TP03 - EX1

April 28, 2024

0.1 TP 03 - EX 1

Como em algumas tecnicas criptograficas o HNP é definido como um jogo com um oráculo, quando o oráculo é consultado com algum número específico, ele retorna um valor que revela os bits mais importantes deste número.

p é usado como modulo em operações aritimeticas para definir um campo finito IF_p com p elementos, neste caso ele é o próximo numero primo após 2^{16}.

n é a quantidade de bits de p.

k é a quantidade de bits significativos, baseado na quantidade de bits de p.

$$k = \sqrt{n} + \sqrt{log_n 2}$$

O adversário tem vantagem em resolver o HNP quando k é aproximadamente $\sqrt{logp}$ usando $d = 2\sqrt{n}$, neste caso o k utilizado tem um valor mais significante.

Para a função MSB (Most Significant Bit) é passado um valor de consulta e a função fica em *looping* até que seja encontrado um valor absoluto da subtração do valor passado com um z aleatório que satisfaça a equação:

$$answer < p/2^{(k+1)}$$

A função "cria_oraculo" tem entrada do segredo (secret) e o chamar o oráculo, ele retorna um par (t, MSB), onde t é um valor aleatório e MSB é uma aproximação dos bits mais significativos do produto do *secret* e t, modulado por p.

[135]:
```
p = next_prime(2^16)

n = ceil(log(p, 2))

k = ceil(sqrt(n)) + ceil(log(n, 2))

d = 2 * ceil(sqrt(n))

def msb(query):
    """Retorna o MSB da query, baseado nos parametros p e k"""
    while True:
        z = randint(1, p-1)
        answer = abs(query - z)
```

```
        if answer < p / 2^(k+1):
            break
    return z

def cria_oraculo(secret):
    """Retorna um oraculo MSB randomizado usando o valor de alpha"""
    alpha = secret
    def oraculo():
        random_t = randint(1, p-1)
        return random_t, msb((alpha * random_t) % p)
    return oraculo
```

A função *basis_v* constrói uma base para uma rede (lattice) usando as entradas do oraculo e parametros definidos no HNP, como o numero primo p e a precisão dos bits significativos (d). Cada vetor base é composto por zeros e um numero significativo, definido como p. O ultimo vetor da base inclui as entradas do oraculo e o resltado é uma matriz que serve como a base para um lattice.

Já a função *cvp* vai encontrar o vetor mais proximo da rede (lattice), primeiramente aplica a redução via LLL a matrix base retornada da função *basis_v* o que faz com que os vetores da base sejam mais curtos e ortogonais, posteriormente usa-se o algoritmo do plano mais proximo de Babai, que faz uso da projeção ortogonal em um conjunto com arredondamento para determinar o ponto da rede mais próximo do vetor dado. O retorno desta função são os coeficientes do vetor aproximadamente mais próximo que encontrou, subtraindo do vetor de entrada v a diferença calculada.

O grupo tentou utilizar da função *closest_vector()* do Sagemath, porém não obteve sucesso, visto que esta função é extremamente lenta e com alto custo computacional, portanto, utilizamos a técnica citada acima.

[139]:
```
def basis_v(oracle_inputs):
    basis_vectors = []
    for i in range(d):
        p_vector = [0] * (d+1)
        p_vector[i] = p
        basis_vectors.append(p_vector)
    basis_vectors.append(list(oracle_inputs) + [QQ(1)/QQ(p)])
    return Matrix(QQ, basis_vectors)

def cvp(basis, v):
    BL = basis.LLL()
    G, _ = BL.gram_schmidt()
    _, n = BL.dimensions()
    small = vector(ZZ, v)
    for i in reversed(range(n)):
        c = QQ(small * G[i]) / QQ(G[i] * G[i])
        c = c.round()
        small -= BL[i] * c
    return (v - small).coefficients()
```

[142]:
```
secret = randint(1, p-1)
print("SEGREDO: ", secret)

# Cria um oraculo usando o secret como escalar
oracle = cria_oraculo(secret)

inputs, answers = zip(*[ oracle() for _ in range(d) ])

u = vector(ZZ, list(answers) + [0])
print("Vetor de respostas do Oraculo:\n%s\n" % str(u))

lattice = basis_v(inputs)
print("Matrix CVP com Lattice:\n%s\n" % str(lattice))

v = cvp(lattice, u)
print("Vetor aproximado:\n%s\n" % str(v))

recovered_secret = (v[-1] * p) % p
print('Segredo resolvido?', recovered_secret == secret)
print("SEGREDO RECUPERADO: %d" % recovered_secret)
```

```
SEGREDO:  32400
Vetor de respostas do Oraculo:
(34968, 9914, 47797, 17000, 45367, 27856, 44171, 32768, 41763, 42842, 0)

Matrix CVP com Lattice:
[  65537       0       0       0       0       0       0       0       0       0
0]
[      0   65537       0       0       0       0       0       0       0       0
0]
[      0       0   65537       0       0       0       0       0       0       0
0]
[      0       0       0   65537       0       0       0       0       0       0
0]
[      0       0       0       0   65537       0       0       0       0       0
0]
[      0       0       0       0       0   65537       0       0       0       0
0]
[      0       0       0       0       0       0   65537       0       0       0
0]
[      0       0       0       0       0       0       0   65537       0       0
0]
[      0       0       0       0       0       0       0       0   65537       0
0]
[      0       0       0       0       0       0       0       0       0   65537
0]
[  17601   29407   37574   32589    5568   58881   60793   38682   38124   20603
1/65537]
```

```
Vetor aproximado:
[34963, 9894, 47825, 16993, 45376, 27867, 44202, 32749, 41761, 42855,
32400/65537]

Segredo resolvido? True
SEGREDO RECUPERADO: 32400
```