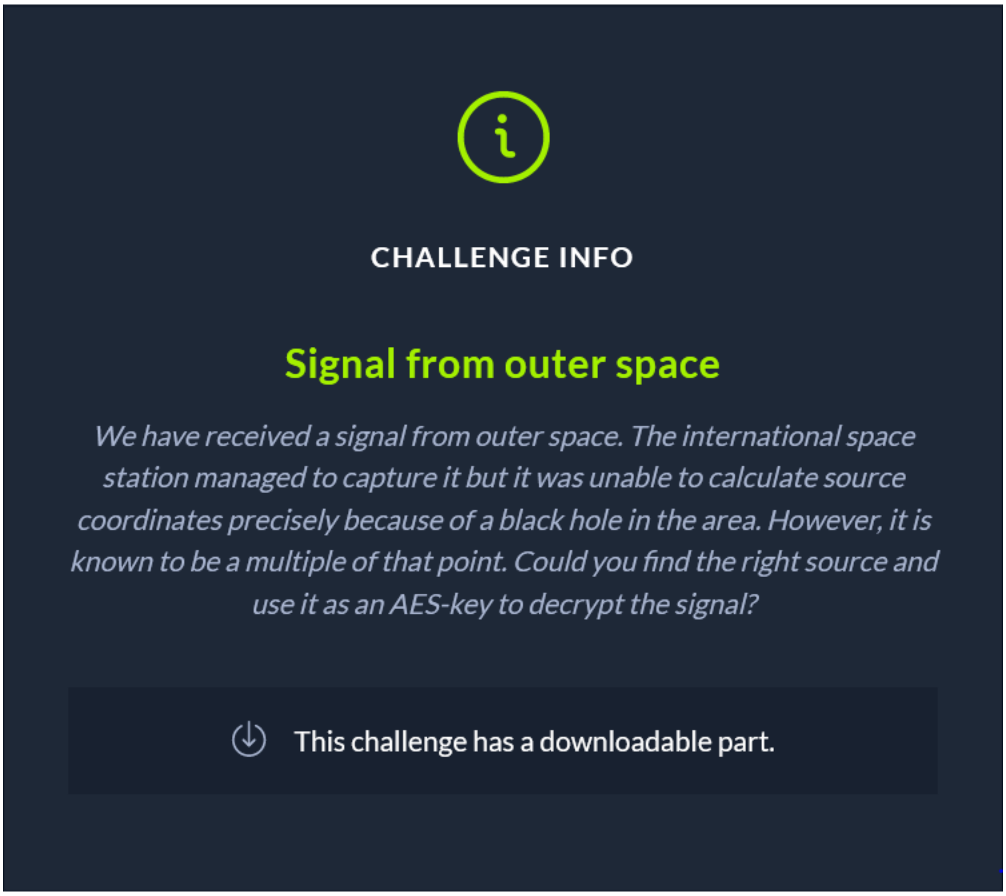
SIGNAL_FROM_OUTER_SPACE



MATERIAL:
info.pdf
flag.enc
challenge.py

FLAG:
HTB{37_c0m3_h0m3_d1nn3r_15_r34dy}

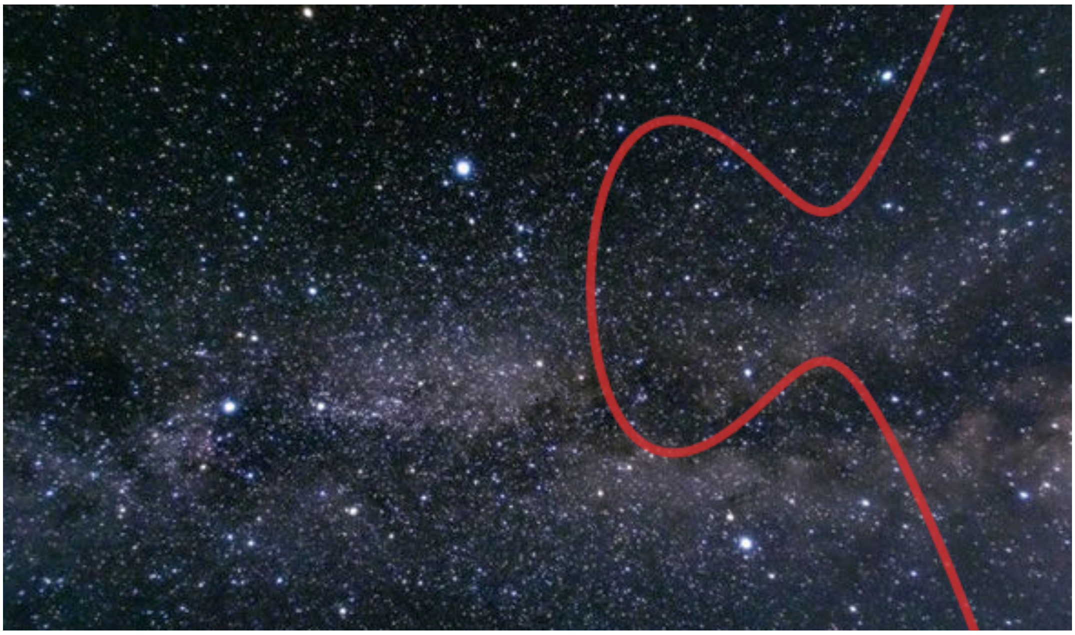
SOLVER:
M1gnus

FOOTHOLD

The challenge provide a script named **challenge.py:**

```
from Crypto.Cipher import AES
from Crypto.Util.number import inverse
from Crypto.Util.Padding import pad, unpad
from collections import namedtuple
from random import randint
import hashlib
import os

# Create a simple Point class to represent the affine points.
Point = namedtuple("Point", "x y")

# The point at infinity (origin for the group law).
O = 'Origin'

def encrypt(secret):
    sha1 = hashlib.sha1()
    sha1.update(str(secret).encode('ascii'))
    key = sha1.digest()[:16]

    dt = open('flag.mp3','rb').read()
    # Encrypt flag
    cipher = AES.new(key, AES.MODE_ECB)

    ciphertext = cipher.encrypt(pad(dt, 16))

    open('flag.enc','wb').write(ciphertext)


def check_point(P: tuple):
    if P == O:
        return True
    else:
        return (P.y**2 - (P.x**3 + a*P.x + b)) % p == 0 and 0 <= P.x < p and 0 <= P.y < p


def point_inverse(P: tuple):
    if P == O:
        return P
    return Point(P.x, -P.y % p)

def point_addition(P: tuple, Q: tuple):
    if P == O:
        return Q
    elif Q == O:
        return P
    elif Q == point_inverse(P):
        return O
    else:
        if P == Q:
            lam = (3*P.x**2 + a)*inverse(2*P.y, p)
            lam %= p
        else:
            lam = (Q.y - P.y) * inverse((Q.x - P.x), p)
            lam %= p
    Rx = (lam**2 - P.x - Q.x) % p
    Ry = (lam*(P.x - Rx) - P.y) % p
    R = Point(Rx, Ry)
    assert check_point(R)
    return R

def double_and_add(P: tuple, n: int):
    Q = P
    R = O
    while n > 0:
        if n % 2 == 1:
            R = point_addition(R, Q)
        Q = point_addition(Q, Q)
        n = n // 2
    assert check_point(R)
    return R

def gen_shared_secret(Q: tuple, n: int):
    S = double_and_add(Q, n)
    return S.x


p = ...
a = ...
b = ...


g_x = ...
g_y = ...
G = Point(g_x, g_y)


n = randint(1, p)

public = double_and_add(G, n)

key = gen_shared_secret(public,n)
encrypt(key)
decrypt(key)
```

a PDF file named **info.pdf:**



Space Complex f: $y^2 = x^3 +$
1696410886787087543102251184740607451350543826913219962992644710511732770297332142913007867361199631338277557931699620223974053371401355978936943390559885 3 x
+
25572699039046829700992365380317352184627808448171753176973856593770071867850509696769380590881559223280110750816063545696219588626128508166946470023808 20

