Solution Challenge SSTIC 2019

Loïc Jourdheuil

22 mai 2019

Table des matières

1. Présentation

Bonjour,

Récemment un individu au comportement suspect nous a été signalé. Il semblerait qu'il s'attaque à la communauté sécurité informatique française avec notamment l'intention de lui nuire.

Sans preuve, il est difficile d'agir à son encontre. Ainsi, nous avons décidé de saisir son téléphone portable afin de collecter des éléments confirmant nos hypothèses. Cependant son téléphone semble posséder plusieurs couches de chiffrement qui nous empêchent d'accéder à ses données.

Dans l'incapacité de contourner ces systèmes de chiffrement, nous avons décidé de faire appel à vous pour nous aider.

Nous avons consacré du temps à rendre possible le démarrage du téléphone sécurisé dans un environnement virtualisé.

Malheureusement le coffre de clef du téléphone ciblé n'a pas pu être copié. Avant de devoir restituer le téléphone, nous avons été en mesure d'enregistrer une trace de consommation de courant lors du démarrage du téléphone. Nous espérons que cela pourra vous être utile.

Des instructions techniques plus précises vous seront fournies.

Bonne chance pour votre mission et nous comptons sur vous pour nous communiquer toutes les preuves que vous pourrez

trouver au cours de votre investigation à l'adresse mail suivante : challenge2019@sstic.org.

La communauté sécurité informatique française dépend de vous !

Le défi consiste à analyser le micro-logiciel d'un téléphone. Les traces laissées sur ce téléphone permettent de remonter à une adresse e-mail (...@challenge.sstic.org). À vous de la récupérer.

2. Démarrer le téléphone

Au démarrage de Qemu, une clef privé RSA nous est demandé. Le message de départ nous indique qu'une `trace de consommation de courant lors du démarrage du téléphone nous est fourni.`

En affichant la trace, on remarque une fenêtre (Figure 1) de consommation différente entre 700000 et 1489000 :

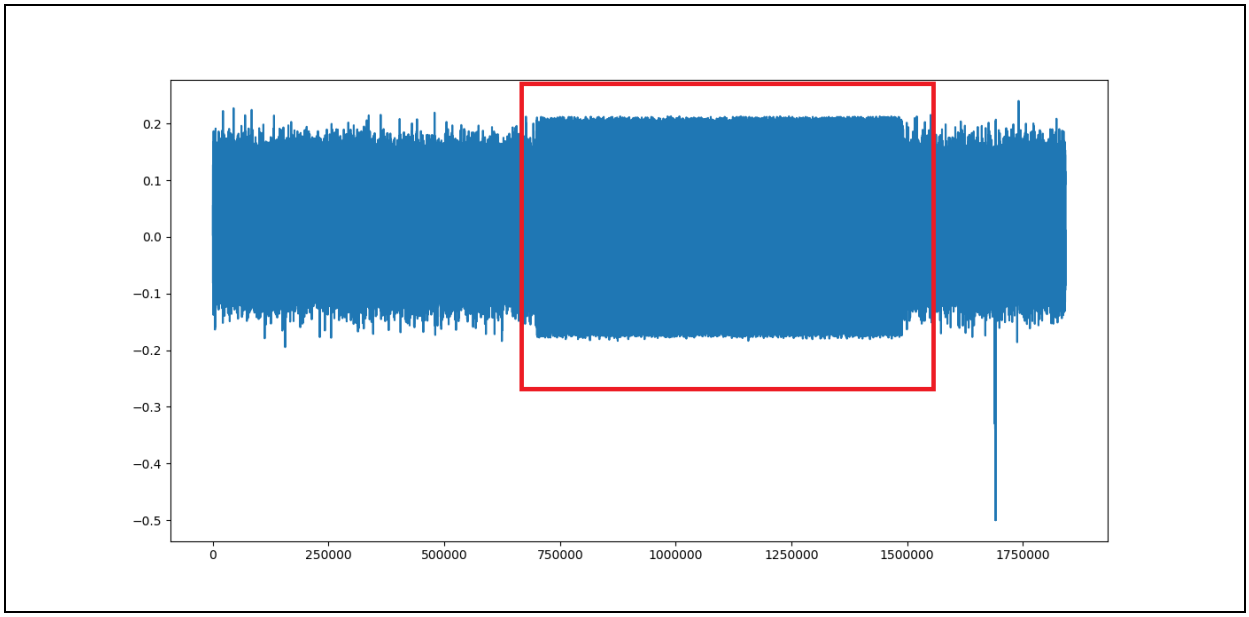


Figure 1 : Analyse de courbe de consomation

En zoomant sur dans cette trace, j'ai remarqué 2 schémas différents qui se répètent 2048 fois (Figure 2). Ces 2 schémas représentent une différence entre le cas d'un bit à 0 (opération puissance de 2) et le cas bit à 1 (opération puissance de 2 et multiplication).

En appliquant un filtre, il est possible de retrouver la clef (Annexe A).

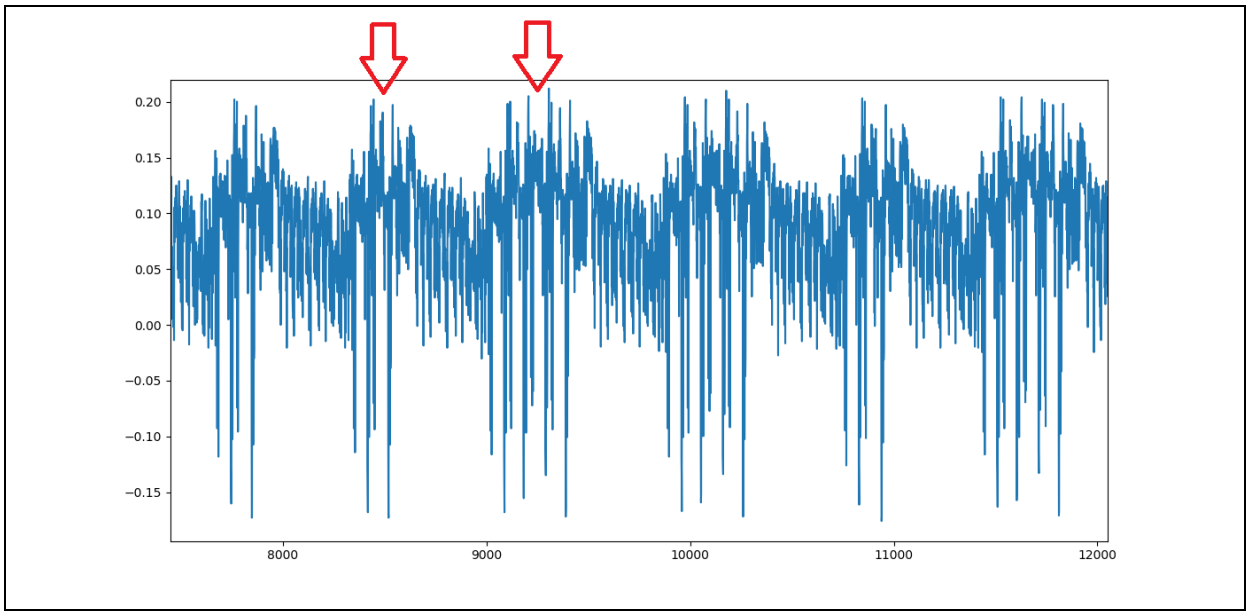


Figure 2 : différence entre opération puissance de 2 et multiplication + puissance de 2

```
##########################################
#     virtual environment detected     #
#          QEMU 3.1+ is needed          #
##########################################
NOTICE:  Booting SSTIC ARM Trusted Firmware
KEYSTORE: AES Key is still encrypted, need decryption
KEYSTORE: Need RSA key to decrypt
KEYSTORE: RSA private exponent is not set, please set it in the keystore or enter hex value :
23d87cdf97bb95abe6273c384190c765f552ab86f6de30a8db74435c95e6e3138f54af689812d8f9359cf0f4d453a0c11ec68ce470216c09e74c
8947adaf23e902415d61ddf2c0ffe459cbb40f7de42bdb7cd14093100a570e8c29819765e2d8d276f86471b52ac29aa2ce2bb72cd45006279e
82bec253ae9675fe45824f6001
KEYSTORE: Key read:
HEXDUMP :
-----------------------------------------------
23 d8 7c df 97 bb 95 ab e6 27 3c 38 41 90 c7 65
f5 52 ab 86 f6 de 30 a8 db 74 43 5c 95 e6 e3 13
8f 54 af 68 98 12 d8 f9 35 9c f0 f4 d4 53 a0 c1
1e c6 8c e4 70 21 6c 09 e7 4c 89 47 ad af 23 e9
02 41 5d 61 dd f2 c0 ff e4 59 cb b4 0f 7d e4 2b
db 7c d1 40 93 10 0a 57 0e 8c 29 81 97 65 e2 d8
d2 76 f8 64 71 b5 2a c2 9a a2 ce 2b b7 2c d4 50
06 27 9e 82 be c2 53 ae 96 75 fe 45 82 4f 60 01
-----------------------------------------------
KEYSTORE: Decrypting ...
```

```
+-------------+-+--+-+-+-+---+---+------+-+--+----+---+-+-+-+-+-+-+--+--+-+---+-+--+---+--+-+-+--+--+-+-+---+--+---+-----+-+--+-+-+-+-+--+--+------+--
+-+-+-+---+-+-+-+---+----+-+----------+--+----+--+-+--+-+---+-+--+---+-+-+-+--+-+-+-+--+----+-+-+---+-+--+----+--+--+--+-+---+--+-----+-+--+-+--+-+---+--
+--+-+--+-+----+-+-+----+---+-+-----+-+-+-+-+--+-+--+-+-+---+---+--+-+----+-+--+-+--+----+-+-+-+--+---+-+-+-+--+---+-+--+-+-+-+---+-+-------+-+---+--+------+-+----+--
+-+-+-----+-+-+--+--+---+--+---------+----+-+---+---+-------+--+----+--+-+---+-+-+-+-+--+-+-+--+-+-+-+--+--+-----+---+-+-+-+--+-+-+-+-+--+-+-+-+-------
+--+-+--+-+-+--+---+-+-+---+-+--+----+---+-+-+-+-+-+-+-+-+-+-+-------+-+--+---+-+-+-+-+--+-+-+--+-+-+-----+-+--+--+-+-+--+--+------+---+-------+---+--+-
+-+-+-+----+---+-+-+-+--+--+-+--+-+--+--+-+-+-+----+--+---+----+---+-+---+--+-+-+---+-+-+-+---+-------+-+--+-+--+-----+-------+-+-+----+---+-+-+---+-+----
+--+-+----+-+---+-+-+-+-+----+------+-+------+--+-+-+-+---+--+----+--+--+-+---+--+-+-+-+-----+-+-+-+---+-+-+---+-+--+-+--+-+---+---+-+-+-+-+----+-+--+-+--+---+-
------+-+---+----+--+-+--+-+-+-+--+--+---+--+--+--+-+-+-+----+-+-+---+----+-+----+-+-+--+-+---+-+-+-+--+--+---+---+-+-+--+--+-+-----+----+--+-+-+--+-+--
+-+--+-+----+--+--+-----+-+----+-+-+-+--+-+--+--+-+-+-+--+-+-----+-+-+--+--+--+--+---+--+--+--+--+-+-+-+-+--+---+-+--+-+-+-+----+-+-----+---+-+------+---
--+-+-+-----+-+-+-+---+-+-+---+----+-+---+-+-+-+-+--+--+--+-+--+---+-+-+--+-+-+--+-+-+-+--+---+-+-+-+-+-+--+-+---+-+-+-+-+-----+-+--+-+-+-+----+
```

Bravo, envoyez le flag **SSTIC{a947d6980ccf7b87cb8d7c246}** à l'adresse challenge2019@sstic.org pour valider votre avancée

3. Communication avec le sécure élément

J'ai maintenant accès au fichier du téléphone. Dans le dossier « /root », j'ai une image d'un schéma avec des portes logique et un fichier python.

Dans le fichier python, j'ai trouvé le texte suivant :

```
"""

TODO :

        - Implémentation de la communication avec le secure element

                * Entrées A et B (A0 = bit de poids faible)

                * Entrée OP

                * Sortie Out (Out0 = bit de poids faible)

                * Les boutons permettent à l'utilisateur de rentrer sa combinaison secrète
pour le déchiffrement

        - Supprimer les docs de conception

"""
```

J'ai donc regardé l'image pour traduire les portes logiques en code python (Annexe B).

Une fois les tests des cas « bouton appuyer » et « bouton relâcher » passés, j'ai constaté qu'il n'y a que 4 milliards de combinaisons possibles. J'ai donc lancé le programme en brute force.

La clef trouvée est la suivante :

```
5fb3a83d1fd97137076019ad6e96c6a366fb6b32618d162e00cdee9bad427a8a
```

En ajoutant la clef, j'ai trouvé le flag 2 :

```
SSTIC{5fb3a83d1fd97137076019ad6e96c6a366fb6b32618d162e00cdee9bad427a8a}
```

4. Clef du sécure OS

J'ai maintenant accès au fichier suivant : « /root/safe_01/decrypted_file ».

J'ai désassemblé le programme et constaté qu'après une vérification du nombre de paramètres, il envoie une exception, la rattrape et affiche le message « Not good ».

Je n'ai pas d'indication sur la longueur du flag qu'il prend en entrée.

En testant plusieurs cas, je constate qu'il y a une différence de temps d'exécution entre les flags de longueur 32 et les autres.

Je me suis donc concentré sur ce qu'il se passe entre l'exception et l'affichage.

La commande *objdump* me permet de trouver un comportement étrange, un saut loin en arrière dans la partie Dwarf du programme:

```
00000090 000000000000001c 00000094 FDE cie=00000000
pc=0000000000402e34..0000000000402e68

  DW_CFA_advance_loc: 1 to 0000000000402e35

  DW_CFA_def_cfa_offset: 32

  DW_CFA_offset: r29 (x29) at cfa-32

  DW_CFA_offset: r30 (x30) at cfa-24

  DW_CFA_val_expression: r28 (x28) (DW_OP_skip: -12222)

  DW_CFA_nop

  DW_CFA_nop
```

Cela correspond à la partie *.gnu.hash* du elf.

Pour analyser le fonctionnement du code en Dwarf, j'ai fait un simulateur en python (Annexe C) :

J'ai pu isoler une fonction *main* et 7 sous-fonctions.

La fonction la plus grosse en taille (la fonction 7) n'est appelée, dans la fonction 6, qu'avec 2 cas.

J'ai donc utilisé 2 constantes à la place de la fonction 7.

1. Fonction Main :

La fonction main est plutôt simple :

```
def main(a, b, c, d, e, f, g, h):
  for _ in range(4):
    a, b, c, d = fonction_2(a, b, c, d, *fonction_1(e, f, g, h))
    e, f, g, h = fonction_2(e, f, g, h, *fonction_1(a, b, c, d))
  return e, f, g, h, a, b, c, d
```

En l'inversant j'ai constaté qu'il n'était pas nécessaire d'inverser la fonction 1.

```
def main_rev(e, f, g, h, a, b, c, d):
  for _ in range(4):
    e, f, g, h = fonction_rev_2(e, f, g, h, *fonction_1(a, b, c, d))
    a, b, c, d = fonction_rev_2(a, b, c, d, *fonction_1(e, f, g, h))
  return a, b, c, d, e, f, g, h
K3,K4,K1,K2,K7,K8,K5, K6 = main_rev(0x658302a6,0x8e8e1c24,0xdc7564f1,0x612e5347,0xd9c69b74,0xa86ec613,0x65850b36,0xe76aaed5)
print(f"0x{K1:08x}{K2:08x} 0x{K3:08x}{K4:08x} 0x{K5:08x}{K6:08x} 0x{K7:08x}{K8:08x}")
```

