

CS152-13

Hongchen Cao 2019533114

2020/12/20

1

```
1 from itertools import product
2 import random
3
4 server_data = "01001101" # change server data here
5
6
7 def omega_bijection(i: int) -> tuple:
8     if i == 1:
9         return 1, 1, 1
10    elif i == 2:
11        return 1, 1, 2
12    elif i == 3:
13        return 1, 2, 1
14    elif i == 4:
15        return 1, 2, 2
16    elif i == 5:
17        return 2, 1, 1
18    elif i == 6:
19        return 2, 1, 2
20    elif i == 7:
21        return 2, 2, 1
22    elif i == 8:
23        return 2, 2, 2
24
25
26 def query_y(alpha_beta_gamma: tuple) -> int:
27     if alpha_beta_gamma == (1, 1, 1):
28         return int(server_data[0])
29     elif alpha_beta_gamma == (1, 1, 2):
30         return int(server_data[1])
31     elif alpha_beta_gamma == (1, 2, 1):
32         return int(server_data[2])
33     elif alpha_beta_gamma == (1, 2, 2):
34         return int(server_data[3])
35     elif alpha_beta_gamma == (2, 1, 1):
36         return int(server_data[4])
37     elif alpha_beta_gamma == (2, 1, 2):
38         return int(server_data[5])
39     elif alpha_beta_gamma == (2, 2, 1):
40         return int(server_data[6])
41     elif alpha_beta_gamma == (2, 2, 2):
42         return int(server_data[7])
43
44
```

```

45 def calculate_a(index) -> int:
46     res = 0
47     for i in index:
48         res += query_y(i)
49     return res % 2
50
51
52 def calculate_A1(u_v_w: tuple, q000) -> tuple:
53     if u_v_w.count(1) == 1:
54         oplus_index = u_v_w.index(1)
55         if oplus_index == 0:
56             res1 = product(q000[0].symmetric_difference({1}), q000[1], q000[2])
57             res2 = product(q000[0].symmetric_difference({2}), q000[1], q000[2])
58         elif oplus_index == 1:
59             res1 = product(q000[0], q000[1].symmetric_difference({1}), q000[2])
60             res2 = product(q000[0], q000[1].symmetric_difference({2}), q000[2])
61         elif oplus_index == 2:
62             res1 = product(q000[0], q000[1], q000[2].symmetric_difference({1}))
63             res2 = product(q000[0], q000[1], q000[2].symmetric_difference({2}))
64
65     res1 = calculate_a(res1)
66     res2 = calculate_a(res2)
67     return res1, res2
68
69
70 def calculate_A2(u_v_w: tuple, q111) -> tuple:
71     if u_v_w.count(1) == 2:
72         oplus_index = u_v_w.index(0)
73         if oplus_index == 0:
74             res1 = product(q111[0].symmetric_difference({1}), q111[1], q111[2])
75             res2 = product(q111[0].symmetric_difference({2}), q111[1], q111[2])
76         elif oplus_index == 1:
77             res1 = product(q111[0], q111[1].symmetric_difference({1}), q111[2])
78             res2 = product(q111[0], q111[1].symmetric_difference({2}), q111[2])
79         elif oplus_index == 2:
80             res1 = product(q111[0], q111[1], q111[2].symmetric_difference({1}))
81             res2 = product(q111[0], q111[1], q111[2].symmetric_difference({2}))
82
83     res1 = calculate_a(res1)
84     res2 = calculate_a(res2)
85     return res1, res2
86
87

