

Capture the Flag

50.042 Foundations of Cybersecurity 2019

Reverse Engineering

Abinaya 1002941 | Glenn 1003118 | Kenneth 1002691 | Sumedha 1002876

Introduction	2
1. SMTB	2
2. Sartech	5
3. Caesar Salad	7
4. Pinky Pinky Unicorn	10
5. GFCS	14
6. CIA	18
Insights & Reflection	21

Introduction

We have compiled our solutions to the 6 challenges that we successfully completed in this report. We have also attached the screenshots included in this report, the codes required and other files in the respective challenge folders.

1. SMTB

<http://flask-env.dy2nh6w2mr.ap-southeast-1.elasticbeanstalk.com>

Flag:

```
CTF{thisIsTheFinalFlagThatYouAndYourGroupmatesHaveWorkedHardForToGetForTheLastCoupleOfDays:p}
```

Step 1: SH Attered

From the header, we know that SHA encryption was used.

Given: b7a875fc1ea228b9061041b7cec4bd3c52ab3ce3

Go to : <https://hashkiller.co.uk/Cracker/SHA1>

Found: letmein

Steb

Firstly, we went to www.factordb.com and using n, we found p and q. We are now able to derive the private key using mod inverse method. The decrypted RSA text is 's3cr3t*c0d3'.

RSA is weak

[illegible]

Step 3.1: AES Key Derivation

Next, we had to use `pyaes` and `pdkdf2` to derive our AES encryption key, which is `b'c531048c47c13a7092d6ad8b36c0cc86bcef6fd4b8063acb7831c4fd2b523d68'`.

AES-y Key

Using the pyaes and pbkdf2 libraries and the password from previous page, and the salt below, get the AES encryption key!

passwordSalt = b'\x7f\x8a\x91\xab\xcc\x0c\x06\x08\x0d\x0f\x07\x0a\x02\x08\x01\x01'

AES Encryption Key (you will need it for later)

Okay, next please!

Step 3.2: Read the bytes

Once solved, we selected the 4 strings that were the real ciphertext.

Is is encrypted ???

4 of these strings are real ciphertext, can you tell us which ones?

- ☐ xcvhgbrnjyhtgrbytrthgrbntbvb
- ☐ cdfvrebtrrb5465bttb34gv4b5b4
- ☐ @5*%@&^&^/
- ☒ cee979b86e2c59f6caab3c6055fbfa
- ☒ c9ee7aa5693542f0d7b63a615af6
- ☐ 23453465vrev514rfergr3rtg5g
- ☐ vf34v43rv6786by
- ☒ ccf27fb3733944e0
- ☒ c9f371b069
- ☐ b76n98766v5vvt4r5g54
- ☐ f45b6n78ygf435b67897867m7bytr
- ☐ dfgfcvrtbytrghttrtrtrty
- ☐ 3443576575554757
- ☐ revtryrurturbryr
- ☐ FERERVYBTURTURRURTRI
- ☐ FERVTEURT34G0Y568

Are these correct?

Once you have selected correctly, you will be redirected to the next page to decrypt the 4 ciphertexts. The plaintexts are - 'transpositional', 'substitutional', 'vigenere' and 'shift' respectively.

AES Decryption

Using the IV given and AES encryption key derived in part II, decrypt the 4 strings you singled out from the previous page!

(in the order of appearance from the previous page)

IV = 57116448576878005380785937564945681393249968307171981972
903895716101015138040

first plaintext

second plaintext

third plaintext

fourth plaintext

Next

Step 4: AES Decryption (CTR Block Mode)

Based on the way 'CIPHER' is being arranged, we can derive that transposition cipher was used. We guessed that 'transpositional' was the key used. We used an online tool. It is cap sensitive and the decrypted RSA text is 'This is the flag that you have finally attained after using transpositional cipher and RSA and AES and password solver'.

C I
P H
E R

The flag is just around the corner!

"IYTNANDTLRTRSEEDSENRLIFOAPHLUISWSOTGLDSGAIEDSTANAPSVANTSNASH
IRIELHTAINARIUATCAOHVENHAETAYSOAOFFAPRD"

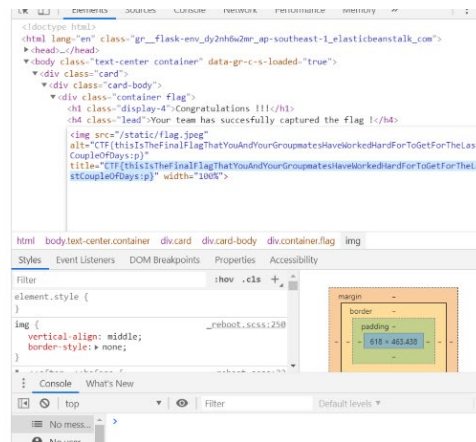
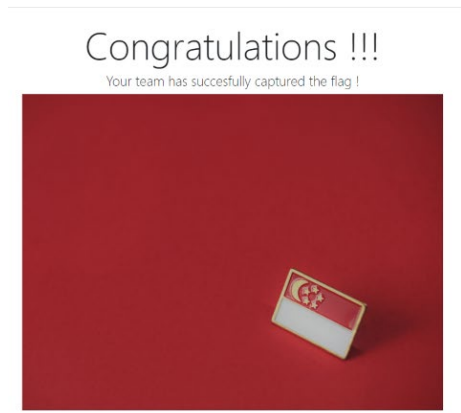
decrypted RSA text

Give me the flag please!

