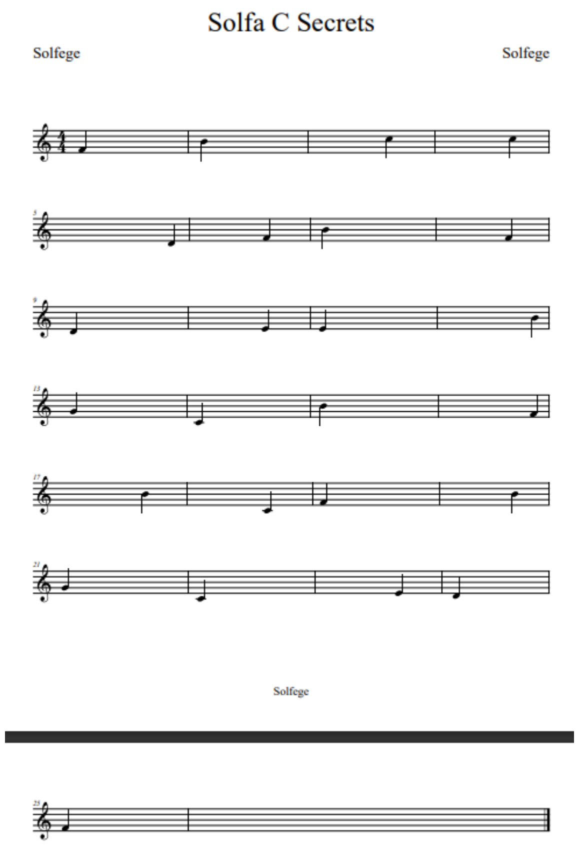
BOH2021 Cryptography Write Up

<u>Solfa's Secret [Easy]</u>

Challenge Description:

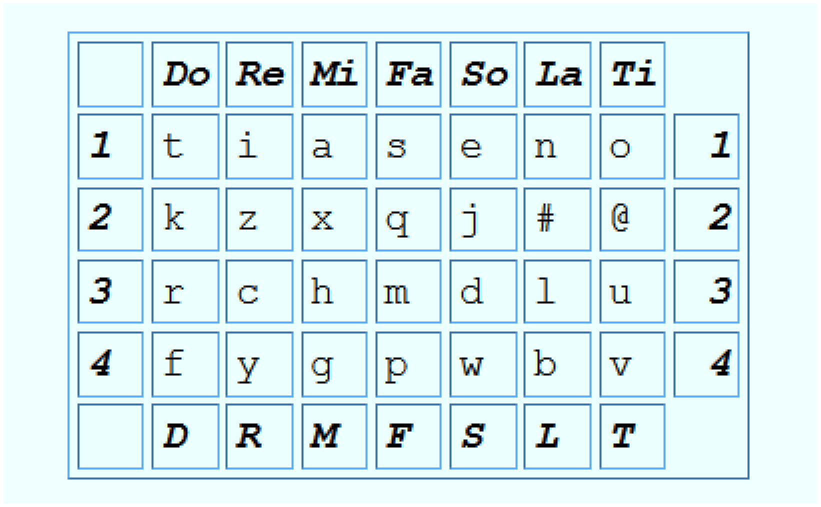
Solfa went missing and she didn't leave a note. Her parents found her laptop screen still on, with only 2 files on her Desktop.

Challenge Writeup:



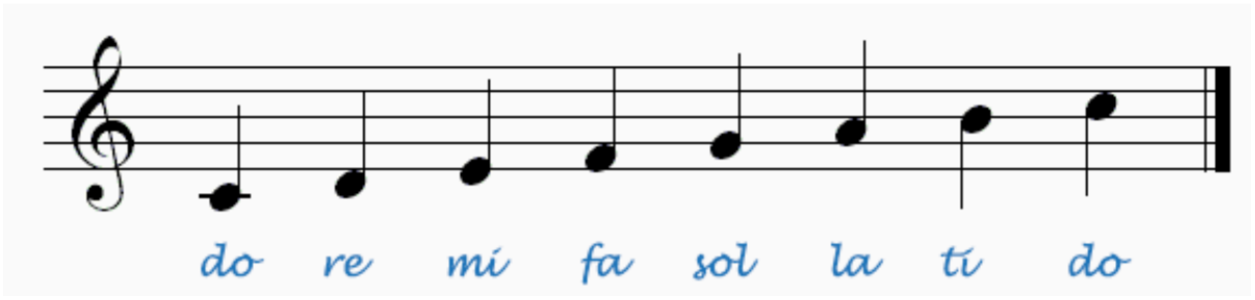
In this challenge, we are given a piano sheet and a password protected message. This music piece is called Solfa C Secrets and the word "solfege" appeared multiple times in the sheet. Seems like a hint of solfa cipher.

Solfa Cipher encodes each letter as a scale degree (Do, Re, Mi, etc) and note length (1, 2, 3, 4). To decode it, we would have to decode it manually :>, but how?



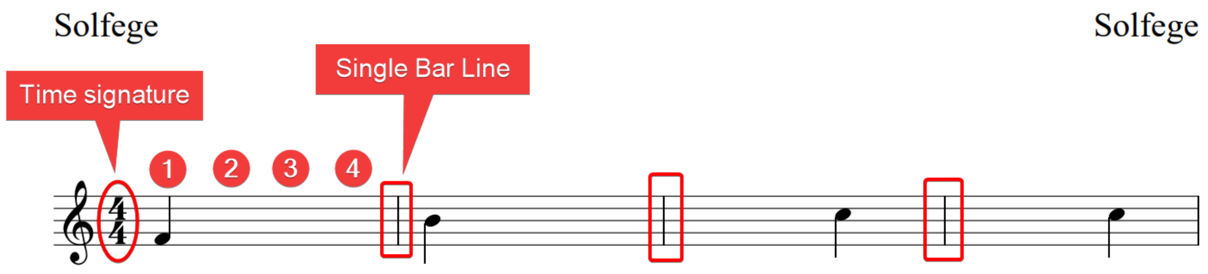
	Do	Re	Mi	Fa	So	La	Ti	
1	t	i	a	s	e	n	o	1
2	k	z	x	q	j	#	@	2
3	r	c	h	m	d	l	u	3
4	f	y	g	p	w	b	v	4
	D	R	M	F	S	L	T	

Searching the title "Solfa C Secrets" on google returns a page named Solfa Cipher Secrets that provided the cipher grid for solfa cipher. The rows represent the units of the notes (1,2,3,4) The columns represent the solfege (Do, Re, Mi, Fa, Sol, La, Ti).



The solfege syllables can be easily found on google.

Solfa C Secrets



Back to the piano sheet, the time signature of the sheet is 4/4, meaning there are always 4 beats per measure. The unit of the solfa can be determined by assigning the value based on their placement. Now we can decode the cipher manually by using the cipher grid. You would end up with:

Fa,1 – s

Ti,1 – o

Do,3 – r

Do, 3 – r

Re,4 – y

Fa,3 – m

Ti,1 – o

Fa,3 – m

Re,1 – i

Mi,3 – h

Mi,1 – a

Ti,4 – v

So,1 – e

Do,1 – t

Ti,1 – o

Fa,4 – p

Ti,3 – u

Do,3 – r

Fa,1 – s

Ti,3 – u

So,1 – e

Do,1 – t

Mi,3 – h

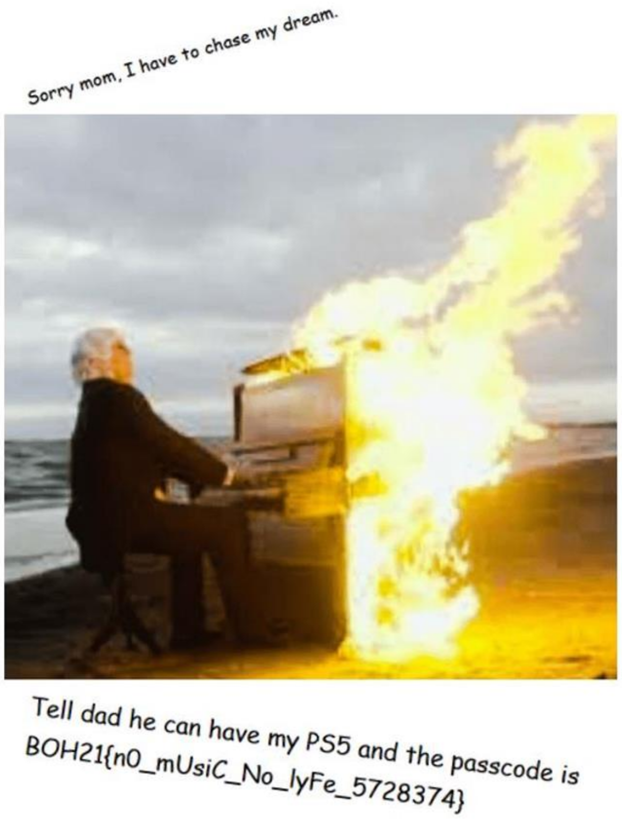
Re,1 – i

Fa,1 – s

Joining the letters, you would get:

sorrymomihavetopursuethis

Use “sorrymomihavetopursuethis” as the key to extract the password protected message.7z and the flag is yours:


Sorry mom, I have to chase my dream.
Tell dad he can have my PS5 and the passcode is BOH21{n0_mUsiC_No_lyFe_5728374}

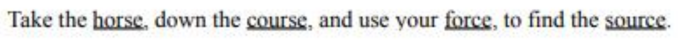
Flag: BOH21{n0_mUsiC_No_lyFe_5728374}

Military Operation Plan [Easy]

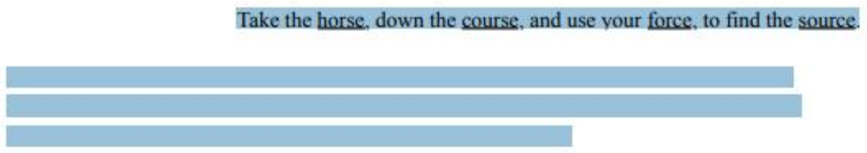
Challenge Description:

I've intercepted our enemy's military comms, and these are the files retrieved. Can you figure out what it is? Looks like an operation plan.

