0x13, 0x88, 0x22, 0xd1, 0x52, 0x59, 0xb1,
    0x33, 0x71, 0x19, 0x44, 0x0d, 0x0a, 0x8e, 0x95, 0xce, 0x34, 0x2c, 0x88, 0x6b, 0xb0, 0x6a, 0x39,
    0xa0, 0xd9, 0xff, 0xd9, 0x11, 0x1a, 0x76, 0x19, 0x2f, 0x32, 0x62, 0x43, 0x9a, 0x7f, 0x20, 0xe2,
    0x44, 0xc3, 0x71, 0xfc, 0x4e, 0x55, 0x8c, 0xe4, 0x82, 0xc5, 0x36, 0x7e, 0xcc, 0xa5, 0x36, 0x39,
    0xd8, 0xf4, 0x61, 0x33, 0x2f, 0x3b, 0x7b, 0x46, 0xc9, 0xe7, 0xfc, 0x33, 0x47, 0x3a, 0xab, 0x7a,
    0x97, 0xa9, 0x48, 0xd8, 0xe7, 0xa3, 0x52, 0xd4, 0x2a, 0xd4, 0x20, 0xd8, 0xc1, 0xb4, 0x5f, 0xbe,
    0x4e, 0x3c, 0xc0, 0x13, 0x60, 0x2d, 0xae, 0xf1, 0x4c, 0xac, 0xf3, 0x31, 0xc8, 0x17, 0x79, 0xb0,
    0x72, 0xc2, 0x42, 0xe1, 0x9f, 0xba, 0xc8, 0x7a, 0x30, 0xd7, 0xf4, 0xf2, 0xcb, 0x69, 0x77, 0x4a,
    0xfa, 0xc4, 0xc2, 0x70, 0xd8, 0xdf, 0x23, 0x5a, 0x2c, 0x47, 0x6d, 0x27, 0x25, 0xe8, 0x31, 0x1a,
    0xe4, 0x92, 0x3e, 0x05, 0x26, 0x67, 0x9a, 0x95, 0x7b, 0xd9, 0x57, 0x1e, 0x1c, 0x54, 0xd2, 0x6a,
    0x92, 0xdc, 0x3b, 0x6a, 0xc0, 0x38, 0x27, 0x0f, 0x50, 0xdd, 0xc7, 0xda, 0x32, 0x24, 0xba, 0xb4,
    0x65, 0xe1, 0x28, 0x21, 0x62, 0x1a, 0x3e, 0x95, 0x67, 0xbe, 0xee, 0xb2, 0xd9, 0x08, 0x89, 0xfc,
    0x39, 0x85, 0xbe, 0xa4, 0x21, 0x7f, 0x65, 0xf0, 0x3e, 0xd3, 0x96, 0x47, 0x90, 0x46, 0x77, 0x2f,
    0xd6, 0x57, 0xe7, 0x61, 0xff, 0x98, 0xb3, 0x50, 0xbd, 0x19, 0x8b, 0x24, 0x2f, 0x63, 0x9e, 0x0c,
    0x26, 0x42, 0x83, 0xdd, 0x9b, 0xf9, 0x39, 0xc2, 0x0e, 0x60, 0xc5, 0x92, 0x6f, 0x5b, 0x0b, 0xc5,
    0xfa, 0xa2, 0xdd, 0xdb, 0x04, 0x1b, 0x0e, 0x48, 0x46, 0x02, 0x40, 0x5f, 0xa3, 0x96, 0xab, 0xf3,
    0x72, 0xee, 0x1b, 0x40, 0xb5, 0x6c, 0x55, 0xd8, 0x2f, 0xcd, 0xcb, 0x61, 0x0f, 0x5f, 0x9b, 0xa3,
    0xd4, 0x8b, 0xb2, 0xad, 0x08, 0xa0, 0xcf, 0x5b, 0x53, 0x17, 0x7f, 0x7f, 0x11, 0x5c, 0x85, 0x9d,
    0xc1, 0x3d, 0xdc, 0x29, 0x39, 0x40, 0xe9, 0x14, 0x13, 0xa2, 0x6a, 0x8e, 0x79, 0xe1, 0x2a, 0xb1,
    0xb4, 0x9f, 0x9c, 0x00, 0x10, 0x04, 0x95, 0xc3, 0x89, 0xb6, 0xc8, 0xd1, 0x01, 0xc5, 0x9d, 0x8e,
    0xe8, 0xa6, 0x80, 0x1e, 0x12, 0x35, 0xcd, 0x16, 0x7c, 0xae, 0x54, 0x59, 0x60, 0x2a, 0x4a, 0x53,
    0x5b, 0x18, 0xdb, 0x44, 0xb3, 0xef, 0xfc, 0x1c, 0x10, 0x12, 0xa4, 0xd0, 0xc1, 0x2b, 0x71, 0x5c,
    0x6d, 0xa9, 0xb4, 0xe6, 0x1e, 0x0c, 0xfc, 0x15, 0x94, 0x46, 0x61, 0x6f, 0xe9, 0x2d, 0x09, 0x3e,
    0x23, 0xfb, 0x25, 0x17, 0xb9, 0x14, 0x9d, 0x98, 0xd9, 0x9e, 0x06, 0xc2, 0x81, 0xdc, 0x86, 0x2a,
    0x2b, 0x98, 0xe7, 0x8d, 0x54, 0x15, 0xb2, 0x7c, 0x45, 0xa4, 0x3d, 0x5d, 0xdc, 0x6b, 0xa9, 0x78,
    0x94, 0x4e, 0x14, 0xf7, 0x9b, 0x28, 0x37, 0x47, 0xe4, 0xe9, 0x3f, 0x4c, 0x82, 0x7f, 0xde, 0x4c,
    0xdc, 0xd4, 0xbd, 0xce, 0xd8, 0xbd, 0x32, 0x31, 0x40, 0x89, 0xeb, 0xf5, 0xd5, 0x9a, 0x8d, 0x93,
    0xff, 0x13, 0x28, 0x53, 0x06, 0x29, 0xcf, 0x02, 0xf4, 0x8c, 0x47, 0x73, 0x8a, 0x9d, 0x2f, 0xfc,
    0xc0, 0xea, 0xde, 0x6f, 0x11, 0x30, 0x33, 0x31, 0x55, 0xf2, 0xcd, 0xf4, 0x8f, 0xdb, 0x4b, 0x48,
    0x88, 0x56, 0x41, 0x76, 0x33, 0x0c, 0x0c, 0x0e, 0xd5, 0x3c, 0x01, 0x96, 0xfd, 0xbf, 0xf6, 0xd1,
    0x4f, 0xd2, 0x48, 0x18, 0x5d, 0x7a, 0x18, 0x6c, 0x0b, 0xe0, 0x7f, 0xed, 0xd3, 0xd6, 0xc9, 0x64,
    0x9e, 0x92, 0x17, 0xbe, 0xb4, 0x18, 0x7e, 0x76, 0x2e, 0xc4, 0x8b, 0xa0, 0xc0, 0x12, 0x16, 0x8a,
    0x98, 0x76, 0x34, 0x49, 0xb6, 0xc4, 0x8a, 0x44, 0x63, 0xdf, 0xdc, 0x0f, 0x3c, 0xe4, 0x7c, 0x07,
    0x57, 0x99, 0xa3, 0x85, 0x78, 0x4e, 0x64, 0x97, 0x74, 0x62, 0x90, 0xe5, 0x0e, 0x43, 0xc0, 0xfa,
    0xf8, 0xa9, 0xdb, 0xc1, 0xcd, 0xc5, 0xc7, 0xa7, 0xa4, 0x46, 0x59, 0xb7, 0x86, 0x4f, 0x68, 0x3d,
    0x6a, 0xdf, 0x9d, 0x74, 0x75, 0x2f, 0xb8, 0xcf, 0xbb, 0xe8, 0x14, 0xc0, 0x53, 0xbd, 0x7f, 0x66,
    0x10, 0xa4, 0x89, 0xb6, 0xb4, 0x5d, 0xe8, 0xae, 0x24, 0x4e, 0xd4, 0x51, 0x07, 0x79, 0x80, 0xd1,
    0xf8, 0x98, 0x6a, 0xc6, 0xde, 0x62, 0xaa, 0x25, 0xed, 0x47, 0x58, 0x75, 0xd1, 0x8d, 0x60, 0x3e,
    0x9a, 0xde, 0x5b, 0x3e, 0xb7, 0xf7, 0x55, 0x7b, 0x9f, 0x7b, 0x0c, 0x5f, 0x3d, 0x05, 0x3e, 0x61,
    0x33, 0xd7, 0x15, 0x33, 0xa2, 0x4f, 0xd1, 0x8f, 0xd9, 0xaf, 0xe5, 0x73, 0x56, 0xa1, 0x48, 0x0b,
    0xec, 0xa1, 0x6b, 0x3a, 0xc3, 0x54, 0xed, 0xba, 0x09, 0xfe, 0x7d, 0x9f, 0x6f, 0x95, 0xf7, 0xd6,
    0xd4, 0x2e, 0x65, 0xa1, 0x85, 0xac, 0xc2, 0xc3, 0x07, 0x2e, 0x77, 0xc6, 0x3e, 0xea, 0x3b, 0xfd,
    0xb3, 0xbc, 0x21, 0x51, 0xe9, 0x83, 0xa6, 0x65, 0x90, 0xce, 0x01, 0xab, 0xc8, 0x02, 0x96, 0x94,
    0x6d, 0x2e, 0x39, 0xf3, 0x12, 0x4a, 0x51, 0x3f, 0xbe, 0x0a, 0x81, 0x73, 0x91, 0xbb, 0x01, 0xe8,
    0x2e, 0x35, 0xb7, 0x4b, 0x93, 0x45, 0x54, 0x36, 0xe9, 0x06, 0x19, 0x59, 0xd9, 0x53, 0xe4, 0xb2,
    0xec, 0x75, 0x46, 0xb9, 0xa5, 0x3a, 0x07, 0xd8, 0x40, 0x53, 0xfb, 0xa5, 0x88, 0xca, 0xaf, 0x7f,
    0x8e, 0x57, 0xa6, 0x06, 0x36, 0x44, 0x42, 0x17, 0xf5, 0x3d, 0x69, 0x20, 0x20, 0x76, 0xa9, 0x3d,
    0x06, 0xf7, 0x5a, 0x92, 0x71, 0x96, 0x0f, 0xd3, 0x8a, 0x19, 0x44, 0x6e, 0x6c, 0x8a, 0xa1, 0x4b,
    0x83, 0x9d, 0xc0, 0x43, 0xce, 0xd6, 0x01, 0xba, 0x4b, 0x9a, 0x38, 0x90, 0x98, 0x69, 0x25, 0x9f,
    0x82, 0xbb, 0x47, 0xe3, 0x7f, 0xb9, 0x1a, 0x5e, 0xd8, 0x28, 0x97, 0xb4, 0xcc, 0xb9, 0x35, 0x32,
    0xc9, 0x12, 0xb3, 0x4e, 0xa3, 0x46, 0xf9, 0x03, 0x13, 0x91, 0x02, 0x32, 0x90, 0x75, 0xec, 0x73,
    0x90, 0xfe, 0x3b, 0x4a, 0x55, 0x06, 0x78, 0xb4, 0x6d, 0x0a, 0x82, 0x35, 0x3e, 0x3c, 0x2a, 0x6a,
    0xeb, 0x60, 0xe0, 0x22, 0x67, 0x31, 0xf2, 0x0d, 0x10, 0x49, 0x8e, 0x03, 0xc2, 0x6c, 0xc4, 0x9a,
    0xb6, 0xa5, 0xbc, 0x73, 0x1f, 0xbf, 0x6a, 0x58, 0x20, 0xdd, 0x78, 0xb4, 0x85, 0x99, 0x60, 0x63,
    0xb6, 0x5d, 0xed, 0x76, 0x3e, 0xc8, 0xaa, 0xe1, 0x04, 0xf7, 0xbe, 0x69, 0x0a, 0x36, 0x0d, 0x1d,
    0xb1, 0x74, 0x5f, 0xb9, 0x29, 0x04, 0x9a, 0xbe, 0xfe, 0x6a, 0xcc, 0xb9, 0x13, 0x89, 0xd6, 0x16,
    0x67, 0x58, 0x78, 0x2b, 0x73, 0x62, 0xc3, 0x96, 0x97, 0x7f, 0xa4, 0xa6, 0x8c, 0x3b, 0xfd, 0x4d,
    0x4c, 0x2f, 0x23, 0x51, 0x2d, 0x9e, 0x86, 0xa2, 0x31, 0xf4, 0xa8, 0x8f, 0xcf, 0x0a, 0x4d, 0x6e,
    0x1d, 0x07, 0xc9, 0xcc, 0xe8, 0xee, 0x41, 0x84, 0x86, 0xde, 0xa5, 0xce, 0x9c, 0xd1, 0xb7, 0xf8,
    0x04, 0x16, 0x06, 0x43, 0x78, 0x6c, 0xb3, 0x36, 0xdc, 0x69, 0x20, 0x73, 0xd5, 0x15, 0x90, 0x20,
    0x87, 0x5a, 0x92, 0x1e, 0xb0, 0x1a, 0x55, 0x6d, 0x50, 0x60, 0xfc, 0x67, 0xdb, 0x7c, 0x1a, 0xeb,
    0x00, 0x49, 0x9b, 0x21, 0x45, 0x09, 0x35, 0xf3, 0x8e, 0x46, 0x28, 0xda, 0xdd, 0xb9, 0x06, 0x6f,
    0x16, 0xae, 0x0e, 0xbb, 0xeb, 0xa5, 0x51, 0x85, 0x4d, 0x67, 0x16, 0x10, 0xa6, 0x06, 0xb7, 0xf9,
    0x5d, 0xfa, 0xb2, 0x55, 0x8a, 0xae, 0xfa, 0x13, 0xeb, 0xd6, 0x1f, 0x94, 0x7c, 0xa0, 0xbb, 0x73,
    0x28, 0x69, 0x33, 0x76, 0x87, 0xc7, 0xd7, 0x4b, 0x5b, 0x2b, 0x4b, 0x3c, 0x97, 0x8b, 0xe6, 0xa6,
    0xba, 0x9e, 0x45, 0x78, 0x19, 0xcb, 0xc3, 0x51, 0x0d, 0x43, 0xf9, 0x7f, 0xec, 0xef, 0x6f, 0x93,
    0x86, 0xa8, 0x5a, 0xf4, 0x31, 0x66, 0x99, 0xd9, 0xc3, 0x5b, 0x0d, 0x07, 0x39, 0x69, 0xd4, 0x13,
    0x1e, 0x59, 0x2c, 0x74, 0xb9, 0x62, 0xc0, 0x1e, 0x2e, 0x5a, 0x9b, 0x3f, 0x37, 0x56, 0x49, 0x32,
    0x7b, 0x0d, 0x45, 0x7a, 0x8b, 0x04, 0xec, 0x56, 0x82, 0x41, 0xb0, 0x58, 0x0c, 0x76, 0x78, 0x8a,
    0x4a, 0xda, 0x6d, 0xd5, 0x5c, 0x20, 0x5b, 0x83, 0xf2, 0xca, 0xeb, 0xe5, 0x31, 0xff, 0x9e, 0xd9,
    0x91, 0x58, 0x45, 0xf9, 0x01, 0x7a, 0x5f, 0x7e, 0xc6, 0xb8, 0x94, 0x2b, 0x6c, 0x29, 0x42, 0x16,
    0x4d, 0x7f, 0x07, 0x28, 0xa3, 0xb9, 0x56, 0xc3, 0xf7, 0x42, 0x4c, 0x47, 0xc5, 0x9d, 0xdf, 0x6f,
```

