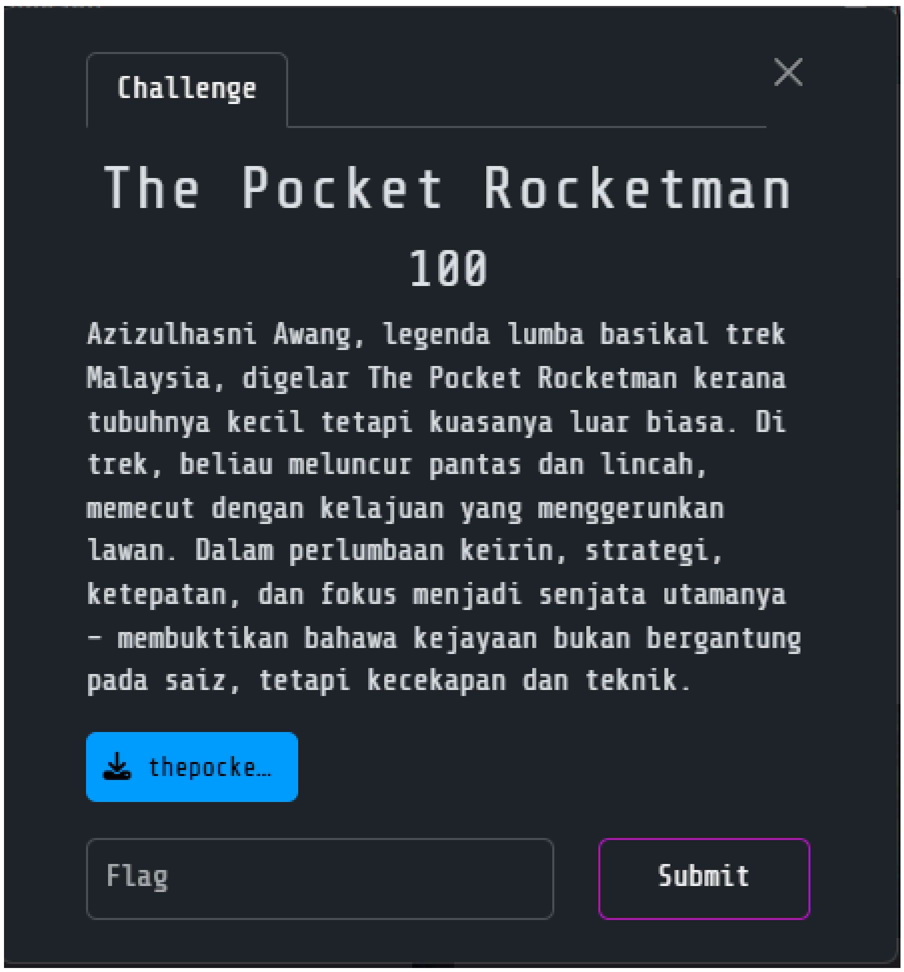
# The Pocket Rocketman

| | |
|---|---|
| Types | cryptography |
| CTF | 3108 |




check the given pdf

STRATEGY

```python
from sympy import nextprime
from random import randint

def readFlag():
    with open('flag.txt', 'r') as f:
        return f.readline().strip()


def main():
    size = 4096
    p = nextprime(randint(2**size, 2**(size+1)))
    q = nextprime(p)
    n = p * q
    e = 65537
    print("n:", n)
    print("e:", e)
    flag = readFlag()
    message = int.from_bytes(flag.encode(), "big")
    ciphertext = pow(message, e, n)

    with open("output.txt", "w") as f:
        f.write(f"n: {n}\n")
        f.write(f"e: {e}\n")
        f.write(f"ciphertext: {ciphertext}\n")

    print("done output.txt")


if __name__ == "__main__":
    main()
```

```
None

n:
3571131588460723593757070157654148429120124149684435888798837617488512620460724
2466833941812974308022247302690002757936526058025640569173350735885687265223807
9303419976598117480026740710061426230310189728940563265579836377515841194693915
6482623842120156542688764597434688290963639995275979828057718304296497569918097
7699785394729830503560548515239635986606226579687773961376590383953991503756101
38357365570164661622284555578860395753403969045101188808039477161027338675256055
5408743521269583978370164813692200577554016321731380788703445000505521717971852
3753063798416220847080541367777173765182710590733234620135256341270557436066226
1222331553242850437385626739813256724982306550378559795368920392399923328058183
0234002904557630319011913357801033447322062883022805863662308652478477926099501
9268418429689046325914455138561047914228711074105204033993301377743095176525065
2413033005102508317286437167886580010379183211122103546201625832791526610109542
5626402963515034379887464718462664831811433395297668160524088112539175136803106
0254408054415427537478054081279251706602194821913181531064699726191086545305561
1399754180537367760586784502084500092088467821267622133932205872758978737924929
6664645413169312349939268399511623479787762713351542956176353721624967813859564
1829463750883770625799175804502545275678885163827283338763461010146040941016883
5676621704374910072584202494846468674063327590676052333279295450036016182339376
4159138286404327889500446181434984756628103365408698584640001883055169031488316
6082370034270169063112144689965361964585430273853271710798113019418005067815257
2078203919404648745574849092680047869108473589036048094400971504964546866396031
4316785253744150635697487304015863501753502295744467004108494802766542734247563
8318960765964003561584499289002485585075587799034528998064804637521614522572414
5257907224162897473540663933376504401856090250001881727187859966252697675807799
0925063478927907518632768493690339831001738155338819956525133875205803790261270
4611914505013118734241860181922062471771091947215062227467550287406709584380818
2212822389545068986200394674553578104940918706968799993812182293658247043970547
7571828811651689805125016425640895350968865676761470122461397229485154559543064
0383958693509984016997185290119842033349866736057763437830306230767736164597258
7860013602039427531576217711843211599891372140362944340903764661105134915114410