Challenge Writeup:
In this challenge, we are given an accesscode.pdf and a password protected file, secretplan.7z.


Take the horse, down the course, and use your force, to find the source.

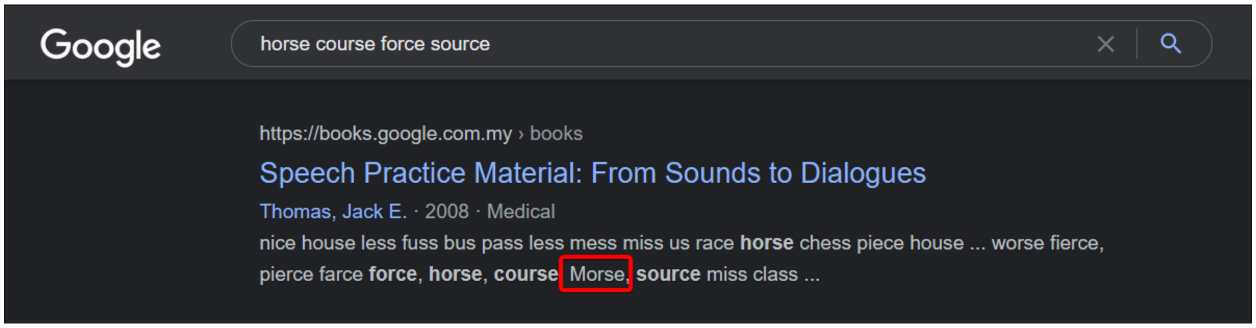
The content of the accesscode.pdf includes an image of a camouflaged soldier and a string. The image is a hint that something is hiding.



Take the horse, down the course, and use your force, to find the source.

Pressing ctrl+A shows the hidden content in the pdf. Lets take a look at it:

“@@@@ @# @@ @#@@ @# @#@ @ @# @@@@@ @#### @@#@ ### @#@ @ @@@# @ @#@ @@#@ @@@ @ #@#@ @@@ @@@ @# @#@ ## #@## @#### @#### ##### #@@@@ @#### @#### ##### ##@@@”


horse course force source

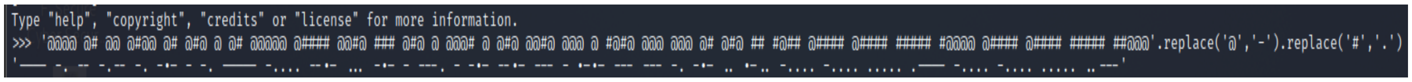
https://books.google.com.my › books

Speech Practice Material: From Sounds to Dialogues

Thomas, Jack E. · 2008 · Medical

nice house less fuss bus pass less mess miss us race **horse** chess piece house ... worse fierce, pierce farce **force**, **horse**, **course**, Morse, **source** miss class ...

From the string (not the hidden one) provided in the pdf, the words "horse", "course", "force", and "source" are emphasized, and they all rhyme with morse, hence hinting its morse code?


```
Type "help", "copyright", "credits" or "license" for more information.
>>> '@@@@ @# @@ @#@@ @# @#@ @ @# @@@@@ @#### @@#@ ### @#@ @ @@@# @ @#@ @@#@ @@@ @ #@#@ @@@ @@@ @# @#@ ## #@## @#### @#### ##### #@@@@ @#### @#### ##### ##@@@'.replace('@','-').replace('#','.')
'---- -. -- -.-- -. -.- - -. ----- -.... --.- ... -.- - ---. - -.- --.- --- - .-.- --- --- -. -.- .. .-.. -.... -.... ..... .---- -.... -.... ..... ..---'
```

Morse code is represented in '-' and '.'. Open up python idle and replace '@' to '-' and '#' to '.' returns a morse code:

---- -. -- -.-- -. -.- - -. ----- -.... --.- ... -.- - ---. - -.- --.- --- - .-.- --- --- -. -.- .. .-.. -.... -.... ..... .---- -.... -.... ....---