```

```

88 # Main Part
89 def query(k: int) -> tuple:
90     choice = [{1}, {2}, {1, 2}, set()]
91     q000 = [choice[random.randint(0, 3)], choice[random.randint(0, 3)], choice[random.randint(0, 3)]]
92     q111 = [set(), set(), set()]
93     right = omega_bijection(k)
94     q111[0] = q000[0].symmetric_difference({right[0]})
95     q111[1] = q000[1].symmetric_difference({right[1]})
96     q111[2] = q000[2].symmetric_difference({right[2]})
97     return q000, q111
98
99
100 def answer1(q000: list):
101     a000 = calculate_a([list(q000[0])[0], list(q000[1])[0], list(q000[2])[0]])
102     A100 = calculate_A1((1, 0, 0), q000)
103     A010 = calculate_A1((0, 1, 0), q000)
104     A001 = calculate_A1((0, 0, 1), q000)
105
106     return a000, A100, A010, A001
107
108
109 def answer2(q111: list):
110     a111 = calculate_a(product(q111[0], q111[1], q111[2]))
111     A011 = calculate_A2((0, 1, 1), q111)
112     A101 = calculate_A2((1, 0, 1), q111)
113     A110 = calculate_A2((1, 1, 0), q111)
114
115     return a111, A011, A101, A110
116
117
118 def restruction(k: int):
119     omega = omega_bijection(k)
120     ans1 = answer1(query(k)[0])
121     ans2 = answer2(query(k)[1])
122     res = ans1[0] + ans2[0]
123
124     for h in range(3):
125         res += ans1[h + 1][omega[h] - 1]
126         res += ans2[h + 1][omega[h] - 1]
127
128     return res % 2
129
130
131 if __name__ == '__main__':
132     print("res: ", restruction(int(input())))

```

2

Follow the hint we can represent the database as a 4-dimension cube and use the similar method to 2-server PIR.

```
1 from itertools import product
2 import random
3
4 server_data = "0101101001011010" # change server data here
5
6
7 def omega_bijection(k: int) -> tuple:
8     if k == 1:
9         return 1, 1, 1, 1
10    elif k == 2:
11        return 1, 1, 1, 2
12    elif k == 3:
13        return 1, 1, 2, 1
14    elif k == 4:
15        return 1, 1, 2, 2
16    elif k == 5:
17        return 1, 2, 1, 1
18    elif k == 6:
19        return 1, 2, 1, 2
20    elif k == 7:
21        return 1, 2, 2, 1
22    elif k == 8:
23        return 1, 2, 2, 2
24    elif k == 9:
25        return 2, 1, 1, 1
26    elif k == 10:
27        return 2, 1, 1, 2
28    elif k == 11:
29        return 2, 1, 2, 1
30    elif k == 12:
31        return 2, 1, 2, 2
32    elif k == 13:
33        return 2, 2, 1, 1
34    elif k == 14:
35        return 2, 2, 1, 2
36    elif k == 15:
37        return 2, 2, 2, 1
38    elif k == 16:
39        return 2, 2, 2, 2
40
41
42 def query_y(alpha_beta_gamma: tuple) -> int:
43     if alpha_beta_gamma == (1, 1, 1, 1):
44         return int(server_data[0])
45     elif alpha_beta_gamma == (1, 1, 1, 2):
46         return int(server_data[1])
47     elif alpha_beta_gamma == (1, 1, 2, 1):
```

```

48     return int(server_data[2])
49 elif alpha_beta_gamma == (1, 1, 2, 2):
50     return int(server_data[3])
51 elif alpha_beta_gamma == (1, 2, 1, 1):
52     return int(server_data[4])
53 elif alpha_beta_gamma == (1, 2, 1, 2):
54     return int(server_data[5])
55 elif alpha_beta_gamma == (1, 2, 2, 1):
56     return int(server_data[6])
57 elif alpha_beta_gamma == (1, 2, 2, 2):
58     return int(server_data[7])
59 elif alpha_beta_gamma == (2, 1, 1, 1):
60     return int(server_data[8])
61 elif alpha_beta_gamma == (2, 1, 1, 2):
62     return int(server_data[9])
63 elif alpha_beta_gamma == (2, 1, 2, 1):
64     return int(server_data[10])
65 elif alpha_beta_gamma == (2, 1, 2, 2):
66     return int(server_data[11])
67 elif alpha_beta_gamma == (2, 2, 1, 1):
68     return int(server_data[12])
69 elif alpha_beta_gamma == (2, 2, 1, 2):
70     return int(server_data[13])
71 elif alpha_beta_gamma == (2, 2, 2, 1):
72     return int(server_data[14])
73 elif alpha_beta_gamma == (2, 2, 2, 2):
74     return int(server_data[15])
75
76
77 def calculate_a(index) -> int:
78     res = 0
79     for k in index:
80         res += query_y(k)
81     return res % 2
82
83
84 def calculate_A1(u_v_w: tuple, q0000: list) -> tuple:
85     if u_v_w.count(1) == 1:
86         oplus_index = u_v_w.index(1)
87         if oplus_index == 1: # 0100
88             res1 = product(q0000[0], q0000[1].symmetric_difference({1}), q0000[2], q0000[3])
89             res2 = product(q0000[0], q0000[1].symmetric_difference({2}), q0000[2], q0000[3])
90         elif oplus_index == 2: # 0010
91             res1 = product(q0000[0], q0000[1], q0000[2].symmetric_difference({1}), q0000[3])
92             res2 = product(q0000[0], q0000[1], q0000[2].symmetric_difference({2}), q0000[3])