Space Field:
1993088520382554010851391632346461960789052061388211026899931540250248399271362405607084777848506578838769556716212080583083796423749616366477723336083585 3

ISS Coordinates as Reference Point:
(1161231603880147996240072369842788331013016317629133875991128103417055660974345777303237596562042993167541038922051234283466308912665944552370153392231323 8,
1449247055997660990961492015030253389066228153987454263980821398295631671083766711468149633636190671874805388451346834323605554533752205571314107661596724 3)

Miscalculated Source Point:
(7164474552184432113371383549103608203100392224670997505488756019786699488776533696709844504937760383149392739907928750304772097389051381620161039728228 87,
2577636835404844606086593513656242872409629115084680797009677298030035019056074660428408865132307063442440986937998752115629728860272800311376841076830297)

and the encrypted flag.
The provided script uses [ECC] to encrypt a symmetric key and [AES] with [ECB] mode to encrypt the flag (which is a mp3 file). The PDF file gives us the info needed to afford the problem:
the curve:

y^2 = x^3 + 16964108867870875431022511847406074513505438269132199629926447105117327702973321429130078673611996313382775579316996202239740533714013559789369433905598853x + 2557269903904682970099236538031735218462780844817175317697385659377007186785050969676938059088155922328011075081606354569621958862612850816694647002380820

Char of the field:

19930885203825540108513916323464619607890520613882110268999315402502483992713624056070847778485065788387695567162120805830837964237496163664777233360835853

G:

(11612316038801479962400723698427883310130163176291338759911281034170556609743457773032375965620429931675410389220512342834663089126659445523701533922313238, 14492470559976609909614920150302533890662281539874542639808213982956316710837667114681496336361906718748053884513468343236055545337522055713141076615967243)

P:

(716447455218443211337138354910360820310039222467099750548875601978669948877653369670984450493776038314939273990792875030477209738905138162016103972822887, 257763683540484460608659351365624287240962911508468079700967729803003501905607466042840886513230706344244098693799875211562972886027280031137684107683O297)

THE ATTACK

By analyzing the curve is possible to see that is an anomalous curve: the cardinality of the curve is equal to the char of the field. This means that is possible to perform the Smart Attack against the curve to compute efficiently discrete_log(P, base=G), then recover the symmetric key and decrypt the flag.

