

# A hypothetical way to reduce the search space when finding the structure seed from structure positions

## Assumptions

Let's assume we found a structure and we know its chunk position. Here's the relationship, assuming the [document](#) is true:

$$s_0 = (w + Px_r + Qz_r + salt) \oplus M \pmod{2^{48}}$$

$$x_c = x_r \cdot spacing + (s_1 \gg 17) \bmod bound$$

$$z_c = z_r \cdot spacing + (s_2 \gg 17) \bmod bound$$

I will also claim the following relationships are true:

$$x \gg n = \frac{x - (x \bmod 2^n)}{2^n}$$

$$(m \cdot x) \bmod m \cdot n = m \cdot (x \bmod n)$$

where  $m, n \in \mathbb{N}$

If  $(a \bmod n) + (b \bmod n) \geq n$ :

$$(a + b) \bmod n = (a \bmod n) + (b \bmod n) - n$$

If  $(a \bmod n) + (b \bmod n) < n$ :

$$(a + b) \bmod n = (a \bmod n) + (b \bmod n)$$

## Derivation

Let's rearrange the equations in terms of  $s_i$ :

$$s_0 = (w + Px_r + Qz_r + salt) \oplus M \pmod{2^{48}}$$

$$(s_1 \gg 17) \bmod bound = x_c - x_r \cdot spacing$$

$$(s_2 \gg 17) \bmod bound = z_c - z_r \cdot spacing$$

Let's do a bit of renaming here:

- $Px_r + Qz_r + salt = D$
- $x_c - x_r \cdot spacing = \Delta x$
- $z_c - z_r \cdot spacing = \Delta z$

We get:

$$s_0 = (w + D) \oplus M \pmod{2^{48}}$$

$$(s_1 \gg 17) \bmod bound = \Delta x$$

$$(s_2 \gg 17) \bmod bound = \Delta z$$

Notice that  $\lfloor \log_2(M) \rfloor = 34$ , which means that it affects this amount of low bits plus one. But we still have  $48 - 35 = 13$  unaffected high bits. And some other bits are sprinkled here and there in the middle.

Rewriting the first equation to account for that:

$$s_0 = 2^{35} \cdot (w + D) \gg 35 + e_0 \pmod{2^{48}}$$

$$(s_1 \gg 17) \bmod bound = \Delta x$$

$$(s_2 \gg 17) \bmod bound = \Delta z$$

where:

- $e_0 = (w + D) \oplus M \pmod{2^{35}}$

So, at the cost of adding another variable, albeit small, we are able to make the expression slightly more algebraically malleable. We could keep  $2^{35} \cdot (w + D) \gg 35$  as it is, but the problem is that this expression varies for each structure. Since we are trying to combine constraints here, we'll have to split it up.

What I know for certain is that we can define bitshifts as such:

$$x \gg n = \frac{x - x \bmod 2^n}{2^n}$$

Also, splitting  $(a + b) \bmod 2^n$  into  $a \bmod 2^n + b \bmod 2^n$  is possible, the problem is that when we do that, we get a carry condition, which is basically another variable, although it can only take on two states, which is a lot easier to deal with, than a full on variable.

At the same time,  $D$  is already known for each structure. Then, after splitting the mod bracket, we'll find ourselves with  $w - w \bmod 2^{35}$ , which can be effectively called  $w_h$ ,  $h$  stands for "the high bits of  $w$ ". Feels like this is a good option to avoid collecting another variable.

With that in mind, let's do a bit of algebra on  $s_0$ :

$$s_0 = 2^{35} \cdot (w + D) \gg 35 + e_0 \pmod{2^{48}}$$

Expanding our bitshift definition:

$$s_0 = w + D - (w + D) \bmod 2^{35} + e_0 \pmod{2^{48}}$$

Now, we know that:

If  $w \bmod 2^{35} + D \bmod 2^{35} \geq 2^{35}$ :

$$(w + D) \bmod 2^{35} = w \bmod 2^{35} + D \bmod 2^{35} - 2^{35}$$

If  $w \bmod 2^{35} + D \bmod 2^{35} < 2^{35}$ :

$$(w + D) \bmod 2^{35} = w \bmod 2^{35} + D \bmod 2^{35}$$

Effectively, our expression becomes this:

$$(w + D) \bmod 2^{35} = w \bmod 2^{35} + D \bmod 2^{35} - 2^{35} \cdot c_0$$

where:

- $c_0 \in \{0, 1\}$

Notice that we also will subtract  $D \bmod 2^{35}$  from  $D$ , which will also leave us with just the higher bits of  $D$ . This means that:

$$2^{35} \cdot (w + D) \gg 35 = w + D - w \bmod 2^{35} - D \bmod 2^{35} + c_0 \cdot 2^{35} = 2^{35} \cdot (w \gg 35 + D \gg 35 + c_0)$$

This property extends to arbitrary  $a$  and  $b$ , so we found a way to give bitshifts almost linear properties. Almost is good enough here.

As discussed earlier, let's rename  $w \gg 35$  to  $w_h$  and  $D \gg 35$  to  $D_h$  to reduce clutter.

Let's use our new property, then:

$$s_0 = 2^{35} \cdot (w_h + D_h + c_0) + e_0 \pmod{2^{48}}$$

where:

- $e_0 = (w + D) \oplus M \pmod{2^{35}}$

In other words:

$$2^{35} \cdot w_h = s_0 - 2^{35} \cdot (D_h + c_0) - e_0 \pmod{2^{48}}$$

where:

- $e_0 = (w + D) \oplus M \pmod{2^{35}}$

Now let's look at the other two equations:

$$(s_1 \gg 17) \bmod bound = \Delta x$$

$$(s_2 \gg 17) \bmod bound = \Delta z$$

Let's expand  $\bmod bound$ :

$$(s_1 \gg 17) = \Delta x + k_1 \cdot bound$$

$$(s_2 \gg 17) = \Delta z + k_2 \cdot bound$$

Substitute in our definitions of  $s_1$  and  $s_2$ :

$$(((s_0 \cdot M + A) \bmod 2^{48}) \gg 17) = \Delta x + k_1 \cdot bound$$

$$(((s_0 \cdot M^2 + A \cdot (M + 1)) \bmod 2^{48}) \gg 17) = \Delta z + k_2 \cdot bound$$

Now, remember the definition of  $s_0$ :

$$s_0 = 2^{35} \cdot (w_h + D_h + c_0) + e_0 \pmod{2^{48}}$$

Let's.. look at the first equation.

$$(((s_0 \cdot M + A) \bmod 2^{48}) \gg 17) = \Delta x + k_1 \cdot bound$$

Let's substitute in our definition of  $s_0$ :

$$((((2^{35} \cdot (w_h + D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17) = \Delta x + k_1 \cdot bound$$

If we expand the left hand side of our equation using the bitshift formula, we get the following:

$$(((2^{35} \cdot (w_h + D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) - (((2^{35} \cdot (w_h + D_h + c_0) + e_0) \cdot M + A) \bmod 2^{17})$$

Say, we wanna extract only  $w_h$ , as that's constant across... many equations. We'll be splitting that left bracket into two mod brackets, albeit with additional carry added to the mix. My point is, let's drop only  $w_h$  in the second term:

$$(((2^{35} \cdot (w_h + D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) - (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{17})$$

If we split purely the first bracket, we get:

$$((2^{35} \cdot M \cdot w_h) \bmod 2^{48}) + (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) - c_1 \cdot 2^{48}$$

where:

- $c_1 = 1$  if:

$$((2^{35} \cdot M \cdot w_h) \bmod 2^{48}) + (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \geq 2^{48}$$

- $c_1 = 0$  if:

$$((2^{35} \cdot M \cdot w_h) \bmod 2^{48}) + (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) < 2^{48}$$

Notice that the terms inside the first bracket match exactly. And we know that:

$$(a \bmod 2^{48}) \bmod 2^{35} = a \bmod 2^{35}$$

With that, it looks like we can use our definition of a bitshift here. With this in mind, we get:

$$\begin{aligned} 2^{35} \cdot ((M \cdot w_h) \bmod 2^{13}) + 2^{17} \cdot (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17 - c_1 \cdot 2^{48} = \\ = 2^{17} \cdot (\Delta x + k_1 \cdot bound) \end{aligned}$$

Divide everything by  $2^{17}$ :

$$\begin{aligned} 2^{18} \cdot ((M \cdot w_h) \bmod 2^{13}) + (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17 - c_1 \cdot 2^{31} = \\ = \Delta x + k_1 \cdot bound \end{aligned}$$

Leave the constant part on the left:

$$\begin{aligned} 2^{18} \cdot ((M \cdot w_h) \bmod 2^{13}) = \Delta x + k_1 \cdot bound - (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17 + \\ + c_1 \cdot 2^{31} \end{aligned}$$

We have.. still quite a lot of variables to deal with. But, if we take this expression mod bound, we can get rid of  $k_i$ :

$$2^{18} \cdot ((M \cdot w_h) \bmod 2^{13}) = \Delta x - (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17 + c_1 \cdot 2^{31} \pmod{\text{bound}}$$

Also note that we can transform this constraint to work for the second equations as well by replacing  $M$  with  $M^2$ ,  $A$  with  $A \cdot (M + 1)$ ,  $\Delta x$  with  $\Delta z$ ,  $c_1$  with  $c_2$ .

