# Научно-исследовательская практика

## Cryptohack.org: kaitenzushi (HackTM CTF)

Гервятович Олег Игоревич
Коршунов Владислав Вячеславович

Институт физико-математических наук и информационных технологий БФУ им. И. Канта

8 июля 2023 г.

# Задача

Даны 2 файла: chall.sage (исходный код алгоритма kaitenzushi) и output.txt (полученные данные в ходе выполнения алгоритма). Найти значение флага, использующееся в алгоритме.

# Исходные данные. Часть 1

В файле chall.sage даны следующие секретные переменные:
p, q - простые числа, длиной 768 бит;
e - экспонента шифрования, являющаяся простым числом, длиной 256 бит;
$x_1$, $x_2$ - первый и второй элемент вектора x;
$y_1$, $y_2$ - первый и второй элемент вектора y.
Также указаны следующие условия:
$\text{НОД}((p-1)\times(q-1), e) = 1$;
$x_1^2 + e \times y_1^2 = x_2^2 + e \times y_2^2 = p \times q$.
В данном алгоритме выполняются следующие действия. Сначала находят переменную n (выводим её в output.txt) произведением чисел p и q. Далее идёт шифрование флага с помощью RSA. Результат шифрования присваиваем переменной c (выводим её в output.txt). После этого выбираем случайное значение $\theta$ в поле вещественных чисел по модулю 1337 в диапазоне $[-\pi, \pi]$.

# Исходные данные. Часть 2

Следом создаётся матрица R (размером 2x2) со следующими значениями: $\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$.

В конце находим новые векторы x (произведение старого вектора x на матрицу R) и y (произведение старого вектора y на матрицу R) и выводим их в output.txt.

# Решение

Создаём следующие переменные: $X_1$ и $X_2$ (которым присваивается значение первого и второго элемента новой матрицы x); $Y_1$ и $Y_2$ которым присваивается значение первого и второго элемента новой матрицы y). Далее находим экспоненту зашифрования:
$e = \frac{(2 \times n - (X_1^2 + X_2^2))}{(Y_1^2 + Y_2^2)}$.

# Код

```
from Crypto.Util.number import *

n = 990853953648382437503731888872568785013804329239290721076418541795771569507440261620612308640652961121590348037 2
c = 312168688094168684887530746663711142224819184527420449851136749248641895825646649162310024737395663075921549510 2
x = (9.936594001232774709263276764788831406973765090102977665128451398814873486374779171951795139705201088081161950
y = (9.028997440419990155494803623588970372177953039010859370710391718828352975635459590153366480167720023963554513 0

X1, X2 = x
Y1, Y2 = y

# находим e
e = floor((2*n - (X1**2 + X2**2)) / (Y1**2 + Y2**2))
assert is_prime(e)

# находим x1, x2, y1, y2
F = RealField(1337)
PR.<cos_theta, sin_theta, x1, x2, y1, y2> = PolynomialRing(QQ)
f1 = (x1*cos_theta + x2*sin_theta)^2 + e*(cos_theta*y1 + sin_theta*y2)^2 - n
f2 = (x2*cos_theta - x1*sin_theta)^2 + e*(cos_theta*y2 - sin_theta*y1)^2 - n

f_sin = f1.resultant(f2, sin_theta)
sin_roots = f_sin.subs({x1: X1, x2: X2, y1: Y1, y2: Y2}).univariate_polynomial().roots(ring=F, multiplicities=False)
```

# Код

```
23
24      f_cos = f1.resultant(f2, sin_theta)
25      cos_roots = f_cos.subs({x1: X1, x2: X2, y1: Y1, y2: Y2}).univariate_polynomial().roots(ring=F, multiplicities=False)
26
27      found = False
28      for sin_theta in sin_roots:
29          if sin_theta == 0:
30              continue
31
32          for cos_theta in cos_roots:
33              if cos_theta == 0:
34                  continue
35
36              R = matrix(F, [[cos_theta, -sin_theta], [sin_theta, cos_theta]])
37              M = matrix(F, [[X1, Y1], [X2, Y2]])
38              x1, y1, x2, y2 = (R**(-1) * M).list()
39              if x1 < 0 or x2 < 0 or y1 < 0 or y2 < 0:
40                  continue
41
42              x1 = round(x1)
43              x2 = round(x2)
44              y1 = round(y1)
```

# Код

```
45              y2 = round(y2)
46
47              found = True
48              break
49          if found:
50              break
51
52      assert x1**2 + e*y1**2 == n
53      assert x2**2 + e*y2**2 == n
54
55      p = gcd(x1*y2 - y1*x2, n)
56      q = n // p
57
58      # расшифровываем
59      d = pow(e, -1, (p-1)*(q-1))
60      m = pow(c, d, n)
61      #flag = ((3^3)^3)^3
62
63      flag = ((((m ** x1) ** y1)) ** x2) **y2
64      print(flag)
65      print(long_to_bytes(flag))
```

# Результат

В результате работы будет выведено значение флага:

# Ссылки

- Код презентации и код задания:
  https://github.com/KorkunoVl/Practice2023-Gervyatovich-Korshunov