The screenshot shows the 'Columnar Transposition Cipher Tool' interface on the Boxentriq website. The tool has a dark header with the site name and navigation links. The main area contains a text input field with the encrypted text: "IYTNANDTLRTRSEEDSENRLIFOAPHLUISWSOTGLDSGAIEDSTANAPSVANTSNASHIRIELHTAINARIUATCAOHVENHAETAYSOAOFFAPRD". Below the input field are buttons for 'Copy', 'Paste', and 'Text Options'. A search bar shows 'transpositional' and a language dropdown is set to 'English'. There are buttons for 'Decode', 'Encode', 'Auto Solve (without key)', 'Instructions', and 'Show grid'. Under 'Auto Solve Options', there are input fields for 'Min Key Length' (2), 'Max Key Length' (9), 'Max Results' (20), and a 'Spacing Mode' dropdown set to 'Automatic'. The 'Results' section shows a 'Decoded message' box containing the decrypted text: "THIS IS THE FLAG THAT YOU HAVE FINALLY ATTAINED AFTER USING TRANSPOSITIONAL CIPHER AND RSA AND AES AND PASSWORD SOLVER". At the bottom of the results box are 'Copy' and 'Text Options' buttons.

<https://www.boxentriq.com/code-breaking/columnar-transposition-cipher>

Finally, the flag is within the image and we do an inspect to retrieve it. The flag is CTF {this Is The Final Flag That You And Your Groupmates Have Worked Hard For To Get For The Last Couple Of Days :p }.



2. Sartech

Flag: CTF {indicator}

Step 1: Finding Keys

After the last round of hints, Sartech group revealed the key in the poem is KILL & DICT and that the ciphertext is PHIB3FB8DKGEE.

Title: PHIBFB8DKGEE

Death is not the end we seek
It is the beginning, for I
Cannot help but kneel
To the colossal hades' will.

Step 2: Substitution Cipher with KILL

In my first attempt using their special substitution table I derived the wrong text because I was encrypting and not working backwards as shown below in the picture.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9
a	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9
b	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9	
c	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9		
d	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9			
e	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9				
f	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9					
g	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9						
h	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9							
i	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9								
j	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9									
k	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9										
l	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9											
m	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9												
n	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9													
o	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9														
p	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9															
q	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																
r	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																	
s	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																		
t	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																			
u	u	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																				
v	v	w	x	y	z	0	1	2	3	4	5	6	7	8	9																					
w	w	x	y	z	0	1	2	3	4	5	6	7	8	9																						
x	x	y	z	0	1	2	3	4	5	6	7	8	9																							
y	y	z	0	1	2	3	4	5	6	7	8	9																								
z	z	0	1	2	3	4	5	6	7	8	9																									
0	0	1	2	3	4	5	6	7	8	9																										
1	1	2	3	4	5	6	7	8	9																											
2	2	3	4	5	6	7	8	9																												
3	3	4	5	6	7	8	9																													
4	4	5	6	7	8	9																														
5	5	6	7	8	9																															
6	6	7	8	9																																
7	7	8	9																																	
8	8	9																																		
9	9																																			

On my second attempt I derived:

PHIBFB8DKGEE

KILLKILLKILL

F97053x2a833

Step 3: Transpositional Cipher with DICT

Realising that I would have to decrypt here too, I separated **F97053x2a833** into 4 columns.

C D I T

F 0 x 8

9 5 2 3

7 3 a 3

Then I joined the characters starting from DICT (given the clue 0x is the first two and alphabetically), **Key: 0XF852933A73**.

Step 4: Decrypting Images

We were given 7 images that were encrypted with ECB. After getting the hex key, I used the given ecb.py code to decrypted each image.

```
nonchalantcocoa$ python3.7 ecb.py -i random-1.png -o rd1.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-2.png -o rd2.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-3.png -o rd3.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-4.png -o rd4.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-5.png -o rd5.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-6.png -o rd6.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$ python3.7 ecb.py -i random-7.png -o rd7.jpeg -k 0XF852933A73 -m d
nonchalantcocoa$
```

The results were 7 images of hashes.

```
c337b66e9655a710d5a292bbe8365946
5578f65081824f19e2f253e59c91671f
2ed074586ca648092321eb284101350b
ae7bc1462aec26b6eb44514af9041c76
9b76f2b398575b43d292063423931bb2
ee052027e2fded28c0e6acb12e2409cd
7100de6fd95ea13a2b32872371deeda7
```

Step 5: Cracking Hash

Using the online hash cracker hashkiller.co.uk/Cracker/MD5, we found the message.

```
c337b66e9655a710d5a292bbe8365946 MD5 2a535 or osCommerce 2a:535
5578f65081824f19e2f253e59c91671f MD5 15dic or osCommerce 15:dic
2ed074586ca648092321eb284101350b MD5 49i37 or osCommerce 49:i37
ae7bc1462aec26b6eb44514af9041c76 MD5 524n9 or osCommerce 52:4n9
9b76f2b398575b43d292063423931bb2 MD5 o4258
ee052027e2fded28c0e6acb12e2409cd MD5 ry560
7100de6fd95ea13a2b32872371deeda7 MD5 363t9 or osCommerce 36:3t9
```

As their hint indicated the flag only contained alphabets, we found the word dictionary in the above: adicinoryt -> dictionary. However, the flag CTF {dictionary} didn't work. Hence we tried to form other words using the jumbled letters and got the word indicatory, which was successful!

3. Caesar Salad

Flag: CTF{SUTD_lib_2100h}

Step 1

We had to figure out with pubKey encrypts which ciphertext. pubKey1 -> ct2 and vice versa.

Step 2

We used an online tool to find p and q for pubKey1 and pubKey2.

For pubKey1:

```
131175012912718250806592304873426282086400000000001
```

Result:

number

[1311750129...01](#)<51> = [293049347](#) · [447620901583917369406703727865417431168683](#)<42>

For pubKey2:

```
124357060441566807002668863235325077648677273600000000001
```

Result:

```
> = 52750358975649354449407<23> · 23574637757246841069594107658796543<3
```

Step 3

Use b64decode when reading the ciphertxts

This is the output derived:

```
Finding private key 1 (t):
t: 71550006799144008666913601163666847756899423153803
ct1: 113264822024947893022000910723816575818390378749238
dec_ct1: bytearray(b'\\UR%$##[p')

Finding private key 2 (t):
t: 97322916867313153306434656690777503657225869233641807519
ct2: 426067942678669650384389884991364363010626560442555009417
dec_ct2: bytearray(b'6G9nFHG7R_')
```

Step 4

We used an online tool to do an ascii shift cipher for each dec_ct(x).

For dec_ct1:

The output is iib_2100h}

The screenshot shows an online ASCII Shift Decoder tool. The input text is '\\UR%\$##[p' and the output is 'iib_2100h}'. The tool also shows a list of results on the left and a sidebar with links to other cipher tools.

Results
+113 kkda432j
+112 l1eb5433k
+111 mmfc6544l
+110 nngd7655m
+109 oohe8766n
+108 ppif9877o
+98 zzspCBAAy
+43 iib_2100h}
+115 jje`3211i~
+114 jje`3211i~
+107 qqjg:988p
+44 00)&yxww/D
+42 22+({zyy1F
+106 rrrkh;:99q
+99 yyroBA@x
+97 {{tqDCBBz

ASCII Shift Decoder

★ ASCII SHIFTED CIPHERTEXT

\\UR%\$##[p

TRY ALL POSSIBLE SHIFTS (FROM 1 TO 127)

USE A SHIFT OF 64

★ RESULTS FORMAT

- ASCII CHARACTERS (PRINTABLE ONLY)
- HEXADECIMAL 00-7F-FF
- DECIMAL 0-128-256
- OCTAL 000-177-377
- BINARY 00000000-11111111

DECRYPT

See also: ASCII Code – ROT-47 Cipher

ASCII Shift Encoder

For dec_ct2:

The output is CTF{SUTD_I

The screenshot shows an online ASCII Shift Decoder tool. On the left, a list of shifted characters is displayed, with 'CTF{SUTD_]' circled in red. The main interface has '6G9nFHG7R_' entered in the input field. The 'TRY ALL POSSIBLE SHIFTS' option is selected. The 'RESULTS FORMAT' is set to 'ASCII CHARACTERS (PRINTABLE ONLY)'. The 'DECRYPT' button is visible. On the right, there are links for 'Similar tools' and 'Support'.

Step 6

We combined the 2 outputs to get CTF{SUTD_liib_2100h}, how ever there seems to be a typo so we removed the extra 'i'. The final flag is CTF{SUTD_lib_2100h}.