2. Fonction 1:

La fonction 1 est principalement une SBox :

```
tab1 = [0x5963B39B,0x30F75ADD,0x103FECBC,0x00392E7A,0x35DF7ADF,0xE19ABD13,0xDAF8B35C,0xF8798214,
    0xB30C2305,0x067980E9,0x6900D940,0x035E876F,0xA3857014,0x56C8E162,0xE9748F56,0x91D4E409,
    0xDCC75A09,0xAC65F52F,0x8571DD07,0x019EDCF6,0x51CEF9EB,0x1EB1B17D,0x0ABE446F,0x3B277CFE,
    0x843869BC,0xB23EA298,0x7C296F51,0xCD799972,0x62180A64,0x0AC052D5,0xF0076205,0x13183193,
    0xB908CB94,0x4BF4CD3C,0xDEE5D48A,0xF64F9A74,0xD64A15D0,0xB2CAD434,0x64E9013B,0xF46CC1D2,
    0x9D78E9DB,0x11789216,0x335689E6,0x074C7EDB,0xE6EB6185,0xD020170B,0xF304AA15,0xBCF2B69E,
    0x4EB3D2EA,0xD78D4D5C,0x7ED2BFC4,0x58EBF0F3,0x8B591C3F,0xD3041F6B,0x005CAE88,0xBA696F5C,
    0xC16C8EDE,0x9ABCBB27,0x56D78D77,0x765B3E20,0xCF37212D,0x192E2DCF,0x8CAF2806,0xBC9A575B,
    0x776421CE,0x527FB9EB,0x69F84340,0xADBC7BD7,0x73F2C329,0x737F8A7F,0xE301D3E4,0x057EBEB2,
    0x5859B858,0x2CC41979,0xEC69A639,0x53B0D523,0x39A2F532,0x8B29E35D,0x44E2CE81,0xCC10A16D,
    0x44D9FF58,0x77102C14,0xFB57817D,0x3CF7C8C8,0x1222868A,0x4173D5D1,0x3529EE32,0x7A9DF58E,
    0x513525AC,0x81954BAC,0xCE53CCF5,0x79168728,0xA2D660F8,0xF30CC9CE,0xF0B89C76,0x089FB3A9,
    0xC919DBA8,0x1F9E4DC3,0xA2594E0C,0x34FFE178,0xB04414FB,0xD31FB33A,0x184D0278,0x2C816A9A,
    0xB993F2F2,0xE4D8601C,0x49E2EEDE,0x9CD50CE1,0xC03E1E77,0xA901869E,0x7579DE50,0x726AC4AB,
    0x38D04840,0xEABE1270,0x8C40812D,0xE84976B7,0x172B04AD,0x756606C4,0x66258491,0xB5A0BEF8,
    0x6BCC5CF3,0xA535AE94,0xC97A87AA,0x9103A8F6,0xCC3B9E5F,0xBB20BE1F,0xFFCFEF97,0x90954F16,
    0x501AE1A6,0x6ED589CD,0x6826B02B,0x565FF263,0x8E8C369B,0x6990BE7A,0x3525B840,0x1847D7BB,
    0x355A40C7,0xA3579F10,0xE9EDECAE,0xD0337AB1,0x6355E5BA,0x88975355,0x5EC0F3CF,0xA0D6213D,
    0x75389387,0xE40216F0,0xD980CCE0,0x6C88C67C,0x829D419C,0x3BF6451B,0x11F07BFA,0xC4C1154E,
    0xBD0735EB,0x9CF8DF9D,0xE457BE75,0x63A6BD18,0xEFE77FD3,0x83421B63,0x7F83072D,0x44940F61,
    0xF8BDCDF7,0x61C802CA,0x0A30F9A8,0x7FF03B37,0xA26CC5A9,0xE10E570D,0x95EA0C16,0xA05E6B02,
    0xC81D5384,0x7785DB05,0x92C84C5F,0x05584617,0x82BCFE8D,0x559EA1DA,0x4FD5CDB0,0x9D871FED,
    0xDD6F5539,0x4ED1EF26,0xFE6813C4,0x1CFA71D5,0xD5613AEA,0x0F1C9B8C,0x2BCAC45D,0x65D00F41,
    0x689BE0D8,0x68B01100,0x635BD280,0x954D5D4B,0x72887F79,0xCE027A75,0xFCF01C66,0x006A1BD3,
    0x199A1C8E,0x87D6EE25,0x938E9F08,0xD8A11D4D,0x2B9A4D81,0xB6F5D2E5,0xD15C325A,0x64EAAFC1,
    0xFD33B61C,0x43C1BD57,0x37B8F048,0x5CBA7CF2,0x72810CD0,0xABFEF454,0xA76384BA,0xD8861440,
```

```
    0x36DE5837,0x0F6A03F1,0x10D48FA1,0x5883EC2F,0xA8C00C9B,0x618FFEA4,0xA05DA206,0xFFB9E97A,
    0x8A376781,0x3156B479,0xE4AF5ECD,0x87D9E06F,0xB4D4D459,0xEB9A7D25,0x59DFFEAA,0xDC8BF553,
    0x6DCE3C3A,0x2162970E,0xE8C9929D,0x6C3A9BF4,0x45DA5392,0x9CEEE7B0,0x3F68D4EB,0xCD29434F,
    0x0E4DF712,0xB1A8C69A,0x1C190F46,0x2B45873C,0x46AFDFC9,0x61E8883F,0x979118C7,0x70F991B1,
    0x1F82604D,0xC18BF48F,0xB327F4FF,0x519A7508,0xFA619B0D,0x268D1490,0x567E37C2,0x25A07691,
    0x424359C0,0x13320C53,0xEFF742FD,0x48B945BA,0xCFA8E711,0x8F5FB519,0x2B7332A5,0x10AA767C]


def fonction_1(a, b, c, d):
  for _ in range(4):
    t1 = ((b + c) ^ d) & 0x00000000ffffffff
    d = (tab1[a & 0xFF] + t1) & 0x00000000ffffffff
    a = d ^ (a >> 8)
    b, c = (b - t1) & 0x00000000ffffffff, b & c
  return a, b, c, d
```

3. Fonction 2

La fonction 2 utilise les fonctions : 3, 4, 5 et 6.

```
def fonction_2(a, b, c, d, e, f, g, h):
  for i in range(15):
    f61, f62, f63, f64 = e, f, g, h
    for j in range(i+1):
      f61, f62, f63, f64 = fonction_6(f61, f62, f63, f64, j)
    t3, t4 = fonction_3(c, d, *fonction_5(a,b))
    c = t3 ^ f63
    d = t4 ^ f64
    t3, t4 = fonction_4(a, b, *fonction_5(c, d))
    a = t3 ^ f61
    b = t4 ^ f62
  return a, b, c, d
```

En l'inversant, j'ai constaté qu'il n'était pas nécessaire d'inverser les fonctions 6 et 5.

```
def fonction_rev_2(a, b, c, d, e, f, g, h):
  for i in range(14, -1, -1):
    f61, f62, f63, f64 = e, f, g, h
    for j in range(i+1):
      f61, f62, f63, f64 = fonction_6(f61, f62, f63, f64, j)
    a, b = fonction_rev_4(a ^ f61, b ^ f62, *fonction_5(c, d))
    c, d = fonction_rev_3(c ^ f63, d ^ f64, *fonction_5(a, b))
  return a, b, c, d
```

4. Fonctions 3 et 4

Les fonctions 3 et 4 ne font que des rotations et des XOR.

```
def rot(a, d):
    return ((a << d) | (a >> (0x20 - d))) & 0x00000000ffffffff


def fonction_3 (b, d, f, h):
    s2 = h ^ rot(b ^ f, 0xe)
    s4 = f ^ rot(d ^ h, 0x4)
    return s2, s4


def fonction_4 (b, d, f, h):
    s2 = rot(b ^ h ^ f, 0x12)
    s4 = rot(d ^ f, 0x1a)
    return s2, s4
```

Même si elles perdent de l'information, il est possible de les inverser car la fonction 2 nous apporte ces valeurs.

```
def fonction_rev_3 (bs, ds, f, h):
    s2 = f ^ rot(bs ^ h, 0x12)
    s4 = h ^ rot(ds ^ f, 0x1c)
    return s2, s4


def fonction_rev_4 (bs, ds, f, h):
    s2 = rot(bs, 0xe) ^ f ^ h
    s4 = rot(ds, 0x6) ^ f
    return s2, s4
```