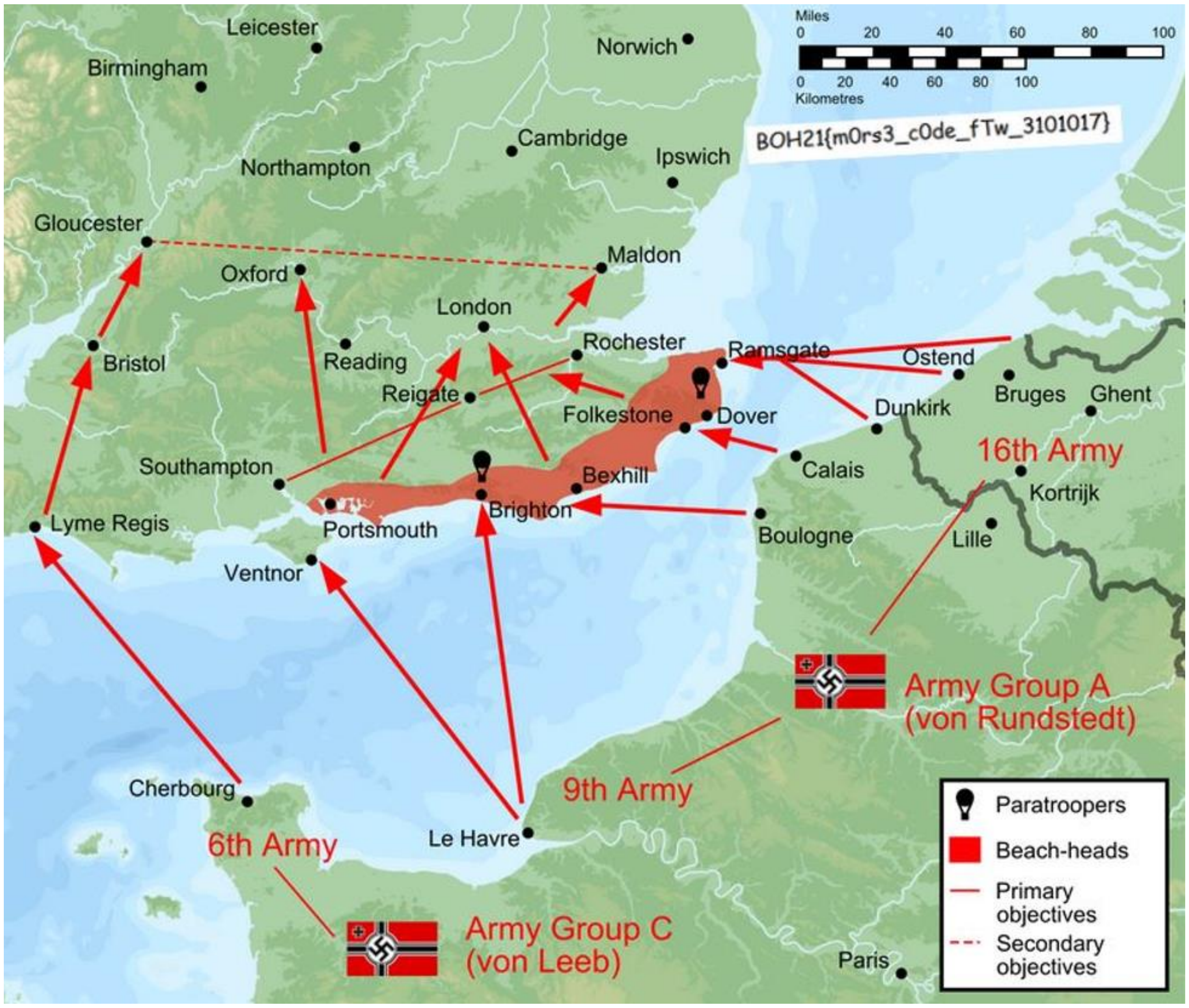
which decodes to:

ĤNMYNKTN06QSKTÓTKQOTÄOONKIL66516652

However, the decoded string is the incorrect password for the secretplan.7z. Thus, let's try replacing '@' to '.' and '#' to '-' and decode the returned morse code, you will get:

HAILAREA51FOREVERFSECSSARMY11061107

Use the decoded string to unzip the secretplan.7z and the flag is on the map.



Flag: BOH21{m0rs3_c0de_fTw_3101017}

Crypt4fun [Medium]

Challenge Description:

The flag is encrypted with a python program. Happy Decoding!

Challenge Writeup:

Players are provided with 2 files:

1. flag.enc – the encoded flag.

2. crypt.py – the python program used to encode the flag.

```
import hashlib

inp = input("inp:")
result = ""
c = ""

for e in inp:
   e  = hashlib.md5(e.encode())
   e  = e .hexdigest()
   c +=str(e )

for l in c:
   l  = hashlib.sha1(l .encode())
   l  = l .hexdigest()
   result += str(l)

f = open('flag.enc','w')
f.write(result)
f.close
```

The encryption of the flag is easy to understand. The crypt.py requests for an input flag. Each character in the flag is encoded with md5 hash, and each character in the md5 hash is encoded with sha1 hash. The program writes the encoded string to 'file.enc'.

