# Cryptography

Practicum of Attacking and Defense of Network Security

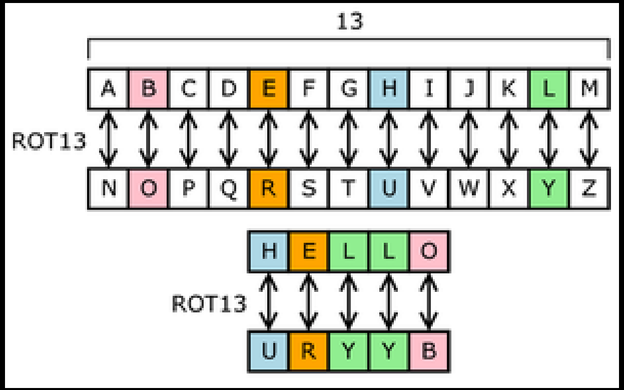
2025.04

# Outline

- Classic cryptography
- Modern cryptography
    - Symmetric-key cryptography
    - Public-key cryptography
- TLS
- Steganography

# Classic cryptography

# Affine Cipher




Substitution Cipher Ref.

- $\gcd(a, n) = 1$
- n is the size of the alphabet
- $EK(m) = (a*m + b) \bmod n$
- $DK(c) = a^{-1}(c - b) \bmod n$

- When a = 1, Affine Cipher is Caesar cipher

# Modular Inverse

- How to compute a^-1 (mod n)?
- Tools available
  - sympy.invert in sympy
  - inverse_mod in Sage

```
import sympy
print sympy.invert(11,26) ##output : 19

from Crypto.Util.number import inverse
print inverse(11,26) ##output : 19

import gmpy2
print gmpy2.invert(11,26) ##output : 19
```

# Exercise - Affine Cipher

- Ciphertext =ifpmluglesecdlqp_rclfrseljpkq
- for each letter of cipher text its position in the alphabet is the position of the original letter multiplied by 4 and shifted by 15
  character shift over alphabet is cyclic, so 'z' shifted by 1is '_' and '_' shifted by 1is 'a' alphabet consists of letters from 'a' to 'z' and symbol '_'
  letter 'a' has position 0, symbol '_' has position 26 ( following 'z' )
  please find the flag

hint : ord('a')        chr(97)
>>97        >> a

# Solution

```
import string

s = string.ascii_lowercase # a-z s += '_'
d = {}
for c in range(len(s)):  d[s[(c*4 +
   15)%27]] = s[c]
ciphertext = 'ifpmluglesecdlqp_rclfrseljpkq' s1 = ''
for x in ciphertext:  s1 += d[x]
print(s1) # flag_is_every_haxor_love_math
```

# Solution with sympy

```
import sympy
original = 'abcdefghijklmnopqrstuvwxyz_'
encrypted = 'ifpmluglesecdlqp_rclfrseljpkq'


def main():
    result = ''
    a = sympy.invert(4,27)
    for e in encrypted:
        result += original[((original.index(e)-15) * a) % 27]
    print(result)


if __name__ == '_main_':
    main()
```