5. Fonction 5

```
tab5 = [0x489dddde,0x067990f1,0x95bf74a9,0x77941ee7,
        0x0e6d80e3,0x2dedaf8b,0xfb92cd42,0xd0e867c0,
        0xf2b3a3fb,0x6c39ce47,0xe74f99e0,0x5a24f221]


def fonction_5 (b, d):
    for i in range(6):
        s2 = ((tab5[i*2] + b) & 0x00000000ffffffff) ^ d
        s4 = (tab5[1+i*2] | s2) ^ b
        b = s4
        d = s2
    s2 = b
    s4 = d
    return s2, s4
```

6. Fonction 6

La fonction 6 appelle la fonction 7 deux fois. Mais avec des constantes en entrée. J'ai donc remplacé les appels de la fonction par la valeur de retour.

```
tab6 = [0xd6378fea,0xe23ca8c4,0x84e3b1bc,0xce5e10bf,0xa2b364da,
    0x41f250f0,0x0fe97040,0x1cc05266,0x16f87e4b,0x515e26b7,
    0xeea48dcb,0x62b357e4,0x39bd2041,0x72cd387a,0xf37aac8b]


def fonction_6 (a, b, c, d, i):
  tmp1 = d ^ tab6[i]
  tmp2 = (c + 0x45786532) & 0x00000000ffffffff
  s2 = tmp2 ^ b
  if s2 & 0x80000000:
    tmp4 = 0x60bf080f
  else:
    tmp4 = 0x818f694a
  s1 = rot(a, 0x4)
  s3 = (tmp2 ^ tmp4) ^ s1
  s4 = (tmp1 - tmp2) & 0x00000000ffffffff
  return s1, s2, s3, s4
```

7. Résultat

En appliquant la valeur comparée dans l'exécution du *dwarf* à ma fonction inverse, je trouve le résultat suivant :

```
53535449437b44773472665f564d5f31735f636f306c5f69736e5f7
```

Qui se traduit en ASCII par :

```
SSTIC{Dw4rf_VM_1s_co0l_isn_t_It}
```

5. Analyse des secure OS

Après un reboot, j'ai accès au fichier suivant : « /root/safe_02/decrypted_file ».

1. decrypted_file

Ce programme teste si la longueur de l'entrée est de 64 caractères au format hexadécimal.

Il ouvre le fichier « /dev/sstic » pour faire appel à 4 fonctions :

- Fonction 0xC0150300 : il envoie une table de longueur 0x101010,
- Fonction 0xC0150301 : il envoie le tag.

Dans une boucle *while* :

- Fonction 0xC0150302 : pas de paramètre,

- Fonction 0xC0150303 : retourne une valeur indiquant si le tag est bon ou non.

Ces fonctions se trouvent dans le fichier *sstic.ko* que j'avais identifié en observant le fonctionnement du programme add_key.py.

2. sstic.ko

La fonction sstic_ioctl dispatch les entrées 0xC015030x vers le boot loader 31 avec des « System Monitor Call » (SCM). La correspondance est la suivante :

- 0xc0105300 -> 0x83010004
- 0xC0150301 -> 0xf2005003
- 0xC0150302 -> 0xf2005001
- 0xC0150303 -> 0xf2005002

En se référençant à la convention de nommage des SMC, on peut poser l'hypothèse que la fonction 0 est traitée par le boot loader 31 et les autres par le boot loader 32.

Je me suis donc intéressé au boot loader.

3. Boot loader 1 et 2

J'ai passé Qemu en mode gdb server et désassemblé le fichier « rom.bin ». En me plaçant à l'adresse 0x00002c64, j'ai pu récupérer l'adresse et le binaire du Boot loader 2. De même à l'adresse 0x0e02207c, j'ai récupéré les adresses et les binaires des Boot loader 31, 32, 33 (ce dernier n'était pas utile).

- Le boot loader 31 est à l'adresse 0x0e030000,
- Le boot loader 32 est à l'adresse 0x0e200000.

J'ai analysé ces binaires pour y trouver les dispatcher.

4. Boot loader 31 et 32

J'ai commencé par regarder le dispatcher du boot loader 32 (adresse 0x0e200c08). J'ai constaté que la sortie de 0xf2005001 est l'entrée de 0xf2005002. J'ai donc affiché sous gdb cette valeur pour me faire une trace d'exécution.

J'ai obtenu les statistiques d'utilisation suivante :

Opération	Répétition	Opération	Répétition	Opération	Répétition	Opération	Répétition
0x0C	98	**0xdc**	4	**0xcc**	128	**0x8c**	32
0x4c	1179	**0x10**	676	**0x60**	192	**0xa4**	1
0x2c	726	**0x00**	1861	**0xec**	128		
0x04	592	**0x7c**	960	**0x08**	8		
0x5c	272	**0x20**	1160	**0x3c**	1		
0xb0	32	**0x9c**	836	**0x30**	32		

Je me suis donc intéressé à seulement ces cas-là dans la sous-fonction de 0xf2005002 (à l'adresse 0x0e2005a4). Ces opérations se répartissent en celles qui font appel au boot loader 31 et celles qui utilisent les exceptions el1.

Je me suis concentré en premier sur les fonctions du boot loader 31. Elles sont traitées à l'adresse 0x0e31034.

J'y retrouve la fonction 0x83010004 qui stocke les datas à l'adresse 0x0e053000. Cette adresse physique est également mappée virtuellement à l'adresse 0x00413000.

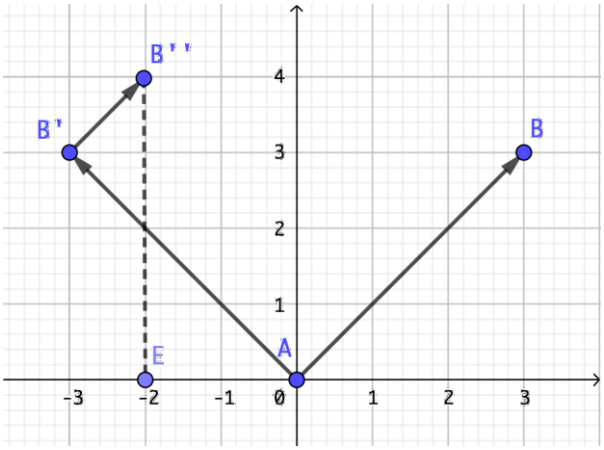
Les fonctions 0x83010001 et 0x83010002 m'ont surpris par l'utilisation d'un seul dernier tour AES (respectivement chiffrement et déchiffrement) avec la moitié de mon flag. J'ai fini par comprendre qu'elles décrivent des fonctions d'accès à 15 registres (respectivement en lecture et écriture), le 15ème étant le PC de la VC.

L'analyse des autres fonctions du Boot loader 31 m'a permis de créer un désassembleur (Annexe D) que j'ai appliqué sur mes traces.

Avec ces traces, j'ai déduit le fonctionnement des opérations suivantes : 0x0c, 0x4c, 0x5c, 0x9c, 0x00 et 0xb0 (qui utilise les exceptions el1).

L'opération 0xcc ne fait qu'écrire des constantes à des adresses fixes.

Il est intéressant de noter la façon de faire une soustraction en faisant une addition avec rotation dans l'ensemble des complexes :



Les 2 seules opérations qui me manquaient étaient 0xdc et 0xec. Je les ai ignorées dans un premier temps.

En suivant pas à pas la sortie de 0x83010001, j'ai constaté que la fonction fait un *load* à une adresse invalide. Cela déclenche une faute traitée par une fonction *sync_exception_sp_elx* à l'adresse 0x0e203200. Dans les cas de *load* ou de *store*, l'adresse est virtualisée par une fonction qui utilise un des instructions SM4. J'ai donc utilisé gdb pour dumper la mémoire avec un petit script :

```
b *0x0e200f40

c

set $x5 = 0

while 1

x/16x $x20

set $x21 = $x5

set $x5 = $x5 + 0x10

jump *0x0e200ed0

end
```

J'ai constaté qu'il y a 2 parties à ce fichier :

- Mes opérations sur 3 bytes en LSB,
- Un tableau de taille 0x100000.