To decode the flag, reverse the encryption by decoding Sha1 hash first and then md5. Sha1 hash consist of 40 characters, therefore, to decode it, for each 40 characters in the 'file.enc' will be compared to the hash of every key available on your keyboard until the matching hash is found and return the letter of that hash. After decoding Sha1, the same goes to md5, the only difference is that the length of an md5 hash is 32 characters long. The full script to decode the flag is provided below:

```
import hashlib


def compare_sha(shaChar):
    # every characters in keyboard // If we save it hash it can reduce
execution time
    keys = ['`', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=',
'q', 'w', 'e', 'r', 't', 'y', 'u', 'i',
            'o', 'p', '[', ']', 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', ';',
"'", 'z', 'x', 'c', 'v', 'b', 'n', 'm',
            ',', '.', '/', '~', '!', '@', '#', '$', '%', '^', '&', '*', '(',
')', '_', '+', 'Q', 'W', 'E', 'R', 'T',
            'Y', 'U', 'I', 'O', 'P', '{', '}', '|', 'A', 'S', 'D', 'F', 'G',
'H', 'J', 'K', 'L', ':', '"', 'Z', 'X',
            'C', 'V', 'B', 'N', 'M', '<', '>', '?']
    for key in keys:
        hash_key = hashlib.sha1(key.encode())
        hash_key = hash_key.hexdigest()
        if hash_key == shaChar:
            return key


def compare_md5(md5char):
    # every characters in keyboard // If we save it md5 it can reduce
execution time
    keys = ['`', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=',
'q', 'w', 'e', 'r', 't', 'y', 'u', 'i',
            'o', 'p', '[', ']', 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', ';',
"'", 'z', 'x', 'c', 'v', 'b', 'n', 'm',
            ',', '.', '/', '~', '!', '@', '#', '$', '%', '^', '&', '*', '(',
')', '_', '+', 'Q', 'W', 'E', 'R', 'T',
            'Y', 'U', 'I', 'O', 'P', '{', '}', '|', 'A', 'S', 'D', 'F', 'G',
'H', 'J', 'K', 'L', ':', '"', 'Z', 'X',
            'C', 'V', 'B', 'N', 'M', '<', '>', '?']
    for key in keys:
        md5 = hashlib.md5(key.encode())
        md5 = md5.hexdigest()
        if md5 == md5char:
            return key


with open("hash.txt", "r") as myfile:
    data = myfile.readlines()

data = data[0]

# Decoding the sha1

# Got by dividing hashed length by md5 length
eachCharacterSha1Length = 40

# separating each character hashing
words = [(data[i:i + eachCharacterSha1Length]) for i in range(0, len(data),
eachCharacterSha1Length)]
deHashedWords = ""

for letter in words:
    # Comparing hash and getting pre hashed character
    deHash = compare_sha(letter)
    deHashedWords += deHash

# Decoding MD5

# Got by dividing md5 length by original character length
eachCharacterMd5Length = 32

# separating each character md5 from de-hashed words
words = [(deHashedWords[i:i + eachCharacterMd5Length]) for i in range(0,
len(deHashedWords), eachCharacterMd5Length)]
decryptedWords = ""

for letter in words:
    # Comparing md5 and getting pre md5 character
    decrypted = compare_md5(letter)
    decryptedWords += decrypted

# This should be an original text
print decryptedWords
```

Flag: BOH21{mD5_sHa1_S0_cRyT0}

Crypt4fun2 [Medium]

Challenge Description:

The flag is encrypted with a python program.....again.....so....Happy Decoding!

Challenge Writeup:
2 files are provided:

1. output.txt – the encrypted flag

```
885, 822, 674, 33, 835, 564, 144, 856, 407, 644, 560, 160, 48, 609, 838, 615,
412, 154, 367, 741, 703, 77
```

2. crypt.py – python program used to encrypt the flag

```
def magic(x):
   magiclist = []
   for i in range(100):
      x = ((x<<5)*x-~x<<8)%971
      magiclist.append(x)
   return magiclist


flag = open('f').read()
output = []
r = randint(0,1000)
g = magic(r)

for i in range(len(flag)):
   output.append(ord(flag[i]) ^ g[i])

print(output)
```

Again….reverse encryption. According to the encryption code, each character of the flag is xor'ed with a list of numbers generated by the magic() function. The magic function gets a parameter 'x' and generate an array list of 100 numbers using the 'x' in a complicated equation 'x = ((x<<5)*x-~x<<8)%971'. The list is then stored as 'g' variable, which is then xor'ed with the flag.

However, here's the catch. The 'x' is randomly generated by the randint() function and the random number is between 0 to 1000. So, every time the encryption is executed, the output will be unique and different from previous output because the flag would be xor'ing with a different list of number.

In order to get the flag from the output.txt, we would have to determine the random number used for the magic() function.

Since it is about xor encryption, then it can be reversed easily as:

A ^ B = C

A ^ C = B

B ^ C = A

To apply this in our challenge:

1. ‘g’ = magic(x)

2. Flag ^ g = Output.txt

   Flag ^ Output.txt = g

   g ^ Output.txt = Flag

3. We need to find ‘x’ first in order generate the complete ‘g’ list, then we can find the flag by xor’ing the ‘g’ list with the output.txt as Flag = g ^ Output.txt.