```
    0x4c, 0x03, 0xf6, 0xaf, 0x6b, 0x9a, 0x2f, 0xbd, 0xce, 0x3a, 0x18, 0x33, 0xcb, 0x9f, 0xcc, 0xbe,
    0xa0, 0x29, 0x36, 0xfd, 0x28, 0x17, 0x87, 0xd1, 0x1c, 0x46, 0xfb, 0xd3, 0xb8, 0xfa, 0x4c, 0xfe,
    0x2d, 0xc3, 0x34, 0x1e, 0xf1, 0x7c, 0x09, 0x98, 0xd3, 0x1e, 0x1b, 0x7b, 0xda, 0xf6, 0xd4, 0x63,
    0x02, 0x5f, 0xef, 0xa6, 0x58, 0x51, 0x4e, 0x73, 0x8d, 0xd0, 0xa1, 0x18, 0x80, 0x6c, 0x0a, 0xfd,
    0x5d, 0x38, 0x39, 0x36, 0x38, 0xa3, 0x83, 0x30, 0x65, 0x7f, 0xd4, 0xb2, 0x63, 0x57, 0xfd, 0x92,
    0x06, 0x85, 0xdc, 0xb5, 0x1e, 0x50, 0xbe, 0xc4, 0x96, 0x3e, 0xbf, 0x3e, 0x2e, 0x55, 0x98, 0x76,
    0x20, 0x2e, 0xf4, 0xbc, 0x43, 0xe5, 0xa4, 0xc3, 0xb6, 0xe9, 0x95, 0x9b, 0x1e, 0x94, 0x4e, 0xa8,
    0x20, 0xa5, 0xac, 0x82, 0x5f, 0x07, 0x67, 0xc9, 0x1c, 0xac, 0xa3, 0x6e, 0x5e, 0x3d, 0xfa, 0x37,
    0x09, 0xf6, 0x02, 0x0a, 0xcd, 0x9a, 0x11, 0xe6, 0xd4, 0x20, 0x03, 0x83, 0x14, 0x60, 0x1d, 0x18,
    0x6a, 0x92, 0x4b, 0x8c, 0x6a, 0xe3, 0x9a, 0xb8, 0xa7, 0xe0, 0x7d, 0x5c, 0x35, 0xfd, 0x44, 0x5b,
    0xb3, 0x24, 0xcf, 0x9b, 0x4f, 0x41, 0xb1, 0xa6, 0x1e, 0x07, 0x85, 0xa6, 0xa1, 0xb7, 0xda, 0x75,
    0x43, 0x8b, 0x88, 0x3c, 0xab, 0xaf, 0x7c, 0x22, 0x3a, 0x45, 0xb8, 0xce, 0xef, 0x2f, 0xa3, 0xb5,
    0x7c, 0x06, 0x12, 0xd1, 0x66, 0xe8, 0x3a, 0x6b, 0x31, 0xc3, 0x91, 0x48, 0xae, 0x9c, 0x58, 0xa6,
    0xfe, 0xa8, 0x1d, 0xc9, 0xf5, 0xa3, 0xed, 0x69, 0x98, 0x36, 0xcf, 0x62, 0xf9, 0xfd, 0x14, 0xd4,
    0x63, 0xae, 0xa1, 0x8d, 0x0f, 0x41, 0xdc, 0xf8, 0xa8, 0x32, 0x93, 0x65, 0x5e, 0x48, 0x64, 0x2a,
    0x7f, 0xd8, 0x1e, 0x77, 0x7b, 0x12, 0x07, 0x29, 0x7a, 0x2a, 0xbc, 0x21, 0x9e, 0x27, 0x19, 0x13,
    0xd6, 0xee, 0xf6, 0xe2, 0xbc, 0x4c, 0x62, 0x57, 0x96, 0x7b, 0xdf, 0x56, 0x15, 0x2d, 0xfc, 0x19,
    0xdf, 0xa6, 0x5e, 0x51, 0xdd, 0x86, 0x64, 0x81, 0x42, 0x96, 0xbc, 0x1d, 0x30, 0xe7, 0x52, 0xab,
    0x1f, 0x04, 0x5c, 0x65, 0x71, 0x68, 0x8d, 0x61, 0xd0, 0x07, 0x78, 0xac, 0x25, 0x12, 0xa2, 0x01,
    0x49, 0x21, 0x9c, 0xa3, 0xb0, 0x1f, 0x27, 0x1d, 0xc8, 0xda, 0x5c, 0xa8, 0x24, 0x85, 0x68, 0xf7,
    0x80, 0x83, 0xea, 0x7f, 0xc0, 0xe5, 0x12, 0xfa, 0x49, 0xc5, 0xf2, 0x17, 0x8e, 0x94, 0x8d, 0x0d,
    0xc8, 0x77, 0xa9, 0x3d, 0x5f, 0x07, 0xf7, 0x14, 0x0e, 0x68, 0x37, 0x30, 0x66, 0xde, 0x8b, 0x76,
    0xd1, 0x80, 0x21, 0xe3, 0xc3, 0x0a, 0x45, 0x04, 0x1d, 0x55, 0x6c, 0xd9, 0xb4, 0x9f, 0x88, 0xe4,
    0x86, 0xb0, 0x40, 0x81, 0x07, 0xe8, 0x0e, 0x95, 0xb2, 0x8e, 0x49, 0xc6, 0xde, 0x9c, 0x7b, 0x94,
    0x56, 0x32, 0x37, 0x8a, 0x2e, 0xce, 0x61, 0x3f, 0x96, 0xfc, 0x50, 0xee, 0x0a, 0x32, 0x01, 0xe8,
    0x18, 0x5c, 0x1c, 0xd1, 0xca, 0x8d, 0xb7, 0xb8, 0xf1, 0xa3, 0x9b, 0xe2, 0xd6, 0x30, 0xc2, 0x80,
    0xb7, 0xc5, 0xa6, 0x72, 0xc3, 0xaa, 0x3b, 0x8e, 0xfa, 0x3e, 0xe7, 0x4a, 0x3a, 0x7c, 0x59, 0xcd,
    0x0d, 0x98, 0x4c, 0x0d, 0x9f, 0xa3, 0xec, 0xeb, 0x65, 0xa4, 0xea, 0xa2, 0xbd, 0x6f, 0x5c, 0xd7,
    0x53, 0x52, 0xee, 0xe0, 0xcf, 0xbd, 0xd2, 0xc2, 0x26, 0xb5, 0xf5, 0xff, 0x66, 0x14, 0x1f, 0x6e,
    0x04, 0x53, 0x3e, 0x85, 0x18, 0xf8, 0xf4, 0xd1, 0xc0, 0x6f, 0xd1, 0x22, 0x5b, 0xd5, 0xe4, 0x1d,
    0x24, 0xd0, 0x8f, 0xc4, 0xf3, 0x06, 0x76, 0xcf, 0xc0, 0x8e, 0xd0, 0x95, 0xe1, 0xc3, 0xdf, 0x66,
    0x49, 0xcd, 0x38, 0x1a, 0x01, 0x57, 0x99, 0x69, 0x5a, 0x30, 0x2a, 0xbb, 0x6c, 0x19, 0xd4, 0xd4,
    0xf2, 0x6d, 0x83, 0x45, 0x15, 0x46, 0x52, 0xc0, 0xb5, 0x14, 0x73, 0xf9, 0x3e, 0x4d, 0xd8, 0x71,
    0x06, 0xc0, 0x57, 0xa5, 0x7b, 0x14, 0x12, 0x1d, 0xb2, 0xdb, 0x05, 0x88, 0xba, 0xb1, 0xd8, 0xf7,
    0xd8, 0x61, 0xc1, 0x2a, 0x60, 0xc6, 0x5b, 0xde, 0xb8, 0xab, 0xfe, 0x0d, 0xff, 0x69, 0x45, 0xd1,
    0x0e, 0xef, 0x43, 0x86, 0x66, 0x9d, 0xc1, 0x48, 0xe9, 0x85, 0x26, 0x3c, 0x3c, 0x86, 0x48, 0x9c,
    0xd4, 0x77, 0x20, 0x2d, 0x9b, 0x8a, 0xd5, 0xa3, 0x07, 0x4c, 0x49, 0x14, 0x61, 0x29, 0xac, 0xa8,
    0xe6, 0xa3, 0xf0, 0x1b, 0x25, 0x62, 0x9b, 0xb8, 0x4d, 0x6f, 0x49, 0x34, 0xac, 0x3b, 0x9e, 0xf1,
    0xbe, 0x7a, 0xd4, 0xb1, 0x15, 0x7d, 0x43, 0x39, 0xe9, 0x7c, 0xf8, 0x25, 0x4f, 0xa2, 0x3a, 0x81,
    0x62, 0x15, 0xed, 0x88, 0x99, 0xfc, 0x58, 0x01, 0x8d, 0xac, 0xd3, 0x63, 0x7e, 0x87, 0x7f, 0xfe,
    0xd3, 0x5c, 0xfd, 0xfd, 0xe9, 0xc2, 0x17, 0xa0, 0x8c, 0x15, 0xeb, 0xd3, 0x2b, 0xfc, 0x68, 0x38,
    0x09, 0x3e, 0x9e, 0xb4, 0xa6, 0x64, 0xd5, 0x7f, 0xf0, 0x2e, 0xcd, 0x76, 0x04, 0xdc, 0xf9, 0x7a,
    0x1c, 0x43, 0x14, 0xc3, 0x89, 0xa3, 0x37, 0x44, 0x87, 0x6f, 0xe4, 0x02, 0xa5, 0x8c, 0x50, 0x76,
    0x3d, 0xf4, 0x3f, 0xfc, 0xac, 0x9b, 0x00, 0x51, 0xeb, 0xdc, 0x5e, 0x87, 0xce, 0xcb, 0x6d, 0xda,
    0x0c, 0xf6, 0xbc, 0x97, 0x84, 0x0c, 0x30, 0x83, 0x87, 0xae, 0xf6, 0x21, 0x23, 0x87, 0x38, 0x02,
    0xaf, 0x2e, 0x8a, 0x93, 0xb4, 0x74, 0xb8, 0xe0, 0xd0, 0x32, 0xa7, 0xe1, 0x07, 0x4e, 0xf1, 0x6d,
    0x38, 0xc4, 0x4c, 0xda, 0xaf, 0x84, 0x25, 0x14, 0xb8, 0x76, 0xa3, 0x0d, 0x66, 0x3f, 0xc1, 0xdc,
    0xb8, 0xb2, 0x90, 0x08, 0xae, 0x17, 0x8f, 0x78, 0x2a, 0x6e, 0xc7, 0x12, 0x98, 0xfa, 0x42, 0x12,
    0x7c, 0x32, 0xde, 0x27, 0xed, 0x0d, 0xc4, 0x8d, 0xea, 0x8d, 0x32, 0x74, 0x30, 0x08, 0x76, 0x30,
    0xa7, 0xf7, 0x3b, 0x3f, 0xdb, 0x63, 0xdf, 0xdb, 0xb3, 0x3d, 0x9e, 0x17, 0x9e, 0x73, 0x0b, 0xef,
    0x3a, 0x86, 0x2e, 0x21, 0x32, 0x1e, 0x34, 0x33, 0xa7, 0xfb, 0x5f, 0xc6, 0xfc, 0x67, 0x3f, 0xd9,
    0xfa, 0x10, 0x66, 0x4d, 0x09, 0x68, 0xcc, 0xf0, 0xbb, 0x71, 0x8a, 0xd3, 0x81, 0x80, 0x9b, 0xdb,
    0xa6, 0x8f, 0x11, 0x4b, 0x8f, 0x1c, 0x82, 0xcb, 0xb2, 0xa1, 0xda, 0xd7, 0xf4, 0x87, 0xfc, 0xef,
    0x3b, 0xb7, 0x9f, 0x74, 0x23, 0x68, 0x5c, 0xd0, 0x28, 0x21, 0x33, 0xdc, 0x4b, 0x2a, 0xce, 0xb6,
    0xed, 0x8f, 0xd9, 0xa3, 0xf3, 0xbb, 0xf5, 0x96, 0x79, 0xcc, 0xf0, 0xd9, 0x90, 0xcb, 0xdd, 0x4d,
    0xec, 0xea, 0x1c, 0xf0, 0xff, 0x61, 0x41, 0xbe, 0x9b, 0xca, 0xbf, 0x49, 0x99, 0x6c, 0x01, 0x0a,
    0x59, 0xbb, 0xa1, 0xb9, 0xd4, 0xd2, 0x48, 0xee, 0x5a, 0x61, 0x1d, 0x7e, 0xb1, 0x4a, 0xd5, 0xf5,
    0x35, 0x1d, 0xa2, 0x16, 0x1d, 0xca, 0x22, 0xf9, 0x06, 0xcf, 0x27, 0x04, 0x86, 0x44, 0xa9, 0x9d,
    0xb2, 0xd0, 0x3d, 0xb1, 0x20, 0x76, 0x0f, 0x08, 0x27, 0x92, 0x65, 0x93, 0xb6, 0x8a, 0xa1, 0xc9,
    0x19, 0xd8, 0xac, 0x8c, 0xb2, 0xa7, 0xc6, 0x46, 0xc8, 0x63, 0x5c, 0x9b, 0xa9, 0xfa, 0x3c, 0x04,
    0x0e, 0x73, 0xea, 0x59, 0xf0, 0xbb, 0x40, 0x07, 0x16, 0x8c, 0xd6, 0xb9, 0x92, 0x82, 0x38, 0xe6,
    0xc3, 0x64, 0x18, 0x2b, 0x91, 0x27, 0x36, 0x75, 0xc5, 0x1b, 0x7f, 0xcb, 0x93, 0xb3, 0xae, 0x6b,
    0xf9, 0x34, 0xcc, 0xef, 0x02, 0x8c, 0xa0, 0x45, 0xec, 0x52, 0xd2, 0xe8, 0xe0, 0xea, 0x71, 0xb0,
    0xa3, 0xe0, 0x8d, 0x12, 0xe3, 0x29, 0x70, 0xc7, 0xc5, 0xff, 0x14, 0x8d, 0x1e, 0x0f, 0xaa, 0x3d,
    0x22, 0xf4, 0x89, 0xcf, 0xe9, 0x2f, 0x29, 0x08, 0xd8, 0x52, 0xa5, 0xcc, 0xd4, 0xa1, 0xf0, 0x0e,
    0xb8, 0x8d, 0x6b, 0xf6, 0x52, 0x93, 0x38, 0xe9, 0xed, 0x5d, 0xf3, 0x3c, 0x1f, 0x02, 0x63, 0x2b,
    0xcf, 0x9a, 0x1e, 0x15, 0x42, 0x4a, 0xe1, 0xf3, 0x17, 0xee, 0x2e, 0x22, 0x3b, 0x11, 0x6c, 0xc5,
    0x4c, 0xd1, 0xf5, 0x6e, 0x10, 0x1d, 0xf6, 0x4f, 0x87, 0x93, 0x3e, 0x7b, 0x71, 0xd3, 0x3c, 0x0b,
    0x8b, 0x90, 0x2e, 0x61, 0x93, 0x02, 0x64, 0x9e, 0xd7, 0xe3, 0xbc, 0x39, 0xd9, 0x27, 0x52, 0xbc,
    0xfa, 0x02, 0x87, 0x37, 0x9d, 0x8b, 0x03, 0x40, 0xbb, 0x6e, 0xde, 0x19, 0xd3, 0x61, 0xc5, 0x24,
    0x59, 0x67, 0x98, 0xf0, 0x8c, 0x39, 0xb0, 0x3f, 0x9d, 0xfa, 0x5b, 0xdb, 0x87, 0x37, 0xdb, 0x42,
    0xbe, 0x83, 0xef, 0x7d, 0x50, 0x21, 0x45, 0x27, 0xd1, 0x26, 0x7a, 0xf7, 0x71, 0xfe, 0x4f, 0xc2,
    0xc6, 0xe7, 0x44, 0x21, 0x93, 0x32, 0x33, 0x1f, 0x69, 0x8b, 0x1b, 0x54, 0x6f, 0xd8, 0xc5, 0x42,
    0x0f, 0x10, 0x10, 0x6a, 0xe8, 0x63, 0x1a, 0xd8, 0xe8, 0xff, 0xe3, 0xca, 0x0e, 0x73, 0x9f, 0x7f,
    0x5b, 0xd0, 0xa8, 0x7c, 0x7b, 0x3a, 0x5e, 0x66, 0xba, 0x24, 0x47, 0x40, 0x59, 0x0d, 0xd1, 0x8a,
    0x60, 0x3f, 0x8b, 0x1c, 0x13, 0x86, 0x54, 0x7c, 0x38, 0xcc, 0x9e, 0x2d, 0xa8, 0x04, 0xf8, 0xbf,
```

