Writeup della challenge, key in the haystack

Contiene trick figo per trovare radice con molteplicità n in un dato polinomio

Questa è la parte di codice interessante:

```
bale = [p, q]
bale.extend(prime() for _ in range(1 << 6))

def add_hay(stack, to_add):
    x = stack[0]
    for i in range(1, len(stack)):
        y = stack[i]
        stack[i] = y + (to_add * x)
        x = y
    stack.append(to_add * x)

stack = [1]
add_hay(stack, p)
add_hay(stack, q)
for straw in bale:
    add_hay(stack, straw)

print("size:", len(stack))
for x in stack:
    print(b64enc(x))
```

In particolare p e q sono i numeri primi utilizzati per generare il modulo usato nell'rsa. Ovviamente, vogliamo sapere questi 2 valori per poter decryptare la flag.

Andiamo ad analizzare il codice:

- bale, sarà composto da p e q seguiti da 64 altri numeri primi.
- Il valore dello stack iniziale sarà 1.
- `add_hay(stack,p)` renderà lo stack = $\{1{,}p\}$.
- `add_hay(stack,q)` renderà lo stack = $\{1{,}p + q{,}p * q\}$, qui l'occhio esperto potrebbe notare che si tratta dei coefficienti del polinomio $(x + p)(x + q) = x^2 + x(p + q) + pq$. (ricorda regola somma e pro to)

Dopo aver iniziato il for loop in cui verranno aggiunti nello stack tutti i valori di bale:
- `add_hay(stack,bale[0] = p)` renderà lo stack = $\{1{,}2p + q{,}p^2 + 2pq,\ p^2q\ \}$. Ora, avevamo ipotizzato che i valori dello stack potrebbero rappresentare i coeff di un polinimio, verifichiamo! $(x + p)(x + q)(x + p) = (x^2 + x(pq) + pq)(x + p) = x^3 + x^2(2p + q) + x(p^2 + 2pq) + p^2q$, propio come ci aspettavamo!

Possiamo asserire, o perlomeno aspettarci che alla fine delle rimanenti 65 iterazioni avremo un polinomio di questo tipo:

$$P(x) = (x + p)(x + q)(x + p)(x + q) \prod_{i=1}^{64} (x + p_i)$$

dove p_i rappresentà l'$i$esimo primo aggiunto tramite l'extend.

Ora, dati i coefficienti, per fattorizzare il polinomio e dunque trovare facilmente p e q, utilizzeremo questo trick.

Il trick

In generale, dato un polinomio definito in questo modo

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$$

Se possiamo scomporlo in questo modo: $P(x) = (x + p)(x + q)(x + p_1)\dots(x + p_k)$,

Le sue radici sono $-p, -q, -p_1, \dots, -r_k$.

Ora andiamo a definire il polomio inverso in questo modo

$$Q(X) = X^n P\left(\frac{1}{X}\right).$$

Dunque $P\left(\frac{1}{X}\right) = a_n\left(\frac{1}{X}\right)^n + a_{n-1}\left(\frac{1}{X}\right)^{n-1} + \dots + a_1\left(\frac{1}{X}\right) + a_0$, ora moltiplicando per X^n otteniamo

$$Q(X) = a_0 X^n + a_1 X^{n-1} + \dots + a_{n-1} X + a_n$$

Ovvero, lo stesso polinomio con i **coefficienti invertiti**.

Il **trick** si basa su questo: se il polinomio ha radici con molteplicità n, saranno presenti anche nella derivata prima e cosi via fino alla $n - 1$ esima, come ci insegna l'algebra.

Dunque, il massimo comune divisore del polinomio tra il polinomio inverso e la sua derivata prima, sarà proprio il polinomio che contiene le radici con molteplicità, in quanto sarà presente in entrambi i polinomi. $D(X) = \gcd(Q(X), Q'(X))$.

In questo caso, dunque il gcd ci restituirà (supponendo non vengano generati altri primi ripetuti) il polinomio $(X + p)(X + q)$ permettendoci dunque di ricavare p e q e risolvere la chall!

exploit: