Cryptography

Diophantine Equation

Tan Kel Zin

Table of Contents

Basics

Introduction

Diophantine equation is an equation where only the integer solutions matter.

Examples

- Pythagorean Triples

$$a^2 + b^2 = c^2$$

- linear Equation

$$ax + by = c$$

- Fermat Last's Theorem

$$a^n + b^n = c^n$$

It is not easy to solve a diophantine equation

Hilbert's tenth problem

Seek for a prove for a general algorithm that solves every diophantine equation. Solved it negatively by Yuri Matiyasevich by showing such algorithm cannot exist

Application in Cryptography

RSA

$m^e \equiv c \mod n$

Diffie–Hellman key exchange

$g^a \equiv A \mod p$

Elliptic Curve cryptography

$y^2 \equiv x^3 + ax + b \mod p$

Examples

Cryptography is coupled with all kinds of equations very much!

```
#!/usr/bin/env python3

from Crypto.Util.number import *
from flag import flag

l = len(flag)
m_1, m_2 = flag[: l // 2], flag[l // 2:]

x, y = bytes_to_long(m_1), bytes_to_long(m_2)

k = '''
000bfdc32162934ad6a054b4b3db8578674e27a165113f8ed018cbe9112
4fbd63144ab6923d107eee2bc0712fcbdb50d96fdf04dd1ba1b69cb1efe
71af7ca08ddc7cc2d3dfb9080ae56861d952e8d5ec0ba0d3dfdf2d12764
'''.replace('\n', '')

assert((x**2 + 1)*(y**2 + 1) - 2*(x - y)*(x*y - 1) == 4*(int(k, 16) + x*y))
```

Flag is split into 2 parts

```
m_1, m_2 = flag[: l // 2], flag[l // 2:]
x, y = bytes_to_long(m_1), bytes_to_long(m_2)
```

The only hint

```
assert((x**2 + 1)*(y**2 + 1) - 2*(x - y)*(x*y - 1) == 4*(int(k, 16) + x*y))
```

Rewrite with math notation

$$(x^2 + 1)(y^2 + 1) - 2(x - y)(xy - 1) = 4(k + xy)$$

Simplifying the equation

$$
\begin{aligned}
(x^2+1)(y^2+1) - 2(x-y)(xy-1) &= 4(k+xy) \\
(x^2+1)(y^2+1) - 2(x-y)(xy-1) - 4xy &= 4k \\
x^2y^2 - 2x^2y + 2xy^2 + x^2 - 4xy + y^2 + 2x - 2y + 1 &= 4k \\
(x+1)^2(y-1)^2 &= 4k \\
(x+1)(y-1) &= 2\sqrt{k}
\end{aligned}
$$

Then we can find all possible x and y by factorising $2\sqrt{k}$

Simplifying the equation

$$
\begin{aligned}
(x^2+1)(y^2+1) - 2(x-y)(xy-1) &= 4(k+xy) \\
(x^2+1)(y^2+1) - 2(x-y)(xy-1) - 4xy &= 4k \\
x^2y^2 - 2x^2y + 2xy^2 + x^2 - 4xy + y^2 + 2x - 2y + 1 &= 4k \\
(x+1)^2(y-1)^2 &= 4k \\
(x+1)(y-1) &= 2\sqrt{k}
\end{aligned}
$$

Then we can find all possible x and y by factorising $2\sqrt{k}$

Use SageMath

```
>>> var('x, y')
(x, y)
>>> ((x**2 + 1)*(y**2 +1) - 2*(x-y)*(x*y -1) - 4*x*y).factor()
(x + 1)^2*(y - 1)^2
>>> ZZ(sqrt(k)).factor()
2 * 3 * 11^2 * 19 * 47 * ...
```

Flag : CCTF{S1mPL3_4Nd_N!cE_Diophantine_EqUa7I0nS!}

Crypto CTF 2021 (RSAPHANTINE)

RSA and solving equations, but should be a real mathematician to solve it with a diophantine equation?

```
2*z**5 - x**3 + y*z = 477698647067501615811...
x**4 + y**5 + x*y*z = 8970186379449474157 92...
y**6 + 2*z**5 + z*y = 477698647067501615811...

p = nextPrime(x**2 + z**2 + y**2 << 76)
q = nextPrime(z**2 + y**3 - y*x*z ^ 67)
n, e = p * q, 31337
m = bytes_to_long(FLAG)
c = pow(m, e, n)
c = 486675922771716096231737399040548486325...
```

This question is easy, just solve

$$\begin{cases} 2z^5 - x^3 + yz = a_1 \\ x^4 - y^5 + xyz = a_2 \\ y^6 - 2z^5 + yz = a_3 \end{cases}$$

But ... How?

$$\begin{cases} 2z^5 - x^3 + yz - a_1 = 0 \\ x^4 - y^5 + xyz - a_2 = 0 \\ y^6 - 2z^5 + yz - a_3 = 0 \end{cases}$$

If it is possible to combine the equation and rewrite in terms of one variable

$$c_1 x^{a_1} + c_2 x^{a_2} + \cdots + c_n x^{a_n} = 0$$

Then this can be easily solved using numerical method (e.g Newton's method)

Crypto CTF 2021 (RSAPHANTINE)

Use SageMath

```
>>> var('x y z')
(x, y, z)
>>> p1 = 2*z**5 - x**3 + y*z - a1
>>> p2 = x**4 + y**5 + x*y*z - a2
>>> p3 = y**6 + 2*z**5 + z*y - a3
>>> p1.resultant(p2,x)
16*z^20 + 56*y*z^16 + y^15 - ...
>>> p1.resultant(p2,x).resultant(p3, y).roots()
[(298968066749556920280253653682020210357225489348275334602970 89, 1)]
```

Flag : CCTF{y0Ur_jO8_C4l13D_Diophantine_An4LySI5!}

One day, I tried to swap x and y coordinates of a Point on the Curve.

```
from params import p, a, b, flag, y
x = int.from_bytes(flag, "big")

assert 0 < x < p
assert 0 < y < p
assert x != y

EC = EllipticCurve(GF(p), [a, b])
assert EC(x,y)
assert EC(y,x)

print("p = {}".format(p))
print("a = {}".format(a))
print("b = {}".format(b))
```

Simplification of the question, solve

$$\begin{cases} y^2 = x^3 + ax + b \pmod{P} \\ x^2 = y^3 + ay + b \pmod{P} \end{cases}$$

It can be solved using a similar method above

With SageMath

```
x,y = PolynomialRing(ZZ, ['x','y']).gens()
f = x**3 + a*x + b - y**2
g = y**3 + a*y + b - x**2

poly = f.resultant(g, y).univariate_polynomial().change_ring(GF(p))
for root in poly.roots():
    m = root[0]
    print(long_to_bytes(m))
```

Conclusion

Takeaway

- Techniques to solve certain Diophantine Equation
- Appreciate abstraction given by SageMath
- Love Crypto more!

References

- https://en.wikipedia.org/wiki/Diophantine_equation
- https://blog.cryptohack.org/cryptoctf2021-easy
- https://blog.cryptohack.org/cryptoctf2021-medium
- https://mechfrog88.github.io/acsc-2021/swap