Ex1

May 1, 2024

1 Estruturas Criptográficas - Criptografia e Segurança da Informação

Grupo 03

(PG54177) Ricardo Alves Oliveira

(PG54236) Simão Oliveira Alvim Barroso

1.1 TP3 - Exercício 1

1. No capítulo 5 dos apontamentos é descrito o chamado Hidden Number Problem. No capítulo 8 dos apontamentos é discutida um artigo de Nguyen & Shparlinsk , onde se propõem reduções do HNP a problemas difíceis em reticulados. Neste trabalho pretende-se construir, com a ajuda do Sagemath, uma implementação da solução discutida nos apontamentos para resolver o HNP com soluções aproximadas dos problemas em reticulados.

1.2 Resolução

Ao longo deste playbook iremos apresentar os passos tomados para resolver o exercício proposto juntamente com as respetivas funções implementadas.

Para tal, começamos por importar a biblioteca `sagemath` e definir as variáveis necessárias seguindo o defenido por Nguyen & Shparlinsk. Isto define que para um dado `d` temos as seguintes variáveis:

- `p` 2^d
- `k`$>\sqrt{d}+\log 2d$
- `n`$>2\sqrt{d}$

Tendo em conta estas definições, a implementação do HNP, que dita `p` como um número primo, e seja `s` um qualquer número inteiro tal que `1<s<p`, temos então:

[ ]:
```
from sage.all import *
from sage.matrix.matrix_rational_dense import *

d = 12
p = next_prime(2^d)
n = 4 * ceil(sqrt(d))
k = ceil(sqrt(d) + log(d, 2))
s = randint(1, p - 1)
```

```
print(f"d = {d}")
print(f"p = {p}")
print(f"n = {n}")
print(f"k = {k}")
print(f"s = {s}")
```

```
d = 12
p = 4099
n = 16
k = 8
s = 3036
```

De seguida, definimos a função auxiliar `represent_in_Zq` que, recebendo um número `x` e um módulo `q`, devolve a representação de `x` no intervalo `[0, q-1]`.

[ ]:
```
def represent_in_Zq(val, max):
    return val % max
```

A próxima função a ser implementada é fundamental para o cálculo do `HNP`, sendo esta a `msb_k`. Esta deve, para qualquer `y` pertencente a `ZZp`, devolver um `u E ZZp` tal que:

$$u = \text{msb}_k(y) \quad se\ e\ s\ se \quad 0 \leq y - B\,u < B$$

Onde,

$$B \equiv p/\lambda \quad para \quad \lambda \equiv 2^k$$

[ ]:
```
def msb_k(y, k, p):
    B = p // (2 ** k)
    return y // B
```

Com esta função implementada, podemos agora criar os pares que serão utilizados para a resolução do `HNP`. Para tal, definimos a função `generate_pairs` que tem por objetivo gerar `n` pares `(x_i, u_i)` tal que, para um qualquer `x_i` pertencente a `ZZp`, `u` é o resultado da função `msb_k` aplicada a `s * x_i % q`.

Esta regra pode ser representada pela seguinte fórmula, presente no capítulo 5 dos apontamentos:

$$u_i \ = \ \text{msb}_k(s \times x_i \bmod q)$$

Com base nisto podemos então gerar o oráculo que será utilizado para a resolução do `HNP`.

[ ]:
```
def generate_pairs(N, q, k, s):
    pairs = []
    for i in range(N):
        x_i = randint(0, q - 1)
        u_i = msb_k((s * x_i) % q, k, q)
        pairs.append((x_i, u_i))
    return pairs
```

```
pairs = generate_pairs(n, p, k, s)
print(pairs)
```

```
[(2112, 140), (37, 73), (2441, 196), (2316, 170), (769, 74), (1838, 55), (54,
203), (507, 5), (1909, 134), (766, 5), (2540, 142), (2298, 17), (2418, 185),
(3434, 242), (2231, 30), (3547, 251)]
```

Tendo agora os varios pares `(x_i, u_i)` gerados, podemos então começar a implementar a resolução do `HNP` conforme o capítulo 8 dos apontamentos.

Para isso devemos começar por construir a matriz geradora tal que:

$$G \in \mathbb{Q}^{m \times m} \quad com \quad m = n + 2$$

$$G' \equiv \begin{bmatrix} p & 0 & \cdots & 0 & 0 & 0 \\ 0 & p & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & p & 0 & 0 \\ x_1 & x_2 & \cdots & x_n & A & 0 \\ -B\,u_1 & -B\,u_2 & \cdots & -B\,u_n & 0 & M \end{bmatrix}$$

Onde `x1, x2, ..., xn` e `u1, u2, ..., un` são os valores `(x_i,u_i)` anteriormente gerados pela função `generate_pairs`. A variável `B` mantém-se igual à definição anterior. `A` deve ser definido como `1/lambda`, relembrando que `lambda=2^k`. Finalmente `M`, segundo o descrito nos apontamentos, deve ser "um qualquer grande inteiro `M>>lambda`".

[ ]:
```
def builg_G(oracle_inputs, secon, p, k):
    l = len(oracle_inputs) + 2  # Dimension of the lattice
    basis_vectors = []

    # Add basis vectors for oracle inputs
    for i in range(l-2):
        p_vector = [0] * (l)
        p_vector[i] = p
        basis_vectors.append(p_vector)


    # Add the last basis vector with appropriate values
    A = 1 / (2 ** k)
    M = p * (2 ** k)
    last_basis_vector = oracle_inputs + [A,0]
    basis_vectors.append(last_basis_vector)
    last_last = secon + [0, M]
    basis_vectors.append(last_last)
    return Matrix(QQ, basis_vectors)
```

```python
B = p / (2 ** k)
G = builg_G([x_i for x_i, _ in pairs], [-B * u_i for _, u_i in pairs], p,k)
print(G)
```

```
[        4099            0            0            0            0            0
0            0            0            0            0            0            0
0            0            0            0            0]
[           0         4099            0            0            0            0
0            0            0            0            0            0            0
0            0            0            0            0]
[           0            0         4099            0            0            0
0            0            0            0            0            0            0
0            0            0            0            0]
[           0            0            0         4099            0            0
0            0            0            0            0            0            0
0            0            0            0            0]
[           0            0            0            0         4099            0
0            0            0            0            0            0            0
0            0            0            0            0]
[           0            0            0            0            0         4099
0            0            0            0            0            0            0
0            0            0            0            0]
[           0            0            0            0            0            0
4099            0            0            0            0            0
0            0            0            0            0            0]
[           0            0            0            0            0            0
0         4099            0            0            0            0            0
0            0            0            0            0]
[           0            0            0            0            0            0
0            0         4099            0            0            0            0
0            0            0            0            0]
[           0            0            0            0            0            0
0            0            0         4099            0            0            0
0            0            0            0            0]
[           0            0            0            0            0            0
0            0            0            0         4099            0            0
0            0            0            0            0]
[           0            0            0            0            0            0
0            0            0            0            0         4099            0
0            0            0            0            0]
[           0            0            0            0            0            0
0            0            0            0            0            0         4099
0            0            0            0            0]
[           0            0            0            0            0            0
0            0            0            0            0            0            0
4099            0            0            0            0]
[           0            0            0            0            0            0
```

```
0             0             0             0             0             0             0
0          4099             0             0             0]
[             0             0             0             0             0             0
0             0             0             0             0             0             0
0             0          4099             0             0]
[          2112            37          2441          2316           769          1838
54            507          1909           766          2540          2298          2418
3434          2231          3547          1/256            0]
[  -143465/64  -299227/256   -200851/64  -348415/128  -151663/128  -225445/256
-832097/256   -20495/256  -274633/128   -20495/256  -291029/128   -69683/256
-758315/256  -495979/128   -61485/128 -1028849/256             0       1049344]
```

O próximo passo é aplicar a redução `BKZ` a `G` para a sua forma reduzida, ou aproximada. Para tal, definimos a função `reduced_matrix` que recebe a matriz `G` e devolve a sua forma reduzida, com base no algoritmo em uso.

No entantom, apesar de a redução `BKZ` estar disponivel na documentação do [SageMath] (https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_rational_dense.html#sage.matrix.m esta apresentou problemas na sua execução dependendo da versão do SageMath utilizada. Assim, optamos por implementar utilizar a função `LLL` para obter a lattice reduzida, ou a sua aproximação.