J'ai désassemblé le binaire et j'ai réécrit le programme en python (Annexe E). Je l'ai ensuite inversé.

Je me suis confronté aux 2 opérations que je ne connaissais pas (0xdc et 0xec). 0xec semblait être un jump conditionné par un registre donné en entrée. Mais si le programme saute, il devient impossible d'inverser mes traces.

Pour information, mes traces ne sautent pas les 2 première fois mais sautent le reste du temps (sauf cas aléatoire). J'ai donc supposé que 0xdc et 0xec ne devraient rien faire en cas normal (sans debug ou strace).

Je me suis donc intéressé aux fonctions qui utilisent les exceptions el1. Dans un premier temps, je n'arrivais pas à les désassembler en aarch64. Mais en interprétant comme du arm et arm thrump mode, j'ai pu les traduire. Sauf les 2 cas suivants : 0xef001338 et 0xef1337 qui sont interprétés par la fonction sync_exception_aarch32 (adresse 0xe0203600) et qui respectivement font appel à SMC et sortent du contexte d'exception el1.

J'ai confirmé les fonctionnements de toutes les opérations. J'ai découvert que 0xdc ne fait qu'écrire à l'adresse 0 à 0x9010008, que 0xec ne fait que lire à 0x9010000 et le comparer à 5. Il s'agit de l'adresse du pl031 qui est une horloge de 1 seconde. Comme le programme prend 3 secondes pour s'exécuter sans gdb, j'ai confirmé qu'il s'agit bien d'un piège. Je l'ai retiré grâce au script gdb suivant :

```
set *0x0e2050e8=0xe1a00000

set *0x0e2050ec=0xe2800003
```

J'ai inversé le code d'exécution (Annexe F). Je me suis trouvé confronté à 2 problèmes :

- Il y avait plusieurs solutions pour mes cas de test (j'ai donc utilisé un parcours de graphe),
- La valeur attendue ne donnait pas de solution.

Je me suis rappelé que dans la fonction de map de ma table, il y avait un comportement étrange : sous certaines conditions mon adresse physique était différente. Cela ne m'avait pas étonné que qemu recherche la manière dont il est lancé (avec l'option s ou gdb). J'ai donc utilisé gdb pour dumper les datas en forçant le mapping au démarrage :

```
b *0x0e2017cc

c

set $x0 =  0xe054000
```

Avec cette nouvelle table, j'ai pu inverser la valeur attendue :

acadaa8b5b55306fb3c6dfc3b2d1c80770084644225febd71a9189aa26ec740e

Que j'ai appliqué dans *add_key* :

SSTIC{acadaa8b5b55306fb3c6dfc3b2d1c80770084644225febd71a9189aa26ec740e}

6. Analyse des fichiers du téléphone

J'ai constaté que le dernier fichier « /root/safe_03/decrypted_file » est un tar.gz.

J'ai donc ouvert l'archive et extrait la base de données SQL des messages. J'y ai trouvé le message suivant :

> *Mission accomplie, comme d'habitude la perspective d'une tournée de shooter au cactus a suffi à corrompre le CO et à choisir la date de publication du challenge. Pour être sûr que les experts sont toujours occupés, j'ai redirigé l'adresse* ***9e915a63d3c4d57eb3da968570d69e95@challenge.sstic.org*** *vers votre boîte mail. Tant que vous ne voyez passer aucun mail, la voie est libre...*

7. Annexes

Annexe A

Source code Python 3 qui analyse le fichier 'power_consumption.npz' :

```
from numpy import load

data = load('power_consumption.npz')

data = data.f.arr_0[700000:700000+789000]

data = [x < -0.15 for x in data]


data2 = ''.join("1"*x or "0" for x in data)


key = ""

pos = back_pos = data2.find("1") + 1

next_1 = 0

find_0 = False

while (next_1 != -1) and (pos < len(data2)):

        next_1 = data2[pos:].find("1")

        pos, back_pos = next_1 + pos + 1, pos

        diff = pos - back_pos

        if diff < 50:

                continue

        elif diff < 150:

                find_0 = True

        elif diff < 250:

                find_0 = False
```

```
                        key += "1"
            elif find_0:
                        find_0 = False
                        key += "0"
key += find_0 * "0" or ""
print(key)
print(key.replace("0", "-").replace("1", "+-"))
print(f"{int(key[::-1],2):0256x}")
```

Annexe B

Source code Python 3 de la fonction *secure_device* :

```
def secure_device(a,b,op):
    out = [False for _ in range(8)]
    a = [x=="1" for x in f"{a:08b}"]
    b = [x=="1" for x in f"{b:08b}"]
    op = [x=="1" for x in f"{op:02b}"]
    B = [x=="1" for x in f"{boutons:04b}"]
    B = B[::-1]
    if B[2]:
        t = a.pop(0)
        a.append(t)
    a = a[::-1]
    if B[3]:
        t = b.pop(0)
        b.append(t)
    b = b[::-1]
    op = op[::-1]
    op[0] ^= B[0]
    op[1] ^= B[1]
    if op[1]:
        out[0] = a[0] ^ b[0]
    else:
        if op[0]:
            out[0] = a[0] or b[0]
        else:
            out[0] = a[0] and b[0]
    r = a[0] and b[0]

    for i in range(1,8):
        if op[1]:
            if op[0]:
                out[i] = r ^ (a[i] ^ b[i])
            else:
                out[i] = a[i] ^ b[i]
        else:
            if op[0]:
```

```
                                        out[i] = a[i] or b[i]
                                else:
                                        out[i] = a[i] and b[i]
                        r = (r and (a[i] ^ b[i])) or (a[i] and b[i])
                out = out[::-1]
                out = int(''.join("1"*o or "0" for o in out),2)
                return out
```

