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 θ 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

Изначально у нас есть набор скрытых значений, удовлетворяющих некоторым свойствам. У нас также есть ротация двух векторов, обозначенная как $x = (x_1, x_2)$ и $y = (y_1, y_2)$ в R^2.
Ротация не влияет на норму векторов, поэтому $|x|^2 = |xrot|^2 = x_1^2 + x_2^2$. Мы можем восстановить e, сложив два уравнения и решив их с нашими значениями $x_1^2 + x_2^2$ и $y_1^2 + y_2^2$.
В теории, поскольку ротация выполняется над вещественными числами, у нас нет точных значений норм исходных векторов, но у нас есть прииближение для обоих из них, и в частности восстановленное e имеет много бесполезных θ в десятичных знаках, поэтому мы можем округлить число до ближайшего целого и затем использовать то, что, поскольку наше приближение имеет большую точность, чем у $|\mathrm{x}|$, мы можем использовать вновь восстановленные $|\mathrm{y}|$ и e для вычисления эффективных норм исходных $|\mathrm{x}|$.
Наши векторы X,Y находятся в первой и четвертой четверти в R^2.

Решение

Предполагая, что наши исходные векторы находятся в первой четверти, что является разумным предположением, учитывая, что мы используем эти значения в XOR, мы можем с уверенностью сказать, что $\frac{-\pi}{2} \leq \theta \leq \frac{\pi}{2}$.
Исходя из этого предположения, мы можем использовать бинарный поиск для возможных значений, таких что исходные векторы находятся в первой четверти. Это позволяет нам сказать, что исходное значение находится между $\frac{-\pi}{8}$ и 0. Теперь существует монотонная связь между углом и длиной компоненты векторов. Чем меньше θ, тем больше становится x_1,y_1 (и тем меньше x_2,y_2), и наоборот.
Напомним, что первые два уравнения точно связывают значения x-координат векторов с y-координатами и позволяют снова использовать бинарный поиск для θ. Это дает нам правильные значения исходных векторов.

Решение

Теперь факторизация:
Во-первых: $x_1^2 + ey_1^2 = pq$ означает, что $(x_1 + \sqrt{-e}y_1)(x_1 - \sqrt{-e}y_1) = pq$ над $Z(\sqrt{-e})$.
Если можно было бы найти гомоморфизм $f : Z(\sqrt{-e}) \rightarrow \frac{Z}{nZ}$, такой что, $f(a) = a$, мы смогли бы отобразать гомоморфно делители n на себя, получив $f((x_1 + \sqrt{-e}y_1)(x_1 - \sqrt{-e}y_1) = f(x_1 + \sqrt{-e}y_1)f(x_1 - \sqrt{-e}y_1) = f(pq) = f(p)f(q) = 0 mod n$.
Отметим, что $x_1^2 + ey_1^2 = n$, означает, что $(\frac{x_1}{y_1})^2 + e = 0 mod n$, поэтому наш выбор m правильный, мы получаем кратные ненулевые делители, берем НОД и получаем p и q.

Код

```
from Crypto.Util.number import long_to_bytes

n = 9908539536483824375037318888725687850138043292392907210764185417
c = 3121686880941686848875307466637111422248191845274204498511367492
rot_x = (9.936594001232774709263276764788831406973765090102977665128
rot_y = (9.028997440419990155494803623588970372177953039010859370710

F = RealField(1337)
rot_x = vector(F, rot_x)
rot_y = vector(F, rot_y)
nxsq_old =rot_x * rot_x
nysq_old =rot_y * rot_y
e = (2*n - nxsq_old) / nysq_old
e = ZZ(QQ(e.numerical_approx(prec  = 700, digits = 400)))
nysq = ZZ(nysq_old.round())
nxsq = ZZ(2*n - e*nysq)

lb = -pi/2
up = pi/2

theta = (lb+up) / 2
R = matrix(F, [[cos(theta), -sin(theta)], [sin(theta), cos(theta)]])
Rinv = R^-1
newx = Rinv * vector(rot_x)
newy = Rinv * vector(rot_y)
tmpx1 = round(newx[0])
tmpx2 = round(newx[1])
```

Код

```
    tmpy1 = round(newy[0])
    tmpy2 = round(newy[1])
    if (tmpx2 < 0 or tmpy2 < 0):
        up = theta
    elif (tmpx1 < 0 or tmpy1 < 0):
        lb = theta
    elif (tmpx1.nbits() >768 or tmpy1.nbits()>640):
        up = theta
    elif (tmpx2.nbits() >768 or tmpy2.nbits()>640):
        lb = theta

    for j in range(8000):
        theta = (lb+up) / 2
        R = matrix(F, [[cos(theta), -sin(theta)], [sin(theta), cos(theta)]])
        Rinv = R^-1
        newx = Rinv * vector(rot_x)
        newy = Rinv * vector(rot_y)
        tmpx1 = round(newx[0])
        tmpx2 = round(newx[1])
        tmpy1 = round(newy[0])
        tmpy2 = round(newy[1])
        lhs1 = tmpx1^2 + e*tmpy1^2
        lhs2 = tmpx2^2 + e*tmpy2^2
        if (lhs1 > n or lhs2 <n):
            up = theta
        else:
            lb = theta
```

Код

```
    if (abs(lhs1 - n) == 0 and abs(lhs2 - n)==0):
        print("found")
        x1 = tmpx1
        x2 = tmpx2
        y1 = tmpy1
        y2 = tmpy2
        break

assert x1.nbits() <= 768 and x2.nbits() <= 768
assert y1.nbits() <= 640 and y2.nbits() <= 640
assert x1 ** 2 + e * y1 ** 2 == n
assert x2 ** 2 + e * y2 ** 2 == n
assert x1^2 + x2^2 == nxsq
assert y1^2 + y2^2 == nysq

tmp = int(x1 * pow(y1, -1, n))
assert((tmp^2 + e) % n == 0)
pfake =(x2 + tmp*y2) % n
qfake =(x2 - tmp*y2) % n
p = gcd(pfake, n)
q = n//p
phi = (p-1)*(q-1)
d = pow(e, -1, phi)
m = pow(c, d, n)
m = int(m) ^^ x1 ^^ x2 ^^ y1 ^^ y2
print(m)
print(long_to_bytes(m))
```

Результат

В результате работы будет выведено значение флага:

```
found
214555659008471306907293162637105662335968238779266594600393027927045601288193394
b'HackTM{r07473_pr353rv35_50m37h1n6}'
```

Ссылки

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