```
    0xd5, 0xce, 0xcd, 0x86, 0x0e, 0x3f, 0x47, 0xd0, 0x88, 0x7b, 0xf7, 0xfb, 0x67, 0x33, 0xfa, 0x24,
    0x7f, 0xa6, 0x68, 0x59, 0x68, 0xc5, 0xee, 0x07, 0x42, 0x96, 0x9c, 0x13, 0xc9, 0xb0, 0xa0, 0x05,
    0x27, 0xe7, 0xea, 0xb4, 0x0f, 0x38, 0x38, 0xe5, 0x68, 0x4c, 0xd6, 0x1b, 0x66, 0xf1, 0xed, 0x1a,
    0xed, 0xce, 0xb2, 0x59, 0x69, 0x93, 0x2c, 0x60, 0xa2, 0xd1, 0x03, 0xec, 0x37, 0xe1, 0xf6, 0x92,
    0x91, 0x48, 0xe1, 0x64, 0xb3, 0x25, 0xe7, 0xe0, 0xfe, 0x47, 0xb7, 0x52, 0x8c, 0x4f, 0x50, 0xaa,
    0x16, 0xaf, 0x3d, 0xd8, 0x09, 0x5e, 0x55, 0x83, 0x96, 0x0b, 0x41, 0x47, 0x13, 0xe9, 0x4f, 0x1e,
    0xc3, 0x82, 0x39, 0x50, 0x1e, 0x9d, 0xcc, 0x67, 0x8c, 0x91, 0xec, 0x1c, 0x67, 0xeb, 0xf6, 0x97,
    0xf6, 0x25, 0xf6, 0xce, 0x74, 0x34, 0x20, 0xfa, 0x18, 0xdd, 0x04, 0x0a, 0xf8, 0xc0, 0x38, 0x58,
    0xf1, 0x41, 0xc0, 0x0d, 0x0e, 0xaa, 0x80, 0x63, 0x39, 0x54, 0x00, 0x08, 0xe9, 0xf2, 0xaa, 0xe1,
    0xec, 0x1b, 0x6d, 0xb8, 0x46, 0x6e, 0xee, 0x93, 0x2d, 0xa6, 0x9a, 0x63, 0xfb, 0x2e, 0x98, 0x1e,
    0x08, 0x3c, 0x83, 0xc5, 0xef, 0xcd, 0x47, 0x0a, 0xa4, 0xbf, 0x1e, 0xac, 0xd0, 0x9a, 0xde, 0x35,
    0x89, 0xe8, 0xf5, 0x87, 0x92, 0xbb, 0xe9, 0x06, 0x37, 0xb8, 0x53, 0x68, 0x1f, 0x34, 0x37, 0x18,
    0xc2, 0x80, 0x6f, 0xb8, 0xf4, 0xb1, 0xe6, 0xc9, 0xb5, 0xb6, 0x99, 0xed, 0x07, 0xda, 0x04, 0x0f,
    0x3e, 0xfb, 0x9d, 0x14, 0x97, 0x9f, 0xf8, 0xe8, 0xcd, 0x0c, 0x86, 0xc0, 0xfa, 0x5e, 0x80, 0xa4,
    0x08, 0xea, 0x4d, 0x24, 0x29, 0x43, 0x6d, 0x6a, 0xa4, 0x35, 0x2c, 0x4e, 0x0f, 0x8c, 0x30, 0x44,
    0x4e, 0x1a, 0x9b, 0x16, 0xd0, 0xea, 0x8e, 0x4e, 0x43, 0x9f, 0x0c, 0x69, 0xaa, 0x0d, 0x5f, 0x84,
    0xe1, 0x33, 0x87, 0x5a, 0x7c, 0x79, 0x42, 0xad, 0x1d, 0x90, 0xe6, 0xbb, 0x31, 0x9d, 0xc4, 0x41,
    0xd3, 0x71, 0x69, 0xef, 0xda, 0x5d, 0x23, 0xfe, 0xb4, 0xc6, 0x9d, 0x17, 0x9f, 0x9a, 0x90, 0xbc,
    0xb1, 0x51, 0x30, 0x19, 0xd0, 0xf2, 0xb6, 0x15, 0x93, 0xe2, 0x6d, 0xa5, 0xdb, 0xd8, 0x20, 0x1a,
    0x57, 0xb5, 0x14, 0x4f, 0x7d, 0x36, 0xe3, 0x9f, 0x97, 0xdd, 0xad, 0xcc, 0x2c, 0x51, 0x6f, 0x0b,
    0x97, 0x9a, 0x79, 0xbf, 0xce, 0x54, 0xd0, 0x0c, 0x8b, 0x9f, 0xa5, 0x99, 0xbc, 0x1b, 0xe3, 0x87,
    0xf3, 0x7a, 0x77, 0xec, 0xbf, 0xef, 0x57, 0xfa, 0x1e, 0x60, 0x50, 0xa3, 0x42, 0xe8, 0x8b, 0xd6,
    0xcd, 0xff, 0x92, 0x90, 0x72, 0xed, 0x59, 0xae, 0xa2, 0xf4, 0x6e, 0xc7, 0x4a, 0x08, 0x81, 0x16,
    0x03, 0x3f, 0xbc, 0xe6, 0xa7, 0x12, 0xab, 0x75, 0x57, 0x33, 0x40, 0xae, 0xe6, 0xc4, 0x45, 0x06,
    0xb3, 0xa8, 0x92, 0x98, 0x4d, 0x23, 0xf9, 0x20, 0xc5, 0x80, 0xfc, 0x80, 0x79, 0x1e, 0x81, 0xff,
    0x6b, 0x8e, 0x87, 0xff, 0xa5, 0x43, 0x6c, 0xcc, 0x34, 0x7b, 0x8b, 0xd0, 0xdd, 0xc5, 0x15, 0xce,
    0x1b, 0x34, 0x7d, 0x38, 0x65, 0xc1, 0xdd, 0x2e, 0x28, 0xc0, 0x5d, 0x9c, 0x3e, 0xab, 0xeb, 0x5f,
    0xc9, 0xb6, 0xbe, 0xce, 0x16, 0x0c, 0xd7, 0x7a, 0x37, 0xdf, 0x32, 0xd1, 0xdf, 0xf6, 0x50, 0x58,
    0xc2, 0xcc, 0x56, 0x54, 0x17, 0xae, 0x55, 0x36, 0x4a, 0x31, 0xae, 0x52, 0x50, 0xd1, 0xac, 0x51,
    0xbb, 0x48, 0x32, 0x27, 0x71, 0x48, 0x31, 0xab, 0xa6, 0xa4, 0xb5, 0x68, 0xb7, 0x63, 0xab, 0xba,
    0x75, 0x89, 0xe5, 0x12, 0x8b, 0x26, 0x0c, 0xbb, 0x3e, 0xd4, 0x3e, 0x27, 0xf3, 0xb2, 0x40, 0xcb,
    0x43, 0x07, 0x32, 0x74, 0x26, 0x1f, 0x8e, 0xa6, 0xa8, 0xe6, 0xfc, 0x23, 0xb2, 0x5e, 0xd1, 0x18,
    0x36, 0x75, 0xea, 0x90, 0xc4, 0xf5, 0xda, 0xb8, 0xc3, 0xdd, 0x11, 0x38, 0x76, 0x6b, 0xee, 0xf1,
    0x57, 0x9a, 0xb7, 0x36, 0x8a, 0xef, 0x5f, 0x48, 0x8b, 0x9f, 0x3d, 0x96, 0x35, 0xf4, 0xf4, 0xed,
    0x59, 0xd0, 0x2c, 0x8a, 0x88, 0x52, 0xf8, 0x41, 0xa5, 0xdb, 0x5c, 0x9f, 0x12, 0x27, 0xa4, 0xe2,
    0x94, 0x10, 0x64, 0x66, 0xc4, 0xaf, 0x23, 0x27, 0xe4, 0x8e, 0x76, 0x96, 0x4e, 0xe8, 0xd9, 0x19,
    0x89, 0xcd, 0xe7, 0xe7, 0x67, 0xb8, 0x62, 0x5a, 0x46, 0x41, 0x68, 0x4c, 0x3d, 0x7f, 0xb5, 0x35,
    0x20, 0x70, 0x47, 0x64, 0xbe, 0x66, 0x1a, 0x88, 0xdb, 0x38, 0x97, 0x52, 0x84, 0xdf, 0xee, 0xd5,
    0xd7, 0x36, 0x21, 0xa9, 0xc0, 0x85, 0xf5, 0xdd, 0x3d, 0x96, 0xe2, 0x9d, 0x24, 0x77, 0xac, 0xf7,
    0x30, 0xfb, 0xc4, 0x65, 0x6a, 0xfa, 0x78, 0x50, 0x24, 0x28, 0x76, 0x94, 0xd6, 0x60, 0x5b, 0x00,
    0x3f, 0xc9, 0x75, 0xad, 0x3c, 0x38, 0x82, 0x49, 0x0b, 0xf0, 0x53, 0x5d, 0x9b, 0x47, 0xfc, 0x0f,
    0xf3, 0x38, 0x8c, 0x14, 0x61, 0xb3, 0x31, 0xcc, 0xc5, 0x7f, 0x15, 0x4e, 0x70, 0xc5, 0x70, 0x9b,
    0xf9, 0x7a, 0xb0, 0x7b, 0x38, 0x77, 0x49, 0xd7, 0xa5, 0x0d, 0x71, 0x9e, 0x12, 0x19, 0x5a, 0x21,
    0x1a, 0xa2, 0x1d, 0x43, 0x91, 0xd4, 0xf2, 0x70, 0x11, 0x1f, 0xeb, 0x1e, 0xc6, 0x52, 0xfc, 0x24,
    0x8f, 0xe8, 0x45, 0x43, 0x18, 0x8d, 0x09, 0x1b, 0x85, 0xf1, 0xb0, 0x73, 0x9c, 0x9a, 0xa5, 0x45,
    0xb0, 0xb7, 0xdb, 0x3a, 0x5c, 0x83, 0x53, 0xf9, 0xe3, 0x4c, 0xf3, 0xa7, 0x78, 0x02, 0x26, 0x62,
    0xe1, 0xbf, 0x4b, 0xb3, 0xd5, 0x56, 0xe7, 0x56, 0x70, 0xcc, 0x0c, 0xb4, 0x1a, 0xc3, 0x21, 0xae,
    0x6d, 0x2d, 0xbf, 0x3f, 0xee, 0xa8, 0xda, 0x7b, 0x2c, 0x41, 0x0b, 0x42, 0xe9, 0xf1, 0x44, 0xcb,
    0x5c, 0x01, 0xcb, 0x1f, 0x5a, 0x1d, 0xc0, 0x9e, 0x5c, 0x70, 0x9e, 0x2f, 0x9f, 0xeb, 0xff, 0x0a,
    0xce, 0x6d, 0x43, 0xbd, 0x42, 0xef, 0x42, 0x44, 0x0a, 0xeb, 0x1e, 0xf7, 0x6e, 0x99, 0x56, 0xad,
    0xbb, 0xcb, 0x7b, 0xde, 0x7b, 0x9d, 0xdf, 0x2c, 0xba, 0x34, 0xcb, 0xe9, 0x7d, 0x41, 0x42, 0x78,
    0x29, 0x07, 0x82, 0xec, 0x2c, 0x01, 0xb5, 0x0e, 0x0f, 0xfa, 0x4a, 0x09, 0x15, 0x54, 0x53, 0x84,
    0x99, 0x9f, 0x83, 0x81, 0xdd, 0xdc, 0x1e, 0xdc, 0x81, 0x33, 0x6f, 0x1d, 0x30, 0x87, 0x0a, 0x2d,
    0x54, 0xb4, 0xd0, 0x14, 0xb6, 0x3d, 0xbb, 0x5d, 0xdc, 0xb1, 0x49, 0x3f, 0x3c, 0x5d, 0x7d, 0xd3,
    0x09, 0xe2, 0x37, 0x33, 0xa3, 0xeb, 0x30, 0x8c, 0x7e, 0x91, 0xe6, 0xfd, 0x6f, 0x59, 0x00, 0xea,
    0xf5, 0xab, 0xc3, 0x30, 0x42, 0x2b, 0x6d, 0x11, 0x77, 0xcb, 0xfd, 0x92, 0x3b, 0x59, 0x65, 0x48,
    0xb3, 0x10, 0xbe, 0xc4, 0xba, 0xc8, 0xac, 0x0e, 0xfd, 0xaa, 0x67, 0x89, 0xee, 0xa4, 0x40, 0x0a,
    0x14, 0x44, 0xde, 0xd4, 0x32, 0x95, 0x24, 0x9f, 0xe4, 0xe4, 0x16, 0x52, 0x6e, 0xc1, 0xf8, 0x17,
    0xb1, 0x6a, 0xb2, 0x27, 0x5f, 0x6c, 0xc9, 0x3d, 0x70, 0x44, 0x56, 0xe6, 0xbd, 0xf6, 0x46, 0xf7,
    0x49, 0x52, 0x20, 0xae, 0xf7, 0xae, 0x4d, 0x05, 0x2b, 0x03, 0xb6, 0x0a, 0x19, 0xf5, 0x2c, 0x77,
    0x28, 0xe1, 0x04, 0x9f, 0xec, 0xc1, 0xf6, 0x8b, 0x5b, 0xc6, 0xdc, 0x85, 0xf6, 0x48, 0x51, 0x28,
    0x82, 0xdf, 0x78, 0x06, 0x69, 0x5b, 0x01, 0xf1, 0x27, 0x99, 0x7f, 0xfb, 0x2b, 0x0c, 0x70, 0xb7,
    0x21, 0x37, 0x42, 0xf3, 0xba, 0xa3, 0xdb, 0xaf, 0xc6, 0xfa, 0xd8, 0x57, 0xd6, 0x02, 0xcb, 0xbd,
    0x20, 0xbb, 0xed, 0xde, 0xb2, 0x4a, 0x34, 0x5d, 0x8e, 0xf3, 0x2c, 0xf5, 0x91, 0xbb, 0x4e, 0x5d,
    0x5f, 0x4b, 0x5c, 0xc1, 0x39, 0xef, 0x84, 0xf7, 0x7a, 0xd7, 0x78, 0x70, 0x2b, 0xcc, 0x95, 0xe6,
    0x83, 0x05, 0x0e, 0x6d, 0x82, 0xd2, 0xb0, 0xae, 0xd9, 0x22, 0x43, 0xcf, 0x1f, 0x72, 0x8e, 0xe9,
    0xa5, 0x6e, 0x4b, 0xf4, 0x85, 0x03, 0x67, 0xb9, 0xf9, 0x67, 0x61, 0x76, 0x3e, 0x3a, 0xe6, 0x13,
    0x37, 0x50, 0x48, 0xb2, 0x67, 0x35, 0xfb, 0x73, 0xaf, 0x92, 0x36, 0x1f, 0xb3, 0xac, 0xcb, 0x3a,
    0x7a, 0x56, 0xfb, 0x7a, 0x6f, 0x2d, 0x28, 0xca, 0xf2, 0x20, 0xba, 0xf2, 0x0b, 0x5d, 0x80, 0x9f,
    0x4a, 0x92, 0x83, 0x3b, 0x61, 0xa2, 0xaa, 0x5e, 0x44, 0x6e, 0xd9, 0x0c, 0x39, 0xfc, 0xa7, 0x7b,
    0x1b, 0x2e, 0xb9, 0x2d, 0x9d, 0x09, 0xfa, 0xf3, 0x9e, 0x0a, 0x62, 0x79, 0xd3, 0x81, 0x7c, 0xc5,
    0xa0, 0x0d, 0xfc, 0x2b, 0xe1, 0x25, 0x09, 0xb0, 0xdb, 0x65, 0xea, 0x9e, 0x93, 0xb0, 0x0f, 0x38,
    0x61, 0x88, 0x44, 0x67, 0xe1, 0x00, 0x0b, 0x47, 0x54, 0x7f, 0x89, 0x68, 0x48, 0xe3, 0x82, 0x3d,
    0xdc, 0x2f, 0x13, 0xc9, 0x5b, 0xc5, 0xb5, 0x79, 0xa3, 0xbd, 0xa8, 0x7e, 0x59, 0xb0, 0xc1, 0x9d,
    0x6f, 0x30, 0xd5, 0x0a, 0xc8, 0x24, 0x01, 0x30, 0x42, 0x6d, 0x3e, 0x35, 0xc9, 0x1b, 0x6c, 0x55,
    0x27, 0x5d, 0x98, 0xc1, 0x5a, 0xba, 0xea, 0xc1, 0xfa, 0xcf, 0x0e, 0xab, 0x0d, 0x7c, 0xd6, 0x48,
```

