Si tu regardes ce fichier depuis GitHub.com, regarde plutôt le fichier : `WRITEUP.pdf`. En effet, le markdown GitHub n'est pas interopérable avec les visionneurs markdown usuels…

BreizhCTF 2025 | Write-up : Demi-tour [Crypto]

Auteur : skilo

Quelques notations

- $[\beta]$: l'ensemble $\{-\beta, -\beta+1, \dots, 0, \dots, \beta-1, \beta\}$.
- E^k : l'ensemble des vecteurs de taille k à coefficients dans E.
- $E^{k\times l}$: l'ensemble des matrices de taille $k \times l$ à coefficients dans E.
- $\mathbb{Z}_n$: l'anneau des entiers modulo n, aussi noté $\mathbb{Z}/n\mathbb{Z}$.
- Si $\mathbf{x}$ est un vecteur de taille n, alors $x_1, x_2, \dots, x_n$ sont ses coordonnées.

Paramétrage

Paramètre	Valeur
k	48
l	50
β	2
n	256

Objectif

L'objectif de ce challenge est de réussir à pré-image la fonction `hash()`.

```
def hash(msg: str):
    msg = msg.encode()
    msg = sanitize(msg, l)

    x = vector(Zn, msg)
    A = Matrix(Zn, k, l, get_a())

    return bytes(A*x)
```

Analysons, *petit pas par petit pas*, cette fonction.

`sanitize`

```
def sanitize(a: bytes, l: int):
    a = pad(a, BLOCK_LEN)
    out = b"\x00"*BLOCK_LEN

    for i in range(0, len(a), BLOCK_LEN):
        out = xor(out, a[i:i+BLOCK_LEN])

    return btq(out)[:l]
```

Voici le fonctionnement de cette fonction :

1. Le message - après avoir été paddé avec PKCS #7 - est groupé en blocs de 16 octets.
2. Les blocs sont tous XORés entre eux dans `out`.
3. La variable `out` est convertie dans une base à 5 éléments puis tronquée au l-ème élément (i.e. elle est convertit en un vecteur de $[\beta]^l$) ; c'est cette version tronquée qui est retournée.

L'entrée de cette fonction est une chaîne d'octets de longueur 16 (cf. `assert` du fichier `chal.sage`), et la sortie, un vecteur de $[\beta]^l$.

```
>>> log(256**16, 2) # Taille en bits de l'ensemble de départ
128.0
>>> log(5**50, 2)   # Taille en bits de l'ensemble d'arrivé
116.09640474436812
```

Cette fonction induit une légère compression - 12 bits - donc rien de méchant, elle ne sera pas dure à reverser avec du bruteforce (en effet, nous avons une condition d'arrêt : le format du flag). Analysons maintenant la suite de la fonction `hash()`.

`A*x`

Nous avons $\mathbf{A} \in \mathbb{Z}_n^{k \times l}$: la matrice constante (cf. fichier `A.txt`), et $\mathbf{x} \in [\beta]^l$: notre message après passage dans la fonction `sanitize()`. Le hash $\mathbf{h}$ est lui un élément de $\mathbb{Z}_n^k$, obtenu par le produit : $\mathbf{Ax} \pmod n$.

Le problème qui consiste à retrouver $\mathbf{x}$ dans $[\beta]^l$ (comprendre : trouver un $\mathbf{x}$ avec uniquement des petits coefficients ; dans $[\beta]$) à partir de $\mathbf{A}$ et $\mathbf{h}$, s'appelle le ISIS (pour Inhomogeneous Secret Integer Solution).

Dans notre cas, le paramètrage est faible, nous devons alors résoudre un ISIS de *petite* dimension, ce qui est très efficace avec l'algorithme `LLL` (ou sa variante `BKZ`).

Résolution

Une implémentation complète de la solution est disponible dans le fichier `solve.sage`

TL;DR

- Nous allons construire la matrice qui a pour noyau l'ensemble des $x \in \mathbb{Z}_n^k$ tel que $h = Ax \pmod n$.
- Ensuite, nous allons lancer `LLL()` sur la matrice engendrant ce noyau afin de chercher une petite solution (et oui, nous cherchons un $\mathbf{x}$ petit).
- Magie (ou pas …) : le petit $\mathbf{x}$ que nous cherchons sera dans la matrice réduite.
- On bruteforce pour reverse `sanitize()`. Fini.

Si je construis la matrice :

$$L = \left(\mathbf{A} \;\middle|\; -\mathbf{h} \;\middle|\; n\mathbf{I}_k \right)$$

On remarque, en developpant le produit matriciel, qu'avec $y = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \\ 1 \\ q_1 \\ q_2 \\ \vdots \\ q_l \end{pmatrix}$, le produit $\mathbf{Ay} = 0$ (ssi les q_i sont bien choisis pour retirer ou ajouter le bon nombre de fois le module n).

Le vecteur $\mathbf{y}$ se trouve donc dans le noyau de $\mathbf{L}$. Il suffit maintenant de calculer ce noyau. Étant donné que $\mathbf{y}$ a une petite norme, l'application de `LLL()` ou `BKZ()` sur la matrice engendrant ce noyau nous permettra de retrouver $\mathbf{y}$ et donc $\mathbf{x}$.

Voici l'implémentation de ce procédé en sagemath :

```
L = block_matrix([A, -h, n*identity_matrix(k)])
kerL = L.right_kernel().matrix()
llled = kerL.LLL()

for idx, vec in enumerate(llled.rows()):
    vec = vec[:l]

    if all([v in range(-beta, beta+1) for v in vec]):
        x = list(vec)
        break
```

Il suffit ensuite de bruteforce les caractères manquant du flag jusqu'à ce que ce dernier, en bytestring, match le format du flag.