This is a writeup for the challenge "Starter ECC", part of the Crypto CTF (https://cr.yp.toc.tf):

1 Setup

For this challenge, we are given the files: starter_ecc.sage:

```
#!/usr/bin/env sage

from Crypto.Util.number import *
from secret import n, a, b, x, flag

y = bytes_to_long(flag.encode('utf-8'))

assert y < n
E = EllipticCurve(Zmod(n), [a, b])

try:
	G = E(x, y)
	print(f'x = {x}')
	print(f'a = {a}')
	print(f'b = {b}')
	print(f'n = {n}')
	print('Find the flag :P')
except:
	print('Ooops, ERROR :-(')
```

output.txt:

x = 1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858127594672917553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418215304647702061791760188485382761123235545522396603959014362279280380087918603392415017391292520 8583

a = 31337

b = 6682641856848707718142539698474390546418947007246683388463694730650738034236238648870370281267332736737938697025227896368293908050246850645288426053494912096733853206898330706136368698753940821664424971895036532207864306766680284572093911175830902634323977955553651771829275456163150456098992678515298364903 5

n = 1172249882296274364826596736243245584619897371637339915298109877814501606885400013667788242452752877573733898873197392416842445457455832125128139491720780790427758251453129000175126609316678535670608103315419275681028600398981161822485972918994987905181059093903310986306909778587676700610269319381529248399 36

Find the flag :P

2 Elliptic curves

In cryptography, elliptic curves are curves of the form

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

Given x, a, b, n, it is generally difficult to work out the value of y. The concept of modular square roots is generally known as finding the "quadratic residue", and this is in general difficult to solve.

3 Vulnerability in this cryptosystem

Noting that n ends in "6", we can immediately observe that the modulus is divisible by 2 and hence not prime. Because of this, we may be able to break this congruence into many smaller congruences, find the solution to these and then use the Chinese Remainder Theorem to find solutions to the original congruence.

We repeatedly divide by 2 until the modulus is no longer of even. It turns out that we can do this 63 times, so 2^{63} is a factor of n.

We then find the remaining factors by using Magma to factorise $\frac{n}{2^{63}}$:

```
SetVerbose("Factorization", 1);
SetVerbose("MPQS",1);
n := 127095586908149752947247243554945108215999796884519952414972546467589101
Factorisation(n);
<output omitted>
ECM
    x: 127095586908149752947247243554945108215999796884519952414972546467589
    2904544398018302170960976396069774264206261125722622008107406078658225211
    8107247376247754452810108426723003737691474060434813326707597243467045547
    26727327620195872367526078463055605698328066999917469244741097566417
    (290 digits)
    Initial B1: 5000, limit: 0
    Initial Pollard p - 1, B1: 45000
    Step 1; B1: 5000 [0], digits: 290, elapsed time: 0.079
    Step 10; B1: 5650 [0], digits: 290, elapsed time: 0.880
    Step 20; B1: 6420 [0], digits: 290, elapsed time: 1.859
    Step 30; B1: 7240 [0], digits: 290, elapsed time: 2.980
    Step 40; B1: 8110 [0], digits: 290, elapsed time: 4.229
    Step 50; B1: 9030 [0], digits: 290, elapsed time: 5.599
    Step 60; B1: 10000 [0], digits: 290, elapsed time: 7.129
    Step 70; B1: 11016 [0], digits: 290, elapsed time: 8.840
    Step 80; B1: 12081 [0], digits: 290, elapsed time: 10.640
    Step 90; B1: 13196 [0], digits: 290, elapsed time: 12.730
    Step 100; B1: 14361 [0], digits: 290, elapsed time: 14.990
    Factor: 690712633549859897233 (21 digits)
    Factor divides as power: 690712633549859897233^6
```

```
  Cofactor: 65113226288318917167620946699307 3^5 (33 digits)
  ECM time: 16.910

Total time: 16.929

[ <690712633549859897233, 6>, <65113226288318917167620946699307 3, 5> ]
```

So, we can write

$$n = 2^{63} \times 690712633549859897233^6 \times 651132262883189171676209466993073^5$$