```
    0x58, 0x7a, 0xa0, 0xd7, 0x14, 0xce, 0xb2, 0x36, 0xb8, 0xbb, 0xb6, 0x50, 0x1b, 0xf5, 0x91, 0xe7,
    0xd5, 0x9e, 0xfb, 0x00, 0x10, 0xb0, 0xfe, 0x97, 0x94, 0x67, 0xa6, 0xdb, 0x71, 0x00, 0xbc, 0x8b,
    0x95, 0x19, 0xfb, 0xef, 0xab, 0x6d, 0x78, 0x0c, 0x88, 0xf0, 0xba, 0x0b, 0xba, 0xeb, 0x36, 0xd2,
    0xa4, 0x24, 0x58, 0xef, 0x35, 0xf3, 0xae, 0xb5, 0x89, 0x1b, 0x0e, 0x99, 0x06, 0x6f, 0x2e, 0xf9,
    0xc8, 0xf3, 0x84, 0x62, 0x4d, 0xf4, 0xa0, 0xb9, 0xb8, 0x57, 0xf8, 0xa9, 0x5d, 0x07, 0xc7, 0xd4,
    0x5b, 0x05, 0xed, 0x62, 0x2c, 0x74, 0x05, 0xdc, 0x10, 0x06, 0x79, 0x6e, 0x09, 0x76, 0xdd, 0x31,
    0xdb, 0x69, 0x7e, 0x7b, 0x62, 0x18, 0x65, 0x70, 0x82, 0x0f, 0xa0, 0x59, 0x6e, 0x2a, 0xe5, 0x00,
    0xe0, 0x0b, 0xb6, 0xb9, 0x67, 0x29, 0x88, 0xe3, 0x24, 0x9e, 0xfb, 0xa5, 0x19, 0x13, 0x63, 0x26,
    0xe4, 0x67, 0xa6, 0x06, 0xc8, 0xfd, 0xfd, 0x8a, 0x07, 0x50, 0xdf, 0xfb, 0xf1, 0x79, 0x9a, 0x44,
    0x97, 0x71, 0x42, 0xf3, 0x39, 0x7e, 0xca, 0xdc, 0x8b, 0x39, 0x56, 0x6f, 0xd4, 0xcd, 0x29, 0xa3,
    0x61, 0xec, 0x18, 0xa1, 0x4e, 0xa2, 0x7e, 0xe1, 0x8f, 0x5c, 0x63, 0x54, 0xe9, 0xfc, 0x2e, 0x93,
    0x66, 0xbf, 0xb4, 0xf8, 0x46, 0x64, 0xa0, 0xfd, 0xfb, 0xa1, 0x07, 0xa4, 0xe7, 0x52, 0x92, 0xf7,
    0xa6, 0x01, 0xc2, 0x95, 0xfa, 0xc4, 0x6e, 0x8b, 0xb2, 0x48, 0x72, 0xb0, 0xbb, 0xf8, 0xa0, 0xef,
    0x3b, 0x12, 0x1f, 0x38, 0x28, 0x5b, 0xb6, 0x7a, 0x82, 0x56, 0xcc, 0x5b, 0x5b, 0x0c, 0xb2, 0x11,
    0x27, 0x75, 0xce, 0x59, 0x45, 0x5c, 0xb5, 0x48, 0xde, 0x6b, 0xf5, 0x37, 0x6b, 0x09, 0xe5, 0x2a,
    0x2e, 0x7d, 0x06, 0xfa, 0xef, 0xf0, 0xac, 0x44, 0xa0, 0x5d, 0xb2, 0xf2, 0xe8, 0x55, 0x87, 0x6a,
    0x71, 0xa2, 0x8d, 0x03, 0x2a, 0xc1, 0x21, 0x83, 0xda, 0x72, 0xa2, 0x4b, 0x58, 0x70, 0x39, 0xc5,
    0xd6, 0x8d, 0x6e, 0x45, 0x36, 0x8e, 0x13, 0x1a, 0x20, 0xeb, 0x2e, 0x4f, 0xdf, 0x16, 0xc5, 0x44,
    0xaa, 0x0c, 0x43, 0xf4, 0x17, 0x04, 0x06, 0xad, 0x60, 0x09, 0x57, 0xad, 0xaf, 0xf8, 0xe7, 0xcf,
    0xb6, 0xd4, 0xfa, 0x79, 0xf4, 0x2a, 0x05, 0xcc, 0xb2, 0xc0, 0xbb, 0x37, 0xad, 0x83, 0x88, 0x78,
    0x27, 0x8e, 0xf3, 0xaf, 0x2c, 0x90, 0xed, 0xed, 0x8b, 0x84, 0x26, 0x0b, 0x5c, 0x8e, 0x9a, 0x08,
    0x0c, 0x95, 0x69, 0x28, 0x87, 0xf5, 0xa7, 0x12, 0x40, 0xf9, 0x62, 0x7f, 0x19, 0x94, 0x4c, 0x6e,
    0x49, 0x9d, 0xfc, 0x2f, 0xec, 0x89, 0x5c, 0x32, 0x37, 0xeb, 0xd2, 0x7a, 0x09, 0xc8, 0x87, 0x3b,
    0x4c, 0xd7, 0x91, 0x9f, 0xf0, 0x9c, 0x97, 0x34, 0xea, 0x21, 0x39, 0xd0, 0xb6, 0x2d, 0x4c, 0x52,
    0xee, 0x01, 0xec, 0xff, 0x3c, 0x41, 0xbf, 0x91, 0x67, 0x1d, 0x15, 0xe8, 0xce, 0x48, 0x2f, 0x01,
    0x8b, 0x79, 0x3e, 0x9c, 0xe9, 0x5b, 0x5e, 0x39, 0xcb, 0x37, 0x32, 0xbf, 0x5a, 0xcb, 0x0a, 0x81,
    0x59, 0x05, 0x72, 0x85, 0x35, 0xcb, 0x60, 0xe2, 0x2b, 0x6b, 0x3b, 0x88, 0xc1, 0x59, 0x6d, 0x1f,
    0xcb, 0x28, 0xcb, 0x80, 0x93, 0xec, 0x30, 0x17, 0x82, 0x7f, 0x61, 0x24, 0x6f, 0x5c, 0x55, 0x4c,
    0x91, 0x99, 0x6c, 0x4e, 0x7a, 0xce, 0x53, 0xf6, 0x25, 0xa5, 0x15, 0x2e, 0x60, 0x98, 0x8c, 0xc3,
    0xb0, 0xdd, 0xac, 0x43, 0xc7, 0xed, 0x5c, 0xe1, 0x00, 0x86, 0x09, 0x57, 0x8b, 0xcf, 0x9b, 0xc6,
    0xdb, 0x8a, 0xa5, 0x3d, 0x59, 0xbd, 0x45, 0xc3, 0x0c, 0x9f, 0x22, 0x65, 0x37, 0xd3, 0xf9, 0xb4,
    0x9b, 0x99, 0x1f, 0xb0, 0x21, 0x4b, 0xd4, 0x52, 0xba, 0x5a, 0xbd, 0x71, 0xfe, 0x45, 0xdc, 0x1f,
    0x35, 0x35, 0x9a, 0xfe, 0x84, 0xe5, 0xd2, 0x5e, 0x1a, 0x88, 0x11, 0x7b, 0x89, 0x88, 0x6e, 0x5f,
    0xee, 0xd5, 0x48, 0xba, 0x46, 0x36, 0x98, 0x85, 0xf5, 0xdd, 0xc9, 0xee, 0x4d, 0x96, 0x2a, 0x4f,
    0x4f, 0x79, 0xb9, 0x84, 0xeb, 0x96, 0xee, 0x42, 0x33, 0xd4, 0x5a, 0x4e, 0x9e, 0xc3, 0x2e, 0x4a,
    0x8c, 0xf8, 0x83, 0xf5, 0xa0, 0x90, 0x32, 0x35, 0xbf, 0x47, 0xfd, 0x06, 0x00, 0xff, 0xd8, 0xc1,
    0x0b, 0x19, 0x9a, 0xae, 0x38, 0x5e, 0xb8, 0x70, 0x0c, 0x62, 0xbb, 0xbc, 0x5b, 0x0a, 0x63, 0xdb,
    0x64, 0xd8, 0x28, 0x2d, 0xd7, 0x11, 0x8c, 0xf3, 0xc0, 0x80, 0x36, 0xeb, 0xe6, 0xc6, 0x31, 0x73,
    0x1c, 0x92, 0x69, 0x82, 0xe7, 0x8a, 0x8f, 0xa8, 0xf6, 0x27, 0x5a, 0x4a, 0x05, 0xbb, 0x85, 0x80,
    0xb2, 0x74, 0xf2, 0x35, 0x53, 0x45, 0xb0, 0xe3, 0x4a, 0xf8, 0x56, 0xf9, 0x4b, 0x1a, 0x6d, 0x1e,
    0x44, 0x2a, 0xc3, 0x37, 0x61, 0xd6, 0x9a, 0xbc, 0x07, 0x29, 0xb4, 0xb3, 0xc6, 0x51, 0x35, 0x9b,
    0x68, 0xd4, 0x52, 0x8c, 0xe7, 0x70, 0x22, 0xba, 0xed, 0x3f, 0x91, 0x2d, 0x19, 0x71, 0xfc, 0xbb,
    0xcf, 0xd0, 0xfe, 0x08, 0xf0, 0x54, 0x2d, 0x7d, 0x2f, 0x71, 0x7b, 0x63, 0x38, 0x89, 0x14, 0xcf,
    0xcc, 0x25, 0x0c, 0x3a, 0x80, 0xec, 0x6a, 0x33, 0x15, 0x6b, 0x64, 0xbb, 0x8e, 0x9b, 0x29, 0x02,
    0x53, 0xcb, 0x23, 0x45, 0xbc, 0x82, 0x7e, 0x3b, 0x02, 0xd7, 0x26, 0xac, 0xbf, 0x1f, 0xa0, 0x28,
    0xe0, 0x8b, 0x51, 0x15, 0x99, 0x33, 0x55, 0x63, 0x24, 0x7d, 0xda, 0xb0, 0xf0, 0x26, 0xb0, 0x4b,
    0x52, 0x14, 0x11, 0x00, 0xb6, 0xe7, 0xdb, 0x10, 0x6e, 0x23, 0x93, 0x8e, 0xfb, 0xc2, 0xcb, 0x71,
    0x84, 0xa8, 0x1e, 0xd1, 0x7d, 0x27, 0x82, 0x1d, 0x52, 0x9c, 0x67, 0x93, 0xd3, 0xb3, 0x6e, 0x2c,
    0x1f, 0x09, 0x68, 0x1e, 0xb4, 0x57, 0x5f, 0xce, 0x2f, 0xf0, 0x47, 0x0a, 0x6a, 0xc2, 0x2f, 0x67,
    0x2f, 0x49, 0x50, 0x0d, 0xf2, 0xd8, 0xd0, 0x4c, 0x42, 0x37, 0x71, 0xcf, 0x4a, 0x2a, 0x24, 0xea,
    0xd8, 0xc2, 0x1d, 0x2f, 0x83, 0xbb, 0x52, 0xee, 0xb8, 0x72, 0x5d, 0x85, 0xb4, 0x59, 0x4a, 0x0a,
    0x63, 0xd8, 0x86, 0x3e, 0xf8, 0x81, 0xbb, 0x60, 0x4b, 0xed, 0x51, 0xe9, 0x9c, 0x93, 0x45, 0x1c,
    0x0c, 0xd3, 0xe2, 0xec, 0x87, 0xe1, 0xb0, 0x15, 0x5c, 0xfd, 0x1f, 0x5c, 0x97, 0xb7, 0xdb, 0x77,
    0x78, 0x5d, 0xa1, 0xa3, 0x74, 0x50, 0x82, 0x19, 0x7f, 0x9a, 0x70, 0xd9, 0x76, 0xb6, 0x3b, 0x7e,
    0x66, 0x29, 0x71, 0x46, 0x8a, 0x69, 0x0c, 0x2e, 0x75, 0x62, 0x1d, 0xc3, 0x60, 0x46, 0xd2, 0x7b,
    0x3e, 0x72, 0x5d, 0x26, 0x92, 0x3c, 0xfc, 0x28, 0xcd, 0xe0, 0x65, 0x69, 0xd8, 0xce, 0x87, 0x2a,
    0x30, 0xaa, 0xe5, 0x13, 0x0e, 0xa0, 0xe6, 0xee, 0x1c, 0xa7, 0xb0, 0x1e, 0x32, 0x92, 0xe7, 0xbc,
    0x64, 0x4c, 0xb6, 0xcd, 0x91, 0x3b, 0xc2, 0x87, 0x52, 0x31, 0x04, 0x08, 0x1d, 0x9e, 0xa9, 0xc2,
    0x27, 0x39, 0x23, 0x80, 0x14, 0x67, 0xc8, 0x72, 0x74, 0xb9, 0x1e, 0xdb, 0x76, 0x97, 0x80, 0x56,
    0x89, 0xec, 0x67, 0x9e, 0x69, 0xde, 0xdb, 0x04, 0x7e, 0x9b, 0x21, 0x32, 0x0f, 0x7b, 0xef, 0xf8,
    0xbc, 0x82, 0x70, 0x27, 0x17, 0x45, 0xf4, 0x2a, 0x64, 0x72, 0x1c, 0x4c, 0x68, 0x94, 0x23, 0xbb,
    0x23, 0xd8, 0xf6, 0xa3, 0xc9, 0xdc, 0x4b, 0xa6, 0xe1, 0x47, 0xae, 0xe1, 0x5a, 0x2d, 0xcb, 0xb2,
    0xe2, 0x3d, 0x67, 0xb7, 0x7c, 0x41, 0x59, 0xe0, 0xa9, 0x08, 0xaf, 0x3d, 0xea, 0xda, 0xc8, 0x49,
    0xf0, 0x0e, 0xe8, 0xcf, 0x58, 0x60, 0x9c, 0x0f, 0x49, 0x2e, 0x9c, 0xe8, 0x80, 0x49, 0xd2, 0x42,
    0x37, 0x2e, 0xce, 0x9c, 0x92, 0x37, 0x91, 0xb5, 0x74, 0xa0, 0x98, 0x15, 0xa9, 0x05, 0xab, 0xac,
    0x23, 0x55, 0xa4, 0x88, 0x37, 0x1a, 0x7d, 0x64, 0xa8, 0x88, 0x3f, 0x12, 0x68, 0xf2, 0x14, 0xdc,
    0xa4, 0x7f, 0xd0, 0x1f, 0xb8, 0x56, 0xc0, 0x87, 0xc0, 0x5e, 0xb6, 0xcc, 0xb8, 0x8a, 0x38, 0x05,
    0xc7, 0x7b, 0xae, 0xa3, 0x47, 0xf2, 0x63, 0x88, 0x74, 0xcb, 0xbc, 0xa8, 0x97, 0xa9, 0x6f, 0x12,
    0x1b, 0xf3, 0x40, 0xd5, 0xaf, 0x2b, 0x2f, 0xe0, 0x10, 0xb4, 0xe3, 0x06, 0x0b, 0x59, 0x17, 0xb6,
    0xc7, 0xdc, 0xd9, 0x20, 0x0c, 0x13, 0x97, 0xdf, 0x68, 0x64, 0x13, 0xb6, 0xd2, 0x57, 0xe6, 0x5e,
    0x7f, 0xb1, 0xd9, 0xe7, 0x91, 0xaf, 0x51, 0xa0, 0x5e, 0xfe, 0xb9, 0x8b, 0x93, 0x5e, 0x54, 0x49,
    0x43, 0x32, 0x81, 0x28, 0x1e, 0x5f, 0x95, 0x2b, 0x1e, 0x1f, 0x1d, 0x80, 0x16, 0xbd, 0x4f, 0xcb,
    0x7b, 0x7a, 0xbe, 0xdb, 0x75, 0x9a, 0xd0, 0x9e, 0x61, 0x9f, 0x32, 0xcb, 0x3e, 0xca, 0x24, 0x67,
    0x54, 0x66, 0xfc, 0xef, 0xd1, 0x58, 0xdc, 0x3d, 0xdc, 0x3c, 0xce, 0x99, 0xab, 0x0e, 0xbf, 0xd3,
    0x09, 0x09, 0xbc, 0x81, 0x10, 0x0a, 0x66, 0x57, 0x13, 0x81, 0x09, 0xc6, 0xc7, 0x11, 0x8a, 0xc2,
    0x57, 0xfe, 0x37, 0x8a, 0x3d, 0xab, 0x83, 0x1f, 0x97, 0xa0, 0x21, 0x29, 0x08, 0x34, 0x73, 0x99,
```