# Rail Fence Cipher

- A kind of transposition ciphers

- Example:
- m = WE ARE DISCOVERED. FLEE AT ONCE
- c = WECRL TEERD SOEEF EAOCA IVDEN

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

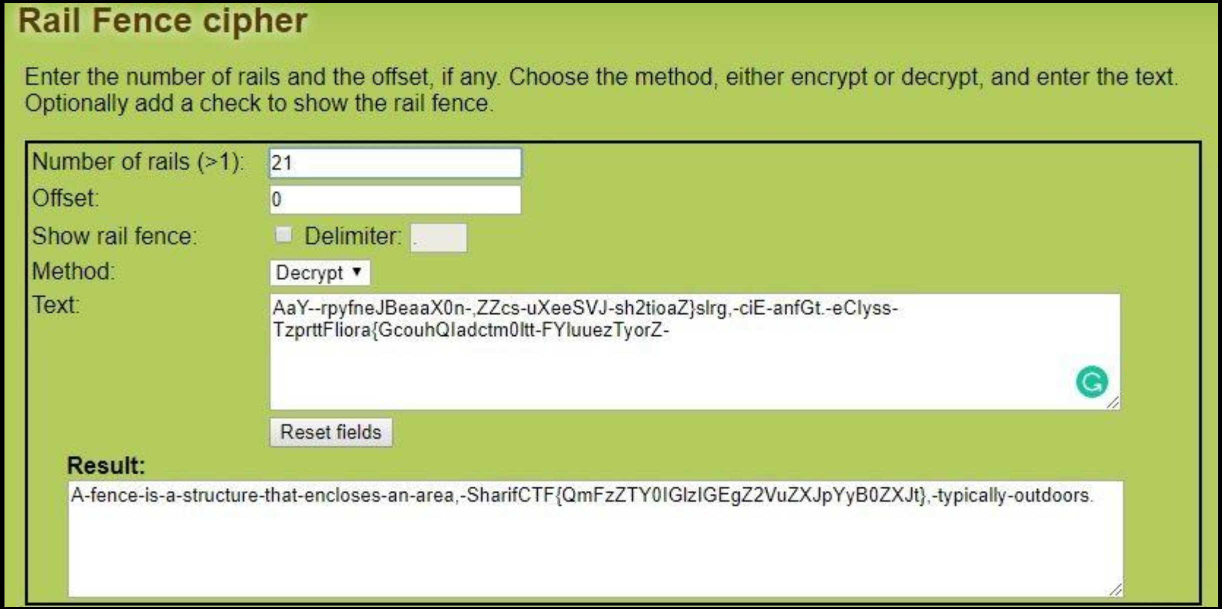
# Exercise - Rail Fence Cipher

Ciphertext :
AaY--rpyfneJBeaaX0n-,ZZcs-uXeeSVJ-sh2tioaZ}slrg,-ciE-anfGt.-eCIyss-TzprttFliora{GcouhQIadctm0ltt-FYluuezTyorZ-

Flag : SharifCTF{flag_is_here}

http://www.geocachingtoolbox.com/index.php?page=railFenceCipher

# Solution



Rail Fence cipher
Enter the number of rails and the offset, if any. Choose the method, either encrypt or decrypt, and enter the text. Optionally add a check to show the rail fence.
Number of rails (>1): 21
Offset: 0
Show rail fence:
Delimiter:
Method: Decrypt
Text: AaY--rpyfneJBeaaX0n-,ZZcs-uXeeSVJ-sh2tioaZ}slrg,-ciE-anfGt.-eClyss-TzprttFliora{GcouhQIadctm0Itt-FYIuuezTyorZ-
Reset fields
Result:
A-fence-is-a-structure-that-encloses-an-area,-SharifCTF{QmFzZTY0IGlzIGEgZ2VuZXJpYyB0ZXJt},-typically-outdoors.

```python
!/usr/bin/env python3

FENCES =1
CIPHERTEXT ="AaY--rpyfneJBeaaX0n-,ZZcs-uXeeSVJ-sh2tioaZ}slrg,-ciE-anfGt.-eCIyss-TzprttFliora{GcouhQIadctm0ltt-FYluuezTyorZ-"

n =len(CIPHERTEXT)

while True:
  FENCES +=1

  Mat =[[] for i in range(FENCES)]
  Msg =[''] * n

  j =0
  d =+1
  for i in range(0, n):
    Mat[j].append(i)
    j +=d
    if j ==0 or j ==FENCES - 1:
      d =-d

  l =0
  for j in range(0, FENCES):
    for i in range(0, len(Mat[j])):
      Msg[Mat[j][i]] =CIPHERTEXT[l]
      l +=1

  m =''.join(Msg)

  if m.find('SharifCTF{') != -1:
    break

print("Number of fences:", FENCES)
print("Plaintext:", m)
```

# Vigenère Cipher

GRASS =Plainttext

CRYPT =Key

EACDZ =Ciphertext

| Key / Plaintext | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| B | Z | 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 |
| C | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
| D | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| E | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| F | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
| G | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| H | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
| I | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
| J | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
| K | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| L | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| M | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| N | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M |
| O | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L |
| P | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K |
| Q | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J |
| R | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I |
| S | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H |
| T | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |
| U | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F |
| V | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E |
| W | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D |
| X | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
| Y | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B |
| Z | 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 | A |

# Tools

- Known key
  - Python - pycipher library
  - online - Vigenère cipher
  - CAP4
- Unknown key
  - Vigenère Cipher Codebreaker
  - Vigenere Solver

```
Vigenere
k: ????????????
p: SECCON{???????????????????????????????????}
c: LMIG}RPEDOEEWKJIQIWKJWMNDTSR}TFVUFWYOCBAJBQ

k=key, p=plain, c=cipher, md5(p)=f528a6ab914c1ecf856a1d93103948fe
```

Hint :

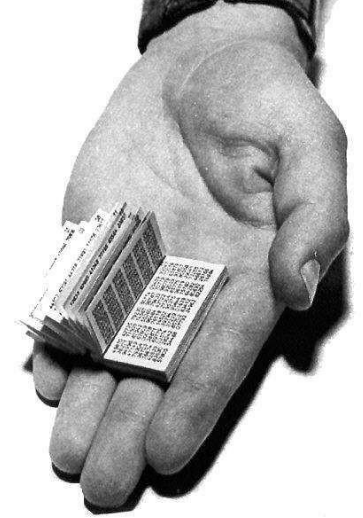
1. len(key)= 12
2. alphabet ='ABCDEFGHIJKLMNOPQRSTUVWXYZ{}'
3. md5(plaintext) =f528a6ab914c1ecf856a1d93103948fe
4. flag will be SECON{.....................}

```
 |ABCDEFGHIJKLMNOPQRSTUVWXYZ{}
-+----------------------------
A|ABCDEFGHIJKLMNOPQRSTUVWXYZ{}
B|BCDEFGHIJKLMNOPQRSTUVWXYZ{}A
C|CDEFGHIJKLMNOPQRSTUVWXYZ{}AB
D|DEFGHIJKLMNOPQRSTUVWXYZ{}ABC
E|EFGHIJKLMNOPQRSTUVWXYZ{}ABCD
F|FGHIJKLMNOPQRSTUVWXYZ{}ABCDE
G|GHIJKLMNOPQRSTUVWXYZ{}ABCDEF
H|HIJKLMNOPQRSTUVWXYZ{}ABCDEFG
I|IJKLMNOPQRSTUVWXYZ{}ABCDEFGH
J|JKLMNOPQRSTUVWXYZ{}ABCDEFGHI
K|KLMNOPQRSTUVWXYZ{}ABCDEFGHIJ
L|LMNOPQRSTUVWXYZ{}ABCDEFGHIJK
M|MNOPQRSTUVWXYZ{}ABCDEFGHIJKL
N|NOPQRSTUVWXYZ{}ABCDEFGHIJKLM
O|OPQRSTUVWXYZ{}ABCDEFGHIJKLMN
P|PQRSTUVWXYZ{}ABCDEFGHIJKLMNO
Q|QRSTUVWXYZ{}ABCDEFGHIJKLMNOP
R|RSTUVWXYZ{}ABCDEFGHIJKLMNOPQ
S|STUVWXYZ{}ABCDEFGHIJKLMNOPQR
T|TUVWXYZ{}ABCDEFGHIJKLMNOPQRS
U|UVWXYZ{}ABCDEFGHIJKLMNOPQRST
V|VWXYZ{}ABCDEFGHIJKLMNOPQRSTU
W|WXYZ{}ABCDEFGHIJKLMNOPQRSTUV
X|XYZ{}ABCDEFGHIJKLMNOPQRSTUVW
Y|YZ{}ABCDEFGHIJKLMNOPQRSTUVWX
Z|Z{}ABCDEFGHIJKLMNOPQRSTUVWXY
{|{}ABCDEFGHIJKLMNOPQRSTUVWXYZ
}|}ABCDEFGHIJKLMNOPQRSTUVWXYZ{
```