Annexe C

Source code Python 3 du simulateur de Dwarf :

```
class DwarfData:
  def __init__(self, buffer, little=False):
    self.baseAddress = 0x00400228
    self.offset = 0
    self.buffer = buffer
    self.maxAddress = self.baseAddress + len(self.buffer) - 1
    self.little = little
    self.size = 64

  def getAt(self, address, nbByte, little = None):
    address -= self.baseAddress
    if little == None:
      little = self.little
    tmp = 0
    for i in range(nbByte):
      if little:
        tmp += self.buffer[address + i] << (8*i)
      else:
        tmp = (tmp << 8) + self.buffer[address + i]
    return  tmp

  def read(self, nbByte, little = None, signed = False):
    tmp = self.getAt(self.baseAddress + self.offset, nbByte, little=little)
    self.offset += nbByte
    if signed and (tmp & (1 << (8*nbByte-1))):
      tmp = -(1 << (8*nbByte)) + tmp

    return tmp

  def readLEB128(self, signed=False):
    result = 0
    shift = 0
    neg = False
    while True:
      byte = self.read(1)
```

```
            result |= (byte & 0x7F) << shift
            shift += 7
            neg = (byte & 0x40) != 0
            if (byte & 0x80) == 0:
                break
        if signed and (shift < self.size) and neg:
            result |= (~0 << shift)
        return result

    def getAddress(self):
        return self.baseAddress + self.offset

    def setAddress(self, address):
        self.offset = address - self.baseAddress

    def isRunnable(self):
        return (self.offset >= 0) and (self.offset <= len(self.buffer))

class Dwarf:
    def __init__(self, data):
        self.dispatch_table = {}
        self.data = DwarfData(data, little=True)
        self.stack = []
        self.externAddress = {}
        self.__init_OP()
        self.log = {}
        self.end = None
        self.exit = False
        self.debug = False

    def interpret(self):
        def default_visitor():
            print(f"op at {self.data.getAddress} not implemented")
            exit(1)

        while self.data.isRunnable():
            address = self.data.getAddress()
            cur_opcode = self.data.read(1)
            cur_opcode_name = self.DW_OP_opcode2name.get(cur_opcode, 'OP:0x%x' % cur_opcode)
```

```
        if (self.end != None) and (address ==self.end):
            self.printStack("")
            if self.exit:
                exit(0)
            break

        # Dispatch to a visitor function
        message = self.dispatch_table.get(cur_opcode,default_visitor)()

        # if (address >= 0x004003a9) and (address <= 0x00400477): # fonction 2
        # if (address >= 0x004002ce) and (address <= 0x00400308): # fonction main
        if self.debug:
            self.printStack(f"{address:08x}: {cur_opcode:02x} {cur_opcode_name:25s} {message:s}")
            input()

        if f"{address:08x}" not in self.log:
            self.log[f"{address:08x}"] = (0, f"{address:08x}: {cur_opcode:02x} {cur_opcode_name:25s} {message:s}")
        c, m = self.log[f"{address:08x}"]
        self.log[f"{address:08x}"] = (c+1, m)

    for k in sorted(list(self.log)):
        c, m = self.log[k]
        print(f"{c:4d} {m}")

def printStack(self, message):
    print("-"*80)
    if len(self.stack) == 0:
        print(message)
    else:
        m = message
        for s in self.stack[::-1]:
            print(f"{m:63s} {s:016x}")
            m = ""

def __init_OP(self):
    self.DW_OP_name2opcode = dict(
        DW_OP_addr=0x03,
        DW_OP_deref=0x06,
```

```
DW_OP_const1u=0x08,
DW_OP_const1s=0x09,
DW_OP_const2u=0x0a,
DW_OP_const2s=0x0b,
DW_OP_const4u=0x0c,
DW_OP_const4s=0x0d,
DW_OP_const8u=0x0e,
DW_OP_const8s=0x0f,
DW_OP_constu=0x10,
DW_OP_consts=0x11,
DW_OP_dup=0x12,
DW_OP_drop=0x13,
DW_OP_over=0x14,
DW_OP_pick=0x15,
DW_OP_swap=0x16,
DW_OP_rot=0x17,
DW_OP_xderef=0x18,
DW_OP_abs=0x19,
DW_OP_and=0x1a,
DW_OP_div=0x1b,
DW_OP_minus=0x1c,
DW_OP_mod=0x1d,
DW_OP_mul=0x1e,
DW_OP_neg=0x1f,
DW_OP_not=0x20,
DW_OP_or=0x21,
DW_OP_plus=0x22,
DW_OP_plus_uconst=0x23,
DW_OP_shl=0x24,
DW_OP_shr=0x25,
DW_OP_shra=0x26,
DW_OP_xor=0x27,
DW_OP_bra=0x28,
DW_OP_eq=0x29,
DW_OP_ge=0x2a,
DW_OP_gt=0x2b,
DW_OP_le=0x2c,
DW_OP_lt=0x2d,
DW_OP_ne=0x2e,
```

```
    DW_OP_skip=0x2f,
    DW_OP_regx=0x90,
    DW_OP_fbreg=0x91,
    DW_OP_bregx=0x92,
    DW_OP_piece=0x93,
    DW_OP_deref_size=0x94,
    DW_OP_xderef_size=0x95,
    DW_OP_nop=0x96,
    DW_OP_push_object_address=0x97,
    DW_OP_call2=0x98,
    DW_OP_call4=0x99,
    DW_OP_call_ref=0x9a,
    DW_OP_form_tls_address=0x9b,
    DW_OP_call_frame_cfa=0x9c,
    DW_OP_bit_piece=0x9d,
)

def _generate_dynamic_values(map, prefix, index_start, index_end, value_start):
    for index in range(index_start, index_end + 1):
        name = '%s%s' % (prefix, index)
        value = value_start + index - index_start
        map[name] = value

_generate_dynamic_values(self.DW_OP_name2opcode, 'DW_OP_lit', 0, 31, 0x30)
_generate_dynamic_values(self.DW_OP_name2opcode, 'DW_OP_reg', 0, 31, 0x50)
_generate_dynamic_values(self.DW_OP_name2opcode, 'DW_OP_breg', 0, 31, 0x70)

self.DW_OP_opcode2name = dict((v, k) for k, v in self.DW_OP_name2opcode.items())

def add(opcode_name, func):
    self.dispatch_table[self.DW_OP_name2opcode[opcode_name]] = func

add('DW_OP_addr', self.visit_OP_addr)

add('DW_OP_const1u', lambda:self.visit_OP_const(1,False))
add('DW_OP_const1s', lambda:self.visit_OP_const(1,True))
add('DW_OP_const2u', lambda:self.visit_OP_const(2,False))
add('DW_OP_const2s', lambda:self.visit_OP_const(2,True))
add('DW_OP_const4u', lambda:self.visit_OP_const(4,False))
```

```
add('DW_OP_const4s', lambda:self.visit_OP_const(4,True))
add('DW_OP_const8u', lambda:self.visit_OP_const(8,False))
add('DW_OP_const8s', lambda:self.visit_OP_const(8,True))
add('DW_OP_constu',  lambda:self.visit_OP_const(-1,False))
add('DW_OP_consts',  lambda:self.visit_OP_const(-1,True))

add('DW_OP_lit0',  lambda:self.visit_OP_lit(0))
add('DW_OP_lit1',  lambda:self.visit_OP_lit(1))
add('DW_OP_lit2',  lambda:self.visit_OP_lit(2))
add('DW_OP_lit3',  lambda:self.visit_OP_lit(3))
add('DW_OP_lit4',  lambda:self.visit_OP_lit(4))
add('DW_OP_lit5',  lambda:self.visit_OP_lit(5))
add('DW_OP_lit6',  lambda:self.visit_OP_lit(6))
add('DW_OP_lit7',  lambda:self.visit_OP_lit(7))
add('DW_OP_lit8',  lambda:self.visit_OP_lit(8))
add('DW_OP_lit9',  lambda:self.visit_OP_lit(9))
add('DW_OP_lit10',  lambda:self.visit_OP_lit(10))
add('DW_OP_lit11',  lambda:self.visit_OP_lit(11))
add('DW_OP_lit12',  lambda:self.visit_OP_lit(12))
add('DW_OP_lit13',  lambda:self.visit_OP_lit(13))
add('DW_OP_lit14',  lambda:self.visit_OP_lit(14))
add('DW_OP_lit15',  lambda:self.visit_OP_lit(15))
add('DW_OP_lit16',  lambda:self.visit_OP_lit(16))
add('DW_OP_lit17',  lambda:self.visit_OP_lit(17))
add('DW_OP_lit18',  lambda:self.visit_OP_lit(18))
add('DW_OP_lit19',  lambda:self.visit_OP_lit(19))
add('DW_OP_lit20',  lambda:self.visit_OP_lit(20))
add('DW_OP_lit21',  lambda:self.visit_OP_lit(21))
add('DW_OP_lit22',  lambda:self.visit_OP_lit(22))
add('DW_OP_lit23',  lambda:self.visit_OP_lit(23))
add('DW_OP_lit24',  lambda:self.visit_OP_lit(24))
add('DW_OP_lit25',  lambda:self.visit_OP_lit(25))
add('DW_OP_lit26',  lambda:self.visit_OP_lit(26))
add('DW_OP_lit27',  lambda:self.visit_OP_lit(27))
add('DW_OP_lit28',  lambda:self.visit_OP_lit(28))