[ ]:
```python
def reduce_lattice(lattice):
    return lattice.LLL()

print(type(G))

reduced_lattice = reduce_lattice(G)
print(reduced_lattice)
```

```
<class 'sage.matrix.matrix_rational_dense.Matrix_rational_dense'>
[          0           0           0           0           0           0           0           0
0           0           0           0           0           0           0           0
-4099/256           0]
[       269         356         111        -427         752        -484         298        1887
-244         280        -709         840        -775         581        -137        -769
-1209/256           0]
[       140        -409         713         433         -81         510         -43        -176
833        1060         -49        -919       -1470         150       -1397         895
493/128           0]
[     -1385        -492         998        -331        -809         -68         279         570
982        1225        -540        -424         196         -20         -87        1247
-339/128           0]
[      1468        -541         -19        1144        -498        -508        -568        -323
-1103         -11         306         -33         982       -1355        1279        -570
301/64           0]
[       529         182         264        -794          16        -708        -953         389
1303        1663        -800         890         594         274         782        -278
-549/256           0]
[       269         356         111        -427         752        -484         298       -2212
```

```
-244       280      -709       840      -775       581      -137      -769
-1209/256          0]
[      -57      -624       266       380     -1226     -1086      -246       423
-954       154      -185       462     -1451     -1525     -1510      -218
-115/64          0]
[     1707       781       232       917      -607      1241        32      -155
524      1213      1657      -460       522       365       563     -1238
-1973/256          0]
[      768      1504       515     -1021       -93      1041      -353       557
-1169     -1212       551       463       134      1994        66      -946
373/256          0]
[     -943        32      1668      -545      1219       371      -618      -337
-897      -113       535      -339      -346      -132     -1394       852
111/64          0]
[     1106       458       304     -1908      -230       -70       890      1069
1252       176      -797       528       684      1185        31       922
1231/256          0]
[      591       645      1341       159      -664       689       609     -1797
1262     -1381       408       -44       -57       926       339      -760
239/256          0]
[     -201      -906       938      1340      -440     -1025      -436      -450
-991      1406      1505       119        61       663     -1010      -553
751/256          0]
[     -409        53      -824        -6      -671       -26      -255     -1711
-589     -1340       758        79     -1854      -731      1534      -126
223/256          0]
[     -143     -1134      -699      -413      -415       943       893       414
-236       674       255     -2077      -548      -446      1749      -475
-363/256          0]
[     1522     -1460      -271       784       232      2031       528      -508
448      1569       697       608       415      -126      -958        68
-453/64          0]
[   855/64   549/256    109/64   897/128  -111/128  3675/256  2463/256  3313/256
-73/128  3057/256  1451/128   973/256   981/256   661/128   211/128   783/256
91/64   1049344]
```

O passo final trata-se de obter o vetor `v` que, segundo a resolução do `HNP`, deve ser:

$$\begin{bmatrix} e_1 & e_2 & \cdots & e_{n+1} & M \end{bmatrix}$$

Onde `e1, e2, ..., en+1` são os primeiros `n+1` elementos da última linha da matriz reduzida `G`.

Com isto, conseguimos então obter o valor de `s` que resolve o `HNP` com soluções aproximadas dos problemas em reticulados. Basta para isso obter o valor de `en+1` que, pelo definido nos apontamentos é `s*A`. Uma vez que `A` é `1/lambda`, basta multiplicar `en+1` por `lambda` para obter o valor de `s`, garantindo que `s E ZZp`.

[ ]:
```
def get_secret_from_reduced_lattice(reduced_lattice, q, k):
    shortest_vector = reduced_lattice.row(-1)
    s = (shortest_vector[-2] * (2 ** k)).ceil()
    return represent_in_Zq(s, q)

found_secret = get_secret_from_reduced_lattice(reduced_lattice, p, k)
print("found_s:",found_secret)
```

```
found_s: 364
```

1.2.1 Testes

Adicionalmente, podemos realizar testes para verificar este processo e a sua correta implementação.