Now that we know the factorisation, we can again use Magma to find the quadratic residues (notably, it will use the Chinese Remainder Theorem work in the background):

```
n := 1172249882296274364826596736243245584619897371637339915298109877814501 60
f1 := 2;
f2 := 690712633549859897233;
f3 := 651132262883189171676209466993073;
a := 31337;
b := 66826418568487077181425396984743905464189470072466833884636947306507380 3
x := 10715086071862673209484250490600018105614048117055336074437503883703510 5
R := ResidueClassRing(n);
y2 := x^3 + a*x + b;
AllSqrts(R!y2: Factorization:=[<f1,63>,<f2,6>,<f3,5>]);

[ 162559115533707434296110721894408231436159386125931969649615840891971155864
4439254438860188973760416673208216652523878433999025546930796293249982656156 3
2138742592372804416730371359298490966137060479292092247190676930029658029277 8
7438370194986944617453973529126667411234471431198160732422894560556834655,
2265567078026161287537455320734674452320163732216942900406130681534739031975 6
4503672884655286961007448150259619730918729324207443166257274431998805226443 2
6650572817854490554214928993747961224043752114606087516310280066993583108691 4
2895684304932123808518743534535403348064633680505741833364006987047 7151,
2344195496168534948530572587274108596410472513462080464641932076945280394838 1
7024351278595912553428732864754460065088108367680920068233016419669289425652 6
8744034581983695052531733908203176496718128442720525597075758941197834622010 3
4315414102575762454237897324140086450406779918940604933249770990866 4481,
2877077992623749931026062992151527188730444473766995907938645039512200166258 0
3080809134890335505484173000751245326001847830287064250195594880304757002576 0
2358777564006756153853777686237226728248219029592249446195954194264966108323 4
1220896030712480778110228608922785873868667903872777822272585724011 2991,
2984171418857621893106920689064700734369042384419703668551904349560307868168 9
7602580277232302138394513694192414543618994291985808541410661526669829036463 4
1520348162557743933709526972102041107055832628117388394318346005225614802919 5
3374078908813424517359287946008748675024270479630273524324321922230 6977,
3517053915312836875602411093942119326689014344724619111848617312127222763958 8
```

365903813352672509044995383018919980453273375459195272337323998730529661338682
513509114458080503503157075013609133858592321498911224343854125829274628923267
02795608369501428412316192307914480984861584645624464133471366553755487,
359568233345521053659552836048155347077932312596975667608441870753776900245134
617971652746735068287123854468404013870211279806542962534898197497578081259625
722855290871000953334837566459130661126029954310355032420402013249699780255152
16992906345937814869507730203961312008283047029973095132329006591942817,
423565825614429748117187646227214560873789299692737987999439098015279647578215
675794552610374026783701937412199461723299872237031809852662708197632042340700
000486648928405841452616872849017122159800372850041312144660719652477798346071
07579554408314435500721636422647934254457952636869781043074515905585313,
748684056681844616709409090016031023746108071944601927298670779799221959307184
460883329814153501992035401576532512200868552217542748468588573197285165467204
775338496384494176060044058818836444901009958691886255958199320245638383902520
84319944382203670408668694675982756723400815033191245888863637019254623,
812681648950753311167043900195090237541965059040364247689668007060724706640265
518706229677792460488613484520327960053957144648031595786353083897339126548279
052969854441899064177823365208722905934780377231572535682458026648416401993444
74906592444580291039882600894669378969575720640087931799609146332897119,
820544490764990677266355626849033651950995937164878004113248146601778842926519
770774069071855019730741955969811993470895048995379285983927282664387546469221
262316030854819514009503856654244433202218010043016343759005914068841553325330
89103890421016677497074138790716209992997183024436562798466786371084449,
873832740410512175515904667336775511182993133195369548442919442858470820068500
376419854701297314936285969569490519562268995255993704180185128727934304161560
623790329057125624141708234457649456355227068730188728671025439375554701956647
58158709702383860735797451638543204227616062873758291688694933702532959,
884542083033899371723990437028092865746852924260640324504245373863281590259599
828596968935493978227320038913607441323983641425868133301691793364441507550290
539947388912224402127283163044130894235988428582702623483264620471619571416248
79690538483393298128288045009402832239172088631333248709212295684726945,
937830332679420869973539477515834724978850120291131868833916670119973567401580
434242754564936273432864052513285967415357587686482551497949639427988265242634
901421687114530512259487540847535917388997487269875008395284145778332720047565
48745357764760481367011357857229826473790968480654977599440443016175455,
945693174493658236072851204169778139387880998415645625257496809661027703687834
686310593958998832674992523962770000832295492033830241695523838195036685163577
110767863527450962091168032293057444656435120081318816471832033198757871379451
62942655741196867824202895753276657497212430865003608598298083054362785,
100969076676256693053048601434883735318373798551140794564849403692253045102091
574413349382263779117157060690656544868538408446431908901328834889509064624465
138839922158485585020894733868294390569020553862100509619609073960153588947030
853529303803573488455416801971963279743387336471900294509043592368005281]

Now, we have a relatively short list of potential values for y. We convert all of these numbers into their ASCII representation and search through the output using the "strings" tool. This gives us the flag CCTF{8E4uTy_0f_L1f7iN9_cOm3_Up!!}