add('DW_OP_lit29',  lambda:self.visit_OP_lit(29))
add('DW_OP_lit30',  lambda:self.visit_OP_lit(30))
add('DW_OP_lit31',  lambda:self.visit_OP_lit(31))
```

```
    add('DW_OP_plus', lambda:self.visit_OP_operationDouble("+"))
    add('DW_OP_minus', lambda:self.visit_OP_operationDouble("-"))
    add('DW_OP_mul', lambda:self.visit_OP_operationDouble("*"))
    add('DW_OP_div', lambda:self.visit_OP_operationDouble("/"))
    add('DW_OP_shl', lambda:self.visit_OP_operationDouble("<<"))
    add('DW_OP_shr', lambda:self.visit_OP_operationDouble(">>"))
    add('DW_OP_and', lambda:self.visit_OP_operationDouble("&"))
    add('DW_OP_xor', lambda:self.visit_OP_operationDouble("^"))
    add('DW_OP_or', lambda:self.visit_OP_operationDouble("|"))

    add('DW_OP_deref', self.visit_OP_deref)
    add('DW_OP_deref_size', self.visit_OP_deref_size)
    add('DW_OP_dup', self.visit_OP_dup)
    add('DW_OP_swap', self.visit_OP_swap)
    add('DW_OP_bra', self.visit_OP_bra)
    add('DW_OP_skip', self.visit_OP_skip)
    add('DW_OP_pick', self.visit_OP_pick)
    add('DW_OP_rot', self.visit_OP_rot)
    add('DW_OP_drop', self.visit_OP_drop)

    add('DW_OP_nop', self.visit_OP_nop)

  def visit_OP_nop(self):
    return ""

  def visit_OP_lit(self, val):
    self.stack.append(val)
    return ""

  def visit_OP_rot(self):
    t1 = self.stack.pop()
    t2 = self.stack.pop()
    t3 = self.stack.pop()
    self.stack.append(t1)
    self.stack.append(t3)
    self.stack.append(t2)
    return ""

  def visit_OP_drop(self):
```

```
    self.stack.pop()
    return ""

def visit_OP_pick(self):
    offset = self.data.read(1)
    self.stack.append(self.stack[-(offset+1)])
    return f"{offset:02x}"

def visit_OP_skip(self):
    offset = self.data.read(2, signed=True)
    self.data.setAddress(self.data.getAddress() + offset)
    return f"{offset:04x}"

def visit_OP_bra(self):
    test = self.stack.pop()
    if test != 0:
        return self.visit_OP_skip()
    else:
        return f"{self.data.read(2, signed=True):04x}"

def visit_OP_dup(self):
    t = self.stack.pop()
    self.stack.append(t)
    self.stack.append(t)
    return ""

def visit_OP_swap(self):
    t1 = self.stack.pop()
    t2 = self.stack.pop()
    self.stack.append(t1)
    self.stack.append(t2)
    return ""

def visit_OP_deref_size(self):
    address = self.stack[-1]

    m = self.visit_OP_deref(log=False)
    val = self.stack.pop()
    size = self.data.read(1)
```

```
    self.stack.append(val & ((1 << (size*8))-1))

    if f"{address:08x}" not in self.log:
        self.log[f"{address:08x}"] = (0, f"{address:08x}:    DATA  0x{m[-(2*size):]}")
    c, m = self.log[f"{address:08x}"]
    self.log[f"{address:08x}"] = (c+1, m)

    return f"{size:02x} < 0x{m[-(2*size):]}"

def visit_OP_deref(self, log=True):
    address = self.stack.pop()
    tmp = 0
    if f"0x{address:016x}" in self.externAddress:
        tmp = self.externAddress[f"0x{address:016x}"]
    elif (address >= self.data.baseAddress) and (address <= self.data.maxAddress):
        tmp = self.data.getAt(address,self.data.size//8)
    else:
        print(address, "not found")
        exit(1)
    self.stack.append(tmp)

    if log:
        if f"{address:08x}" not in self.log:
            self.log[f"{address:08x}"] = (0, f"{address:08x}:    DATA  0x{tmp:016x}")
        c, m = self.log[f"{address:08x}"]
        self.log[f"{address:08x}"] = (c+1, m)

    return f"< 0x{tmp:016x}"

def visit_OP_addr(self):
    address = self.data.read(self.data.size//8)
    self.stack.append(address)
    return f"{address:016x}"

def visit_OP_const(self, nbByte, signed):
    d = 0
    if nbByte == -1:
        d = self.data.readLEB128(signed=signed)
    else:
```

```
            d = self.data.read(nbByte, signed=signed)
        self.stack.append(d)
        return f"{d:016x}"

    def visit_OP_operationDouble(self, operation):
        d1 = self.stack.pop()
        d2 = self.stack.pop()
        d = 0
        if operation == "+":
            d = (d1 + d2) & ((1 << self.data.size)-1)
        elif operation == "-":
            d = (d2 - d1) & ((1 << self.data.size)-1)
        elif operation == "*":
            d = (d1 * d2) & ((1 << self.data.size)-1)
        elif operation == "/":
            d = (d2 / d1) & ((1 << self.data.size)-1)
        elif operation == "&":
            d = d1 & d2
        elif operation == "|":
            d = d1 | d2
        elif operation == ">>":
            d = d2 >> d1
        elif operation == "<<":
            d = (d2 << d1) & ((1 << self.data.size)-1)
        elif operation == "^":
            d = d1 ^ d2
        self.stack.append(d)
        return ""

with open("decrypted_file.gnu.hash", mode='rb') as file:
    dw = Dwarf(file.read())
    dw.exit = True

    dw.data.setAddress(0x00400261)

    dw.stack.append(0x0000ffffffffffb90)
    dw.stack.append(0x0000ffffffffff10)
    dw.externAddress["0x0000ffffffffff10"] = 0x77447b4349545353
    dw.externAddress["0x0000ffffffffff18"] = 0x315f4d565f667234
```

```
dw.externAddress["0x0000ffffffffff20"] = 0x695f6c306f635f73

dw.externAddress["0x0000ffffffffff28"] = 0x7d74495f745f6e73

dw.externAddress["0x0000ffffffffff30"] = 0x6f723d5245535500

dw.end = 0x004002b3


# main

# dw.data.setAddress(0x004002ce)

# dw.stack.append(0x3333333334343434)

# dw.stack.append(0x3131313132323232)

# dw.stack.append(0x3737373738383838)

# dw.stack.append(0x3535353536363636)

# dw.stack = dw.stack[::-1]

# dw.end = 0x00400308


# fonction 1

# dw.data.setAddress(0x0040030b)

# dw.stack.append(0x3737373738383838)

# dw.stack.append(0x3535353536363636)

# dw.stack.append(0x0000000000000000)

# dw.stack = dw.stack[::-1]

# dw.end = 0x004003a0

# dw.debug = True


# # fonction 2

# dw.data.setAddress(0x004003a9)

# dw.stack.append(0xaaaaaaaabbbbbbbb)

# dw.stack.append(0xcccccccccdddddddd)

# dw.stack.append(0xeeeeeeeeffffffff)

# dw.stack.append(0x1111111122222222)

# dw.stack.append(0x0000000000000000)

# dw.stack = dw.stack[::-1]

# dw.end = 0x00400477


# fonction 3

# dw.data.setAddress(0x00400480)

# dw.stack.append(0x0000000011213141)

# dw.stack.append(0x0000000014243444)

# dw.stack.append(0x0000000016263646)

# dw.stack.append(0x0000000018283848)
```

```
# dw.stack = dw.stack[::-1]
# dw.end = 0x004004b2
# dw.debug = True

# fonction 4
# dw.data.setAddress(0x004004b5)
# dw.stack.append(0x00000000abcdef01)
# dw.stack.append(0x0000000053895028)
# dw.stack.append(0x000000001539abd3)
# dw.stack.append(0x00000000fa4599dc)
# dw.stack = dw.stack[::-1]
# dw.end = 0x004004e4
# dw.debug = True

# fonction 5
# dw.data.setAddress(0x004004e7)
# dw.stack.append(0x0000000033333333)
# dw.stack.append(0x0000000034343434)
# dw.stack.append(0x0000000000000000)
# dw.stack = dw.stack[::-1]
# dw.end = 0x00400553
# dw.debug = True

        # fonction 6
# dw.data.setAddress(0x0040055c)
# dw.stack.append(0x0000000000000005)
# dw.stack.append(0xA5A5A5A55A5A5A5A)
# dw.stack.append(0x6969696996969696)
# dw.stack = dw.stack[::-1]
# dw.end = 0x0040061f

        # fonction 7 32bit => 32bit
# dw.data.setAddress(0x00400ab5)
# dw.end = 0x00401980
# dw.stack.append(0x0000000084653217)
# dw.stack = dw.stack[::-1]
# 0000000017246549 => 00000000818f694a
# 0000000084653217 => 0000000060bf080f
dw.interpret()
```

