CS152-13

Hongchen Cao 2019533114

2020/12/20

1

```python
from itertools import product
import random

server_data = "01001101"  # change server data here


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


def query_y(alpha_beta_gamma: tuple) -> int:
    if alpha_beta_gamma == (1, 1, 1):
        return int(server_data[0])
    elif alpha_beta_gamma == (1, 1, 2):
        return int(server_data[1])
    elif alpha_beta_gamma == (1, 2, 1):
        return int(server_data[2])
    elif alpha_beta_gamma == (1, 2, 2):
        return int(server_data[3])
    elif alpha_beta_gamma == (2, 1, 1):
        return int(server_data[4])
    elif alpha_beta_gamma == (2, 1, 2):
        return int(server_data[5])
    elif alpha_beta_gamma == (2, 2, 1):
        return int(server_data[6])
    elif alpha_beta_gamma == (2, 2, 2):
        return int(server_data[7])
```

```python
def calculate_a(index) -> int:
    res = 0
    for i in index:
        res += query_y(i)
    return res % 2


def calculate_A1(u_v_w: tuple, q000) -> tuple:
    if u_v_w.count(1) == 1:
        oplus_index = u_v_w.index(1)
        if oplus_index == 0:
            res1 = product(q000[0].symmetric_difference({1}), q000[1], q000[2])
            res2 = product(q000[0].symmetric_difference({2}), q000[1], q000[2])
        elif oplus_index == 1:
            res1 = product(q000[0], q000[1].symmetric_difference({1}), q000[2])
            res2 = product(q000[0], q000[1].symmetric_difference({2}), q000[2])
        elif oplus_index == 2:
            res1 = product(q000[0], q000[1], q000[2].symmetric_difference({1}))
            res2 = product(q000[0], q000[1], q000[2].symmetric_difference({2}))

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2


def calculate_A2(u_v_w: tuple, q111) -> tuple:
    if u_v_w.count(1) == 2:
        oplus_index = u_v_w.index(0)
        if oplus_index == 0:
            res1 = product(q111[0].symmetric_difference({1}), q111[1], q111[2])
            res2 = product(q111[0].symmetric_difference({2}), q111[1], q111[2])
        elif oplus_index == 1:
            res1 = product(q111[0], q111[1].symmetric_difference({1}), q111[2])
            res2 = product(q111[0], q111[1].symmetric_difference({2}), q111[2])
        elif oplus_index == 2:
            res1 = product(q111[0], q111[1], q111[2].symmetric_difference({1}))
            res2 = product(q111[0], q111[1], q111[2].symmetric_difference({2}))

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2
```

```python
# Main Part
def query(k: int) -> tuple:
    choice = [{1}, {2}, {1, 2}, set()]
    q000 = [choice[random.randint(0, 3)], choice[random.randint(0, 3)], choice[random.randint(0, 3)]]
    q111 = [set(), set(), set()]
    right = omega_bijection(k)
    q111[0] = q000[0].symmetric_difference({right[0]})
    q111[1] = q000[1].symmetric_difference({right[1]})
    q111[2] = q000[2].symmetric_difference({right[2]})
    return q000, q111


def answer1(q000: list):
    a000 = calculate_a([(list(q000[0])[0], list(q000[1])[0], list(q000[2])[0])])
    A100 = calculate_A1((1, 0, 0), q000)
    A010 = calculate_A1((0, 1, 0), q000)
    A001 = calculate_A1((0, 0, 1), q000)

    return a000, A100, A010, A001


def answer2(q111: list):
    a111 = calculate_a(product(q111[0], q111[1], q111[2]))
    A011 = calculate_A2((0, 1, 1), q111)
    A101 = calculate_A2((1, 0, 1), q111)
    A110 = calculate_A2((1, 1, 0), q111)

    return a111, A011, A101, A110


def restruction(k: int):
    omega = omega_bijection(k)
    ans1 = answer1(query(k)[0])
    ans2 = answer2(query(k)[1])
    res = ans1[0] + ans2[0]

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

    return res % 2


if __name__ == '__main__':
    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.

```python
from itertools import product
import random

server_data = "0101101001011010"  # change server data here


def omega_bijection(k: int) -> tuple:
    if k == 1:
        return 1, 1, 1, 1
    elif k == 2:
        return 1, 1, 1, 2
    elif k == 3:
        return 1, 1, 2, 1
    elif k == 4:
        return 1, 1, 2, 2
    elif k == 5:
        return 1, 2, 1, 1
    elif k == 6:
        return 1, 2, 1, 2
    elif k == 7:
        return 1, 2, 2, 1
    elif k == 8:
        return 1, 2, 2, 2
    elif k == 9:
        return 2, 1, 1, 1
    elif k == 10:
        return 2, 1, 1, 2
    elif k == 11:
        return 2, 1, 2, 1
    elif k == 12:
        return 2, 1, 2, 2
    elif k == 13:
        return 2, 2, 1, 1
    elif k == 14:
        return 2, 2, 1, 2
    elif k == 15:
        return 2, 2, 2, 1
    elif k == 16:
        return 2, 2, 2, 2


def query_y(alpha_beta_gamma: tuple) -> int:
    if alpha_beta_gamma == (1, 1, 1, 1):
        return int(server_data[0])
    elif alpha_beta_gamma == (1, 1, 1, 2):
        return int(server_data[1])
    elif alpha_beta_gamma == (1, 1, 2, 1):
```

