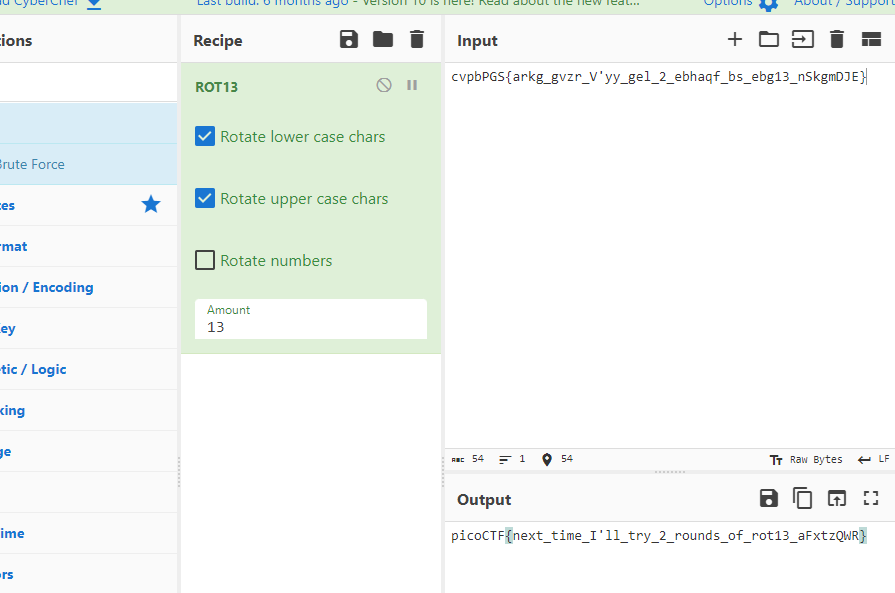
Cryptography

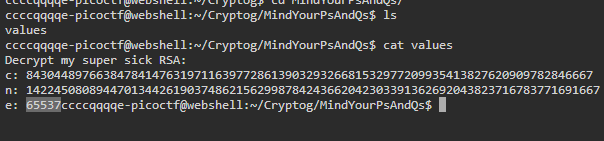
## Mod 26

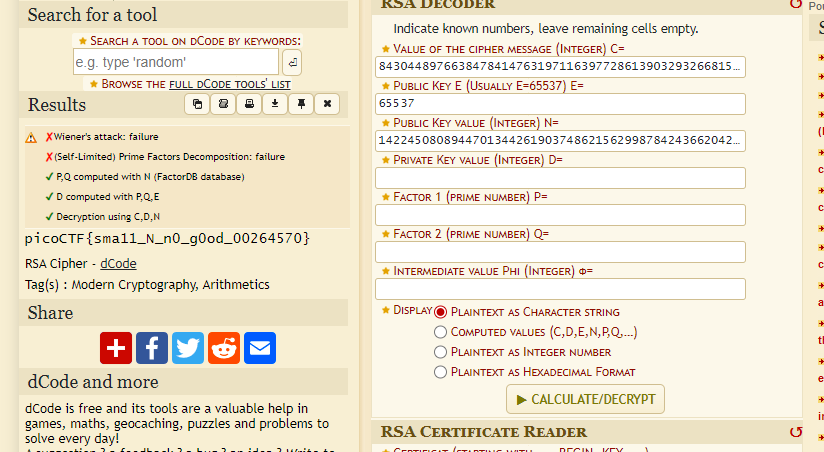
String provedeme přes ROT13 a dostaname flag 

picoCTF{next\_time\_I'll\_try\_2\_rounds\_of\_rot13\_aFxtzQWR}

## Mind your Ps and Qs

Dostali jsme 3 values pro decode RSA šifru a po zadání:





<https://www.dcode.fr/rsa-cipher>

## Easy Peasy

Jedná se o [Vernamovu šifru (one-time pad)](https://cs.wikipedia.org/wiki/Vernamova_%C5%A1ifra)

Obsah *otp.py*:

Python

Copy

#!/usr/bin/python3 -u

import os.path

KEY\_FILE = "key"

KEY\_LEN = 50000

FLAG\_FILE = "flag"

def startup(key\_location):

flag = open(FLAG\_FILE).read()

kf = open(KEY\_FILE, "rb").read()

start = key\_location

stop = key\_location + len(flag)

key = kf[start:stop]

key\_location = stop

result = list(map(lambda p, k: "{:02x}".format(ord(p) ^ k), flag, key))

print("This is the encrypted flag!\n{}\n".format("".join(result)))

return key\_location

def encrypt(key\_location):

ui = input("What data would you like to encrypt? ").rstrip()

if len(ui) == 0 or len(ui) > KEY\_LEN:

return -1

start = key\_location

stop = key\_location + len(ui)

kf = open(KEY\_FILE, "rb").read()

if stop >= KEY\_LEN:

stop = stop % KEY\_LEN

key = kf[start:] + kf[:stop]

else:

key = kf[start:stop]

key\_location = stop

result = list(map(lambda p, k: "{:02x}".format(ord(p) ^ k), ui, key))

print("Here ya go!\n{}\n".format("".join(result)))

return key\_location

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Welcome to our OTP implementation!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

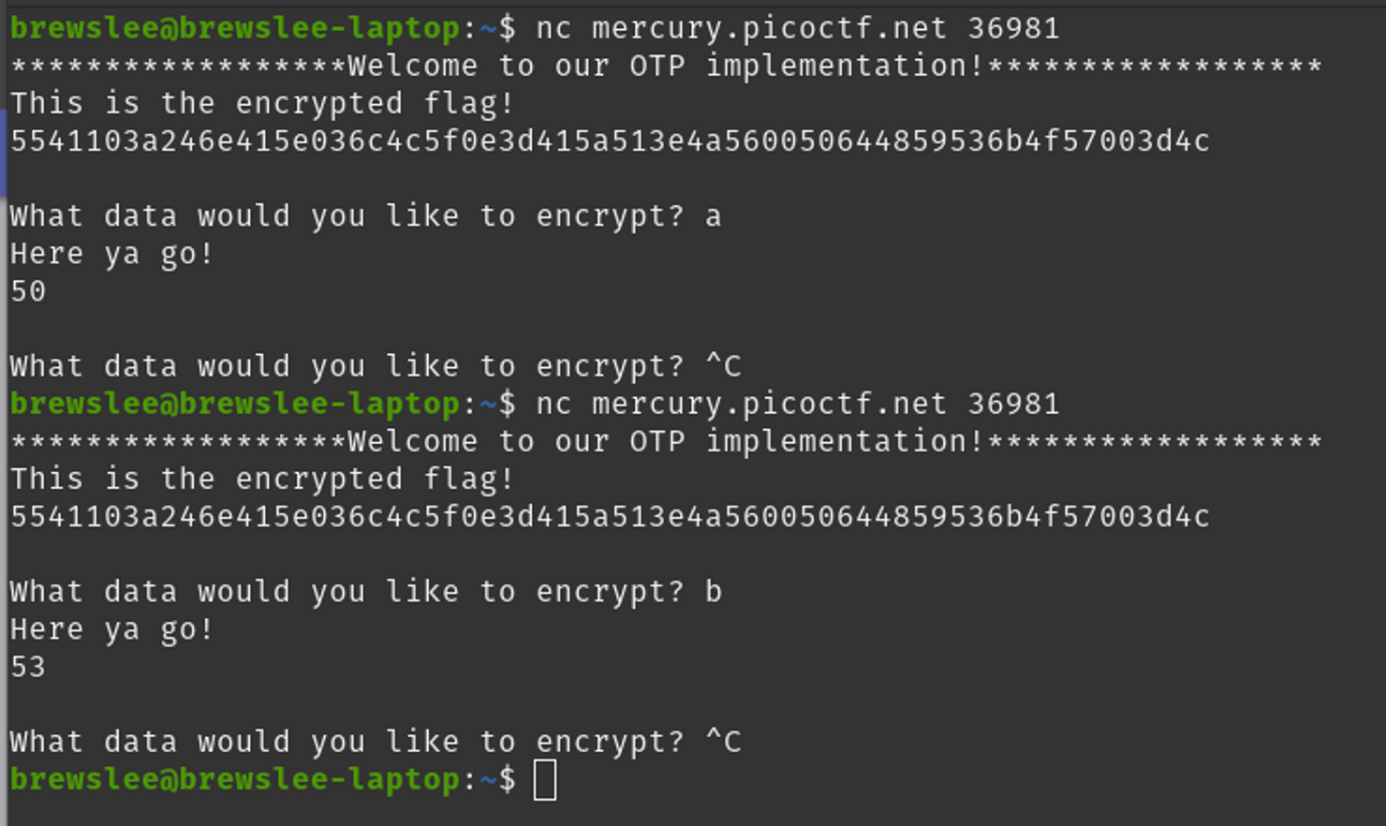
c = startup(0)

while c >= 0:

c = encrypt(c)