We already have the output.txt as provided by the challenge itself. And since the flag would definitely start with “BOH21{”, we can use the start of the flag and xor it with the first 6 numbers of output.txt to get the first 6 numbers of ‘g’ list. The script is shown below:

```
output = [885, 822, 674, 33, 835, 564, 144, 856, 407, 644, 560, 160, 48, 609,
838, 615, 412, 154, 367, 741, 703, 77]
flag = 'BOH21{'

print("flag ^ output:")
for i in range(len(flag)):
    print(ord(flag[i]) ^ output[i], end = ' ')
```

The result is [**823, 889, 746, 19, 882, 591**]. These numbers are the first 6 numbers of ‘g’ list. With these numbers, we can brute force the magic(x) function to see what number was used to generate these numbers. Since the random number generated is between 0 and 1000, we can use the magic(x) function to trace back the numbers between 0 to 1000 that can generate the first value “**823**” in ‘g’ list:

```
def magic(x):
    magiclist = []
    for i in range(100):
        x = ((x << 5) * x - ~x << 8) % 971
        magiclist.append(x)
    return magiclist


print("number for x that will have g start in 823")
for i in range(1000):
    g = magic(i)
    if g[0] == 823:
        print(i, end=' ')
```

The result is **517, 545**. Both numbers generated ‘g’ list that starts with 823, now let’s test which number can generate the 6 exact same values of the ‘g’ list [**823, 889, 746, 19, 882, 591**]:

```python
def magic(x):
    magiclist = []
    for i in range(100):
        x = ((x << 5) * x - ~x << 8) % 971
        magiclist.append(x)
    return magiclist


x = 517
print("Next 6 digits in g for x starting in {x}")
g = magic(x)
for i in range(6):
    print(g[i], end=' ')
```

The result is [**823, 889, 746, 19, 882, 591**] which is identical. This means we found the ‘x’! The number 545 generate the same output as well.

But the point is….we found the ‘x’! Now we can use the ‘x’ to generate the entire ‘g’ list using the magic(x) function and find the flag by xor’ing the ‘g’ list with the output.txt:

```python
def magic(x):
    magiclist = []
    for i in range(100):
        x = ((x << 5) * x - ~x << 8) % 971
        magiclist.append(x)
    return magiclist


output = [885, 822, 674, 33, 835, 564, 144, 856, 407, 644, 560, 160, 48, 609,
838, 615, 412, 154, 367, 741, 703, 77]
x = 517
g = magic(x)

print("Flag:")

for i in range(len(output)):
    print(chr(output[i] ^ g[i]), end='')
```

The flag is all yours <3.

Flag: BOH21{bL4cKmaG!cl33t}

Crypt4fun3 [Medium]

Challenge Description:

Decrypt the keys to decrypt flag. Hint: The railway fences prevented the train that carries our bombe from being derailed.

Challenge Writeup:

Players are given:

1. Five encoded keys with orders from 1 to 5

    Key1 = Sswski

    Key2 = U54K_E1NW_

    Key3 = I0HI2T8__

    Key4 = IMX_3_412

    Key5 = I57I_E1QI_

2. encryptkeys.py – an encryption python program that encrypts the keys

3. encodedsecret – A encoded paragraph that contains the flag.

According to the description, this challenge involves encrypted keys that are used to decrypt the flag. Hence to decrypt the flag, keys are to be decrypted first. The keys are encrypted with 'encryptkeys.py'. That would be the starting point of this challenge. There is a hint for this challenge:

> The railway fences prevented the train that carries our bombe from being derailed.

Railway fences is a hint for railfence cipher, but what is a bombe? Let's find out later.

The encryptkeys.py looks like this:

```python
import string

def encrypt(text, key):

    w = [['\n' for i in range(len(text))]
                  for j in range(key)]

    dir_down = False
    r, c = 0, 0

    for i in range(len(text)):

        if (r == 0) or (r == key - 1):
            dir_down = not dir_down

        w[r][c] = text[i]
        c += 1


        if dir_down:
            r += 1
        else:
            r -= 1

    result = []
    for i in range(key):
        for j in range(len(text)):
            if w[i][j] != '\n':
                result.append(w[i][j])
    return("" . join(result))

def replace(text):
    text = [x.replace(' ', '_') for x in text]
    return ("" . join(text))

t = open(**keys**).read()
k = 3
res = replace(encrypt(t, k))
print(res)
```

The encryption code involves 2 functions – replace(text) and encrypt(text, key). Replace(text) is only to replace the spaces in the encrypted text to '_'. The encrypt(text, key) function is the main encryptor here.

The encrypt(text, key) function requires 2 parameters which are a string of 'text' and an integer 'key'. The function first declares a matrix and named it 'w', the 'key' and 'text' parameters will determine the number of rows and columns in 'w' matrix. Then it starts filling the matrix with characters of 'text'.

After each character is filled, the column will increase its value by 1 constantly, but not the row. The ‘dir_down’ flag (which first declared as false) is used to find the next row of the filling process. When the top or bottom row is reached, the flag changes its value, thus the row changes. After the all the characters are filled, the code then joins the characters in matrix row by row. The replace(text) function then replace the spaces of the cipher with “_”.