```
    0xde, 0xe3, 0xb7, 0x2b, 0x63, 0xad, 0xe1, 0x37, 0x98, 0x4b, 0xaf, 0xcb, 0x26, 0x08, 0x48, 0xc2,
    0xb8, 0x9e, 0xe8, 0x7d, 0x8a, 0xa4, 0xed, 0x27, 0xd5, 0xa0, 0xa4, 0xcf, 0x11, 0x1b, 0xcf, 0xc2,
    0xc1, 0x1c, 0xad, 0x2f, 0xe2, 0x59, 0xbd, 0x27, 0x8f, 0xea, 0x79, 0x17, 0xb5, 0x37, 0xd3, 0xb3,
    0x66, 0x7c, 0x2b, 0x74, 0x53, 0xaf, 0x73, 0x4b, 0x20, 0x5c, 0x45, 0x07, 0x73, 0x5e, 0x6f, 0x82,
    0x71, 0xa3, 0xf9, 0x21, 0x90, 0x67, 0x5e, 0xd8, 0xb7, 0xc5, 0xec, 0x48, 0x1c, 0x3d, 0xd1, 0x47,
    0xde, 0xac, 0xd2, 0x36, 0xce, 0x14, 0x1d, 0xc2, 0xcd, 0x64, 0x1f, 0xdf, 0x67, 0x07, 0x08, 0x72,
    0xd3, 0xc0, 0x17, 0x99, 0x68, 0xe6, 0xd5, 0x03, 0x02, 0xdb, 0x74, 0x62, 0x0e, 0x96, 0x53, 0x8c,
    0x72, 0xcb, 0x3f, 0xb5, 0x5f, 0x04, 0xfd, 0x5c, 0x27, 0x82, 0x66, 0x1a, 0xcc, 0x76, 0x65, 0x31,
    0xfa, 0x4a, 0xe7, 0x0d, 0x2d, 0xe7, 0x53, 0xab, 0xd6, 0xa7, 0xb3, 0x86, 0xeb, 0xc2, 0x26, 0x0c,
    0xfb, 0xb3, 0x4b, 0x8a, 0x0c, 0xe0, 0x3c, 0x78, 0x59, 0xe1, 0xba, 0x44, 0x9b, 0xc7, 0x31, 0x9c,
    0x13, 0x25, 0x9f, 0x89, 0xd7, 0x8b, 0x48, 0x15, 0x4b, 0x53, 0x7a, 0x46, 0x50, 0xef, 0x80, 0x1a,
    0xd7, 0x73, 0xd6, 0xae, 0x55, 0xf4, 0x71, 0xd6, 0x50, 0x57, 0x96, 0x0a, 0xc1, 0x76, 0x8f, 0x75,
    0x38, 0x8b, 0x96, 0xd6, 0x48, 0xb2, 0xa1, 0x94, 0xb8, 0x0c, 0x8a, 0x20, 0x7f, 0x3b, 0xb7, 0xe4,
    0x5b, 0x64, 0x7c, 0x49, 0x7a, 0x82, 0x8b, 0xc0, 0x61, 0x89, 0x78, 0x9d, 0xee, 0xd8, 0x0c, 0xe3,
    0xc6, 0x37, 0x47, 0xeb, 0xcd, 0x36, 0xd2, 0xe1, 0x51, 0xe6, 0xaf, 0x59, 0xbe, 0x18, 0x6e, 0x93,
    0xe2, 0xe7, 0xcc, 0x13, 0x43, 0x9c, 0x32, 0xb4, 0x3b, 0xc2, 0xd2, 0x57, 0x6f, 0x09, 0x03, 0x4e,
    0x10, 0x91, 0x4a, 0xc2, 0x0b, 0x62, 0x4b, 0x16, 0xc7, 0xe2, 0x35, 0xc4, 0x9e, 0x72, 0x34, 0x8a,
    0x3d, 0x1f, 0xdd, 0x23, 0x7b, 0x3b, 0x4e, 0xd6, 0xf7, 0x0b, 0xeb, 0x57, 0x45, 0x7d, 0xcd, 0x3c,
    0xc2, 0x45, 0xf3, 0xe6, 0xac, 0x2c, 0xeb, 0x40, 0x91, 0xb0, 0xb4, 0x3c, 0xa3, 0xbf, 0x59, 0x55,
    0xab, 0xaa, 0x6a, 0x87, 0xc5, 0x99, 0x46, 0xb0, 0xd7, 0xca, 0xee, 0xde, 0xac, 0x81, 0xb2, 0x8e,
    0xb3, 0x11, 0x67, 0x98, 0x31, 0x25, 0x91, 0x2e, 0xfd, 0xab, 0x21, 0x36, 0xfe, 0x06, 0x25, 0x67,
    0xa2, 0x9e, 0x01, 0xe7, 0xa0, 0x04, 0xac, 0x3d, 0x6e, 0xcd, 0xb7, 0xb3, 0x55, 0x52, 0x9b, 0x41,
    0x84, 0x08, 0x21, 0x02, 0x22, 0x27, 0xef, 0x8f, 0x1b, 0x63, 0xb7, 0x1f, 0xc4, 0xaf, 0x5e, 0xd0,
    0x10, 0x4b, 0xaf, 0xe8, 0x76, 0x0b, 0x0a, 0x24, 0x8e, 0x45, 0x85, 0xf1, 0x96, 0xd7, 0x11, 0xf8,
    0x0a, 0xdd, 0x53, 0x39, 0xdf, 0x74, 0x8f, 0x47, 0x11, 0x41, 0xa8, 0xc8, 0x66, 0x89, 0xcc, 0x05,
    0x48, 0x93, 0x5b, 0xf1, 0x8a, 0x46, 0x40, 0x06, 0xa4, 0x6f, 0x9e, 0x8f, 0x11, 0xb2, 0x38, 0xe7,
    0xc0, 0x42, 0xbc, 0x99, 0x39, 0xe0, 0x2a, 0x05, 0xc7, 0x6e, 0xf6, 0x2b, 0x46, 0x71, 0xf2, 0x4f,
    0x68, 0xa0, 0xb2, 0x7d, 0xc3, 0xda, 0x1f, 0x85, 0x12, 0x92, 0x8d, 0xeb, 0xfd, 0x35, 0x42, 0xac,
    0x6e, 0xc4, 0x79, 0x82, 0xf7, 0x3e, 0x4a, 0x78, 0xa0, 0x95, 0x0f, 0x1f, 0x4c, 0x5f, 0x83, 0xd5,
    0x8e, 0xc1, 0xcc, 0x66, 0xa3, 0xe8, 0x02, 0xc1, 0x2e, 0xab, 0xf6, 0x96, 0x02, 0xed, 0xb5, 0xc6,
    0xe9, 0xf3, 0x3c, 0x8a, 0xa9, 0x86, 0x1a, 0x3c, 0xd4, 0x26, 0xf6, 0x35, 0x81, 0x20, 0x11, 0xea,
    0x23, 0xb2, 0xfb, 0x4f, 0x1d, 0x8e, 0x25, 0xf2, 0x73, 0x4a, 0x10, 0x7a, 0xbe, 0xd1, 0xf9, 0x91,
    0x90, 0x7c, 0x6d, 0xee, 0xf5, 0xda, 0x13, 0x70, 0xff, 0xa3, 0xcf, 0x5f, 0x4f, 0x7c, 0x6f, 0xaa,
    0x17, 0xc5, 0xcc, 0x17, 0x66, 0x87, 0x5e, 0xee, 0x21, 0x53, 0x15, 0x4c, 0x3c, 0x97, 0x7b, 0xcd,
    0x8f, 0x90, 0x1a, 0xd2, 0x64, 0x68, 0xba, 0x30, 0xe8, 0x1c, 0x60, 0x8d, 0xf8, 0xa4, 0x58, 0xc7,
    0xb8, 0xf6, 0x92, 0xa2, 0xf1, 0x1b, 0x0e, 0xaf, 0x36, 0x69, 0xd8, 0x3b, 0xda, 0xba, 0xa0, 0xd0,
    0x20, 0x46, 0x5b, 0x45, 0xd6, 0x11, 0x1c, 0xff, 0xc7, 0x69, 0x45, 0x22, 0xc3, 0x93, 0x9b, 0xd0,
    0x97, 0x5a, 0xdd, 0x8d, 0x26, 0x26, 0x28, 0x6d, 0x4c, 0x11, 0x41, 0x2c, 0x4d, 0x8a, 0x9c, 0x30,
    0x78, 0xfe, 0x48, 0xd2, 0x30, 0xee, 0xce, 0x1c, 0x2a, 0x16, 0xa4, 0x1f, 0xdb, 0xcd, 0x2e, 0x2d,
    0x83, 0x96, 0x07, 0x0b, 0xef, 0x65, 0x67, 0x2c, 0x90, 0xfb, 0x68, 0xef, 0x5e, 0x77, 0x50, 0x31,
    0x38, 0x59, 0x63, 0x2f, 0x98, 0xb3, 0xe3, 0xab, 0x47, 0x7e, 0x73, 0xbd, 0x47, 0xd9, 0x8c, 0xa6,
    0xaf, 0xa0, 0x57, 0xd3, 0xfa, 0x2b, 0x87, 0xc3, 0x5e, 0xe3, 0xe6, 0x57, 0xa7, 0x28, 0x05, 0xc5,
    0x32, 0xe1, 0x69, 0x4c, 0xaf, 0xc9, 0x22, 0xdd, 0x91, 0x6c, 0x37, 0x64, 0x42, 0xd4, 0xa9, 0xa2,
    0xb7, 0x5c, 0x50, 0x4e, 0x4a, 0x23, 0x5a, 0x72, 0x8b, 0x09, 0x49, 0x5f, 0x83, 0xb4, 0xf2, 0x1f,
    0x10, 0x0f, 0x81, 0xa1, 0x26, 0x1b, 0xd8, 0xf4, 0x8e, 0x33, 0x6c, 0xab, 0x51, 0xd7, 0x9c, 0xc0,
    0xe9, 0x02, 0x09, 0xf3, 0xe2, 0xc2, 0xbf, 0x6d, 0x13, 0x8d, 0xb9, 0x90, 0xa9, 0xb5, 0x99, 0x65,
    0x4b, 0x86, 0x13, 0x71, 0x13, 0x4f, 0x0f, 0xb0, 0x7d, 0x51, 0xd9, 0x70, 0x1e, 0x42, 0x42, 0x8e,
    0x65, 0x39, 0xa5, 0x00, 0x16, 0xa9, 0x56, 0x32, 0x06, 0xf0, 0xdd, 0xb6, 0xc3, 0x5a, 0xc3, 0xa9,
    0x20, 0xb6, 0x5d, 0x75, 0xc5, 0x35, 0xc5, 0x81, 0xff, 0x3c, 0x14, 0xc1, 0x53, 0xac, 0xe5, 0xb7,
    0xa1, 0xea, 0xb1, 0x54, 0xfd, 0x64, 0x3e, 0xfa, 0x70, 0xa7, 0xda, 0xc0, 0xbe, 0x76, 0xf5, 0xee,
    0x6d, 0x68, 0x60, 0x92, 0x84, 0xc3, 0x3e, 0x98, 0xa2, 0x0e, 0x71, 0x3b, 0x89, 0xb4, 0xbc, 0x80,
    0xd8, 0xe9, 0x70, 0x28, 0x0f, 0x3f, 0x31, 0x24, 0x12, 0xec, 0x24, 0xa4, 0x99, 0x0b, 0x48, 0xdc,
    0x6e, 0x9b, 0x52, 0x95, 0xa5, 0x59, 0x5b, 0xea, 0xe9, 0xd7, 0x72, 0xb0, 0xdd, 0xd4, 0xf3, 0x29,
    0x81, 0x4c, 0x27, 0xdb, 0xeb, 0xe4, 0x26, 0xec, 0x93, 0xf6, 0xe6, 0x59, 0x82, 0xe4, 0xae, 0x9b,
    0x90, 0xb2, 0xe7, 0x7e, 0xd8, 0x4e, 0xe9, 0xd7, 0xe3, 0x4a, 0x05, 0x0a, 0x2a, 0xd1, 0x62, 0xc2,
    0xe1, 0xbb, 0x6b, 0x60, 0x35, 0xda, 0xe0, 0x91, 0x6c, 0xba, 0xef, 0xb6, 0xf9, 0xa7, 0xb6, 0x25,
    0x15, 0x9f, 0x5e, 0xfb, 0xd4, 0x4c, 0xb2, 0x77, 0x06, 0x2f, 0x98, 0x59, 0xad, 0xce, 0xfc, 0x21,
    0xc3, 0xbd, 0x47, 0x7e, 0xeb, 0xf5, 0xea, 0x93, 0x1e, 0xf7, 0x68, 0xbc, 0xd7, 0xac, 0xd6, 0x26,
    0xcd, 0x05, 0x12, 0x87, 0x97, 0xce, 0xb2, 0x75, 0xcf, 0xf9, 0x3b, 0x8b, 0x97, 0x84, 0x56, 0x88,
    0x7a, 0x13, 0xa6, 0x34, 0x68, 0x07, 0x75, 0xb8, 0xbb, 0x73, 0x51, 0xe4, 0xd5, 0x50, 0x53, 0x47,
    0xd7, 0x4b, 0x01, 0x6a, 0x22, 0x24, 0xfd, 0xc2, 0xc2, 0x34, 0x93, 0xca, 0x91, 0xf8, 0x5d, 0x27,
    0x8d, 0x95, 0xa8, 0x7c, 0x8f, 0x47, 0x79, 0x19, 0xcf, 0x06, 0x0e, 0x64, 0x7a, 0xde, 0x62, 0x5a,
    0x33, 0x8a, 0xa0, 0x4e, 0x60, 0x84, 0xd8, 0x83, 0xa8, 0x0d, 0x6d, 0xcd, 0xff, 0x1b, 0xed, 0x57,
    0xc9, 0x2c, 0x30, 0x9a, 0x17, 0x0c, 0xd3, 0xe6, 0x46, 0xea, 0x8e, 0xca, 0xb3, 0x87, 0x90, 0xae,
    0x4d, 0x11, 0x2e, 0xa7, 0xd9, 0xc2, 0xb6, 0x0a, 0xd3, 0x93, 0xbc, 0xaf, 0x7e, 0xa9, 0xe7, 0x11,
    0x4b, 0x90, 0xce, 0xe8, 0xbc, 0xa5, 0xfa, 0xb7, 0x61, 0x38, 0x7b, 0xc9, 0xea, 0x77, 0xa7, 0xde,
    0x70, 0x2a, 0xf3, 0x13, 0x29, 0x74, 0x4c, 0xf5, 0x00, 0x35, 0xe1, 0xd9, 0x13, 0xdc, 0x09, 0x94,
    0x92, 0x6b, 0xc6, 0x5a, 0xb4, 0x15, 0x5a, 0x52, 0xb5, 0x10, 0x96, 0xbb, 0xad, 0xac, 0x11, 0xa9,
    0xe5, 0x49, 0xdf, 0x34, 0x41, 0x15, 0x48, 0x37, 0xb0, 0x7e, 0xbd, 0x95, 0x25, 0x4f, 0x80, 0x21,
    0x00, 0x5f, 0x30, 0xf9, 0x1a, 0xd0, 0xe3, 0xb7, 0x1c, 0x17, 0x38, 0x5a, 0xbb, 0xe3, 0x26, 0x19,
    0x31, 0xde, 0x12, 0x29, 0xef, 0x57, 0x20, 0x13, 0xab, 0xe9, 0x3b, 0x5a, 0x0b, 0x05, 0x6d, 0xa8,
    0x34, 0x37, 0x34, 0xf2, 0x45, 0x53, 0xcf, 0x2a, 0x89, 0x05, 0x76, 0x6e, 0x4f, 0x90, 0x99, 0x11,
    0x2b, 0x6b, 0x3c, 0x32, 0x33, 0x71, 0x37, 0x01, 0xf3, 0xcb, 0x40, 0x27, 0x3a, 0xfe, 0x19, 0x25,
    0xe1, 0x59, 0x55, 0x68, 0xca, 0x16, 0xc8, 0xdd, 0x1d, 0xd4, 0xf9, 0x63, 0x11, 0xdf, 0x63, 0x08,
    0xf7, 0x95, 0xec, 0x3a, 0xb7, 0x99, 0x6f, 0xf1, 0x74, 0x6b, 0xce, 0x11, 0x05, 0xa5, 0x17, 0xab,
    0x3f, 0xba, 0x32, 0x54, 0x94, 0x4f, 0xe6, 0x0c, 0x5e, 0xf2, 0x6e, 0x52, 0xb1, 0xbd, 0x18, 0xea,
```