$\text{bound}$  is usually pretty small. In the case for trial chambers,  $\text{bound} = 22$ . So, I guess we could bruteforce  $w_l$  with  $c_0$ ,  $c_1$  and generate  $4 \cdot \text{bound}$  sets of solutions. But, if we look at the left hand side, we can that it's divisible by  $2^{18}$ . By using the  $(m \cdot x) \bmod m \cdot n = m \cdot (x \bmod n)$  property, if we label  $g = \gcd(2^{18}, \text{bound})$  we get:

$$(2^{18} \cdot ((M \cdot w_h) \bmod 2^{13})) \bmod \text{bound} = g \cdot (((2^{18} \cdot (M \cdot w_h) \bmod 2^{13}) / g) \bmod (\text{bound} / g))$$

This implies that there are only  $\text{bound} / g$  possible values for  $2^{18} \cdot ((M \cdot w_h) \bmod 2^{13})$ .

Let's look at the expression mod  $\gcd(2^{18}, \text{bound})$ . Let's see what happens:

$$0 = \Delta x - (((2^{35} \cdot (D_h + c_0) + e_0) \cdot M + A) \bmod 2^{48}) \gg 17 \pmod{\gcd(2^{18}, \text{bound})}$$

Pretty sure we can also drop  $2^{35}$  here:

$$\Delta x - (((e_0 \cdot M + A) \bmod 2^{48}) \gg 17) = 0 \pmod{\gcd(2^{18}, \text{bound})}$$

- *To prove that you can do that, you'd need to expand the bitshift definition, split off the  $2^{35} \cdot M \cdot (D_h + c_0)$  term from the mod bracket. Notice that you can move  $2^{35}$  from  $\bmod 2^{48}$  as  $2^{35}$  divides  $2^{48}$ . But  $2^{35}$  is divisible by  $\gcd(2^{48}, \text{bound})$ , so that term vanishes.*

We can swap the position of a bitshift and mod here:

$$\Delta x - ((e_0 \cdot M + A) \gg 17) \bmod 2^{31} = 0 \pmod{\gcd(2^{18}, \text{bound})}$$

Drop the  $\bmod 2^{31}$  under our new mod:

$$\Delta x - ((e_0 \cdot M + A) \gg 17) = 0 \pmod{\gcd(2^{18}, \text{bound})}$$

This is quite a nice constraint. We could go a bit further and claim that we can construct a similar relationship for the second equation:

$$\Delta z = (((e_0 \cdot M^2 + A \cdot (M + 1)) \gg 17) \pmod{\gcd(2^{18}, \text{bound})})$$

## The hypothetical algorithm

We'll be bruteforcing  $w_l$ . Let's fix  $c_0$  and  $c_1$ . We'll assume we don't know  $w_h$ . Since we don't know  $w_h$ , we don't know  $2^{18} \cdot ((M \cdot w_h) \bmod 2^{13})$  or  $2^{18} \cdot ((M^2 \cdot w_h) \bmod 2^{13})$ . But what we do know is that for each constraint  $2^{18} \cdot ((M \cdot w_h) \bmod 2^{13})$  will be equal. Same goes for  $2^{18} \cdot ((M^2 \cdot w_h) \bmod 2^{13})$ .

Since we are computing these modulo  $\text{bound}$ , these expressions can take on  $\text{bound}$  values at most.

After we do all that, we should have a few candidates remaining. For each remaining candidate we generate  $w_h$  and check for  $c_0$  and  $c_1$  conditions. If we still have more than one candidate remaining, just plug it in the original constraints and see.