To explain how the code works in detail, let’s take “Hello World!” as the string to be encrypted with this code. Since the key is given which is 3, the encrypt() function then creates a matrix with 3 rows (length of key), and 12 columns (length of string):

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												

The first character of the string is ‘H’ and it will be filed to row 0 and column 0:

	0	1	2	3	4	5	6	7	8	9	10	11
0	H											
1												
2												

After filling the first character, the column value is increased by 1. When the row is 0, the dir_down flag which is initially False, will be changed to True. When dir_down flag is True, the row value will be increased by 1, hence making the next character to be filled at row 1 column 1:

	0	1	2	3	4	5	6	7	8	9	10	11
0	H											
1		E										
2												

Column then increased by 1. When the row value is 1, the dir_down flag remained unchanged, hence the row value is increased by 1, Making the next character to be filled at row 2 column 2:

	0	1	2	3	4	5	6	7	8	9	10	11
0	H											
1		e										
2			l									

Column then increased by 1. When the row value = key – 1, the dir_down flag will have its value inverted. Key value is 3, 3 – 1 = 2, the row value reached 2 after last filing, hence, IF condition is triggered in the code. The dir_down flag is then changed to False, the row value will then be decreased by 1. Finally, making the next character to be filled at row 1 column 3:

	0	1	2	3	4	5	6	7	8	9	10	11
0	H											
1		e		l								
2			l									

When all the characters are filled, the matrix will look like this:

	0	1	2	3	4	5	6	7	8	9	10	11
0	H				o				r			
1		e		l		(space)		o		l		!
2			l				W				d	

Then characters are joined row by row, leading to the final cipher to be:

Horel ol!lWd

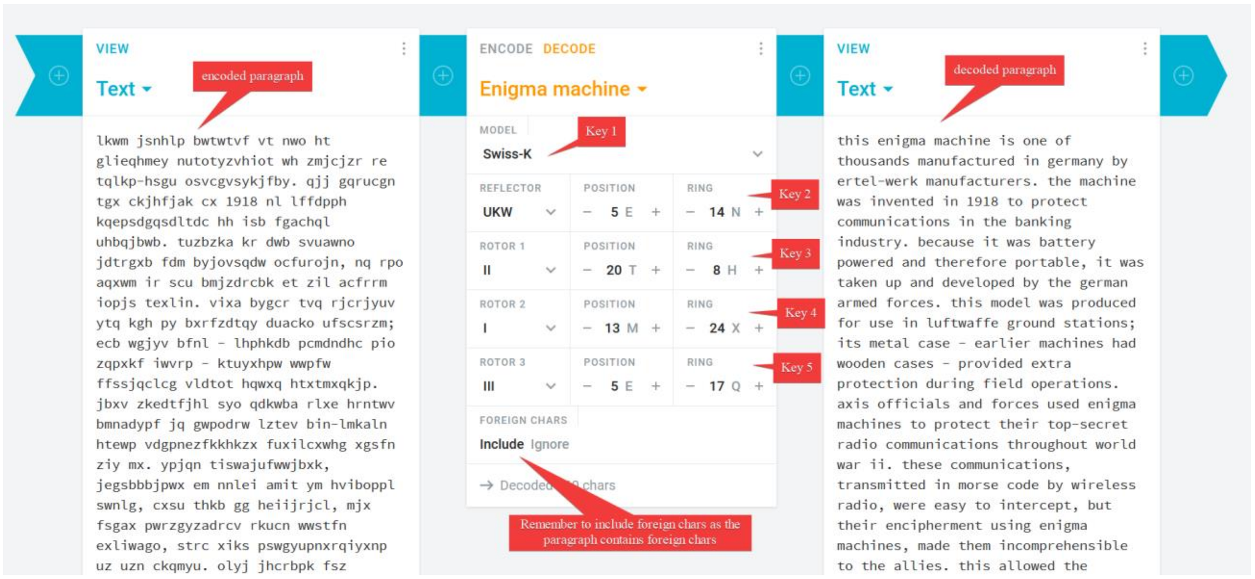
The replace(text) function then replace the space with "_":

Horel_ol!lWd

Knowing how the code works, it can be seen that it's a railfence cipher, which is hinted by the challenge description. To decode railfence cipher, you can simply look up cryptii.com, and set the key as 3 to decode the 5 keys provided by the challenge:

Key1: Sswski	-> Swissk
Key2: U54K_E1NW_	-> UKW_5E_14N
Key3: I0HI2T8__	-> II_20T_8H
Key4: IMX_3_412	-> I_13M_24X
Key5: I57I_E1QI_	-> III_5E_17Q

The 5 keys are decoded, but what are they? Hmm…let's get back to the hint at the challenge description. The railway fences prevented the train that carries our bombe from being derailed. Googling bombe shows it was an electro-mechanical device used to decipher Enigma-machine-encrypted messages. This is a hint for enigma machine cipher. There is an 'encodedsecret' file that contains an encrypted paragraph, use these keys to decode the enigma cipher at cryptii.com, the keys are the rotors' settings of the Enigma Machine used to encode the paragraph:

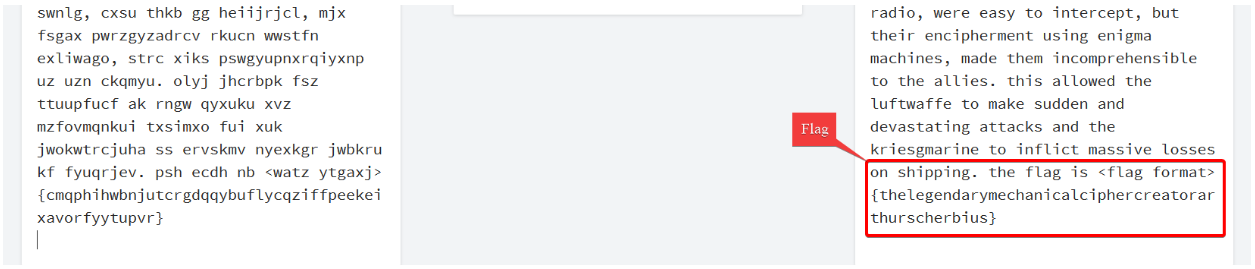


```
lkwm jsnhlp bwtwtvf vt nwo ht
glieqhmey nutotyzvhiot wh zmjcjzr re
tqlkp-hsgu osvcgvsykjfby. qjj gqrucgn
tgx ckjhfjak cx 1918 nl lffdpph
kqepsdgqsdltdc hh isb fgachql
uhbqjbwb. tuzbzka kr dwb svuawno
jdtrgxb fdm byjovsqdw ocfurojn, nq rpo
aqxwm ir scu bmjzdrcbk et zil acfrrm
iopjs texlin. vixa bygcr tvq rjcrjyuv
ytq kgh py bxrfzdtqy duacko ufscsrzm;
ecb wgjyv bfnl - lhphkdb pcmdndhc pio
zqpxkf iwvrp - ktuyxhpw wwpfw
ffssjqclcg vldtot hqwxq htxtmxqkjp.
jbxv zkedtfjhl syo qdkwba rlxe hrntwv
bmnadypf jq gwpodrw lztev bin-lmkaln
htewp vdgpnezfkkhkzx fuxilcxwhg xgsfn
ziy mx. ypjqn tiswajufwwjbxk,
jegsbbbjpwx em nnlei amit ym hviboppl
swnlg, cxsu thkb gg heiijrjcl, mjx
fsgax pwrzgyzadrcv rkucn wwstfn
exliwago, strc xiks pswgyupnxrqiyxnp
uz uzn ckqmyu. olyj jhcrbpk fsz
```

MODEL		
Swiss-K		
REFLECTOR	POSITION	RING
UKW	– 5 E +	– 14 N +
ROTOR 1	POSITION	RING
II	– 20 T +	– 8 H +
ROTOR 2	POSITION	RING
I	– 13 M +	– 24 X +
ROTOR 3	POSITION	RING
III	– 5 E +	– 17 Q +
FOREIGN CHARS		
Include Ignore		

```
this enigma machine is one of
thousands manufactured in germany by
ertel-werk manufacturers. the machine
was invented in 1918 to protect
communications in the banking
industry. because it was battery
powered and therefore portable, it was
taken up and developed by the german
armed forces. this model was produced
for use in luftwaffe ground stations;
its metal case - earlier machines had
wooden cases - provided extra
protection during field operations.
axis officials and forces used enigma
machines to protect their top-secret
radio communications throughout world
war ii. these communications,
transmitted in morse code by wireless
radio, were easy to intercept, but
their encipherment using enigma
machines, made them incomprehensible
to the allies. this allowed the
```

Read the interesting paragraph til the end and you will find what you needed:



```
swnlg, cxsu thkb gg heiijrjcl, mjx
fsgax pwrzgyzadrcv rkucn wwstfn
exliwago, strc xiks pswgyupnxrqiyxnp
uz uzn ckqmyu. olyj jhcrbpk fsz
ttuupfucf ak rngw qyxuku xvz
mzfovmqnkui txsimxo fui xuk
jwokwtrcjuha ss ervskmv nyexkgr jwbkru
kf fyuqrjev. psh ecdh nb <watz ytgaxj>
{cmqphihwbnjutcrgdqqybuflycqziffpeekei
xavorfyytupvr}
```

```
radio, were easy to intercept, but
their encipherment using enigma
machines, made them incomprehensible
to the allies. this allowed the
luftwaffe to make sudden and
devastating attacks and the
kriesgmarine to inflict massive losses
on shipping. the flag is <flag format>
{thelegendarymechanicalciphercreatorar
thurscherbius}
```

Flag: BOH21{thelegendarymechanicalciphercreatorarthurscherbius}

Thank you for reading…or playing my challenges.

This marks my first experience in creating CTF challenges.

I hope you learnt something or had fun. All the best to