```
    0xcd, 0x03, 0xcd, 0xd8, 0x25, 0x9a, 0xb8, 0x44, 0x08, 0xe1, 0x5d, 0x36, 0xe5, 0xa7, 0xd5, 0x23,
    0x80, 0xab, 0x08, 0xc3, 0x22, 0x97, 0x94, 0xa8, 0x00, 0xd7, 0x12, 0xd9, 0x1f, 0x05, 0xad, 0x63,
    0x8e, 0xb3, 0x91, 0x46, 0xcc, 0x77, 0xcf, 0xb4, 0x7b, 0xa8, 0x1d, 0xc3, 0x15, 0x05, 0xcc, 0xb1,
    0x9e, 0xed, 0xa2, 0xd9, 0x3e, 0x3e, 0xa3, 0x48, 0x16, 0x79, 0xfc, 0xf0, 0x2e, 0x16, 0x02, 0x36,
    0xc3, 0xd4, 0x2d, 0x00, 0x32, 0x51, 0xc4, 0xb2, 0xd9, 0xad, 0x86, 0xdb, 0x4a, 0xc6, 0x4e, 0xfc,
    0xf0, 0xd3, 0x69, 0xa6, 0xa7, 0x8a, 0xd1, 0x78, 0x55, 0xd9, 0x62, 0x96, 0x05, 0xbb, 0xe2, 0x12,
    0x22, 0xc8, 0xa9, 0xdf, 0xf9, 0xef, 0x56, 0xc4, 0xa9, 0xb8, 0x37, 0x99, 0x9d, 0x17, 0x75, 0xc0,
    0xd4, 0x5b, 0x4c, 0x81, 0x30, 0x19, 0x77, 0x8e, 0xb1, 0xa6, 0xa3, 0x6c, 0x55, 0x46, 0x8b, 0x84,
    0x54, 0x4d, 0x6f, 0xbf, 0x94, 0xe7, 0xf3, 0x6a, 0xc0, 0x79, 0x66, 0x17, 0xe4, 0x3d, 0x02, 0x0d,
    0x24, 0xc8, 0xae, 0x21, 0x6a, 0x86, 0xd8, 0xeb, 0x52, 0xd5, 0x84, 0xab, 0x29, 0xc3, 0x0e, 0x9e,
    0x56, 0x53, 0x6a, 0x7d, 0x81, 0x57, 0x16, 0x2e, 0x51, 0xa8, 0xa7, 0x41, 0x25, 0x3d, 0xae, 0x0a,
    0xc6, 0x0e, 0xe6, 0xab, 0x53, 0xa2, 0x98, 0xee, 0xae, 0x32, 0xe7, 0xcb, 0x02, 0x91, 0x36, 0x7b,
    0xbc, 0x56, 0x74, 0x0c, 0x72, 0xb7, 0x3c, 0x28, 0x2d, 0xdf, 0xad, 0xbf, 0xa9, 0xe6, 0xf6, 0xed,
    0x86, 0x5e, 0xeb, 0xa1, 0x13, 0x20, 0xb2, 0xeb, 0x8a, 0xcd, 0x10, 0xda, 0xb4, 0xaa, 0x6e, 0x9e,
    0xd6, 0x41, 0xdb, 0xee, 0xf8, 0x35, 0x16, 0x2e, 0xb2, 0x39, 0xa6, 0x7b, 0xd5, 0x77, 0x04, 0xa0,
    0xc4, 0xb9, 0xfb, 0x34, 0x1a, 0xfa, 0xad, 0x6d, 0x23, 0x46, 0x61, 0xe4, 0x83, 0x35, 0x86, 0x9a,
    0x90, 0x75, 0x1d, 0x1b, 0x8c, 0x53, 0x69, 0xd6, 0x24, 0xc1, 0xd3, 0x9e, 0xd2, 0x26, 0x9d, 0x3c,
    0x4e, 0xab, 0xe1, 0x32, 0xc4, 0xae, 0xcf, 0xd2, 0x32, 0x9e, 0x93, 0xa9, 0xb1, 0x16, 0x42, 0xc4,
    0xb5, 0xa0, 0xfd, 0x62, 0x8f, 0xe7, 0x6e, 0xad, 0x77, 0xc7, 0xa2, 0x03, 0x66, 0x74, 0xd1, 0x14,
    0xc2, 0xde, 0xe0, 0x02, 0x04, 0xc0, 0x8f, 0xe4, 0x03, 0x0a, 0x00, 0x03, 0x89, 0xb6, 0x33, 0x23,
    0x8c, 0x7c, 0x71, 0xc5, 0xb9, 0xe7, 0x3a, 0x90, 0xea, 0xf6, 0x8f, 0x6b, 0xa5, 0xfa, 0x83, 0x71,
    0x60, 0xb2, 0xa1, 0x82, 0x94, 0x29, 0xc5, 0x01, 0x8e, 0xfc, 0xab, 0x8d, 0x4e, 0x0f, 0x6b, 0x6c,
    0x98, 0x5c, 0xa6, 0x30, 0x7b, 0x66, 0x9e, 0x78, 0x12, 0x2d, 0x69, 0x00, 0x59, 0x69, 0xd1, 0x9c,
    0x6e, 0x08, 0xa5, 0x55, 0x37, 0x5a, 0x25, 0xa3, 0xcd, 0xb8, 0x3e, 0x53, 0xaf, 0x9f, 0xc1, 0x1e,
    0xa2, 0x80, 0x2e, 0x57, 0xcc, 0x85, 0xc6, 0xcd, 0x6e, 0xdb, 0x5f, 0x97, 0x87, 0xf9, 0xb2, 0xd3,
    0xcd, 0x34, 0xf4, 0x68, 0x05, 0xec, 0xb5, 0xd9, 0xbe, 0x1b, 0xdc, 0x36, 0xdd, 0x15, 0x94, 0x13,
    0x43, 0x9e, 0xbf, 0xdf, 0x79, 0x3e, 0xc0, 0x1e, 0x4b, 0x98, 0x2f, 0x7a, 0xf0, 0x2e, 0xaa, 0xbc,
    0x0c, 0x80, 0x33, 0xd6, 0x6d, 0x84, 0x44, 0x5c, 0x78, 0x00, 0x26, 0x51, 0x86, 0x3f, 0x89, 0x40,
    0x5b, 0x6d, 0x8a, 0xc6, 0x3c, 0x5e, 0xd4, 0xe3, 0xef, 0xbe, 0xd6, 0x4d, 0x9b, 0x0a, 0x23, 0x26,
    0x7a, 0x67, 0x96, 0xa2, 0xba, 0xa7, 0x26, 0xe7, 0x44, 0xde, 0xc3, 0x56, 0x11, 0x19, 0x18, 0x4c,
    0x07, 0x98, 0x73, 0xe7, 0x01, 0xd4, 0x80, 0x78, 0xe6, 0x2a, 0xfd, 0x19, 0x25, 0xdc, 0x92, 0x12,
    0xe7, 0x29, 0x9a, 0xb9, 0x1a, 0x2d, 0x4a, 0xb5, 0xfa, 0x9d, 0x44, 0x5a, 0xbb, 0x5d, 0xca, 0x6b,
    0xd6, 0x2c, 0x07, 0x7c, 0xba, 0xcf, 0x60, 0x45, 0xdf, 0xce, 0x3d, 0xea, 0x60, 0x99, 0x6d, 0xc1,
    0x42, 0xd1, 0xeb, 0xca, 0xe3, 0x7e, 0x79, 0xb7, 0x6e, 0xb7, 0xba, 0x35, 0xfd, 0xe9, 0x7e, 0xfa,
    0xe9, 0xe6, 0x11, 0xe5, 0xeb, 0xc8, 0xbc, 0xca, 0x0f, 0x51, 0xda, 0x06, 0x7d, 0x2a, 0x64, 0x0d,
    0xbd, 0xda, 0x29, 0x20, 0x5d, 0x83, 0xc0, 0x08, 0x2c, 0xf2, 0x7e, 0x00, 0x65, 0xb4, 0x00, 0x0e,
    0x09, 0x91, 0x98, 0xe3, 0xc1, 0x07, 0x1c, 0x7b, 0x62, 0xb7, 0xed, 0x1e, 0xab, 0x68, 0xf6, 0x82,
    0x03, 0x4c, 0xd3, 0x98, 0x34, 0xd5, 0xe7, 0xfc, 0x73, 0xf6, 0x57, 0xed, 0x95, 0xdd, 0x19, 0x9b,
    0x3a, 0x18, 0x8b, 0xcf, 0x06, 0x5d, 0xa5, 0xa6, 0xf2, 0x5b, 0x82, 0xd0, 0x04, 0xf7, 0xc1, 0x18,
    0x56, 0xba, 0x4a, 0x24, 0x95, 0xe6, 0x48, 0xf1, 0xae, 0xf2, 0x35, 0x6a, 0x53, 0x20, 0x7f, 0x48,
    0x32, 0xfc, 0x53, 0xca, 0xc5, 0x6d, 0xe8, 0x9c, 0x56, 0x64, 0xfd, 0x29, 0xb0, 0x41, 0x63, 0x78,
    0x8a, 0xe4, 0x2c, 0x46, 0x1c, 0x01, 0x8c, 0x2a, 0x44, 0x3f, 0xce, 0x4d, 0xa3, 0x8f, 0x15, 0x60,
    0x76, 0xf4, 0x0a, 0xd3, 0x57, 0xf4, 0x88, 0xaa, 0xfb, 0xdc, 0x7c, 0xfd, 0xde, 0x89, 0xc9, 0xf2,
    0xf3, 0x09, 0x78, 0x5d, 0x32, 0xa5, 0xc8, 0x56, 0x6f, 0xf0, 0x5b, 0x08, 0xc4, 0x4f, 0x63, 0xe7,
    0x5e, 0x8e, 0x59, 0x2a, 0x3f, 0x85, 0x1e, 0x97, 0x7b, 0x84, 0xf0, 0x35, 0x7a, 0x4f, 0xe9, 0xde,
    0x61, 0x23, 0x38, 0xc1, 0x9f, 0x85, 0xf0, 0x84, 0xda, 0x90, 0x2a, 0x39, 0xd0, 0xe5, 0xe3, 0x6a,
    0x29, 0xdb, 0x58, 0xc8, 0x6d, 0x1a, 0x02, 0x0d, 0x81, 0x45, 0x36, 0x47, 0x5d, 0xa7, 0x1f, 0xf7,
    0xfe, 0x4f, 0xdf, 0x8b, 0x08, 0x3d, 0xa1, 0x74, 0x2e, 0x42, 0xcc, 0x9e, 0x36, 0xf7, 0xc6, 0x89,
    0x16, 0xa9, 0xbf, 0x0a, 0x8a, 0x19, 0xb2, 0xa5, 0x4d, 0x62, 0x85, 0x58, 0x0d, 0xa5, 0x91, 0x1a,
    0xd3, 0xc4, 0x77, 0x7f, 0xf7, 0xc1, 0x39, 0x35, 0xd9, 0xb1, 0x97, 0x42, 0xfa, 0x39, 0x0f, 0xdd,
    0x37, 0xda, 0x50, 0x72, 0xf2, 0x62, 0x42, 0xcf, 0xde, 0xbb, 0x00, 0x56, 0x6e, 0x72, 0x3b, 0xf4,
    0x16, 0x10, 0x00, 0x69, 0x67, 0x4a, 0x82, 0xf8, 0xfb, 0x71, 0xc7, 0x5a, 0xe6, 0xa7, 0xfc, 0xc9,
    0x9e, 0x6c, 0xe6, 0xbb, 0xc3, 0x50, 0xbf, 0x62, 0xbc, 0x31, 0x69, 0xef, 0x6f, 0x92, 0x85, 0x1f,
    0x31, 0xe2, 0x0f, 0x5e, 0x7a, 0x56, 0x3e, 0x2f, 0xda, 0x03, 0x85, 0x32, 0x11, 0xae, 0x16, 0xe8,
    0x60, 0x02, 0xac, 0x56, 0xc6, 0x0a, 0xb2, 0xed, 0xff, 0xfc, 0xa3, 0xe7, 0x8a, 0xa7, 0xd8, 0x41,
    0x7a, 0x15, 0x8c, 0xde, 0xd6, 0x05, 0xd5, 0x25, 0x0d, 0xac, 0x9b, 0xe9, 0xae, 0x3e, 0xb0, 0x3e,
    0x30, 0xd2, 0xba, 0x0f, 0x2e, 0x16, 0x1c, 0x5f, 0xdb, 0xe0, 0xca, 0xa1, 0xf5, 0x4d, 0xa3, 0xe8,
    0x61, 0x6d, 0x29, 0xfe, 0x88, 0x9a, 0x73, 0x2a, 0xe2, 0x08, 0xd4, 0x2e, 0xd5, 0xa7, 0x4b, 0x58,
    0x4a, 0xa4, 0x1b, 0x5f, 0x77, 0xc2, 0x34, 0xf1, 0xbb, 0xb1, 0xd7, 0x7a, 0xfa, 0x3b, 0x4e, 0x8f,
    0x34, 0x54, 0x19, 0x7c, 0xca, 0x82, 0x32, 0xa1, 0xe4, 0xf8, 0xf0, 0x5c, 0xee, 0xd8, 0xc4, 0x48,
    0x4e, 0x80, 0xb6, 0xbf, 0x5f, 0x7f, 0x6a, 0xd5, 0x2e, 0xdb, 0xae, 0xf4, 0xf4, 0xcc, 0x35, 0xdd,
    0x84, 0x10, 0xd1, 0x76, 0x6a, 0x31, 0xe5, 0xd3, 0x6a, 0xb6, 0x54, 0xc3, 0xca, 0x8f, 0x53, 0x02
    };