Here's the code I sloppily put together in Python to test these ideas:

```
from math import gcd

class LCG:
    def __init__(self, seed: int):
        self.M = 25214903917
        self.A = 11
        self.modulus = 1 << 48
        self.state = (seed ^ self.M) % self.modulus

    def nextSeed(self) -> int:
        self.state = (self.state * self.M + self.A) % self.modulus
        return self.state

    def next(self, numBits: int) -> int:
        if not 1 <= numBits <= 48:
            raise ValueError("numBits must be between 1 and 48")
        self.nextSeed()
        return self.state >> (48 - numBits)

    def nextInt(self, bound: int) -> int:
        if bound <= 0:
            raise ValueError("bound must be positive")

        while True:
            bits = self.next(31)
            value = bits % bound

            if bits - value + (bound - 1) >= 0:
                return value
            print("Warning: nextInt regenerated a value!")

    def getCurrentSeed(self) -> int:
        return self.state

def region_lcg(w, x_r, z_r, salt):
    P = 341873128712
    Q = 132897987541

    return LCG(w + P*x_r + Q*z_r + salt)

if __name__ == "__main__":
```

```

M = 25214903917
A = 11
P = 341873128712
Q = 132897987541

w = 9189798541153775729 % 2**48
spacing = 34
separation = 12
bound = spacing - separation
salt = 94251327

# Generating the info tuple:
# (x_r, z_r, x_c, z_c)
lcgA = region_lcg(w, 1, 1, salt)
infoA = (
    1,
    1,
    1*spacing + lcgA.nextInt(bound),
    1*spacing + lcgA.nextInt(bound),
)

lcgB = region_lcg(w, 2, 2, salt)
infoB = (
    2,
    2,
    2*spacing + lcgB.nextInt(bound),
    2*spacing + lcgB.nextInt(bound),
)

lcgC = region_lcg(w, 3, 3, salt)
infoC = (
    3,
    3,
    3*spacing + lcgC.nextInt(bound),
    3*spacing + lcgC.nextInt(bound),
)

lcgD = region_lcg(w, 64, 34, salt)
infoD = (
    64,
    34,
    64*spacing + lcgD.nextInt(bound),
    34*spacing + lcgD.nextInt(bound),
)

print(f"Structure seed: {w}")
info_list = [infoA, infoB, infoC, infoD]
print(f"Constraints: {info_list}")

```

```

# Testing if 13 higher bits of s_0 are equal to 13 higher bits of w
print("=*10 + f" w_h Check. w_h = {w >> 35} " + "=*10)

x_r, z_r, x_c, z_c = infoA
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0 = 1
else:
    c_0 = 0

w_h = ((s_0 >> 35) - (D_h + c_0)) % 2**48
print(f"infoA w_h = {w_h}, c_0 = {c_0}")

x_r, z_r, x_c, z_c = infoB
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0 = 1
else:
    c_0 = 0

w_h = ((s_0 >> 35) - (D_h + c_0)) % 2**48
print(f"infoB w_h = {w_h}, c_0 = {c_0}")

x_r, z_r, x_c, z_c = infoC
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0 = 1
else:
    c_0 = 0

w_h = ((s_0 >> 35) - (D_h + c_0)) % 2**48
print(f"infoC w_h = {w_h}, c_0 = {c_0}")

```

```

x_r, z_r, x_c, z_c = infoD
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0 = 1
else:
    c_0 = 0

w_h = ((s_0 >> 35) - (D_h + c_0)) % 2**48
print(f"infoD w_h = {w_h}, c_0 = {c_0}")

# Testing the first e_0 constraint formula
print("===== Testing the first constraint formula (all of
these should be 0)")
x_r, z_r, x_c, z_c = infoA
D = (P*x_r + Q*z_r + salt) % 2**48

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

X = ( dx - ((e_0*M + A) >> 17) ) % gcd(2**18, bound)
print(f"infoA: {X}")

x_r, z_r, x_c, z_c = infoB
D = (P*x_r + Q*z_r + salt) % 2**48

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

X = ( dx - ((e_0*M + A) >> 17) ) % gcd(2**18, bound)
print(f"infoB: {X}")

x_r, z_r, x_c, z_c = infoC
D = (P*x_r + Q*z_r + salt) % 2**48

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing

```

```

dz = z_c - z_r*spacing

X = ( dx - ((e_0*M + A) >> 17) ) % gcd(2**18, bound)
print(f"infoC: {X}")

x_r, z_r, x_c, z_c = infoD
D = (P*x_r + Q*z_r + salt) % 2**48

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

X = ( dx - ((e_0*M + A) >> 17) ) % gcd(2**18, bound)
print(f"infoD: {X}")

# Testing the second e_0 constraint formula for all c_0 and c_1
# (the one that barely fits on the page)
print("===== Testing the second constraint formula (X should
be equal for all constraints)")
# Testing infoA
x_r, z_r, x_c, z_c = infoA
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0_true = 1
else:
    c_0_true = 0

w_h = ((s_0 >> 35) - (D_h + c_0_true)) % 2**48

A_true = (2**35 * M * w_h) % 2**48
B_true = ((2**35 * (D_h + c_0_true) + e_0) * M + A) % 2**48

if A_true + B_true >= 2**48:
    c_1_true = 1
else:
    c_1_true = 0

X_true = 2**18 * ((M * w_h) % 2**13) % bound

print(f"infoA: c_0_true = {c_0_true}, c_1_true = {c_1_true}")
print(f"X_true = {X_true}")
for c_0 in range(2):

```

```

        for c_1 in range(2):
            B = ((2**35 * (D_h + c_0) + e_0) * M + A) % 2**48

            X = (dx - (B >> 17) + c_1*2**31) % bound
            print(f"c_0={c_0},\tc_1={c_1},\tX={X}")

# Testing infoB
x_r, z_r, x_c, z_c = infoB
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0_true = 1
else:
    c_0_true = 0

w_h = ((s_0 >> 35) - (D_h + c_0_true)) % 2**48

A_true = (2**35 * M * w_h) % 2**48
B_true = ((2**35 * (D_h + c_0_true) + e_0) * M + A) % 2**48

if A_true + B_true >= 2**48:
    c_1_true = 1
else:
    c_1_true = 0

X_true = 2**18 * ((M * w_h) % 2**13) % bound

print(f"infoB: c_0_true = {c_0_true}, c_1_true = {c_1_true}")
print(f"X_true = {X_true}")
for c_0 in range(2):
    for c_1 in range(2):
        B = ((2**35 * (D_h + c_0) + e_0) * M + A) % 2**48

        X = (dx - (B >> 17) + c_1*2**31) % bound
        print(f"c_0={c_0},\tc_1={c_1},\tX={X}")

# Testing infoC
x_r, z_r, x_c, z_c = infoC
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48

```

```

e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0_true = 1
else:
    c_0_true = 0

w_h = ((s_0 >> 35) - (D_h + c_0_true)) % 2**48

A_true = (2**35 * M * w_h) % 2**48
B_true = ((2**35 * (D_h + c_0_true) + e_0) * M + A) % 2**48

if A_true + B_true >= 2**48:
    c_1_true = 1
else:
    c_1_true = 0

X_true = 2**18 * ((M * w_h) % 2**13) % bound

print(f"infoC: c_0_true = {c_0_true}, c_1_true = {c_1_true}")
print(f"X_true = {X_true}")
for c_0 in range(2):
    for c_1 in range(2):
        B = ((2**35 * (D_h + c_0) + e_0) * M + A) % 2**48

        X = (dx - (B >> 17) + c_1*2**31) % bound
        print(f"c_0={c_0},\tc_1={c_1},\tX={X}")

# Testing infoD
x_r, z_r, x_c, z_c = infoD
D = (P*x_r + Q*z_r + salt) % 2**48
D_h = D >> 35

s_0 = ((w + D) ^ M) % 2**48
e_0 = s_0 % 2**35
dx = x_c - x_r*spacing
dz = z_c - z_r*spacing

if w % 2**35 + D % 2**35 >= 2**35:
    c_0_true = 1
else:
    c_0_true = 0

w_h = ((s_0 >> 35) - (D_h + c_0_true)) % 2**48

A_true = (2**35 * M * w_h) % 2**48
B_true = ((2**35 * (D_h + c_0_true) + e_0) * M + A) % 2**48

```

```

if A_true + B_true >= 2**48:
    c_1_true = 1
else:
    c_1_true = 0

X_true = 2**18 * ((M * w_h) % 2**13) % bound

print(f"infoC: c_0_true = {c_0_true}, c_1_true = {c_1_true}")
print(f"X_true = {X_true}")
for c_0 in range(2):
    for c_1 in range(2):
        B = ((2**35 * (D_h + c_0) + e_0) * M + A) % 2**48

        X = (dx - (B >> 17) + c_1*2**31) % bound
        print(f"c_0={c_0}, \tc_1={c_1}, \tX={X}")

```