Annexe D

Source code Python 3 du désassembleur de la VM du secure OS :

```
import io


ASM = {
    0xa4:(lambda x,y,z:f"return x0"),
    0x04:(lambda x,y,z:f"ldr x{x}, [x{y}]"),
    0x08:(lambda x,y,z:f"str x{y}, [x{x}]"),
    0x0c:(lambda x,y,z:f"str x{x}, #{z:x}"),       # 0x0e205230
    0x2c:(lambda x,y,z:f"add x{x}, x{x}, #{z:x}"),
    0x3c:(lambda x,y,z:f"sub x{x}, x{x}, #{z:x}"),
    0x4c:(lambda x,y,z:f"lsl x{x}, x{x}, #{z:x}"),  # 0x0e205108
    0x5c:(lambda x,y,z:f"lsr x{x}, x{x}, #{z:x}"),  # 0x0e205150
    0x7c:(lambda x,y,z:f"and x{x}, x{x}, #{z:x}"),
    0x8c:(lambda x,y,z:f"b   #{z:x}"),             # 83010028
    0x9c:(lambda x,y,z:f"bn  x{x}, #{z:x}"),       # 0x0e205000
    0xcc:(lambda x,y,z:f"store_key x{x}"),
    0xdc:(lambda x,y,z:f"SET PL031 to 0"),         # 0x0e205250 // set address 0x9010000 to 0
    0xec:(lambda x,y,z:f"TEST if PL031 > #{z:x}"),  # 0x0e2050c0 // test address 0x9010000 > 5 => jump
    0x00:(lambda x,y,z:f"mov x{x}, x{y}"),         # 0x0e205198
    0x10:(lambda x,y,z:f"sub x{x}, x{x}, #1"),     # 83010011
    0x20:(lambda x,y,z:f"add x{x}, x{x}, x{y}"),
    0x30:(lambda x,y,z:f"sub x{x}, x{x}, x{y}"),
    0x60:(lambda x,y,z:f"eor x{x}, x{x}, x{y}"),
    0xb0:(lambda x,y,z:f"rev x{x}, x{x}"),         # 0x0e2051d8
}


def print_trace(data_file):
        with open(data_file) as file:
                instDic = {}
                for l in file.readlines():
                        x = int(l, base=16)
                        inst = ((x >> 0x12) & 0XFF) << 0x2
                        P1 = (x >> 0xe) & 0xF
                        P2 = (x >> 0xa) & 0xF
                        if inst in ASM:
                                print(ASM[inst](P1,P2,x&0x3FFF))
```

```
                    else:
                        print(f"{inst:02x}: {P1:x} {P2:x} {x&0x3FFF:x}")


def print_code(data_file):
    with open(data_file) as file:
        data = file.readline()
        for p in range(0, len(data), 6):
            x = int(data[4+p:4+p+2]+data[2+p:2+p+2]+data[p:p+2], base=16)
            if  x == 0:
                break
            inst = ((x >> 0x12) & 0XFF) << 0x2
            P1 = (x >> 0xe) & 0xF
            P2 = (x >> 0xa) & 0xF
            if inst in ASM:
                print(f"{p//2:03x}: " + ASM[inst](P1,P2,x&0x3FFF))
            else:
                print(f"{inst:02x}: {P1:x} {P2:x} {x&0x3FFF:x}")
```

Annexe E

Source code Python 3 de la fonction exécutée dans le sécure OS:

```
import io


file = open("data.txt","r")

buffer = file.read()

file.close()

data = [int(buffer[i:i+2], base=16) for i in range(0,len(buffer),2)]


def sbox(x7, x1):

    x4 = (x1 >> 8) & 0xFF

    x5 = (x1     ) & 0xFF

    x6 = data[(x7 << 16) + (x5 << 8) + x4 + 0x1000]

    print(f"{(x7 << 16) + (x5 << 8) + x4:06x} = {x6:x}")

    x7 = (x7 - 1) % 10

    x5 = data[(x7 << 16) + (x4 << 8) + x6 + 0x1000]

    print(f"{(x7 << 16) + (x4 << 8) + x6:06x} = {x5:x}")

    x7 = (x7 - 1) % 10

    x4 = data[(x7 << 16) + (x6 << 8) + x5 + 0x1000]

    print(f"{(x7 << 16) + (x6 << 8) + x5:06x} = {x4:x}")

    x7 = (x7 - 1) % 10

    x6 = data[(x7 << 16) + (x5 << 8) + x4 + 0x1000]

    print(f"{(x7 << 16) + (x5 << 8) + x4:06x} = {x6:x}")

    x7 = (x7 - 1) % 10

    return x7, (x6 << 8) + x4


def do_block(x0, x1, x2, x3):

    x7 = 7

    for x14 in range(0x1f, -1, -1):

        x7, x9 = sbox(x7, x1)

        if x14 & 8:

            x8 = x0

            x0 = x9

            x1 = (x14 + 1) ^ x0 ^ x2

                                    x2 = x3

                                    x3 = x8

        else:
```

```
        x8 = x3
        x3 = (x14 + 1) ^ x0
                                    x3 = x3 ^ x1
                                    x0 = x9
                                    x1 = x2
                                    x2 = x8
    input()
  return x0, x1, x2, x3

a,b,c,d = do_block(0xffff,0xffff,0xffff,0xffff)
print(f"0x{a:04x} 0x{b:04x} 0X{c:04x} 0x{d:04x}")
print("0x26ee 0x05ba 0x15b2 0x79b1") # attention c'est la mauvaise table ;)
```

Annexe F

Source code Python 3 de la fonction inverse de la fonction du sécure OS :

```
import io
file = open("data_OK.txt","r")
buffer = file.read()
file.close()
data = [int(buffer[i:i+2], base=16) for i in range(0,len(buffer),2)]


def seek(x7,x6,x4):
    r = []
    for i in range(0x100):
        if x6 == data[(x7 << 16) + (i << 8) + x4 + 0x1000]:
            r.append(i)
    return r


def add_x7(x7):
    x7 += 1
    if x7 == 0xa:
        return 0
    else:
        return x7


def sbox_inv(x7, x9):
    x6p = (x9 >> 8) & 0xFF
    x4p = x9 & 0xFF
    x7_1 = add_x7(x7)
    x7_2 = add_x7(x7_1)
    x7_3 = add_x7(x7_2)
    x7_4 = add_x7(x7_3)
    lx5p = seek(x7_1, x6p, x4p)
    lx1 = []
    for x5p in lx5p:
        lx6 = seek(x7_2, x4p, x5p)
        for x6 in lx6:
            lx4 = seek(x7_3, x5p, x6)
            for x4 in lx4:
                for x5 in seek(x7_4, x6, x4):
```

```
            lx1.append((x4 << 8) + x5)
    return x7_4, lx1
def permutation(x7, x14, x0, x1, x2, x3):
    r = []
    x7, x9_cases = sbox_inv(x7, x0)
    for x9 in x9_cases:
        x0b = x1b = x2b = x3b = 0
        if x14 & 8:
            x0b = x3
            x1b = x9
            x2b = (x14 + 1) ^ x0 ^ x1
            x3b = x2
        else:
            x0b = (x14 + 1) ^ x9 ^ x3
            x1b = x9
            x2b = x1
            x3b = x2
        r.append((x0b,x1b,x2b,x3b))
    return x7, r
def inv_block(x0, x1, x2, x3):
    case_list = {(x0, x1, x2, x3)}
    x7 = -1
    for x14 in range(0x20):
        if len(case_list) == 0:
            return []
        case_list_t = []
        for i in case_list:
            x7_p, r = permutation(x7, x14, *i)
            case_list_t.extend(r)
        x7 = x7_p
        case_list = case_list_t
    return case_list

block = [
    (0x7072, 0x2e61, 0x2e72, 0x6667),
    (0x2e63, 0x6e66, 0x2e66, 0x762e),
    (0x736e, 0x7076, 0x7972, 0x4066),
    (0x6667, 0x7670, 0x2e62, 0x6574),
]
```

```
for i in block:
    R = inv_block(*i)
    if len(R) == 0:
        print(f"KO 0x{i[0]:04x} 0x{i[1]:04x} 0x{i[2]:04x} 0x{i[3]:04x}")
        pass
    else:
        a,b,c,d = R[0]
        print(f"OK 0x{i[0]:04x} 0x{i[1]:04x} 0x{i[2]:04x} 0x{i[3]:04x} avec {len(R)} solutions: 0x{a:04x} 0x{b:04x} 0X{c:04x} 0x{d:04x}")
```