​

Z kódu se dá vyčíst, že program po každém zapnutí využívá stejný klíč



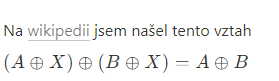
ALT

V každé instanci programu je zašifrovaná flag stejná, znamenaje, že pro a → 50 a b → 53 byl použit stejný klíč

ALT

Délka použité části klíče je vždy stejná jako délka šifrované zprávy

Výsledná zašifrovaná zpráva bude VŽDY dvojnásobně dlouhá v porovnání s plaintext zprávou



tzn.: Mám-li dvě zprávy zašifrované stejným klíčem a provedu XOR (**⊕**) jejich XOR-ovaných hodnot, dostanu XOR těchto dvou zpráv

Python

Copy

start = key\_location

stop = key\_location + len(ui)

kf = open(KEY\_FILE, "rb").read()

if stop >= KEY\_LEN:

stop = stop % KEY\_LEN

key = kf[start:] + kf[:stop]

else:

key = kf[start:stop]

key\_location = stop

​

Tahle část kódu se dá **exploitnout**,

stop je v tuto chvíli délka key\_location + len(ui) znamenaje, že pokud jsou user input (ui) + délka prvního použitého klíče, což je délka první zprávy (flag) rovny 50,000, jak je vidět zde:

Python

Copy

KEY\_FILE = "key"

KEY\_LEN = 50000

FLAG\_FILE = "flag"

​

V takovou chvíli se *stop* bude rovnat 0, protože 50,000 % 50,000 = 0 , potom se *key\_location* bude rovnat 0, protože key\_location = stop , to znamená, že při dalším provedení kódu se *start* bude rovnat také 0, protože start = key\_location , **TO ZNAMENÁ,** že pokud teď zadám 32-znakový string, měl by být **zašifrován stejným klíčem**, jako první zpráva (flag)!

Vytvořím python skriptík:

Python

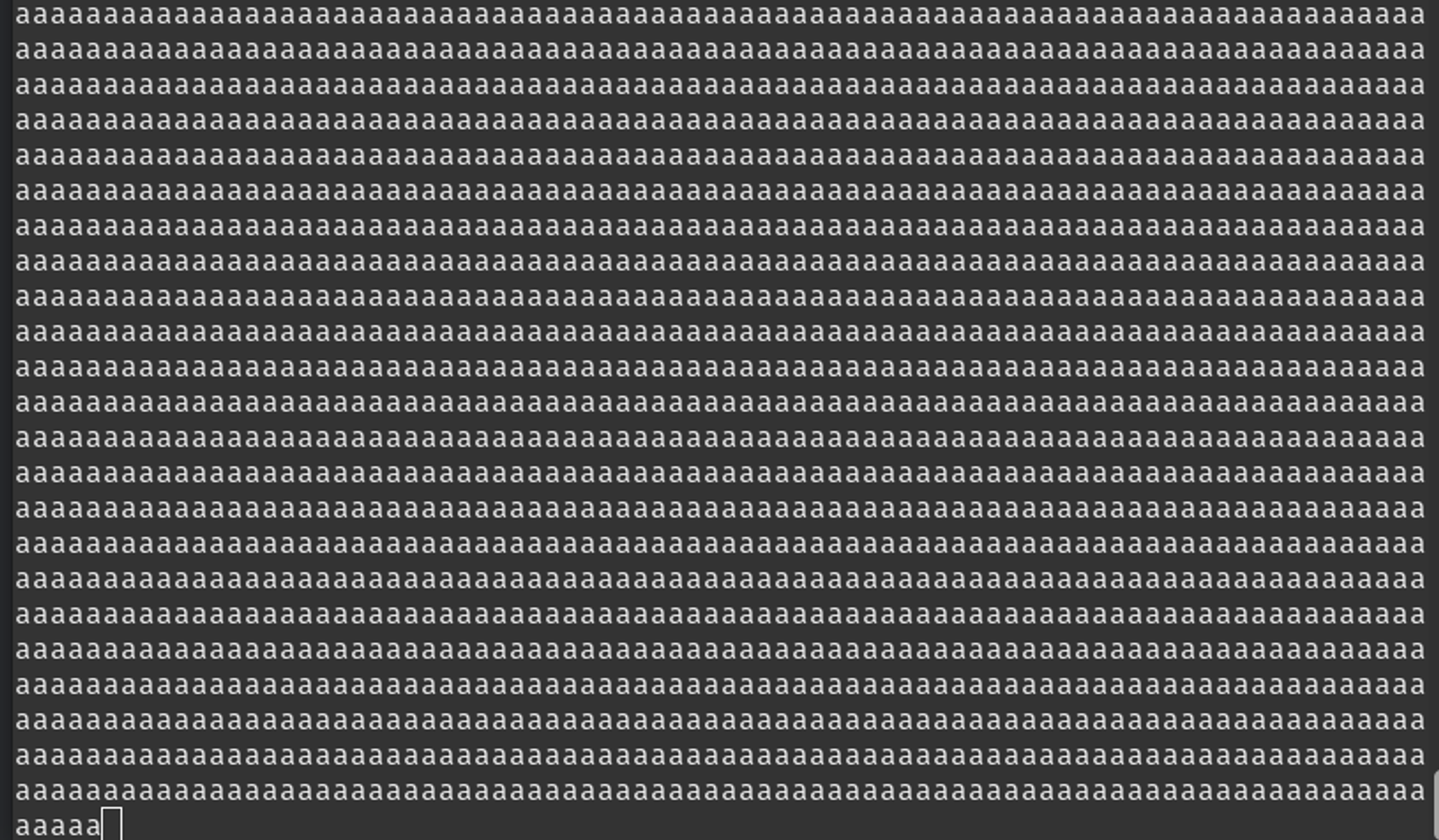
Copy

print("a" \* (50000 - 32))

# Prints 'a' 50,000 times

​

Výsledek vložím jako input:



ALT

Výsledek:



ALT

Délka výsledku je jen **8190!!!** **PROČ? Nikdo neví…** Testoval jsem to a výsledek je stejný pro 4095 ‘áček’…

Zjistím, kolikrát tam musím vložit 4095 ‘áček’, abych se dostal na 50,000

Python

Copy

print((50000-32) // 4095)

print((50000-32) % 4095)

# Outputs:

# 12

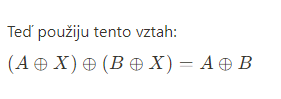
# 828

​

Musím tedy vložit 4095 ‘áček’ 12-krát a poté ještě 828 ‘áček’

Výsledek:

**0346483f243d1959563d1907563d1903543d190551023d1959073d1902573d19**



python skriptík:

Python

Copy

a: str # plaintext flag

b = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" # 12 a's

s1 = "0346483f243d1959563d1907563d1903543d190551023d1959073d1902573d19" # encrypted flag

s2 = "5541103a246e415e036c4c5f0e3d415a513e4a560050644859536b4f57003d4c"# encrypted a's

# funciton to make pairs from s1 and s2 (each pair represents a single hexadecimal number)

def make\_pairs(s: str) -> list[str]:

l = list()

i = 0

while i <= len(s) - 1:

l.append(s[i] + s[i + 1])

i += 2

return l

# list of pairs

s1\_pairs = make\_pairs(s1)

s2\_pairs = make\_pairs(s2)

# convert each of the pairs into a decimal number

s1\_dec = list(map(lambda hex: int(hex, 16), s1\_pairs))

s2\_dec = list(map(lambda hex: int(hex, 16), s2\_pairs))

# result -> XOR of A XOR X and B XOR X (2 messages encrypted using the same key)

result = [a ^ b for a, b in zip(s1\_dec, s2\_dec)]

# a -> message A (plaintext flag)

a = [ord(a) ^ b for a, b in zip(b, result)]

flag = ""

for each in a:

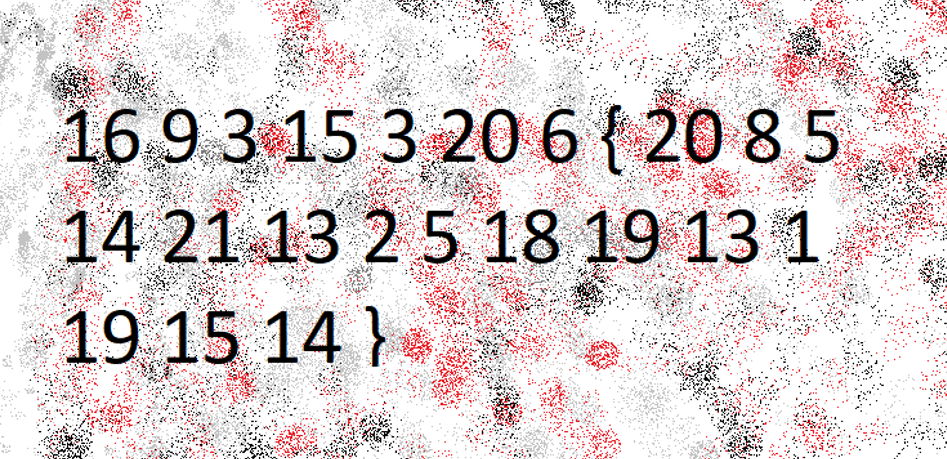
flag += chr(each)

print(flag)

<https://tungsten-gibbon-19b.notion.site/Cryptography-7f5cbf7a506d4d3e8deffcd60564e46e>

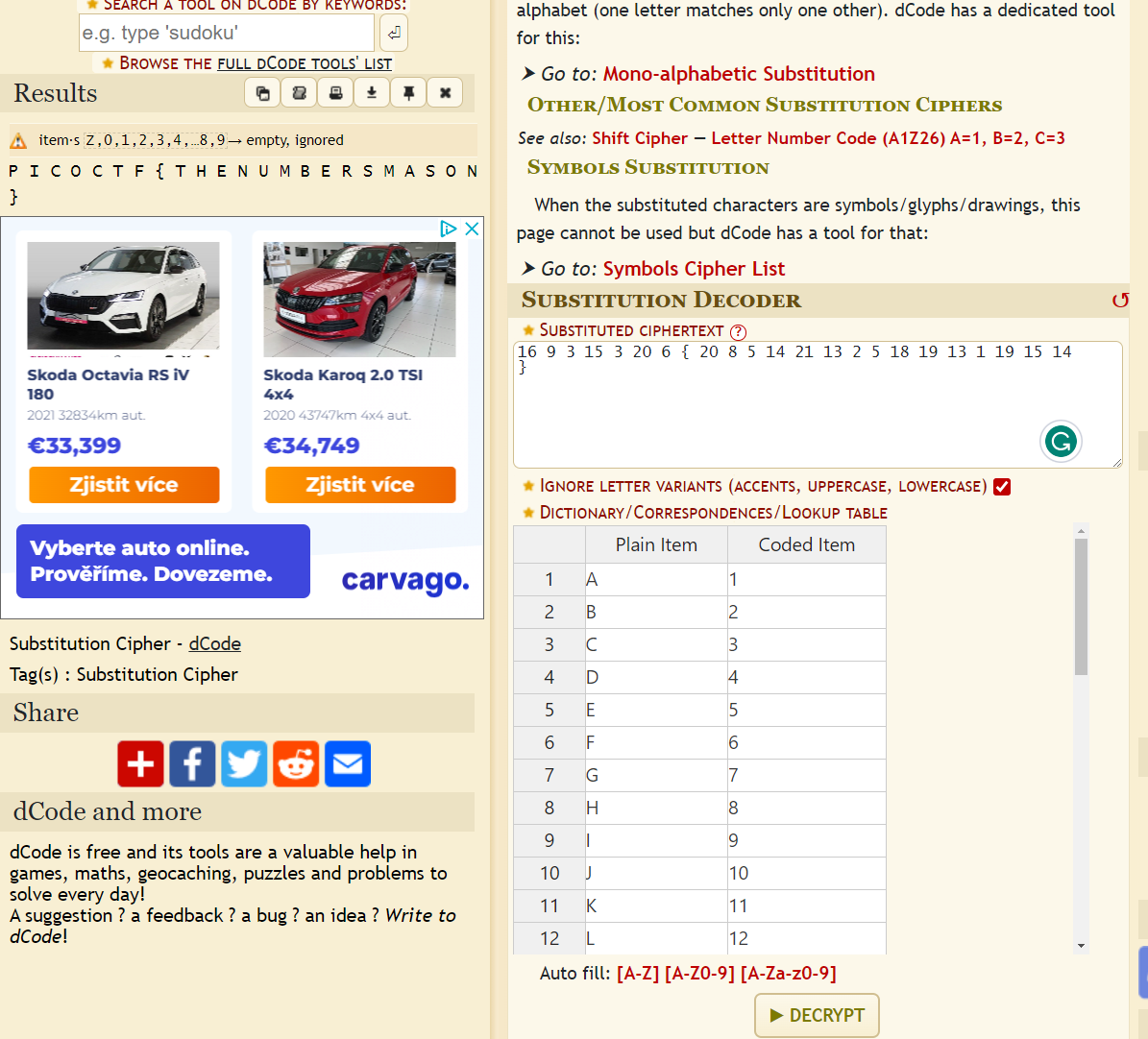
## The numbers

picoctf{thenumbersmason}

dostali jsme obrázek 

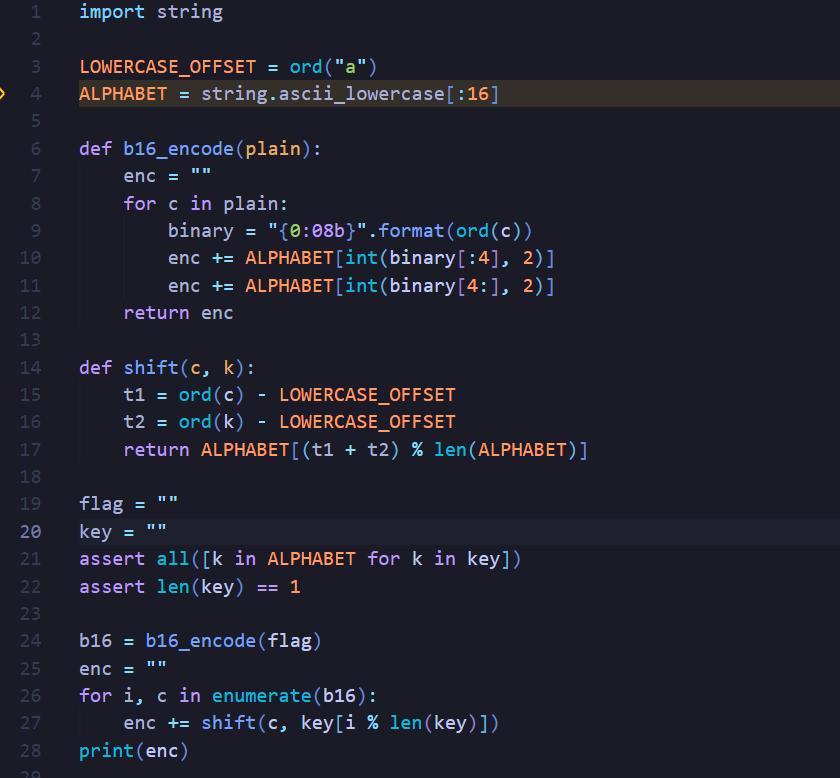
podle toho že flag má prefix picoctf jsem se kouknul do abecedy kde je p i c atd. shodují se (pokud by nebyly ale posun by byl furt stejný tak se jedná o podobnou věc)

poté stačí dát do substitution cipher kde 1 = A, 2 = B atd

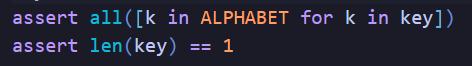


## New Ceasar (Diabolic man…)

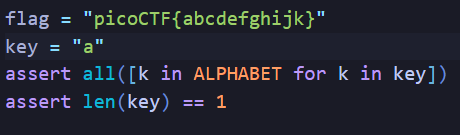
Dělání v VS code, celý začátek pythonu

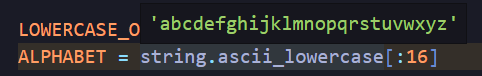
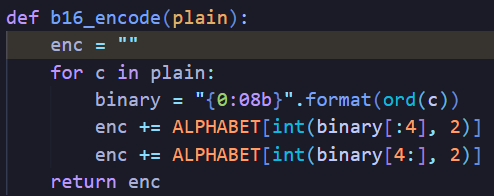
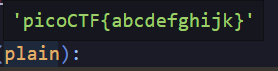
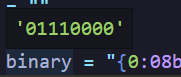
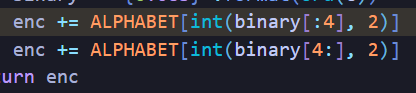
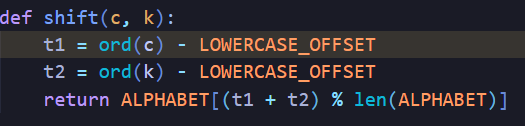


1. Prví máme problém s assertama (něco jako check v tuhle chvíli)



Dá se lehce vyřešit -> key musí obsahovat charaktery jen z prvních 16 z abecedy a musí být 1 charakter dlouhý (16 kombinací key v tento moment)



1. **Dále pojedeme popořadě co se děje**
   1.  je přetvoření na ascii hodnotu 
   2.  abeceda ale jen prvních 16 char
   3.  
      1. Plain je teď naše flaga 
      2. vezme „p“ z plainu, předělá ho do ascii  a pak to přeformátuje do bináru 
      3.  tady se k enc (locální k funkci) přidává (první 4 char našeho bináru, které jsou dány z 0111 do 7 intem, a pak to číslo vybráno z abecedy16 což je „h“)
      4. To se pak stane s druhou polovinou bináru
      5. Tohle nám to returne 
   4. Začíná shift b16 c v tomto přápadě bude “h”
   5. 

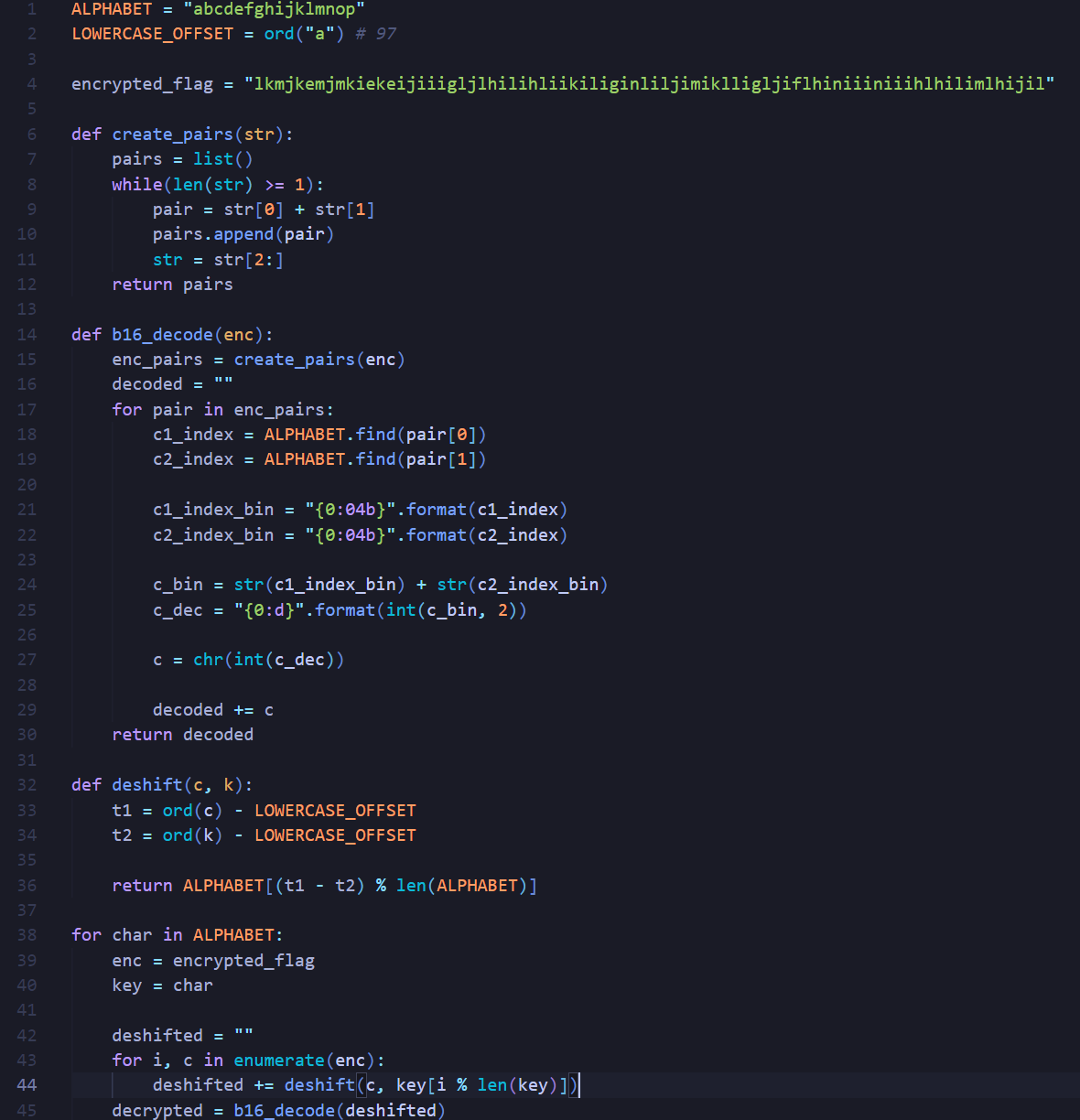
Tady c je „h“ a k je náš klíč => „a“

T1 = ascii h (104) – 97 (ord(“a“)) => 7

T2 = náš klíč (v tomoto případě „a“) (97) – 97 => 0

Return je pak ALPH[ (7+0) mod 16 ] (což v tomto případě je 7 => takže 7. pořadí v abecedě => „h“

Decipher pyhon:



Flag: et\_tu?\_431db62c5618cd75f1d0b83832b67b46

ALPHABET = "abcdefghijklmnop"

LOWERCASE\_OFFSET = ord("a") # 97

encrypted\_flag = "lkmjkemjmkiekeijiiigljlhilihliikiliginliljimiklligljiflhiniiiniiihlhilimlhijil"

def create\_pairs(str):

pairs = list()

while(len(str) >= 1):

pair = str[0] + str[1]

pairs.append(pair)

str = str[2:]

return pairs

def b16\_decode(enc):

enc\_pairs = create\_pairs(enc)

decoded = ""

for pair in enc\_pairs:

c1\_index = ALPHABET.find(pair[0])

c2\_index = ALPHABET.find(pair[1])

c1\_index\_bin = "{0:04b}".format(c1\_index)

c2\_index\_bin = "{0:04b}".format(c2\_index)

c\_bin = str(c1\_index\_bin) + str(c2\_index\_bin)

c\_dec = "{0:d}".format(int(c\_bin, 2))

c = chr(int(c\_dec))

decoded += c

return decoded

def deshift(c, k):

t1 = ord(c) - LOWERCASE\_OFFSET

t2 = ord(k) - LOWERCASE\_OFFSET

return ALPHABET[(t1 - t2) % len(ALPHABET)]

for char in ALPHABET:

enc = encrypted\_flag

key = char

deshifted = ""

for i, c in enumerate(enc):

deshifted += deshift(c, key[i % len(key)])

decrypted = b16\_decode(deshifted)

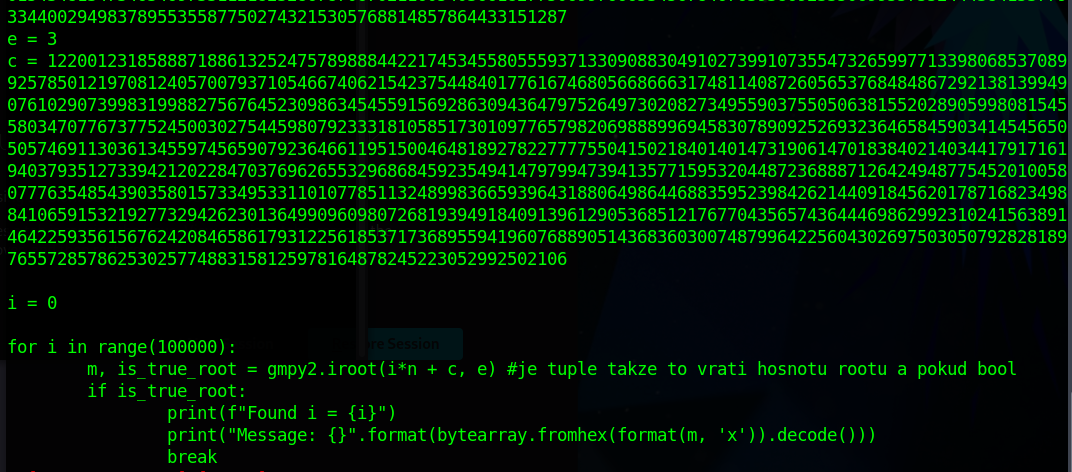
print(decrypted)

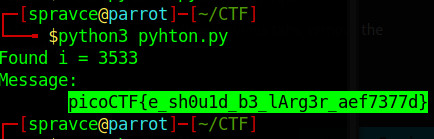
print("=====================")

## Mini RSA

POKUD BY BYLO MENŠÍ - Lze dešifrovat jako jako e tou odmocninu c, 🡪 c = m\*\*e mod n 🡪 a modulo tím pádem nedává smysl ze zadání takže m = odmocnina e céčka

TAKŽE BUDE PLATIT c = m\*\*e mod n -> M\*\*e = k\*n + c 🡪 M = étá odmocnina k\*n+c





## Dachshund Attacks

Teorie:

Wiener [2] shows that a small d can result in a total break of the RSA cryptosystem. mod φ(N), an attacker can efficiently recover d. Since φ(N) = N-p-q+1 and p+q-1 < 3√ an attacker can use N to approximate φ(N). To avoid this attack, and since N is 1024 bits, d must be at least 256 bits long.