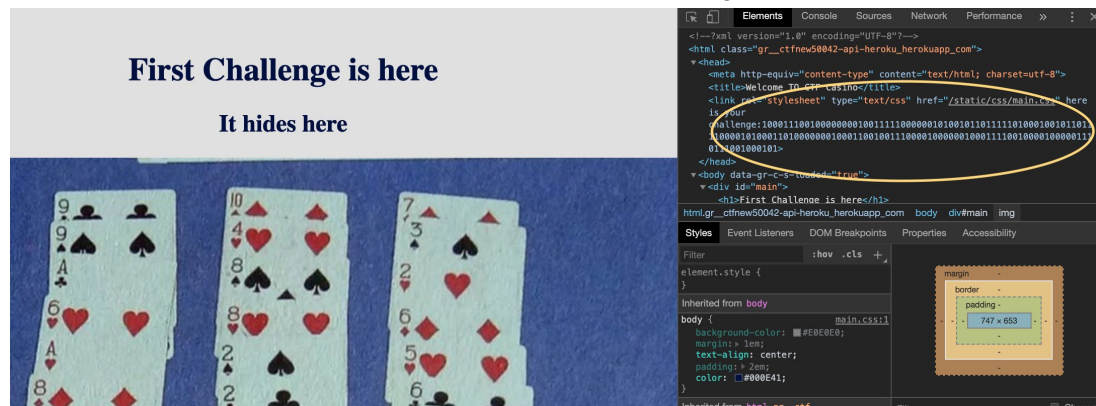
4. Pinky Pinky Unicorn

Flag: CTF{Yas you got it HAHAHA+WellDone+XOR_ECB_FLAG_112715}

Challenge 1: The Card Dealing

Step 1: Finding the cipher

The cipher was hidden in the HTML code.of the webpage provided.



After obtaining the cipher we got the hint that we need to use the word deal as a key to stream cipher with the cipher. However, it was important that we deal the bits like cards in the picture, which is to separate into 3 columns. We had some confusion on separating the cards as there are multiple ways for that. We tried both. (A & B)

A1:	10110011	10000111	10011100	01000011	10000011	10001100	1
A2:	01000001	10011111	01111001	00001000	10100010	00001100	0
A3:	01000001	00100010	00001010	11000100	10001010	01010110	1
B1:	11110001	00100111	00011110	11000001	10001011	00000110	1
B2:	01000001	10011111	01111001	00001000	10100010	00001100	0
B3:	00000101	10000010	10001000	01000110	10000010	11011100	1

Step 2: Stream cipher

Attempt 1: We did not get a coherent word when we decoded both A and B result after XORing with Deal binary using base64.

Deal:	000000000	000000000	00110010	00110010	10110000	10110110	0
AX1:	10110011	10000111	10101110	01110001	00110011	00111010	1
AX2:	01000001	10011111	01001011	00111010	00010010	10111010	0
AX3:	01000001	00100010	00111000	11110110	00111010	11100000	1
BX1:	11110001	00100111	00101100	11110011	00111011	10110000	1
BX2:	01000001	10011111	01001011	00111010	00010010	10111010	0
BX3:	00000101	10000010	10111010	01110100	00110010	01101010	1

Hence we asked for help and checked with other groups who helped us, saying our binary of DEAL was wrong. We then removed the 0 of each 8 bit of character hence deal: 28 bits instead of 32 bits and hence it was 21 zeros padding + 28 deal bits.

```
Deal:0000000 0000000 0000000 1100100 1100101 1100001 1101100
AX1: 1011001 1100001 1110011 0100000 1111001 1101111 1110101
AX2: 0100000 1100111 1101111 1110100 0100000 1101001 1110100
AX3: 0100000 1001000 1000001 1001000 1000001 1001000 1000001
```

In Attempt 2, after we XORed A with the new Deal, we decoded the binary to string using online application and got the below result.

Binary to String☆

Enter the binary text to decode

get sample

Stri

01011001 01100001 01110011 00100000 01111001 01101111 01110101
00100000 01100111 01101111 01101000 00100000 01101001 01110100
00100000 01001000 01000001 01001000 01000001 01001000 01000001

Convert

Load

Browse

The decoded string:

Yas you got it HAHAHA

ANS1: Yas you got it HAHAHA

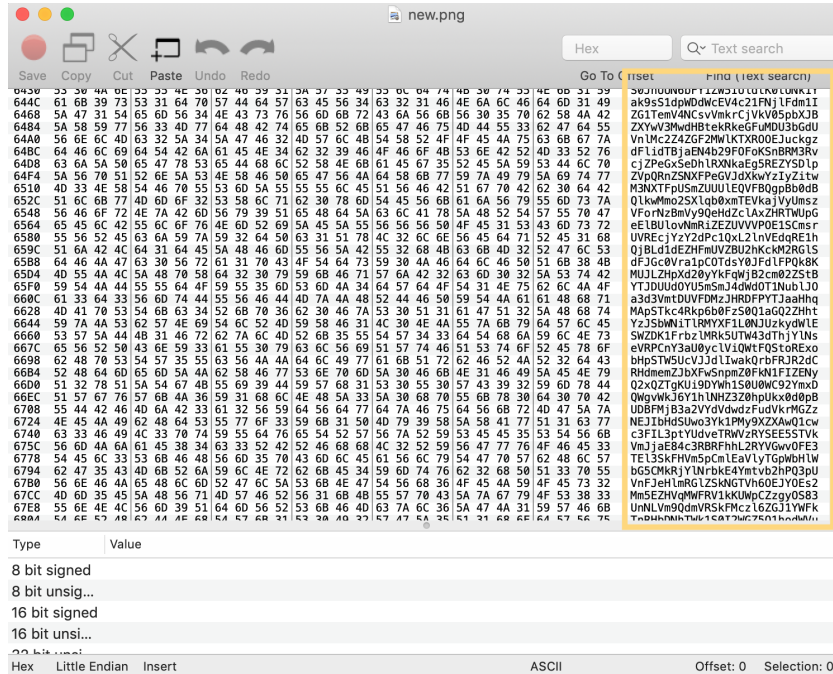
Challenge 2: Image Steganography

Step1: Encrypted.txt

Adding the extension /encrypted.txt to the link provided automatically started a download for encrypted.txt. Now we had to find the key to decrypt this.

Step 2: Finding w hat's in the image

After downloading the image of the ace card from the webpage I opened it using a Hec Editor. At the end we found something that stood out.



Decoding this using an online website with base64 we got:

The Base64 Decode:

```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQCzIV5enHRWm+KT6MXjOIKWiX7VpExsmE69EvmHdmSzex4+/
Vi+
5dWNimrAev0Ws0tpmzDdxan057IgtVylSfxdav1iJMtN8BnrH3tYbu0chCxoE8Z
JpQ3tor6OxIRx8eEsdhH9DFXH9ieZPFvR5qOxeluy0c22f+p3sWLZTJfTQIDAQAB
AoGABY02j6lyjofLlEdj5rRk3TZ+70fW/PxwYrP1dtSYFxiARZ/6dbdFTUuO8MR
k+QTDr63cgcOsT1/igtGjDMaB0KwWDDqfQVASHrC6DiRtRfEsEkkZB97lcBEvQOBO
1BKdzWwm2bAjZ0vrn6e+Aa2CQGNafJbxugNOSnnRNkwwVkcQQC32GD1Oa2
Zhxl0
RNG8FJzoAsKD5hd6dxmc2RmcbNTLaqu/CIS92uiDIfC+Qko9LFNTMn7u8cbSlyTQ
```

Step 3: Decrypting Encrypted.txt

Since it was clear this was a private key for RSA, we used a simple decrypt rsa code similar to Lab and decrypted the encrypted.txt.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

def decrypt(prikey, cipher):
    f = open(prikey, 'r').read()
    key = RSA.importKey(f)
    rsa = PKCS1_OAEP.new(key)
    answer = rsa.decrypt(cipher)
    return answer

#rsakey=RSA.importKey(key)
f = open('encrypted.txt','rb').read()

print(decrypt("key.ppm",f).decode())
```

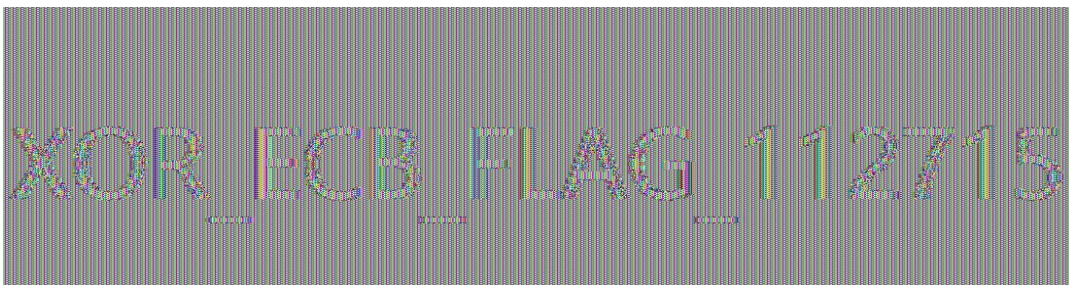
```
Abinayas-MacBook-Pro:PINKY nonchalantcocoa$ python3.7 py.py
WellDone
Abinayas-MacBook-Pro:PINKY nonchalantcocoa$
```

The ANS2 was:

Challenge 3: The Xor Challenge

The hint revealed that it involved some XOR operation. Looking at the encryption code shows that they inserted a random 16 byte nonce at the start, we call this B0. B0 then XORS with B1 to get B1'. B1' then XOR with B2 to get B2 '. This goes on to the last block. Hence to decrypt, we start from the end and XOR the previous block to get the current block. We append all the blocks together and we get the answer. Studying the encryption code also shows that they left the header out from the encryption

ANS3 is



XOR_ECB_FLAG_112715

Joining ANS1+ANS2+ANS3 we obtained the final flag: CTF{Yas you got it
HAHAHA+WellDone+XOR_ECB_FLAG_112715}

5. GFCS

FLAG: CTF{DISCRETIONWILLPROTECTYOUANDUNDERSTANDINGWILLGUARDYOU}

Step 1:

Given n,e and a ciphertext, we did a reverse RSA to get the plaintext 'solomonrocks'.