inline unsigned char revertbyte(unsigned char b) {
    int i;
    unsigned char tmp, res = 0;
    for (i = 0; i < 8; i++) {
        tmp = b & (1<<i);
        tmp = tmp>>i;
        res |= tmp<<(7-i);
    }
    return res;
}

static unsigned int lfsr_backshift(unsigned int *lokey, unsigned int *hikey) {
```

```
534     unsigned int loc, hic, blob;
535
536     loc = *lokey;
537     hic = *hikey;
538     blob = hic & 0x58000000;
539     {
540   blob = (blob ^ blob/2);
541   blob = (blob ^ blob/4);
542   blob = (blob & 0x11111111);
543   blob = (blob * 0x11111111);
544   blob = (blob & 0x10000000)>>28;
545   blob = blob ^ ((hic & 0x80000000)>>31);
546     }
547     *lokey = (*lokey)<<1;
548     *lokey |= blob;
549     loc &= 0x80000000;
550     loc = loc>>31;
551     *hikey = (*hikey)<<1;
552     *hikey |= loc;
553
554     return blob;
555 }
556
557 int main(int argc, char *argv[])
558 {
559     unsigned int lokey, hikey;
560     unsigned int pos, idx;
561     char i;
562     unsigned char mask;
563     int fd;
564     unsigned char *tmask = malloc(sizeof(data));
565     if (tmask == NULL) {
566         printf("malloc failed\n");
567         return -1;
568     }
569
570     pos = sizeof(data);
571     hikey = (unsigned int)(revertbyte(data[pos-1])<<24 | revertbyte(data[pos-2])<<16 |
572             revertbyte(data[pos-3])<<8 | revertbyte(data[pos-4]));
573     lokey = (unsigned int)(revertbyte(data[pos-5])<<24 | revertbyte(data[pos-6])<<16 |
574             revertbyte(data[pos-7])<<8 | revertbyte(data[pos-8]));
575
576     idx = pos-8;
577     while (idx != 0) {
578         mask = 0;
579         for (i = 0; i < 8; i++) {
580             mask |= lfsr_backshift(&lokey, &hikey)<<i;
581         }
582         tmask[idx-1] = mask;
583         idx--;
584     }
585
586     fd = openat(AT_FDCWD, "payload.bin", O_TRUNC|O_CREAT|O_WRONLY, 0666);
587     if (fd < 0) {
588         printf("unable to openfile");
589         return -1;
590     }
591
592     for (idx = 0; idx < pos-8; idx++) {
593         tmask[idx] ^= data[idx];
594     }
595     write(fd, tmask, pos-8);
596
597     lfsr_backshift(&lokey, &hikey);
598
599     printf("key = 0x%02X%02X%02X%02X%02X%02X%02X%02X\n",
600       (hikey&0xff), (hikey&0xff00)>>8, (hikey&0xff0000)>>16, (hikey&0xff000000)>>24,
601       (lokey&0xff), (lokey&0xff00)>>8, (lokey&0xff0000)>>16, (lokey&0xff000000)>>24);
602     return 0;
603 }
```

revert_lfsr.c

microcontroller_disassembler.c

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

enum opcodes {
    INVOP,
    MOVLO,
    MOVHI,
    XOR,
    OR,
    AND,
    ADD,
    SUB,
    MUL,
    DIV,
    CJMP,
    JMP,
    CALL,
    RET,
    LOAD,
    STORE
};

const char * condstr(unsigned int cond)
{
    switch (cond) {
  case 0:
      return "eq";
  case 1:
      return "ne";
  case 2:
      return "ls";
  case 3:
      return "ge";
  default:
      return "??";
    }
}

const char * interpretstr(unsigned int syscall)
{
    switch (syscall) {
  case 0x1:
      return "syscall exit";
  case 0x2:
      return "syscall print";
  case 0x3:
      return "syscall get_cycles";
  default:
      return "??";
    }
}

const char * regstr(unsigned int reg)
{
    switch (reg) {
  case 0:
      return "r0";
  case 1:
      return "r1";
  case 2:
      return "r2";
  case 3:
      return "r3";
  case 4:
      return "r4";
  case 5:
      return "r5";
  case 6:
      return "r6";
  case 7:
```

```
77          return "r7";
78      case 8:
79          return "r8";
80      case 9:
81          return "r9";
82      case 10:
83          return "r10";
84      case 11:
85          return "r11";
86      case 12:
87          return "r12";
88      case 13:
89          return "sp";
90      case 14:
91          return "r14";
92      case 15:
93          return "lr";
94      default:
95          return "r?";
96      }
97  }
98
99  #define wdtobigendian(opcode) \
100     (((opcode) & 0xff)<<8 | ((opcode) & 0xff00)>>8)
101 #define dwtobigendian(opcode) \
102     (((opcode) & 0xff)<<24 | ((opcode) & 0xff00)<<8 |\
103      ((opcode) & 0xff0000)>>8 | ((opcode) & 0xff000000)>>24)
104
105 int main(int argc, char *argv[])
106 {
107     FILE *in, *out;
108     long entry = 0;
109     unsigned int i;
110     size_t size, wsize;
111     unsigned char opcode, opsize;
112     unsigned int inst, offset, cond, jmp, nbinst;
113     unsigned int rout, rin, imm, rin2;
114     int ret;
115
116     if (argc != 3) {
117   printf("Usage: %s binaryfile disassemblefile\n", argv[0]);
118   return 0;
119     }
120
121     in = fopen(argv[1], "r");
122     if (in == NULL) {
123   printf("Open failed (%d): '%s'\n", __LINE__, strerror(errno));
124   return -1;
125     }
126
127     out = fopen(argv[2], "w+");
128     if (out == NULL) {
129   printf("Open failed (%d): '%s'\n", __LINE__, strerror(errno));
130   return -1;
131     }
132
133     fseek(in, entry, SEEK_SET);
134
135     i = entry;
136     nbinst = 0;
137     while (!feof(in)) {
138   size = fread(&opcode, 1, 1, in);
139   if (feof(in))
140       break;
141   if (size != 1) {
142       printf("Opcode read error : %s\n", strerror(errno));
143       break;
144   }
145
146   ret = fseek(in, -1, SEEK_CUR);
147   if (ret == -1) {
148       printf("Seek error : '%s'\n", strerror(errno));
149       break;
150   }
151
152   opsize = opcode > 8 ? 2 : 2;
```

```
153
154   inst = 0;
155   size = fread(&inst, opsize, 1, in);
156   if (size != 1) {
157       printf("Instruction read error : '%s'\n", strerror(errno));
158       break;
159   }
160
161   fprintf(out, "  %08x:\t %04x\t ", i, wdtobigendian(inst));
162   switch ((enum opcodes)((inst & 0xf0)>>4)) {
163       case MOVLO:
164     imm = (inst & 0xff00)>>8;
165     rout = inst & 0xf;
166     fprintf(out, "movlo\t %s, #0x%x\n", regstr(rout), imm);
167     break;
168       case MOVHI:
169     imm = (inst & 0xff00)>>8;
170     rout = inst & 0xf;
171     fprintf(out, "movhi\t %s, #0x%x\n", regstr(rout), imm);
172     break;
173       case CALL:
174     imm = (inst & 0xff00)>>8;
175     imm += (inst & 0x3)<<8;
176     cond = (inst & 0x8);
177     if (cond == 8) {
178         fprintf(out, "syscall\t #0x%x\t // %s\n",
179           imm, interpretstr(imm));
180     } else {
181         if ((imm & 0x200)) {
182       fprintf(out, "call\t 0x%x\t // +0x%x\n",
183         (nbinst+1)*2 + imm, imm);
184         } else {
185       fprintf(out, "call\t 0x%x\t // -0x%x\n",
186         (nbinst+1)*2 - imm, imm);
187         }
188     }
189     break;
190       case CJMP:
191     imm = (inst & 0xff00)>>8;
192     imm += (inst & 0x3)<<8;
193     cond = (inst & 0xc)>>2;
194     if ((imm & 0x200)) {
195         imm = (~imm & 0x3ff) + 1;
196         fprintf(out, "cj.%s\t 0x%x\t // -0x%x\n",
197           condstr(cond), (nbinst+1)*2 - imm, imm);
198     } else {
199         fprintf(out, "cj.%s\t 0x%x\t // +0x%x\n",
200           condstr(cond), (nbinst+1)*2 + imm, imm);
201     }
202     break;
203       case JMP:
204     imm = (inst & 0xff00)>>8;
205     imm += (inst & 0x3)<<8;
206     cond = (inst & 0xc)>>2;
207     if ((imm & 0x200)) {
208         imm = (~imm & 0x3ff) + 1;
209         fprintf(out, "jmp\t 0x%x\t // -0x%x\n",
210           (nbinst+1)*2 - imm, imm);
211     } else {
212         fprintf(out, "jmp\t 0x%x\t // +0x%x\n",
213           (nbinst+1)*2 + imm, imm);
214     }
215     break;
216       case AND:
217     rout = inst & 0xf;
218     rin = (inst & 0xf000)>>12;
219     rin2 = (inst & 0xf00)>>8;
220     fprintf(out, "and\t %s, %s, %s\n",
221       regstr(rout), regstr(rin), regstr(rin2));
222     break;
223       case OR:
224     rout = inst & 0xf;
225     rin = (inst & 0xf000)>>12;
226     rin2 = (inst & 0xf00)>>8;
227     fprintf(out, "or\t %s, %s, %s\n",
228       regstr(rout), regstr(rin), regstr(rin2));
```

```
    break;
      case XOR:
    rout = inst & 0xf;
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    fprintf(out, "xor\t %s, %s, %s\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case SUB:
    rout = inst & 0xf;
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    fprintf(out, "sub\t %s, %s, %s\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case ADD:
    rout = inst & 0xf;
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    fprintf(out, "add\t %s, %s, %s\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case RET:
    rin = (inst & 0xf00)>>8;
    if ((inst & 0x8))
        fprintf(out, "iret\n");