Po nc nám vypadne:

e: 27418143245879684202129626140401269892795954132151652508603509295462646027731903567673339477884393188714636451873377263556494032055928272583209302256707209819123475430552198655874113199600170891872650224644781445485749945660452136205185217382483949606652389794791159033961201060975939849636418381988972647953

n: 88010631228102040816232578309859971494460700874876162263794959471643548594707527081696988549069602588749665401200352722925259073828840989057632494048546036540703002180123224734813828796360727933792317829056002047605951459628987476959009901401354559549491468757946326111687297725226821663028156997218096132113

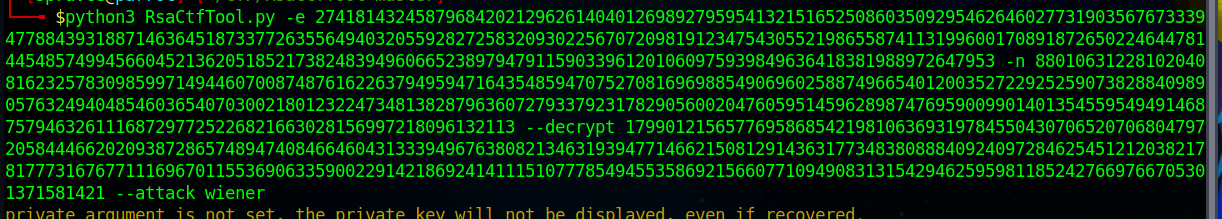
c: 17990121565776958685421981063693197845504307065207068047972058444662020938728657489474084664604313339496763808213463193947714662150812914363177348380888409240972846254512120382178177731676771116967011553690633590022914218692414111510777854945535869215660771094908313154294625959811852427669766705301371581421

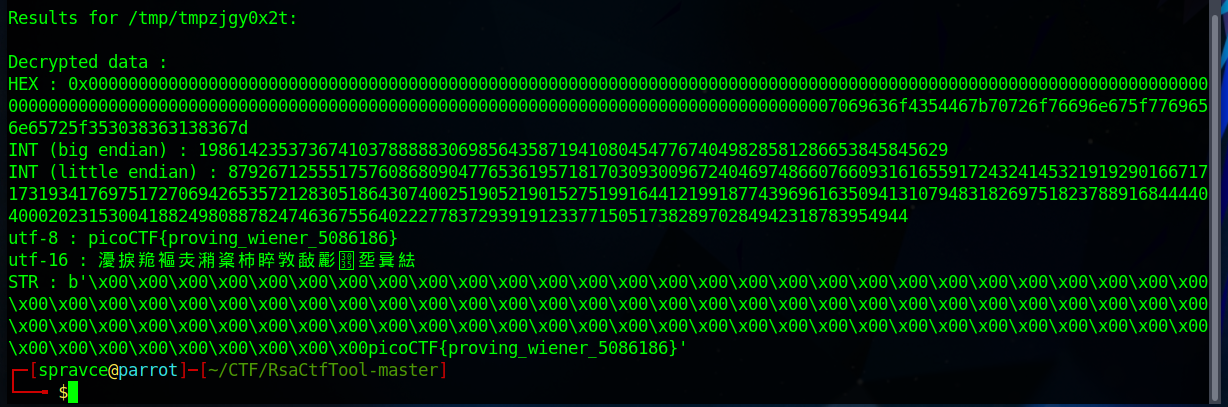
Jde dát do RSA decryption –



picoCTF{proving\_wiener\_5086186}

Použití RSACTFTOOL.PY

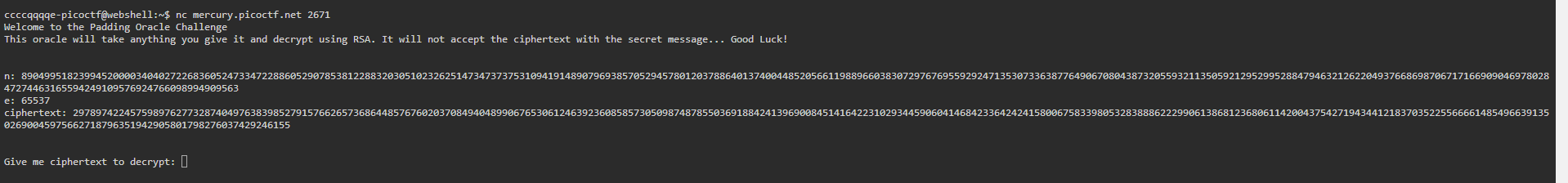


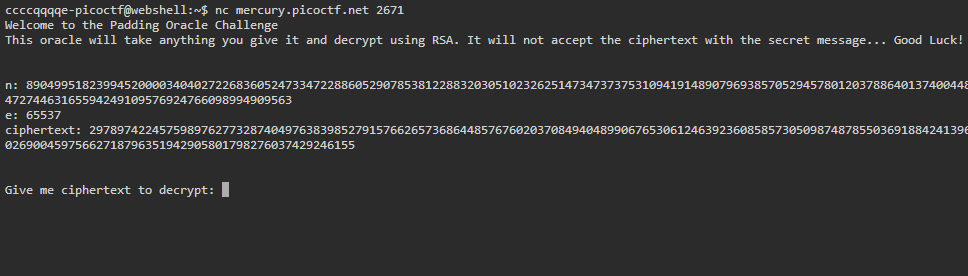


Pokud by jsme dostali jen HEX tak pak překlad z HEX na DECIMAL



## No Padding, No Problem





Po nc jsme dostali tohle nahoře

Jedná se o Homofonní šifru <https://cs.wikipedia.org/wiki/Homofonn%C3%AD_%C5%A1ifra>

Encpypt1 \* encrypt2 = ((m1 \*\* e) \* (m2 \*\*e )) mod n = … = enc(m1 \* m2)

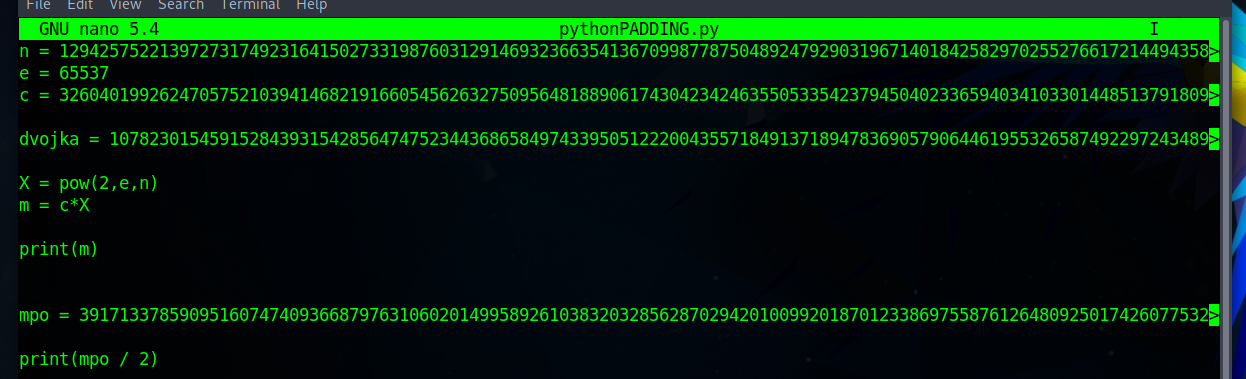
Give me ciphertext to decrypt: 2

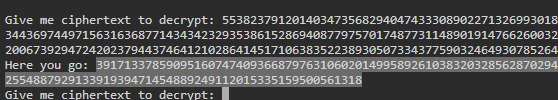
Here you go: 107823015459152843931542856474752344368658497433950512220043557184913718947836905790644619553265874922972434896770750002145846816603645053793578915613790586568345441795304730198418336293758532390067911060063078659229508702307584416462205606495813035118882865675518564220160622169309459634610142278914345086760