Apesar do valor poder ser observado como igual ao inicialmente definido, foi implementada uma função `verify_secret` que verifica se o valor de `s` obtido é igual ao valor inicialmente definido. Esta função deve receber os valores do oráculo e o novo segredo, devolvendo `True` caso, para todos os `x_i`, os valores resultantes da aplicação da função `msb_k` sejam iguais aos valores `u_i` do oráculo. Caso contrário, devolve `False`.

[ ]:
```
def verify_secret(ss, x_values, u_values, q, k):
    for x, u in zip(x_values, u_values):
        if msb_k((ss * x) % q, k, q) != u:
            return False
    return True

# Verificar o segredo extraído
t = [x for x, _ in pairs]
u = [u for _, u in pairs]
is_correct = verify_secret(found_secret, t, u, p, k)

if is_correct:
    print("O segredo extraído está correto.")
else:
    print("O segredo extraído está incorreto.")
```

```
O segredo extraído está correto.
```

[ ]:
```
def test_for_d(dd):
    pp = next_prime(2**dd)
    nn = 4 * ceil(sqrt(dd))
    kk = ceil(sqrt(dd) + log(dd, 2))
    ss = randint(1, pp - 1)


    ppairs = generate_pairs(nn, pp, kk, ss)

    BB = pp / (2 ** kk)
```

```
    GG = builg_G([xx for xx, _ in ppairs], [-BB * uu for _, uu in ppairs], pp,␣
↪kk)

    rreduced_lattice = reduce_lattice(GG)

    ffound_secret = get_secret_from_reduced_lattice(rreduced_lattice, pp, kk)


    tt = [x for x, _ in ppairs]
    uu = [u for _, u in ppairs]
    iis_correct = verify_secret(ffound_secret, tt, uu, pp, kk)

    return iis_correct
```