THE IMPLEMENTATION

```
import hashlib
import sys
from Crypto.Cipher import AES

def HenselLift(P, p, prec):
    E = P.curve()
    Eq = E.change_ring(QQ)
    Ep = Eq.change_ring(Qp(p,prec))
    x_P,y_P = P.xy()
    x_lift = ZZ(x_P)
    y_lift = ZZ(y_P)
    x, y, a1, a2, a3, a4, a6 = var('x,y,a1,a2,a3,a4,a6')
    f(a1,a2,a3,a4,a6,x,y) = y^2 + a1*x*y + a3*y -x^3 -a2*x^2 -a4*x -a6
    g(y) = f(ZZ(Eq.a1()),ZZ(Eq.a2()),ZZ(Eq.a3()),ZZ(Eq.a4()),ZZ(Eq.a6()),ZZ(x_P),y)
    gDiff = g.diff()
    for i in range(1,prec):
        uInv = ZZ(gDiff(y=y_lift))
        u = uInv.inverse_mod(p^i)
        y_lift= y_lift -u*g(y_lift)
        y_lift = ZZ(Mod(y_lift,p^(i+1)))
    y_lift = y_lift+O(p^prec)
    return Ep([x_lift,y_lift])



def SmartAttack(P,Q,p,prec):
    E = P.curve()
    Eqq = E.change_ring(QQ)
    Eqp = Eqq.change_ring(Qp(p,prec))

    P_Qp = HenselLift(P,p,prec)
    Q_Qp = HenselLift(Q,p,prec)

    p_times_P = p*P_Qp
    p_times_Q=p*Q_Qp
    x_P,y_P = p_times_P.xy()
    x_Q,y_Q = p_times_Q.xy()

    phi_P = -(x_P/y_P)
    phi_Q = -(x_Q/y_Q)
    k = phi_Q/phi_P
    k = Mod(k,p)
    return k


p =
19930885203825540108513916323464619607890520613882110268999315402502483992713624056070847778485065788387695567162120805830837964237496163664777233360835
853
a =
16964108867870875431022511847406074513505438269132199629926447105117327702973321429130078673611996313382775579316996202239740533714013559789369433905598
853
b =
25572699039046829700992365380317352184627808448171753176973856593770071867850509696769380590881559223280110750816063545696219588626128508166946470023808
20

E = EllipticCurve(GF(p), [a, b])
print(f"checking if the curve is anomalous...")
if E.cardinality() == p:
    print(f"OK\n")
else:
    print(f"The curve is not anomalous exiting with 1...")
    sys.exit(1)

print(f"Calculating dlog...")
G =
E(11612316038801479962400723698427883310130163176291338759911281034170555660974345777303237596562042993167541038922051234283466308912665944552370153392231
3238,
14492470559976609909614920150302533890662281539874542639808213982956316710837667114681496336361906718748053884513468343236055545337522055713141076615967
243)
P =
E(71644745521844321133713835491036082031003922246709975054887560197866994887765336967098445049377603831493927399079287503047720973890513816201610397282288
7,
25776368354048446060865935136562428724096291150846807970096772980300350190560746604284088651323070634424409869379987521156297288602728003113768410768302
97)
dlog = int(SmartAttack(G,P,p,4096))
print(f"SUCCESS, dlog = {dlog}\n")

Q = dlog*P
secret = str(Q[0])
print(f"Secret: {secret}")

hasher = hashlib.sha1()
hasher.update(secret.encode("ascii"))
key = hasher.digest()[:16]
print(f"Key: {key}")

decryptor = AES.new(key, AES.MODE_ECB)
with open("flag.enc", "rb") as f:
    encflag = f.read()
flag = decryptor.decrypt(encflag)
with open("flag.mp3", "wb") as f:
    f.write(flag)
print(f"Flag written to 'flag.mp3'")
```

Recover the flag

By running the script and listen the decrypted mp3 file we can easily obtain the flag.

CHEESE!

```
sage: load("decrypt_flag.sage")
checking if the curve is anomalous...
OK

Calculating dlog...
SUCCESS, dlog = 115682289480337563168804393093509396734667287518002476162907933
2798372609720444094177754289181167668298093678797314543811021925445047990214789
7654728360094

Secret: 9477361827590594893714115265943990495646189708187931810203425827257486
43729134830502708902047572064771365922808626697099529791166498085469381507183276
7229
Key: b'\xae\x9e_D\x1aw\xd3Hzb\xe9U\xb4\xd8\xb3\x0f'
Flag written to 'flag.mp3'
```