```

```

93         elif oplus_index == 3: # 0001
94             res1 = product(q0000[0], q0000[1], q0000[2], q0000[3].symmetric_difference({1}))
95             res2 = product(q0000[0], q0000[1], q0000[2], q0000[3].symmetric_difference({2}))
96
97         res1 = calculate_a(res1)
98         res2 = calculate_a(res2)
99         return res1, res2
100
101
102     def calculate_A2(u_v_w: tuple, q1111: list) -> tuple:
103         if u_v_w.count(1) == 3:
104             oplus_index = u_v_w.index(0)
105             if oplus_index == 1: # 1011
106                 res1 = product(q1111[0], q1111[1].symmetric_difference({1}), q1111[2], q1111[3])
107                 res2 = product(q1111[0], q1111[1].symmetric_difference({2}), q1111[2], q1111[3])
108             elif oplus_index == 2: # 1101
109                 res1 = product(q1111[0], q1111[1], q1111[2].symmetric_difference({1}), q1111[3])
110                 res2 = product(q1111[0], q1111[1], q1111[2].symmetric_difference({2}), q1111[3])
111             elif oplus_index == 3: # 1110
112                 res1 = product(q1111[0], q1111[1], q1111[2], q1111[3].symmetric_difference({1}))
113                 res2 = product(q1111[0], q1111[1], q1111[2], q1111[3].symmetric_difference({2}))
114
115             res1 = calculate_a(res1)
116             res2 = calculate_a(res2)
117             return res1, res2
118
119
120     def calculate_A3(u_v_w: tuple, q1000: list) -> tuple:
121         if u_v_w.count(1) == 2:
122             if u_v_w[0] == 1 and u_v_w[1] == 1: # 1100
123                 res1 = product(q1000[0], q1000[1].symmetric_difference({1}), q1000[2], q1000[3])
124                 res2 = product(q1000[0], q1000[1].symmetric_difference({2}), q1000[2], q1000[3])
125             elif u_v_w[0] == 1 and u_v_w[2] == 1: # 1010
126                 res1 = product(q1000[0], q1000[1], q1000[2].symmetric_difference({1}), q1000[3])
127                 res2 = product(q1000[0], q1000[1], q1000[2].symmetric_difference({2}), q1000[3])
128             elif u_v_w[0] == 1 and u_v_w[3] == 1: # 1001
129                 res1 = product(q1000[0], q1000[1], q1000[2], q1000[3].symmetric_difference({1}))
130                 res2 = product(q1000[0], q1000[1], q1000[2], q1000[3].symmetric_difference({2}))
131
132             res1 = calculate_a(res1)
133             res2 = calculate_a(res2)
134             return res1, res2
135
136
137     def calculate_A4(u_v_w: tuple, q0111: list) -> tuple:

```

```

137 def calculate_A4(u_v_w: tuple, q0111: list) -> tuple:
138     if u_v_w.count(1) == 2:
139         if u_v_w[1] == 1 and u_v_w[2] == 1: # 0110
140             res1 = product(q0111[0], q0111[1], q0111[2], q0111[3].symmetric_difference({1}))
141             res2 = product(q0111[0], q0111[1], q0111[2], q0111[3].symmetric_difference({2}))
142         elif u_v_w[1] == 1 and u_v_w[3] == 1: # 0101
143             res1 = product(q0111[0], q0111[1], q0111[2].symmetric_difference({1}), q0111[3])
144             res2 = product(q0111[0], q0111[1], q0111[2].symmetric_difference({2}), q0111[3])
145         elif u_v_w[2] == 1 and u_v_w[3] == 1: # 0011
146             res1 = product(q0111[0], q0111[1].symmetric_difference({1}), q0111[2], q0111[3])
147             res2 = product(q0111[0], q0111[1].symmetric_difference({2}), q0111[2], q0111[3])
148
149     res1 = calculate_a(res1)
150     res2 = calculate_a(res2)
151     return res1, res2
152
153
154 # Main Part
155 def query(k: int) -> tuple:
156     choice = [{1}, {2}, {1, 2}, set()]
157     q0000 = [choice[random.randint(0, 3)], choice[random.randint(0, 3)], choice[random.randint(0, 3)],
158             choice[random.randint(0, 3)]]
159     q1000 = [set(), set(), set(), set()]
160     q0111 = [set(), set(), set(), set()]
161     q1111 = [set(), set(), set(), set()]
162
163     right = omega_bijection(k)
164     q1111[0] = q0000[0].symmetric_difference({right[0]})
165     q1111[1] = q0000[1].symmetric_difference({right[1]})
166     q1111[2] = q0000[2].symmetric_difference({right[2]})
167
168     q1111[0] = q0000[0].symmetric_difference({right[0]})
169     q1111[1] = q0000[1].symmetric_difference({right[1]})
170     q1111[2] = q0000[2].symmetric_difference({right[2]})
171     q1111[3] = q0000[3].symmetric_difference({right[3]})
172
173     q1000[0] = q0000[0].symmetric_difference({right[0]})
174     q1111[1] = q0000[1]
175     q1111[2] = q0000[2]
176     q1111[3] = q0000[3]
177
178     q1000[0] = q0000[0]
179     q0111[1] = q0000[1].symmetric_difference({right[1]})
180     q0111[2] = q0000[2].symmetric_difference({right[2]})
181     q0111[3] = q0000[3].symmetric_difference({right[3]})
182

```



```

182
183     return q0000, q1111, q0111, q1000
184
185
186     def answer1(q0000: list):
187         a0000 = calculate_a([(list(q0000[0])[0], list(q0000[1])[0], list(q0000[2])[0], list(q0000[3])[0])])
188         A0100 = calculate_A1((0, 1, 0, 0), q0000)
189         A0010 = calculate_A1((0, 0, 1, 0), q0000)
190         A0001 = calculate_A1((0, 0, 0, 1), q0000)
191
192     return a0000, A0100, A0010, A0001
193
194
195     def answer2(q1111: list):
196         a1111 = calculate_a([(list(q1111[0])[0], list(q1111[1])[0], list(q1111[2])[0], list(q1111[3])[0])])
197         A1011 = calculate_A2((1, 0, 1, 1), q1111)
198         A1101 = calculate_A2((1, 1, 0, 1), q1111)
199         A1110 = calculate_A2((1, 1, 1, 0), q1111)
200
201     return a1111, A1011, A1101, A1110
202
203
204     def answer3(q1000: list):
205         a1000 = calculate_a([(list(q1000[0])[0], list(q1000[1])[0], list(q1000[2])[0], list(q1000[3])[0])])
206         A1100 = calculate_A3((1, 1, 0, 0), q1000)
207         A1010 = calculate_A3((1, 0, 1, 0), q1000)
208         A1001 = calculate_A3((1, 0, 0, 1), q1000)
209
210     return a1000, A1100, A1010, A1001
211
212
213     def answer4(q0111: list):
214         a0001 = calculate_a([(list(q0111[0])[0], list(q0111[1])[0], list(q0111[2])[0], list(q0111[3])[0])])
215         A0011 = calculate_A4((0, 0, 1, 1), q0111)
216         A0101 = calculate_A4((0, 1, 0, 1), q0111)
217         A0110 = calculate_A4((0, 1, 1, 0), q0111)
218
219     return a0001, A0011, A0101, A0110
220
221
222     def reconstruction(k: int):
223         omega = omega_bijection(k)
224         ans1 = answer1(query(k)[0])
225         ans2 = answer2(query(k)[1])
226         ans3 = answer3(query(k)[2])
227         ans4 = answer4(query(k)[3])