```
t: 12028958601759060705133213116987337186653895658914539575255990976160977396833  
ct1: 76957365057130496018644065623476122259695319097532361997851028871694948716071  
dec_ct1: bytearray(b'solomonrocks')
```

Step 2:

Reading up on the PDF format will help:

<https://resources.infosecinstitute.com/pdf-file-format-basic-structure/#gref>
<https://amccormack.net/2012-01-22-anatomy-of-a-pdf-document.html>
<http://lotabout.me/orgwiki/pdf.html>

Initially, the pdf was blank. By removing the '%' from /Contents and /Resources, we were able to load the contents of the pdf.

Before:

```
%% Page 1
%% Original object ID: 4 0
4 0 obj
<<
  /Contents 5 0 R
  /MediaBox [
    0
    0
    595
    842
  ]
  /Parent 3 0 R
  %/Resources <<
    /ProcSet [
      /PDF
      /Text
      /ImageB
      /ImageC
      /ImageI
    ]
    /XObject <<
      /XOb4 7 0 R
    >>
  >>
  /Type /Page
>>
endobj
```

After:

```
%% Page 1
%% Original object ID: 4 0
4 0 obj
<<
  /Contents 5 0 R
  /MediaBox [
    0
    0
    595
    842
  ]
  /Parent 3 0 R
  /Resources <<
    /ProcSet [
      /PDF
      /Text
      /ImageB
      /ImageC
      /ImageI
    ]
    /XObject <<
      /XOb4 7 0 R
    >>
  >>
  /Type /Page
>>
endobj
```

Step 3:

This are the contents in the pdf. Given the hint was Bobby is a paleographer, we went to search up on the ancient handwriting. This are not alphabetic representation but numerics. The fonts were derived from Babylonian number system. From here, we derive '53651'.



1	┐	11	◁┐	21	◁◁┐	31	◁◁◁┐	41	◁◁◁◁┐	51	◁◁◁◁◁┐
2	┐┐	12	◁┐┐	22	◁◁┐┐	32	◁◁◁┐┐	42	◁◁◁◁┐┐	52	◁◁◁◁◁┐┐
3	┐┐┐	13	◁┐┐┐	23	◁◁┐┐┐	33	◁◁◁┐┐┐	43	◁◁◁◁┐┐┐	53	◁◁◁◁◁┐┐┐
4	┐┐┐┐	14	◁┐┐┐┐	24	◁◁┐┐┐┐	34	◁◁◁┐┐┐┐	44	◁◁◁◁┐┐┐┐	54	◁◁◁◁◁┐┐┐┐
5	┐┐┐┐┐	15	◁┐┐┐┐┐	25	◁◁┐┐┐┐┐	35	◁◁◁┐┐┐┐┐	45	◁◁◁◁┐┐┐┐┐	55	◁◁◁◁◁┐┐┐┐┐
6	┐┐┐┐┐┐	16	◁┐┐┐┐┐┐	26	◁◁┐┐┐┐┐┐	36	◁◁◁┐┐┐┐┐┐	46	◁◁◁◁┐┐┐┐┐┐	56	◁◁◁◁◁┐┐┐┐┐┐
7	┐┐┐┐┐┐┐	17	◁┐┐┐┐┐┐┐	27	◁◁┐┐┐┐┐┐┐	37	◁◁◁┐┐┐┐┐┐┐	47	◁◁◁◁┐┐┐┐┐┐┐	57	◁◁◁◁◁┐┐┐┐┐┐┐
8	┐┐┐┐┐┐┐┐	18	◁┐┐┐┐┐┐┐┐	28	◁◁┐┐┐┐┐┐┐┐	38	◁◁◁┐┐┐┐┐┐┐┐	48	◁◁◁◁┐┐┐┐┐┐┐┐	58	◁◁◁◁◁┐┐┐┐┐┐┐┐
9	┐┐┐┐┐┐┐┐┐	19	◁┐┐┐┐┐┐┐┐┐	29	◁◁┐┐┐┐┐┐┐┐┐	39	◁◁◁┐┐┐┐┐┐┐┐┐	49	◁◁◁◁┐┐┐┐┐┐┐┐┐	59	◁◁◁◁◁┐┐┐┐┐┐┐┐┐
10	◁	20	◁◁	30	◁◁◁	40	◁◁◁◁	50	◁◁◁◁◁		

Step 4:

Lastly, under document properties, the final ciphertext to solve was the author's name.

Hint was to derive the encryption scheme from week 1 -6. We tried a transposition cipher using an online tool and manage to get the flag. The plaint ext derived had padding, which explains the extra 'Z's at the back. The flag is CTF{DISCRETIONWILLPROTECTYOUANDUNDERSTANDINGWILLGUARDYOU}.

Document Properties

Description	Security	Fonts	Initial View	Custom	Advanced
Description					
File:	nothing_here				
Title:	<input type="text"/>				
Author:	itioydrdiaucoleuntlnldzrnp cadaggyzdewrtnenwuosiltousilrz				
Subject:	<input type="text"/>				

https://www.dcode.fr/transposition_-cipher

↑↓	↑↓
4,5,1,3	DISCRETIONWILLPROTECTYOUANDUNDER STANDINGWILLGUARDYOUZZZ