8429111721397790162949131317367055345980594062281959388762760611218436558561706
891499953687263663
e: 65537
ciphertext:
1782034476358322681235404510428553062057206727416215401131490654203038859843009
9656525979315403802287799810517002276328321709661906887355737723846113306969755
6910337009363269552357059487784969540167233698642717036973420101844041582291269
2557127102340308408525703192825564686799421602254585650727473175558734508244374
```

```
7857231292412724459630185237350225870052151407326628057844136242491060755463023
9827268868630223930477395073933235196584793144267087989075544664460300671237356
5868272849790709154158553139538029640953276557730292091403809792991384414128016
7288127544980389768705631057270662053158656032738364063576838516798200208770216
1692145766726021333312309469600286207455092050779487046999212952433171378975500
0624760962567194603098802873093086668035611633523150906976105902789418779265498
1169159424244467887304842549058165569198219538715541262389885119473172906086455
9621806411255255397331606488443899464663876341485919295841967809899965682908259
6390778539958919264775036561379490036215148371288929963234705073610044362446103
2254609367862720824605370416249791098511803680045834459757174446340842101667178
4151416389627315118292578411790501053963366769153545772334036292394001395319576
7152934715637263126705194434675589592746070817668193937022863404191610898133070
6111909928216988125149192753759959636441800071888031362296878000803752886305136
1563430374001514512552994152570892143793152039907268593277710954965969316102170
4857026419945920297228044313849160317443754324596229187465984148027251419217547
8998558710982926281946719759907467729527088724768884851265702410498619904131548
8590948111003899510511319757825835068523785866526575732087424338009782024955871
8492782262530010697093746452478461132106278146094838833296640429881117595348570
2602797950732950917053698845062948345844512920329248500467068579290382367321918
6606891905677357590386954887493247647395804112060204885148430632480389567344953
1117818903901962372701663755250682224065975631609264122606144260546380373321672
5706078758052344227199711041161963370941456815138361910552717727015493128222893
7420906388302421173028844061722989750305603137140737267731498058143527695397666
5446480045952214955460054960078482752635925409892920098272011053261932611910020
5747953926557313669426124557523159673520005695744461365408427488381127347911675
7596452679024376731284650930451277296678681723558236054287426314867744655069520
1830610415075065863454625731730704374192101343767002339917576914239602221375963
392526115296558869
```

# 1. Explaining the Challenge Description

The challenge starts by telling the story of **Azizulhasni Awang**, Malaysia's track cycling legend, also known as *The Pocket Rocketman*. Despite his small physique, he has enormous power and speed. The metaphor here is important:

- **Azizul's strength = the RSA system's strength**

Just like Azizul dominates in cycling despite his size, RSA is considered strong because it uses very large numbers.

- **Strategy in keirin = strategy in cryptography**

  Winning in cycling isn't just about size; it's about technique. Similarly, breaking RSA isn't about brute force, but about finding weaknesses.