```



```
    for h in range(3):
        res += ans1[h + 1][omega[h + 1] - 1]
        res += ans2[h + 1][omega[h + 1] - 1]
        res += ans3[h + 1][omega[h + 1] - 1]
        res += ans4[h + 1][omega[h + 1] - 1]

    return res % 2

if __name__ == '__main__':
    print("res: ", reconstruction(int(input())))
```

3

3.1

```
In [1]: p = 1041857  
        q = 716809
```

```
In [2]: N = p * q  
        N
```

```
Out[2]: 746812474313
```

```
In [3]: N_square = N * N  
        N_square
```

```
Out[3]: 557728871789505284821969
```

```
In [4]: g = N + 1  
        g
```

```
Out[4]: 746812474314
```

```
In [5]: phi_N = (p - 1) * (q - 1)  
        phi_N
```

```
Out[5]: 746810715648
```

```
In [6]: pk=(N, g)  
        sk = phi_N
```

```
In [7]: x1 = 726095811532  
        r1 = 270134931749  
        x2 = 450864083576  
        r2 = 378141346340
```

```
In [8]: y1 = mod(pow(g, x1, N_square) * pow(r1, N, N_square), N_square)  
        y1
```

```
Out[8]: 179099620913615548532981
```

```
In [9]: y2 = mod(pow(g, x2, N_square) * pow(r2, N, N_square), N_square)  
        y2
```

```
Out[9]: 237320554928851933110533
```

3.2

```
In [10]: y3 = mod(y1 * y2, N_square)
         y3
```

```
Out[10]: 334681468951155330185827
```

```
In [11]: x3 = mod((int(pow(y3, phi_N, N_square) - 1) / N) * inverse_mod(phi_N, N), N)
         x3
```

```
Out[11]: 430147420795
```

3.3

```
-----  
In [12]: x3 == mod(x1 + x2, N)  
Out[12]: True
```

```

In [1]: import random
p = Primes().unrank(random.randint(1, 1000000))
q = Primes().unrank(random.randint(1, 1000000))
while (p == q):
    q = Primes().unrank(random.randint(1, 1000000))
phi_N = (p - 1) * (q - 1)
N = p * q
N_square = N * N
server_data = [[0, 1, 0], [1, 0, 1], [0, 1, 0]]

In [2]: def omega_bijection(i: int) -> tuple:
    if i == 1:
        return 1, 1
    elif i == 2:
        return 1, 2
    elif i == 3:
        return 1, 3
    elif i == 4:
        return 2, 1
    elif i == 5:
        return 2, 2
    elif i == 6:
        return 2, 3
    elif i == 7:
        return 3, 1
    elif i == 8:
        return 3, 2
    elif i == 9:
        return 3, 3

In [3]: def query(i: int):
    omega = omega_bijection(i)
    r = (random.randint(1, N), random.randint(1, N), random.randint(1, N))
    y = list()
    y.append(pow(r[0], N, N_square))
    y.append(pow(r[1], N, N_square))
    y.append(pow(r[2], N, N_square))
    y[omega[1] - 1] = mod((1 + N) * y[2], N_square)

    return y

In [4]: def answer(y: list):
    a1 = mod(pow(y[0], server_data[0][0], N_square) * pow(y[1], server_data[0][1], N_square) * pow(y[2], server_data[0][2], N_square), N_square)
    a2 = mod(pow(y[0], server_data[1][0], N_square) * pow(y[1], server_data[1][1], N_square) * pow(y[2], server_data[1][2], N_square), N_square)
    a3 = mod(pow(y[0], server_data[2][0], N_square) * pow(y[1], server_data[2][1], N_square) * pow(y[2], server_data[2][2], N_square), N_square)

    return [a1, a2, a3]

In [5]: def restruction(i: int, a: list):
    omega = omega_bijection(i)
    a = a[omega[0] - 1]
    z = mod(pow(a, phi_N, N_square) - 1, N_square)
    x = mod((int(z) / N) * inverse_mod(phi_N, N), N)

    return x

In [6]: i = int(input())
restruction(i, answer(query(i)))

```

6

Out[6]: 1