4,3,6,1,	RINAIDATUTZLGNCLDGEOOTZYNYLORZW EDUNDUURSPONDICWSTILARINARDXXX ZX
3,2,5,6,	INAIIDRTUTZLANCILDGEOOTZGNLYORYWE DUNZUURSPONDICESTILAWINARDRXXXZ

Transposition Decoder

★ TRANSPOSITION CIPHERTEXT

itioydrdiaucoleuntndzrnpdaggyzdewrtnenwuosiltousilrz

★ KEEP SPACES, PUNCTUATION AND OTHER CHARACTERS ☐

1

6. CIA

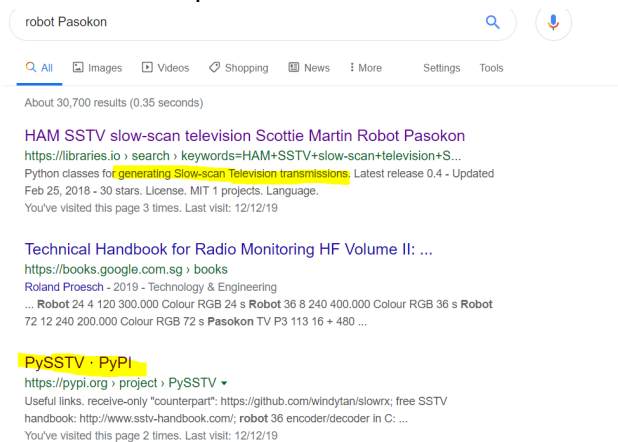
Flag: CTF{THEWORLDHASBEENFOOLEDTHISVIDEOGAMEDOESNOTEXIST}

Step 1: Getting the WAV file

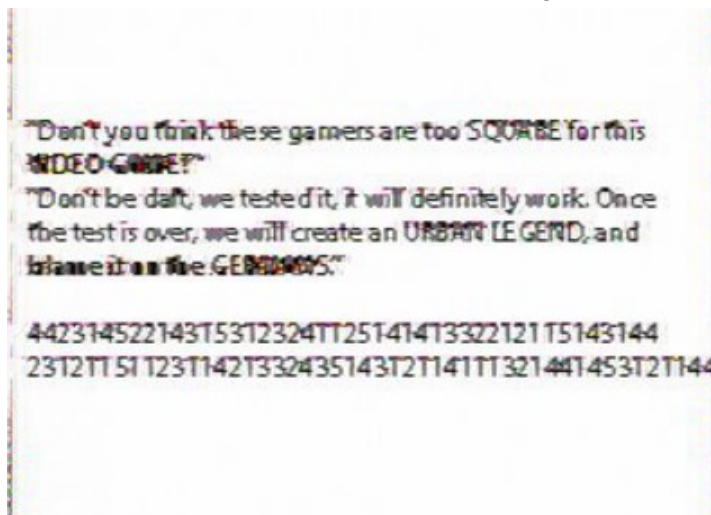
The hint was quite clear to use Lab 1 codes. The also said that the key was their enemy in the hint. From the challenge description, we figured that the key was CIA. Went each letter was represented as a decimal number, we get 391 which is the key for the Caesar Cipher. We then output a wav file

Step 2: Interpreting the WAV file

The hint was to google Robot Pasokon. Based on what we have found, PySSTV was used to convert an image file into WAV file. That inspired us to think of a tool that would be able to decipher the WAV file.



The WAV file given had an awful high frequency which is inaudible. We used a tool called RX-SSTV to help decipher the image from the WAV file.

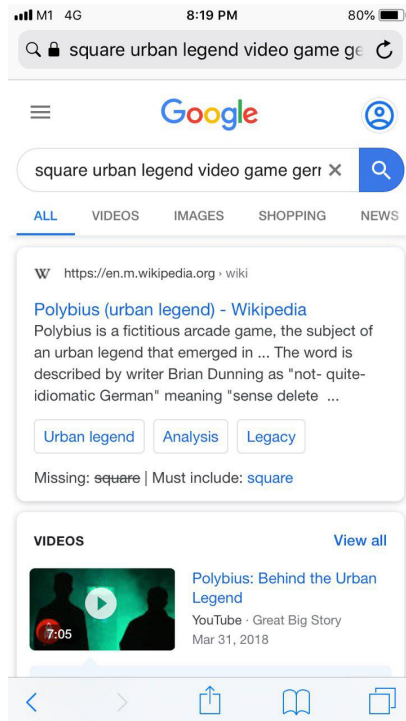


Step 3: Subsequently we got the following

The hint was: There is a relation between the video game and the cipher. the relation is the creator of the game. that relation is the key for the cipher. Please note that for the key AKA the creator, he stands out among the english

The keywords from the previous part was SQUARE, VIDEO GAME, URBAN LEGEND and GERMANS.

We googled all the keywords from STEP 2 and got



Since it was missing square, we searched it and it revealed the Polybius Square Cipher

Googling the German creator revealed that it was Sinneslöschen



- In some versions of this mystery, the players suffered from a series of unpleasant side effects, including amnesia, insomnia, nightmares and night terrors.
What are the other "versions" of this mystery and what are the sources for them. The quoted material is basically directly from the coinop.org source.
 - The supposed creator of Polybius is **Ed Rotberg**, and the company named in most accounts of the game, **Sinneslöschen** (German meaning "deletion/erasure of senses, Sense-delete"), often named as either a secret government organization or a codename for Atari.
Who is the source for tying the game to **Ed Rotberg**? Who is the source tying any of this to Atari? Where are the accounts that do not associate the game with Sinneslöschen?
 - The gameplay is said to be similar to Tempest (a shoot 'em up game using vector graphics), while the game is said to contain subliminal messages which would influence the action of anyone playing it.
Where is the source that says the gameplay was similar to Tempest? Where is the source for "vector graphics"? Where the source for "subliminal messages". The sourced description of the game says it was "kind of abstract, fast action with some puzzle elements".