- **Hidden hint: consecutive primes**

  The description mentions *"strategy, precision, and focus"* — this is a clue that instead of brute forcing huge primes, we need to look carefully at how the primes were chosen.

Then the challenge explains they created **RSAce**, a "big and fast" encryption system. But like any system, if there's a strategic flaw, it can be broken.

The flaw comes from this part of the code:

```
p = nextprime(randint(2**size, 2**(size+1)))
q = nextprime(p)
```

This means:

- `p` is a random prime ~4096 bits.
- `q` is just the **next prime after p**.

So `p` and `q` are **consecutive primes** → they are extremely close together.

That makes RSA insecure here, because we can factor `n` (the modulus) using **Fermat's factorization method**, which is very efficient when the two primes are close.

```
# Fermat factorization and RSA decryption for the user's provided n and ciphe
rtext.
# This code will parse the multi-line numbers, attempt Fermat factoring (effici
ent if p and q are close),
# recover private key d, decrypt ciphertext, and print the recovered plaintext
(flag).
```

```
import re, math, sys
from math import isqrt

n_raw = r"""
3571131588460723593757070157654148429120124149684435888798837617488512620460724
2466833941812974308022247302690002757936526058025640569173350735885687265223807
9303419976598117480026740710061426230310189728940563265579836377515841194693915
6482623842120156542688764597434688290963639995275979828057718304296497569918097
7699785394729830503560548515239635986606226579687773961376590383953991503756101
3835736557016466162228455557886039575340396904510118808039477161027338675256055
5408743521269583978370164813692200577554016321731380788703445000505521717971852
3753063798416220847080541367777173765182710590733234620135256341270557436066226
1222331553242850437385626739813256724982306550378559795368920392399923328058183
0234002904557630319011913357801033447322062883022805863662308652478477926099501
9268418429689046325914455138561047914228711074105204033993301377743095176525065
2413033005102508317286437167886580010379183211122103546201625832791526610109542
5626402963515034379887464718462664831811433395297668160524088112539175136803106
0254408054415427537478054081279251706602194821913181531064699726191086545305561
1399754180537367760586784502084500092088467821267622133932205872758978737924929
6664645413169312349939268399511623479787762713351542956176353721624967813859564
```

```
1829463750883770625799175804502545275678885163827283338763461
010146040941016883
5676621704374910072584202494846468674063327590676052333279295
450036016182339376
4159138286404327889500446181434984756628103365408698584640001
883055169031488316
608237003427016906311214468996536196458543027385327171079811301
9418005067815257
2078203919404648745574849092680047869108473589036048094400971
504964546866396031
43167852537441506356974873040158635017535022957444670041084948
02766542734247563
831896076596400356158449928900248558507558779903452899806480
4637521614522572414
52579072241628974735406639333765044018560902500018817271878599
66252697675807799
09250634789279075186327684936903398310017381553388199565251338
75205803790261270
4611914505013118734241860181922062471771091947215062227467550287
406709584380818
2212822389545068986200394674553578104940918706968799993812182
293658247043970547
757182881165168980512501642564089535096886567676147012246139722
9485154559543064
0383958693509984016997185290119842033349866736057763437830306
230767736164597258
786001360203942753157621771184321159989137214036294434090376466
1105134915114410
8429111721397790162949131317367055345980594062281959388762760611
218436558561706
891499953687263663
"""

c_raw = r"""
17820344763583226812354045104285530620572067274162154011314906
54203038859843009
```

```
9656525979315403802287799810517002276328321709661906887355737723846113306969755
6910337009363269552357059487784969540167233698642717036973420101844041582291269
2557127102340308408525703192825564686799421602254585650727473175558734508244374
7857231292412724459630185237350225870052151407326628057844136242491060755463023
9827268868630223930477395073933235196584793144267087989075544664460300671237356
5868272849790709154158553139538029640953276557730292091403809792991384414128016
7288127544980389768705631057270662053158656032738364063576838516798200208770216
1692145766726021333123094696002862074550920507794870469992129524331713789755OO
0624760962567194603098802873093086668035611633523150906976105902789418779265498
1169159424244467887304842549058165569198219538715541262389885119473172906086455
9621806411255255397331606488443899464663876341485919295841967809899965682908259
6390778539958919264775036561379490036215148371288929963234705073610044362446103
2254609367862720824605370416249791098511803680045834459757174446340842101667178
4151416389627315118292578411790501053963366769153545772334036292394001395319576
7152934715637263126705194434675589592746070817668193937022863404191610898133070
6111909928216988125149192753759959636441800071888031362296878000803752886305136
1563430374001514512552994152570892143793152039907268593277710954965969316102170
4857026419945920297228044313849160317443754324596229187465984148027251419217547
```

```
89985587109829262819467197599074677295270887247688848512657024
10498619904131548
85909481110038995105113197578258350685237858665265757320874243
38009782024955871
8492782262530010697093746452478461132106278146094838833296640
429881117595348570
260279795073295091705369884506294834584451292032924850046706
8579290382367321918
6606891905677357590386954887493247647395804112060204885148430
632480389567344953
111781890390196237270166375525068222406597563160926412260614426
0546380373321672
5706078758052344227199711041161963370941456815138361910552717727
015493128222893
74209063883024211730288440617229897503056031371407372677314980
58143527695397666
544648004595221495546005496007848275263592540989292009827201
1053261932611910020
5747953926557313669426124557523159673520005695744461365408427
488381127347911675
7596452679024376731284650930451277296678681723558236054287426
314867744655069520
183061041507506586345462573173070437419210134376700233991757691
4239602221375963
392526115296558869
"""

# Remove non-digit characters and join
n = int(re.sub(r'[^0-9]', '', n_raw))
c = int(re.sub(r'[^0-9]', '', c_raw))
e = 65537

print("Parsed n bit-length:", n.bit_length())

# Fermat factorization
a = isqrt(n)
```

```
if a * a < n:
    a += 1

def is_perfect_square(x):
    if x < 0:
        return False
    r = isqrt(x)
    return r*r == x

max_iters = 10_000_000  # safety cap
iters = 0
while True:
    b2 = a*a - n
    if is_perfect_square(b2):
        b = isqrt(b2)
        p = a - b
        q = a + b
        if p*q == n:
            break
    a += 1
    iters += 1
    if iters % 1000000 == 0:
        print("Still searching... iterations:", iters, "current a bit-length:", a.bit_lengt
h(), file=sys.stderr)
    if iters > max_iters:
        raise RuntimeError("Exceeded iteration cap; Fermat didn't find factors wit
hin the cap.")

print("Found factors. Iterations:", iters)
p = int(p)
q = int(q)
if p > q:
    p, q = q, p

print("p bit-length:", p.bit_length())
print("q bit-length:", q.bit_length())
```

```
phi = (p-1)*(q-1)
d = pow(e, -1, phi)
m = pow(c, d, n)
# Convert to bytes
m_bytes = m.to_bytes((m.bit_length()+7)//8, 'big')
try:
    plaintext = m_bytes.decode('utf-8')
except Exception as ex:
    plaintext = repr(m_bytes)

print("\nRecovered plaintext:")
print(plaintext)
```