# Solution

```
#!/usr/bin/env python3

from re import match
from hashlib import md5 as md5_
from itertools import product
md5 = lambda x: md5_(x.encode()).hexdigest()

TABLE = "ABCDEFGHIJKLMNOPQRSTUVWXYZ{}"
ENCRYPT, DECRYPT = 1, -1

def vigenere(string, key, mode=ENCRYPT, table=TABLE):
    L = len(key)
    key = [ table.index(i) for i in key ]
    string = [ table.index(i) for i in string ]
    cipher = [ (v + mode * key[i % L]) % len(table) for i, v in enumerate(string) ]
    return ''.join(table[i] for i in cipher)

key        = ".......?????"
plaintext  = "SECCON{..................................}"
ciphertext = "LMIG}RPEDOEEWKJIQIWKJWMNDTSR}TFVUFWYOCBAJBQ"
md5_hash   = "f528a6ab914c1ecf856a1d93103948fe"

known_key = ''
for idx, char in enumerate('SECCON{'):
    for k in TABLE:
        if vigenere(char, k) == ciphertext[idx]:
            known_key += k
            break

print('known_key = %s' % known_key)

for poss in product(TABLE, repeat=len(key) - len(known_key)):
    try_key = known_key + ''.join(poss)
    decrypted = vigenere(ciphertext, try_key, DECRYPT)
    if match(r'SECCON\{[A-Z]{35}\}', decrypted) and \
       md5(decrypted) == 'f528a6ab914c1ecf856a1d93103948fe':
        print('key = %s, plaintext = %s' % (try_key, decrypted))
        exit()
```

# One-Time Pad



**Encryption**

| | |
|---|---|
| ASCII "now" in Binary: | 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 |
| One-time Pad: | 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 |
| Ciphertext (XOR'ed) | 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 0 |

Identical

**Decryption**

| | |
|---|---|
| One-time Pad: | 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 |
| Ciphertext | 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 0 |
| ASCII "now" in Binary: | 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 |

# One-Time Pad

- Plain text : This is an example
- Key : MASKLNSFLDFKJPQ
- This is an example → 19 7 8 18 8 18 0 13 4 23 0 12 15 11 4
- MASKLNSFLDFKJPQ → 12 0 18 10 11 13 18 5 11 3 5 10 9 15 16
- PLUS : 31 7 26 28 19 31 18 18 15 26 5 22 24 26 20
- Mod 26 : 5 7 0 2 19 5 18 18 15 0 5 22 24 0 20
- Ciphertext : FHACTFSSPAFWYAU

https://en.wikipedia.org/wiki/One-time_pad

# Modern cryptography

# Recommended key length

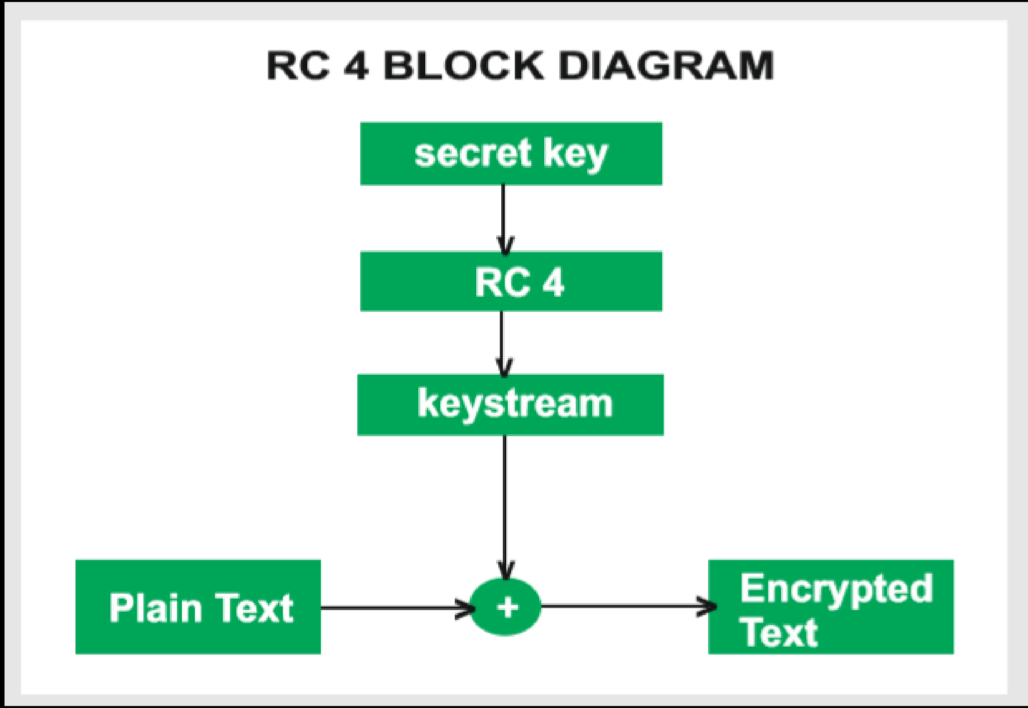
| Date | Security Strength | Symmetric Algorithms | Factoring Modulus | Discrete Logarithm Key | Discrete Logarithm Group | Elliptic Curve | Hash (A) | Hash (B) |
|---|---|---|---|---|---|---|---|---|
| Legacy (1) | 80 | 2TDEA | 1024 | 160 | 1024 | 160 | SHA-1 (2) | |
| 2019 - 2030 | 112 | (3TDEA) (3)<br>AES-128 | 2048 | 224 | 2048 | 224 | SHA-224<br>SHA-512/224<br>SHA3-224 | |
| 2019 - 2030 & beyond | 128 | AES-128 | 3072 | 256 | 3072 | 256 | SHA-256<br>SHA-512/256<br>SHA3-256 | SHA-1<br>KMAC128 |
| 2019 - 2030 & beyond | 192 | AES-192 | 7680 | 384 | 7680 | 384 | SHA-384<br>SHA3-384 | SHA-224<br>SHA-512/224<br>SHA3-224 |
| 2019 - 2030 & beyond | 256 | AES-256 | 15360 | 512 | 15360 | 512 | SHA-512<br>SHA3-512 | SHA-256<br>SHA-512/256<br>SHA-384<br>SHA-512<br>SHA3-256<br>SHA3-384<br>SHA3-512<br>KMAC256 |