```python
        return int(server_data[2])
    elif alpha_beta_gamma == (1, 1, 2, 2):
        return int(server_data[3])
    elif alpha_beta_gamma == (1, 2, 1, 1):
        return int(server_data[4])
    elif alpha_beta_gamma == (1, 2, 1, 2):
        return int(server_data[5])
    elif alpha_beta_gamma == (1, 2, 2, 1):
        return int(server_data[6])
    elif alpha_beta_gamma == (1, 2, 2, 2):
        return int(server_data[7])
    elif alpha_beta_gamma == (2, 1, 1, 1):
        return int(server_data[8])
    elif alpha_beta_gamma == (2, 1, 1, 2):
        return int(server_data[9])
    elif alpha_beta_gamma == (2, 1, 2, 1):
        return int(server_data[10])
    elif alpha_beta_gamma == (2, 1, 2, 2):
        return int(server_data[11])
    elif alpha_beta_gamma == (2, 2, 1, 1):
        return int(server_data[12])
    elif alpha_beta_gamma == (2, 2, 1, 2):
        return int(server_data[13])
    elif alpha_beta_gamma == (2, 2, 2, 1):
        return int(server_data[14])
    elif alpha_beta_gamma == (2, 2, 2, 2):
        return int(server_data[15])


def calculate_a(index) -> int:
    res = 0
    for k in index:
        res += query_y(k)
    return res % 2


def calculate_A1(u_v_w: tuple, q0000: list) -> tuple:
    if u_v_w.count(1) == 1:
        oplus_index = u_v_w.index(1)
        if oplus_index == 1:  # 0100
            res1 = product(q0000[0], q0000[1].symmetric_difference({1}), q0000[2], q0000[3])
            res2 = product(q0000[0], q0000[1].symmetric_difference({2}), q0000[2], q0000[3])
        elif oplus_index == 2:  # 0010
            res1 = product(q0000[0], q0000[1], q0000[2].symmetric_difference({1}), q0000[3])
            res2 = product(q0000[0], q0000[1], q0000[2].symmetric_difference({2}), q0000[3])
```

```python
        elif oplus_index == 3:  # 0001
            res1 = product(q0000[0], q0000[1], q0000[2], q0000[3].symmetric_difference({1}))
            res2 = product(q0000[0], q0000[1], q0000[2], q0000[3].symmetric_difference({2}))

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2


def calculate_A2(u_v_w: tuple, q1111: list) -> tuple:
    if u_v_w.count(1) == 3:
        oplus_index = u_v_w.index(0)
        if oplus_index == 1:  # 1011
            res1 = product(q1111[0], q1111[1].symmetric_difference({1}), q1111[2], q1111[3])
            res2 = product(q1111[0], q1111[1].symmetric_difference({2}), q1111[2], q1111[3])
        elif oplus_index == 2:  # 1101
            res1 = product(q1111[0], q1111[1], q1111[2].symmetric_difference({1}), q1111[3])
            res2 = product(q1111[0], q1111[1], q1111[2].symmetric_difference({2}), q1111[3])
        elif oplus_index == 3:  # 1110
            res1 = product(q1111[0], q1111[1], q1111[2], q1111[3].symmetric_difference({1}))
            res2 = product(q1111[0], q1111[1], q1111[2], q1111[3].symmetric_difference({2}))

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2


def calculate_A3(u_v_w: tuple, q1000: list) -> tuple:
    if u_v_w.count(1) == 2:
        if u_v_w[0] == 1 and u_v_w[1] == 1:  # 1100
            res1 = product(q1000[0], q1000[1].symmetric_difference({1}), q1000[2], q1000[3])
            res2 = product(q1000[0], q1000[1].symmetric_difference({2}), q1000[2], q1000[3])
        elif u_v_w[0] == 1 and u_v_w[2] == 1:  # 1010
            res1 = product(q1000[0], q1000[1], q1000[2].symmetric_difference({1}), q1000[3])
            res2 = product(q1000[0], q1000[1], q1000[2].symmetric_difference({2}), q1000[3])
        elif u_v_w[0] == 1 and u_v_w[3] == 1:  # 1001
            res1 = product(q1000[0], q1000[1], q1000[2], q1000[3].symmetric_difference({1}))
            res2 = product(q1000[0], q1000[1], q1000[2], q1000[3].symmetric_difference({2}))