Pow(2, e, n)

C \* X

/ 2



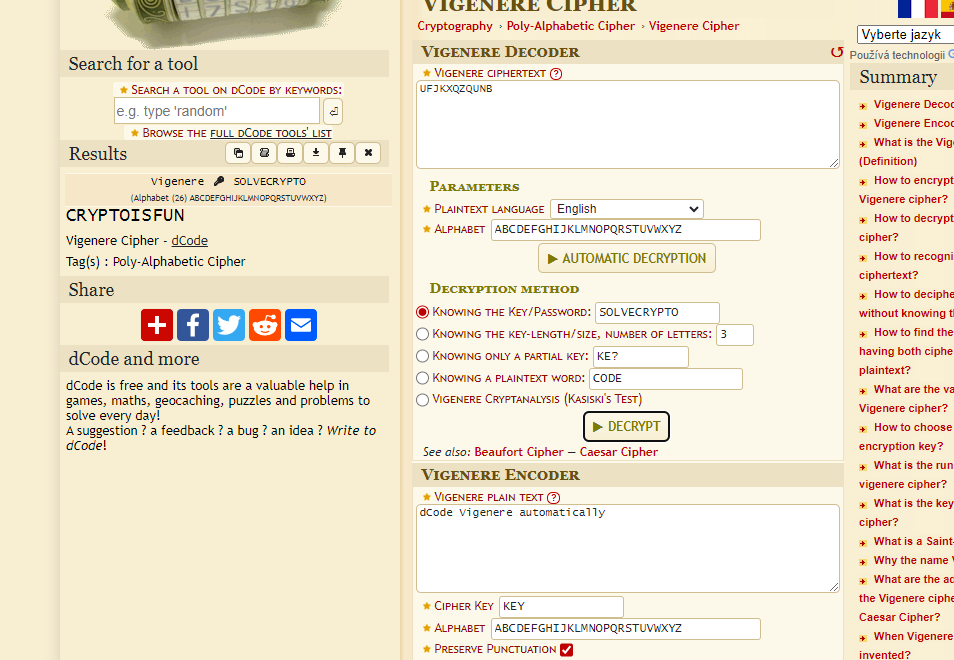


## Easy 1

Zadání:

The one time pad can be cryptographically secure, but not when you know the key. Can you solve this? We've given you the encrypted flag, key, and a table to help UFJKXQZQUNB with the key of SOLVECRYPTO. Can you use this [table](https://jupiter.challenges.picoctf.org/static/1fd21547c154c678d2dab145c29f1d79/table.txt) to solve it?.

Pomocí vigenere ciphry jde rozšifrovat s **klíčem** solvecrypto

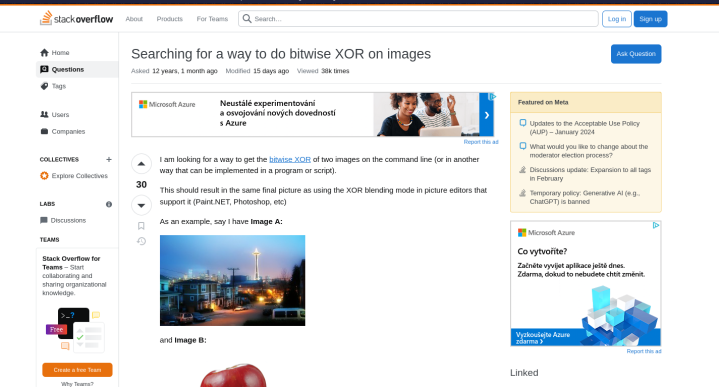


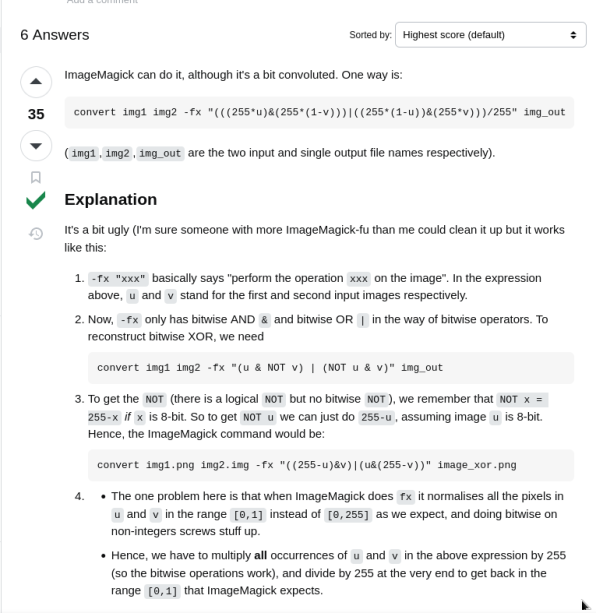
## Pixelated

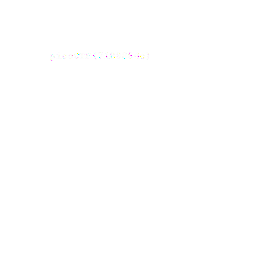
Ze zadaní jsme dostali dva pixelované obrázky a v hintě/jméně se můžeme dostat do <https://en.wikipedia.org/wiki/Visual_cryptography>

Na základě toho jsem si našel jak jde udělat XOR dvou obrázků -> program ImageMagick

Pak jsem dostal výsledek kde jsem jen musel smazat bílou barvu





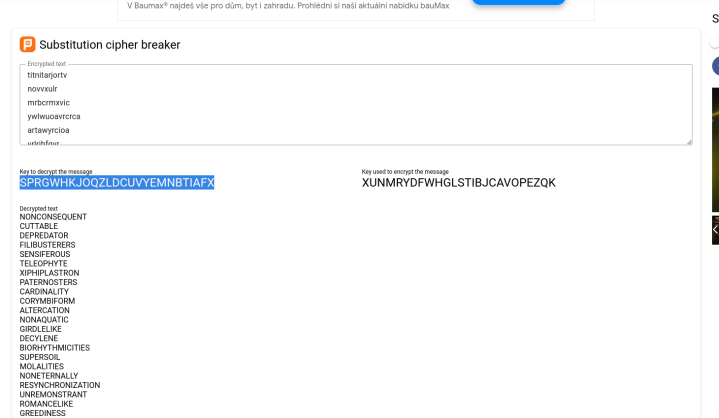




## Spelling

Dostali jsme slovník a podle substitution decyphering můžeme odvodit použitou abecedu

Pak dát na stránku která nám přemění abecdu do naší a dostaneme flagu

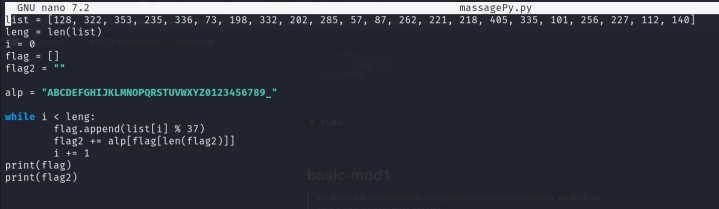


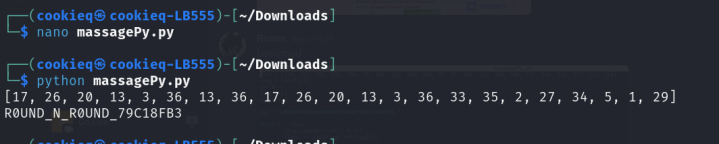


## Basicmod

Postupujeme podle zadání

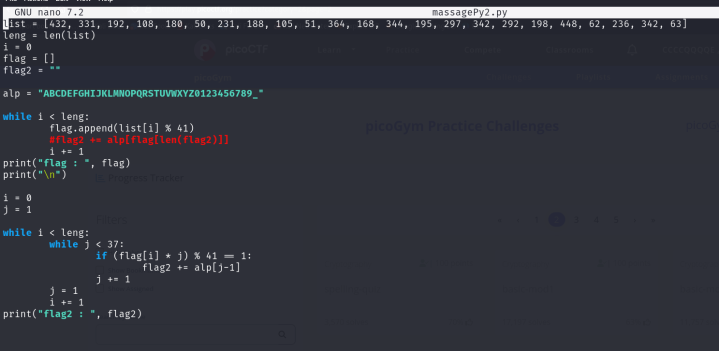
Stáhnout message -> listnout -> první každé číslo mod 37 -> pak .append podle abecedy





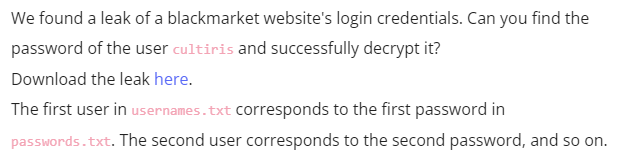
## Basicmod2

Složitější postup ale postup je podobný.