    else
        fprintf(out, "ret\t %s\n", regstr(rin));
    break;
      case STORE:
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    rout = inst & 0xf;
    fprintf(out, "str\t %s, [%s,%s]\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case LOAD:
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    rout = inst & 0xf;
    fprintf(out, "ldr\t %s, [%s,%s]\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case MUL:
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    rout = inst & 0xf;
    fprintf(out, "mul\t %s, %s, %s\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      case DIV:
    rin = (inst & 0xf000)>>12;
    rin2 = (inst & 0xf00)>>8;
    rout = inst & 0xf;
    fprintf(out, "div\t %s, %s, %s\n",
      regstr(rout), regstr(rin), regstr(rin2));
    break;
      default:
    printf("Unknown opcode number 0x%x\n", (inst & 0xf0)>>4);
    fprintf(out, "\n");
    break;
  }
  i += opsize;
  nbinst++;
    }
    fclose(in);
    fclose(out);
}
```

microcontroller_disassembler.c

bin2hex.c

```
#include <sys/types.h>
#include <sys/stat.h>
```

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[]) {
    char c[3] = {'\0','\0','\0'};
    int fdin, fdout;
    int i, j;
    unsigned char buf[16];
    char line[46], tmp[46];
    unsigned long sum;
    unsigned int address;
    struct stat stat;
    ssize_t size, wsize;

    if (argc != 3) {
  printf("Usage: %s file.bin file.hex\n", argv[0]);
  return 0;
    }

    fdin = open(argv[1], 0);
    if (fdin < 0) {
  printf("Can't open '%s' file\n", argv[1]);
  return -1;
    }

    fdout = open(argv[2], O_CREAT|O_TRUNC|O_WRONLY, 0644);
    if (fdout < 0) {
  printf("Can't open '%s' file\n", argv[2]);
  close(fdin);
  return -1;
    }

    fstat(fdin, &stat);

    j = 0;
    address = 0;
    while (j < (stat.st_size+15)/16) {
  size = read(fdin, buf, 16);
  if (size < 0) {
      printf("Can't read\n");
      break;
  }

  snprintf(line, sizeof(line), "%02X%04X00",
    (unsigned int)size, address);
  strcpy(tmp, line);

  for (i = 0; i < size; i++) {
      snprintf(line, sizeof(line), "%s%02X",
        tmp, buf[i]);
      strcpy(tmp, line);
  }

  i = 0;
  sum = 0;
  while (line[i] != '\0' && i < (size*2)+8) {
      c[0] = line[i++];
      c[1] = line[i++];
      sum += strtoul(c, NULL, 16);
  }
  sum = (0x100 - (sum % 0x100)) % 0x100;

  snprintf(line, sizeof(line), ":%s%02X\n",
    tmp, (unsigned int)sum);

  wsize = write(fdout, line, (size*2)+12);
  if (wsize < 0) {
      printf("Can't write : %s\n", strerror(errno));
      break;
  }
  j++;
  address += size;
```

```
    }

    snprintf(line, sizeof(line), ":00000001FF");
    wsize = write(fdout, line, 11);
    if (wsize < 0) {
  printf("Can't write : %s\n", strerror(errno));
    }
    close(fdin);
    close(fdout);
    return 0;
}
```

bin2hex.c

fw.hex

```
:1000000020F01002C803C802C80100000000000070
:10001000000000000000000000000000000000E0
:10002000000000000000000000000000000000D0
:10003000000000000000000000000000000000C0
:10004000000000000000000000000000000000B0
:10005000000000000000000000000000000000A0
:1000600000000000000000000000000000000090
:1000700000000000000000000000000000000080
:1000800000000000000000000000000000000070
:1000900000000000000000000000000000000060
:1000A000000000000000000000000000000000050
:1000B000000000000000000000000000000000040
:1000C000000000000000000000000000000000030
:1000D000000000000000000000000000000000020
:1000E000000000000000000000000000000000010
:1000F000000000000000000000000000000000000
:10010000000000000000000000000000000000EF
:10011000000000000000000000000000000000DF
:10012000000000000000000000000000000000CF
:10013000000000000000000000000000000000BF
:10014000000000000000000000000000000000AF
:100150000000000000000000000000000000000F
:100160000000000000000000000000000000008F
:100170000000000000000000000000000000007F
:100180000000000000000000000000000000006F
:100190000000000000000000000000000000005F
:1001A000000000000000000000000000000000004F
:1001B000000000000000000000000000000000003F
:1001C000000000000000000000000000000000002F
:1001D000000000000000000000000000000000001F
:1001E000000000000000000000000000000000000F
:1001F000000000000000000000000000000000FF
:10020000000000000000000000000000000000EE
:10021000000000000000000000000000000000DE
:10022000000000000000000000000000000000CE
:10023000000000000000000000000000000000BE
:10024000000000000000000000000000000000AE
:100250000000000000000000000000000000009E
:100260000000000000000000000000000000008E
:100270000000000000000000000000000000007E
:100280000000000000000000000000000000006E
:100290000000000000000000000000000000005E
:1002A000000000000000000000000000000000004E
:1002B000000000000000000000000000000000003E
:1002C000000000000000000000000000000000002E
:1002D000000000000000000000000000000000001E
:1002E000000000000000000000000000000000000E
:1002F000000000000000000000000000000000FE
:10030000000000000000000000000000000000ED
:10031000000000000000000000000000000000DD
:10032000000000000000000000000000000000CD
:10033000000000000000000000000000000000BD
:10034000000000000000000000000000000000AD
:100350000000000000000000000000000000009D
:100360000000000000000000000000000000008D
:100370000000000000000000000000000000007D
:100380000000000000000000000000000000006D
:100390000000000000000000000000000000005D
:1003A000000000000000000000000000000000004D
```

```
:1003B000000000000000000000000000000000003D
:1003C000000000000000000000000000000000002D
:1003D000000000000000000000000000000000001D
:1003E000000000000000000000000000000000000D
:1003F00000000000000000000000000000000000FD
:1004000000000000000000000000000000000000EC
:1004100000000000000000000000000000000000DC
:1004200000000000000000000000000000000000CC
:1004300000000000000000000000000000000000BC
:1004400000000000000000000000000000000000AC
:10045000000000000000000000000000000000009C
:10046000000000000000000000000000000000008C
:10047000000000000000000000000000000000007C
:10048000000000000000000000000000000000006C
:10049000000000000000000000000000000000005C
:1004A000000000000000000000000000000000004C
:1004B000000000000000000000000000000000003C
:1004C000000000000000000000000000000000002C
:1004D000000000000000000000000000000000001C
:1004E000000000000000000000000000000000000C
:1004F00000000000000000000000000000000000FC
:1005000000000000000000000000000000000000EB
:1005100000000000000000000000000000000000DB
:1005200000000000000000000000000000000000CB
:1005300000000000000000000000000000000000BB
:1005400000000000000000000000000000000000AB
:10055000000000000000000000000000000000009B
:10056000000000000000000000000000000000008B
:10057000000000000000000000000000000000007B
:10058000000000000000000000000000000000006B
:10059000000000000000000000000000000000005B
:1005A000000000000000000000000000000000004B
:1005B000000000000000000000000000000000003B
:1005C000000000000000000000000000000000002B
:1005D000000000000000000000000000000000001B
:1005E000000000000000000000000000000000000B
:1005F00000000000000000000000000000000000FB
:1006000000000000000000000000000000000000EA
:1006100000000000000000000000000000000000DA
:1006200000000000000000000000000000000000CA
:1006300000000000000000000000000000000000BA
:1006400000000000000000000000000000000000AA
:10065000000000000000000000000000000000009A
:10066000000000000000000000000000000000008A
:10067000000000000000000000000000000000007A
:10068000000000000000000000000000000000006A
:10069000000000000000000000000000000000005A
:1006A000000000000000000000000000000000004A
:1006B000000000000000000000000000000000003A
:1006C000000000000000000000000000000000002A
:1006D000000000000000000000000000000000001A
:1006E000000000000000000000000000000000000A
:1006F00000000000000000000000000000000000FA
:1007000000000000000000000000000000000000E9
:1007100000000000000000000000000000000000D9
:1007200000000000000000000000000000000000C9
:1007300000000000000000000000000000000000B9
:1007400000000000000000000000000000000000A9
:100750000000000000000000000000000000000099
:100760000000000000000000000000000000000089
:100770000000000000000000000000000000000079
:100780000000000000000000000000000000000069
:100790000000000000000000000000000000000059
:1007A0000000000000000000000000000000000049
:1007B0000000000000000000000000000000000039
:1007C000210511E620F6101A2DFC1D002A001A0141
:1007D00038883BBB3999E908F9DB688A711AA7F4B4
:1007E000D8000000000000000000000000000000031
:1007F00000000000000000000000000000000000F9
:00000001FF
```

fw.hex