- RS 1QR 25 2 (talk) 19:51 21 January 2014 (UTC)

Step 4: Final Cipher Breaking

We then use an online tool to break the cipher

Results

THEWORLDHASBEENFOOLEDTHISVIDEOGAMEDOESNOTEXI
ST




GET FOR FREE

Polybius Cipher - dCode

Tag(s) : Substitution Cipher

Share



dCode and you

Polybius Square Decoder

★ POLYBIUS CIPHERTEXT

44	23	14	52	21	43	15	31	23	24	11	25	14	14	13	32	21	21	15	14
31	44	23	12	11	51	12	31	14	21	33	24	35	14	31	21	14	11	13	21
44	14	53	12	11	44														

☐ TRY AN AUTOMATIC DECRYPTION

★ TEXT LANGUAGE English

☐ TRANSFORM IN MONO-ALPHABETIC SUBSTITUTION

☒ I KNOW THE ENCRYPTION GRID / KEY

\	1	2	3	4	5
1	S	I	N	E	L
2	O	C	H	A	B
3	D	F	G	K	M
4	P	Q	R	T	U
5	V	W	X	Y	Z

★ COORDINATES ORDER

☒ LINE, COLUMN (USUAL)

☐ COLUMN, LINE (UNUSUAL)

DECRYPT

Insights & Reflection

In general, solving 8 challenges with little CTF experience was extremely challenging, on top of coping with other modules that have project submissions due on the same week as well. With many information being detained from us, it was difficult to deduce what encryption schemes were used and follow the train of thoughts of the creator.

For our RE challenge, many could easily find the plaintext from the hash value and the image. It was deducing the decompiler that was challenging to most groups and getting the correct timestamp. As creators, we felt otherwise. We used a rainbow table to crack our own hash and unscrambled the image, which we found challenging. However, our hash was easily found through online tools and b64decode respectively, to our dismay. We thought the decompiler was the easiest.

Through the challenges, we got to better appreciate and understand the concepts taught in class and how we can apply it to encrypt and decrypt. It would be great if we had a longer period to solve more challenges and have the CTF on an earlier week instead of week 13. It is too close to finals and many project submissions. This gives us insufficient time for revisions and the weightage for each challenge (0.5%) is too much to ask for the amount of time and effort placed into solving them. For an average student with little CTF experience and have many content being cramped in each week, it takes more than 2hrs per challenge. However, despite the aforementioned, we enjoyed doing the challenges, assuming hints were readily available. The online server website was also very helpful and informative. The scoreboard encouraged us to take on more challenges and very essential in seeking help from groups who completed similar challenges that we were stuck at.

Overall, our team enjoyed the CTF challenge and we are happy that we achieved more than our set goal of 4. We feel that if the timing of the challenge week was better we could have appreciated the experience more and completed more challenges!