[ ]:
```
import time

times=[]

for i in range(5, 126):
    print(f"Testando para d = {i}\t| ", end='')
    #time start
    start = time.time()
    for _ in range(100):
        result=test_for_d(i)
    #time end
    end = time.time()
    times.append((i,(end-start)/100))
    print(f"Resultado: {result}\t| Tempo: {(end-start)/100}")
```

```
Testando para d = 5     | Resultado: True       | Tempo: 0.0042603397369384765
Testando para d = 6     | Resultado: True       | Tempo: 0.003992326259613037
Testando para d = 7     | Resultado: True       | Tempo: 0.003999013900756836
Testando para d = 8     | Resultado: True       | Tempo: 0.003881392478942871
Testando para d = 9     | Resultado: True       | Tempo: 0.003675692081451416
Testando para d = 10    | Resultado: True       | Tempo: 0.005639822483062744
Testando para d = 11    | Resultado: True       | Tempo: 0.005959815979003906
Testando para d = 12    | Resultado: True       | Tempo: 0.006222991943359375
Testando para d = 13    | Resultado: True       | Tempo: 0.006057145595550537
Testando para d = 14    | Resultado: True       | Tempo: 0.006171495914459229
Testando para d = 15    | Resultado: True       | Tempo: 0.006137731075286865
Testando para d = 16    | Resultado: True       | Tempo: 0.005276095867156982
Testando para d = 17    | Resultado: True       | Tempo: 0.008319664001464843
Testando para d = 18    | Resultado: True       | Tempo: 0.008511765003204346
Testando para d = 19    | Resultado: True       | Tempo: 0.008411238193511963
Testando para d = 20    | Resultado: True       | Tempo: 0.009937818050384522
Testando para d = 21    | Resultado: True       | Tempo: 0.01050628662109375
Testando para d = 22    | Resultado: True       | Tempo: 0.008930094242095947
```

```
Testando para d = 23    | Resultado: True       | Tempo: 0.008702223300933837
Testando para d = 24    | Resultado: True       | Tempo: 0.01040555238723755
Testando para d = 25    | Resultado: True       | Tempo: 0.009039254188537597
Testando para d = 26    | Resultado: True       | Tempo: 0.012123892307281494
Testando para d = 27    | Resultado: True       | Tempo: 0.01214590311050415
Testando para d = 28    | Resultado: True       | Tempo: 0.012211501598358154
Testando para d = 29    | Resultado: True       | Tempo: 0.012071468830108643
Testando para d = 30    | Resultado: True       | Tempo: 0.012115566730499268
Testando para d = 31    | Resultado: True       | Tempo: 0.012306311130523682
Testando para d = 32    | Resultado: True       | Tempo: 0.012261695861816406
Testando para d = 33    | Resultado: True       | Tempo: 0.016476263999938966
Testando para d = 34    | Resultado: True       | Tempo: 0.01689007759094238
Testando para d = 35    | Resultado: True       | Tempo: 0.017248847484588624
Testando para d = 36    | Resultado: True       | Tempo: 0.017072958946228026
Testando para d = 37    | Resultado: True       | Tempo: 0.024627585411071778
Testando para d = 38    | Resultado: True       | Tempo: 0.02595285415649414
Testando para d = 39    | Resultado: True       | Tempo: 0.02627246379852295
Testando para d = 40    | Resultado: True       | Tempo: 0.026556451320648194
Testando para d = 41    | Resultado: True       | Tempo: 0.02621768236160278
Testando para d = 42    | Resultado: True       | Tempo: 0.026567769050598145
Testando para d = 43    | Resultado: True       | Tempo: 0.026938538551330566
Testando para d = 44    | Resultado: True       | Tempo: 0.02789154291152954
Testando para d = 45    | Resultado: True       | Tempo: 0.028098878860473634
Testando para d = 46    | Resultado: True       | Tempo: 0.028439717292785646
Testando para d = 47    | Resultado: True       | Tempo: 0.0287129807472229
Testando para d = 48    | Resultado: True       | Tempo: 0.029528167247772217
Testando para d = 49    | Resultado: True       | Tempo: 0.029371669292449953
Testando para d = 50    | Resultado: True       | Tempo: 0.04123117446899414
Testando para d = 51    | Resultado: True       | Tempo: 0.04180846691131592
Testando para d = 52    | Resultado: True       | Tempo: 0.04412210464477539
Testando para d = 53    | Resultado: True       | Tempo: 0.050455501079559324
Testando para d = 54    | Resultado: True       | Tempo: 0.05139308929443359
Testando para d = 55    | Resultado: True       | Tempo: 0.05051387548446655
Testando para d = 56    | Resultado: True       | Tempo: 0.049922688007354735
Testando para d = 57    | Resultado: True       | Tempo: 0.04944596529006958
Testando para d = 58    | Resultado: True       | Tempo: 0.050030138492584225
Testando para d = 59    | Resultado: True       | Tempo: 0.050749857425689694
Testando para d = 60    | Resultado: True       | Tempo: 0.0513713812828064
Testando para d = 61    | Resultado: True       | Tempo: 0.05185266733169556
Testando para d = 62    | Resultado: True       | Tempo: 0.05228433847427368
Testando para d = 63    | Resultado: True       | Tempo: 0.05354949235916138
Testando para d = 64    | Resultado: True       | Tempo: 0.05358516454696655