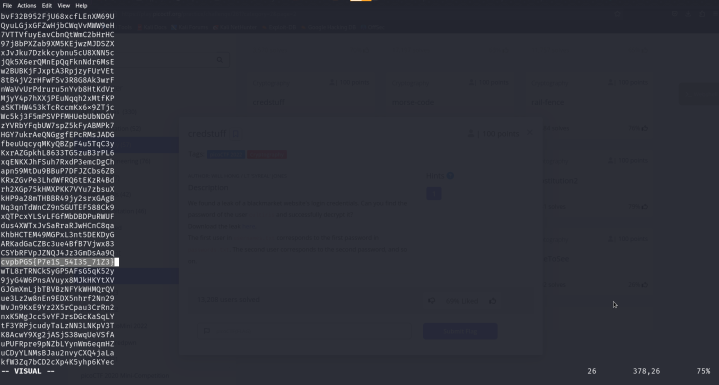


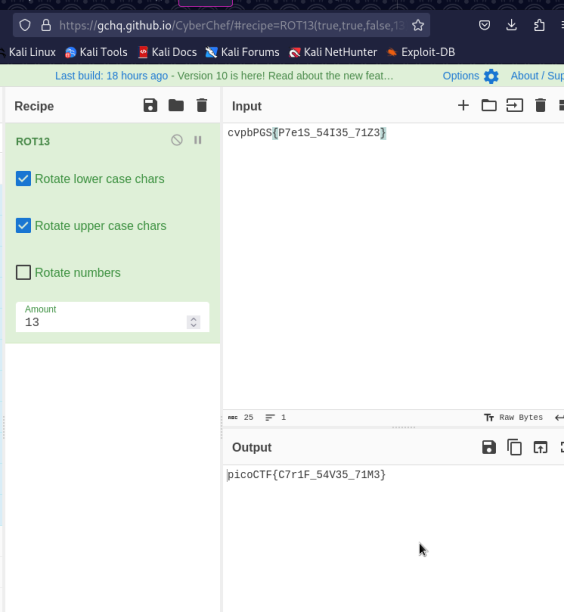
## Credstuff



Po stáhnutí jsem vyhledal jméno „ciltiris“ s grep -na což nám vypíše řádek. Nadále pro urychlení jsem si otevřel vim, kde se zobrazuje řádek a najedeme tedy na 378tý. Pak už jen stačí ROT13 a máme flag.

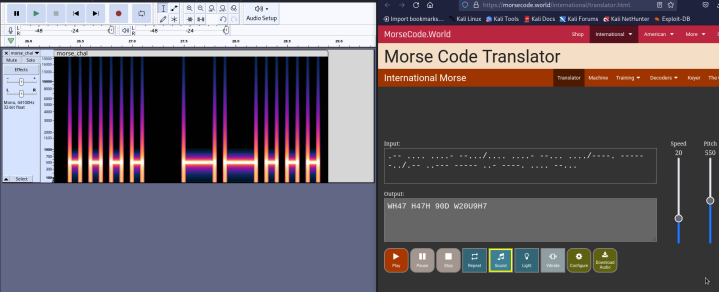






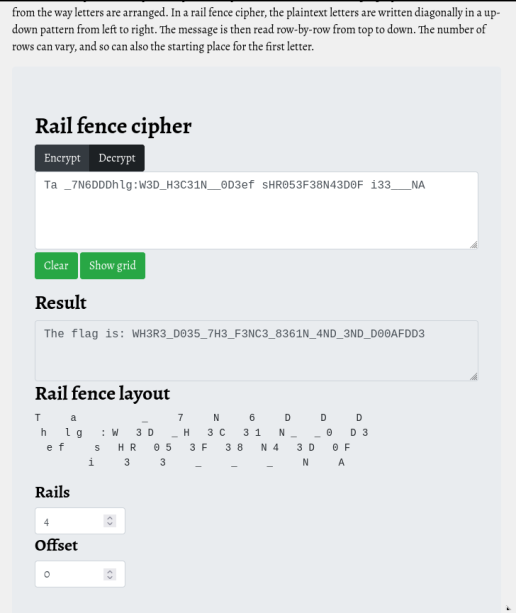
## Morse

Audio po stažení jsem si dal do Audacity -> pak pomalým ručním zápisem jsem to vypsal do online morse decoderu.



## Rail fance

Zkusil jsem online reil fance decoder a po odšifrování nám dal flagu. Ale wiki přečtení bylo hezké -> zajímavá šifra

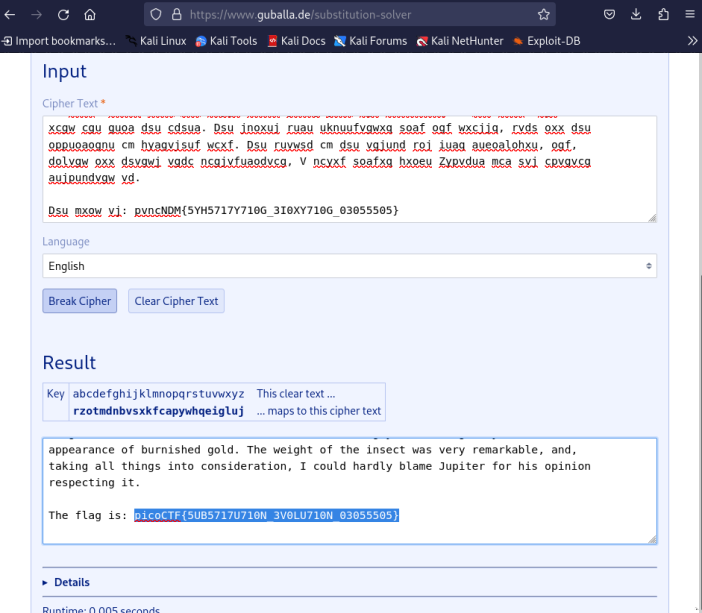


## Substitution0

Platí jak pro sub0 tak i 1 a 2. Vyhledal jsem si substitution solver kde se nějakou záhadou nemuselo ani nic nastavovat a hned to dalo flagu.

Pro další řešení podobných úloh se ale dá použít [https://www.dcode.fr/index-coincidence /](https://www.dcode.fr/index-coincidence%20/) <https://en.wikipedia.org/wiki/Index_of_coincidence#:~:text=6%20See%20also-,Calculation,%2F%20length%20of%20the%20text>).

Což by pomohlo kdyby se jednalo o neznámý jazyk.