# 2. Explaining the Python Exploit Code

Here's what the script does step by step:

## Step 1: Parse the given numbers

We clean up the provided `n` and `ciphertext` from the challenge output:

```
n = int(re.sub(r'[^0-9]', '', n_raw))
c = int(re.sub(r'[^0-9]', '', c_raw))
e = 65537
```

This converts the long multiline values into integers we can work with.

---

## Step 2: Fermat Factorization

Since `p` and `q` are very close, we can use Fermat's method:

1. Start at `a = ceil(sqrt(n))`.
2. Check if `a^2 - n` is a perfect square.
3. If yes, then we found `p = a - b` and `q = a + b`.

In our case, it worked instantly because `q` was just the next prime.

---

### Step 3: Compute Private Key

Once we have `p` and `q`, we compute:

```
phi = (p-1)*(q-1)
d = pow(e, -1, phi)
```

Here `d` is the private exponent.

---

### Step 4: Decrypt the Ciphertext

Now we decrypt with:

```
m = pow(c, d, n)
m_bytes = m.to_bytes((m.bit_length()+7)//8, 'big')
plaintext = m_bytes.decode('utf-8')
```

This recovers the original flag.

### Final Output

The script gave us:

```
3108{Muh4mm4d_Az1zulH4sn1_Th3_P0ck3t_R0ck3tm4n_88}
```

- The challenge uses RSA with consecutive primes.
- This makes factoring `n` easy via Fermat's method.
- Once factored, we calculate the private key and decrypt.
- The flag is revealed.