Testando para d = 65    | Resultado: True       | Tempo: 0.07644004344940186
Testando para d = 66    | Resultado: True       | Tempo: 0.07491888999938964
Testando para d = 67    | Resultado: True       | Tempo: 0.07625390291213989
Testando para d = 68    | Resultado: True       | Tempo: 0.07722525358200073
Testando para d = 69    | Resultado: True       | Tempo: 0.0778765869140625
Testando para d = 70    | Resultado: True       | Tempo: 0.07959930419921875
```

```
Testando para d = 71    | Resultado: True       | Tempo: 0.07967189073562622
Testando para d = 72    | Resultado: True       | Tempo: 0.08427001237869262
Testando para d = 73    | Resultado: True       | Tempo: 0.08536597728729248
Testando para d = 74    | Resultado: True       | Tempo: 0.08585732221603394
Testando para d = 75    | Resultado: True       | Tempo: 0.08568284273147583
Testando para d = 76    | Resultado: True       | Tempo: 0.08553138256072998
Testando para d = 77    | Resultado: True       | Tempo: 0.09387141227722168
Testando para d = 78    | Resultado: True       | Tempo: 0.08771333932876586
Testando para d = 79    | Resultado: True       | Tempo: 0.08849998950958252
Testando para d = 80    | Resultado: True       | Tempo: 0.09159328699111939
Testando para d = 81    | Resultado: True       | Tempo: 0.09088526248931884
Testando para d = 82    | Resultado: True       | Tempo: 0.12084532260894776
Testando para d = 83    | Resultado: True       | Tempo: 0.13188949823379517
Testando para d = 84    | Resultado: True       | Tempo: 0.12375930070877075
Testando para d = 85    | Resultado: True       | Tempo: 0.12503229856491088
Testando para d = 86    | Resultado: True       | Tempo: 0.13410646438598633
Testando para d = 87    | Resultado: True       | Tempo: 0.13790130615234375
Testando para d = 88    | Resultado: True       | Tempo: 0.14004895448684693
Testando para d = 89    | Resultado: True       | Tempo: 0.1394200038909912
Testando para d = 90    | Resultado: True       | Tempo: 0.1430157494544983
Testando para d = 91    | Resultado: True       | Tempo: 0.14417694091796876
Testando para d = 92    | Resultado: True       | Tempo: 0.14484849452972412
Testando para d = 93    | Resultado: True       | Tempo: 0.1386698913574219
Testando para d = 94    | Resultado: True       | Tempo: 0.14724543333053589
Testando para d = 95    | Resultado: True       | Tempo: 0.1400204062461853
Testando para d = 96    | Resultado: True       | Tempo: 0.14738118410110473
Testando para d = 97    | Resultado: True       | Tempo: 0.14537307262420654
Testando para d = 98    | Resultado: True       | Tempo: 0.14861897945404054
Testando para d = 99    | Resultado: True       | Tempo: 0.15026005506515502
Testando para d = 100   | Resultado: True       | Tempo: 0.15860520601272582
Testando para d = 101   | Resultado: True       | Tempo: 0.19349915742874146
Testando para d = 102   | Resultado: True       | Tempo: 0.20509963989257812
Testando para d = 103   | Resultado: True       | Tempo: 0.20231552839279174
Testando para d = 104   | Resultado: True       | Tempo: 0.2132391858100891
Testando para d = 105   | Resultado: True       | Tempo: 0.21884895563125611
Testando para d = 106   | Resultado: True       | Tempo: 0.20387235879898072
Testando para d = 107   | Resultado: True       | Tempo: 0.21980316638946534
Testando para d = 108   | Resultado: True       | Tempo: 0.22205047369003295
Testando para d = 109   | Resultado: True       | Tempo: 0.21265204429626464
Testando para d = 110   | Resultado: True       | Tempo: 0.23283011198043824
Testando para d = 111   | Resultado: True       | Tempo: 0.23673345327377318
Testando para d = 112   | Resultado: True       | Tempo: 0.2511498713493347
Testando para d = 113   | Resultado: True       | Tempo: 0.25604196786880495
Testando para d = 114   | Resultado: True       | Tempo: 0.2753901481628418
Testando para d = 115   | Resultado: True       | Tempo: 0.26415166139602664
Testando para d = 116   | Resultado: True       | Tempo: 0.2513222050666809
Testando para d = 117   | Resultado: True       | Tempo: 0.251125693321228
Testando para d = 118   | Resultado: True       | Tempo: 0.25397356033325197
```

```
Testando para d = 119   | Resultado: True       | Tempo: 0.25383293628692627
Testando para d = 120   | Resultado: True       | Tempo: 0.2439748477935791
Testando para d = 121   | Resultado: True       | Tempo: 0.25560076713562013
Testando para d = 122   | Resultado: True       | Tempo: 0.3193270778656006
Testando para d = 123   | Resultado: True       | Tempo: 0.32118203401565554
Testando para d = 124   | Resultado: True       | Tempo: 0.32684552669525146
Testando para d = 125   | Resultado: True       | Tempo: 0.3273454976081848
```

[ ]:
```python
# make a graph of the results
import matplotlib.pyplot as plt

x = [i for i, _ in times]
y = [t for _, t in times]

plt.plot(x, y)
plt.xlabel("d")
plt.ylabel("Tempo médio (s)")
plt.title("Tempo médio de 100 medições para cada d")
plt.show()
```