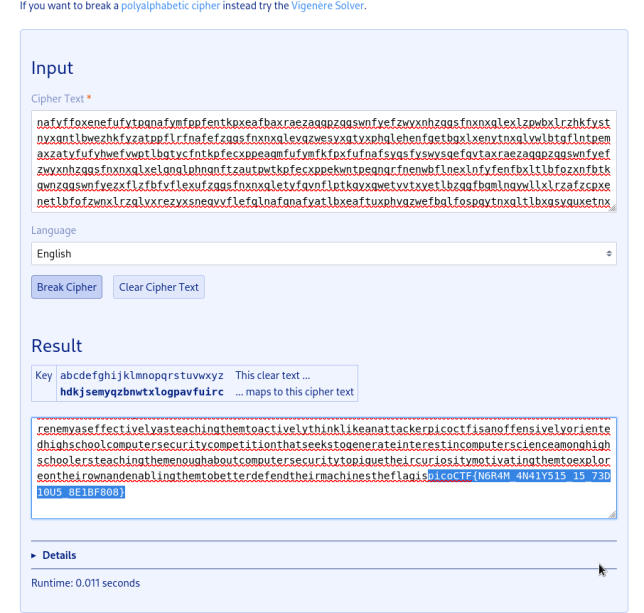
## Substitution1

Stejný postup jak při Sub0



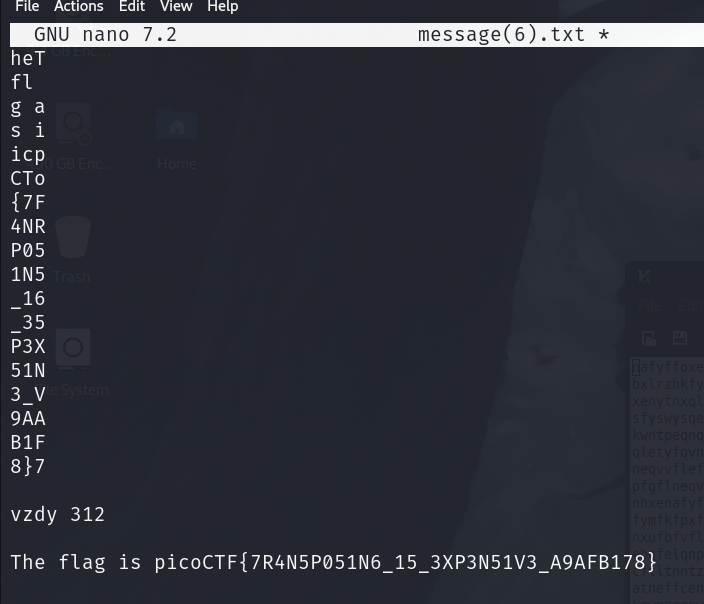
## Substitution2

Stejný postup jak při Sub0



## transposition-trial

děláno ručně v nano 😊 -> po minutě koukání jsem přišel na princip, že se opakuje zamíchání jako třetí první a druhý charakter a tak dokola



Vigenere

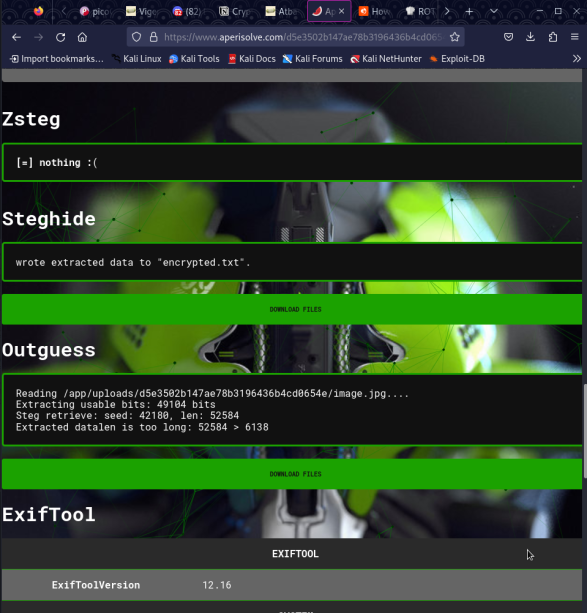
Podle zadání které se jmenuje vigenere šifra můžeme usoudit že se jedná o vigenerovu šifru

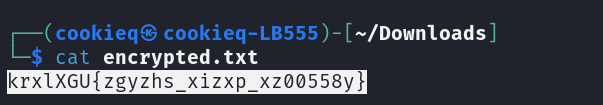
Po stáhnutí a message jsem ji dal do online vigenere decryption a zadal keyword CYLAB a vyšla flaga



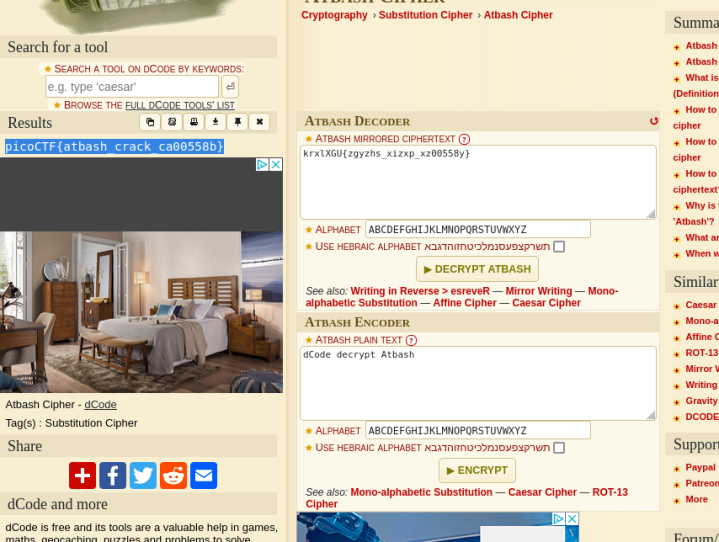
HideToSee

Po online Steghidu jsem extrahoval z obrázku encrypted.txt -> A pak zadal do Atbash decoderu a vyšla my flaga (otevřel jsem si obrázek kde byl Atbash (asi z minulosti) a tak jsem zvolil tento typ šifri)





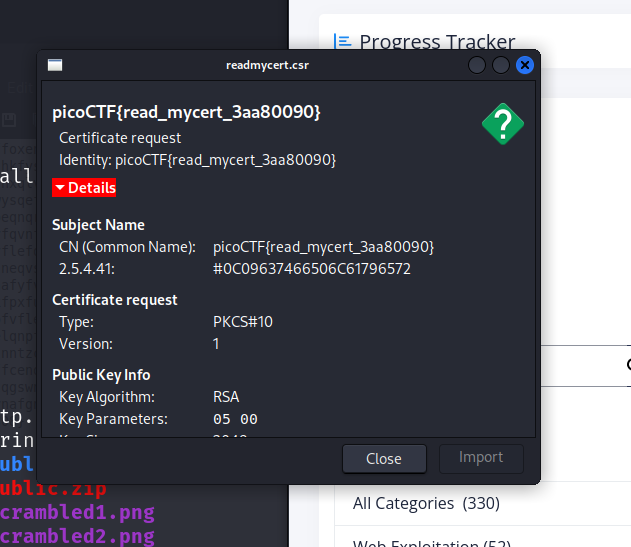
Pak jen stačí protáhnout přes Atbash decrypt



Readmycert

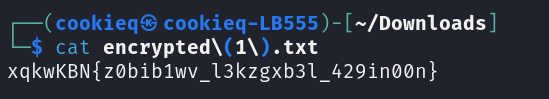
Po otevření a kouknutí se do vlastností tak najdeme flagu

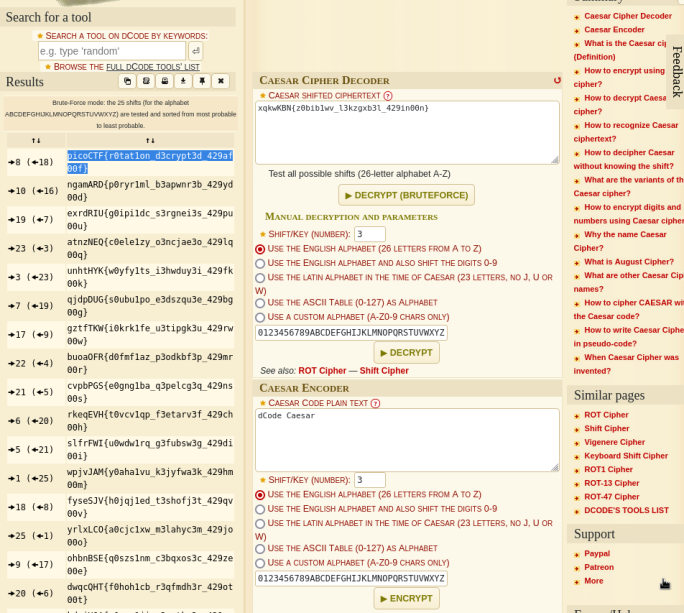
Už název byl hint, protože se tedy buď bude jednat o vlstnosti -> metadata či něco s certifikáty.



Rotation

Doslova jen otevření fily a zkoušení cipher -> jako první Caesarova.





## RSA

* Šifrování s veřejným klíčem a dešifrování soukromím

Example: (? = fí)

p = 13  
q = 7

n = 91

?(n) = (p - 1)(q - 1)  
?(91) = (13 - 1)(7 - 1)

?(91) = 12\*6

gcd(e,?(91)) = 1

e = 71

(d \* 71) % 72 = 1

public = (e,n) => (71,91)  
private = (d,n) => (503,91)

e = 17

public = (e,n) => (17,91)  
private = (d,n) => (89,91)

The public key is (n = 3233, e = 17). For a padded plaintext message m, the encryption function is

c(m)=m17mod3233

The private key is (d = 2753). For an encrypted ciphertext c, the decryption function is

m(c)=c2753mod3233

?(n) = (p - 1)(q - 1)

gcd(e,?(n)) = 1

(d \* e) % PH(n) = 1

c(m)=m na e mod n

m(c)=c na d mod n