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2


def calculate_A4(u_v_w: tuple, q0111: list) -> tuple:
```

```python
def calculate_A4(u_v_w: tuple, q0111: list) -> tuple:
    if u_v_w.count(1) == 2:
        if u_v_w[1] == 1 and u_v_w[2] == 1:  # 0110
            res1 = product(q0111[0], q0111[1], q0111[2], q0111[3].symmetric_difference({1}))
            res2 = product(q0111[0], q0111[1], q0111[2], q0111[3].symmetric_difference({2}))
        elif u_v_w[1] == 1 and u_v_w[3] == 1:  # 0101
            res1 = product(q0111[0], q0111[1], q0111[2].symmetric_difference({1}), q0111[3])
            res2 = product(q0111[0], q0111[1], q0111[2].symmetric_difference({2}), q0111[3])
        elif u_v_w[2] == 1 and u_v_w[3] == 1:  # 0011
            res1 = product(q0111[0], q0111[1].symmetric_difference({1}), q0111[2], q0111[3])
            res2 = product(q0111[0], q0111[1].symmetric_difference({2}), q0111[2], q0111[3])

    res1 = calculate_a(res1)
    res2 = calculate_a(res2)
    return res1, res2


# Main Part
def query(k: int) -> tuple:
    choice = [{1}, {2}, {1, 2}, set()]
    q0000 = [choice[random.randint(0, 3)], choice[random.randint(0, 3)], choice[random.randint(0, 3)],
             choice[random.randint(0, 3)]]
    q1000 = [set(), set(), set(), set()]
    q0111 = [set(), set(), set(), set()]
    q1111 = [set(), set(), set(), set()]

    right = omega_bijection(k)
    q1111[0] = q0000[0].symmetric_difference({right[0]})
    q1111[1] = q0000[1].symmetric_difference({right[1]})
    q1111[2] = q0000[2].symmetric_difference({right[2]})

    q1111[0] = q0000[0].symmetric_difference({right[0]})
    q1111[1] = q0000[1].symmetric_difference({right[1]})
    q1111[2] = q0000[2].symmetric_difference({right[2]})
    q1111[3] = q0000[3].symmetric_difference({right[3]})

    q1000[0] = q0000[0].symmetric_difference({right[0]})
    q1111[1] = q0000[1]
    q1111[2] = q0000[2]
    q1111[3] = q0000[3]

    q1000[0] = q0000[0]
    q0111[1] = q0000[1].symmetric_difference({right[1]})
    q0111[2] = q0000[2].symmetric_difference({right[2]})
    q0111[3] = q0000[3].symmetric_difference({right[3]})

```

```python

    return q0000, q1111, q0111, q1000


def answer1(q0000: list):
    a0000 = calculate_a([(list(q0000[0])[0], list(q0000[1])[0], list(q0000[2])[0], list(q0000[3])[0])])
    A0100 = calculate_A1((0, 1, 0, 0), q0000)
    A0010 = calculate_A1((0, 0, 1, 0), q0000)
    A0001 = calculate_A1((0, 0, 0, 1), q0000)

    return a0000, A0100, A0010, A0001


def answer2(q1111: list):
    a1111 = calculate_a([(list(q1111[0])[0], list(q1111[1])[0], list(q1111[2])[0], list(q1111[3])[0])])
    A1011 = calculate_A2((1, 0, 1, 1), q1111)
    A1101 = calculate_A2((1, 1, 0, 1), q1111)
    A1110 = calculate_A2((1, 1, 1, 0), q1111)

    return a1111, A1011, A1101, A1110


def answer3(q1000: list):
    a1000 = calculate_a([(list(q1000[0])[0], list(q1000[1])[0], list(q1000[2])[0], list(q1000[3])[0])])
    A1100 = calculate_A3((1, 1, 0, 0), q1000)
    A1010 = calculate_A3((1, 0, 1, 0), q1000)
    A1001 = calculate_A3((1, 0, 0, 1), q1000)

    return a1000, A1100, A1010, A1001


def answer4(q0111: list):
    a0001 = calculate_a([(list(q0111[0])[0], list(q0111[1])[0], list(q0111[2])[0], list(q0111[3])[0])])
    A0011 = calculate_A4((0, 0, 1, 1), q0111)
    A0101 = calculate_A4((0, 1, 0, 1), q0111)
    A0110 = calculate_A4((0, 1, 1, 0), q0111)

    return a0001, A0011, A0101, A0110


def restruction(k: int):
    omega = omega_bijection(k)
    ans1 = answer1(query(k)[0])
    ans2 = answer2(query(k)[1])
    ans3 = answer3(query(k)[2])
    ans4 = answer4(query(k)[3])
```

```python
    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: ", restruction(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
```

4

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