http://www.keylength.com/

# RC4

- RC4 is a stream cipher that is roughly divided into two parts by byte encryption
- Including initialization algorithm (KSA) and pseudo-random sub-cipher generation algorithm (PRGA)
  - 1. Generate S box according to Key
  - 2. Generate pseudo-random key stream based on S box
  - 3. xor bitwise encrypted plaintext
- Only XOR operation and S box so the encryption and decryption process is reversible

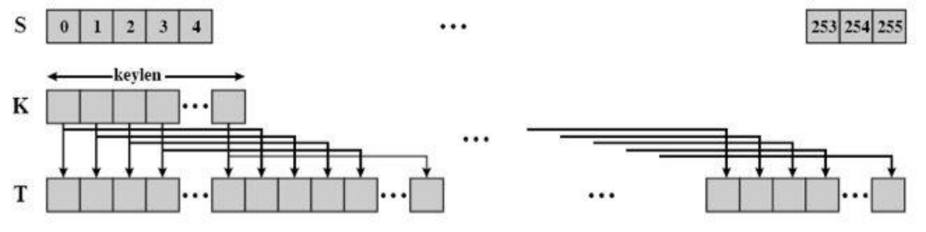
# RC4


RC 4 BLOCK DIAGRAM
secret key
RC 4
keystream
Plain Text
+
Encrypted Text

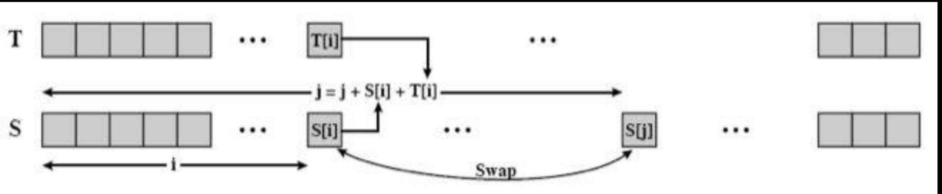
Ref.

# Key-scheduling algorithm (KSA)

- The secret key participates in the generation of the S box
- Initialize the S box with a length of 256. The first for loop loads 0 to 255 non-repetitive elements into the S box and generates a temporary array T. The second for loop scrambles the S box according to the key
- i make sure that every element of Sbox is processed
- j make sure that the scramble of Sbox is random




(a) Initial state of S and T




(b) Initial permutation of S

# Key-scheduling algorithm (KSA)

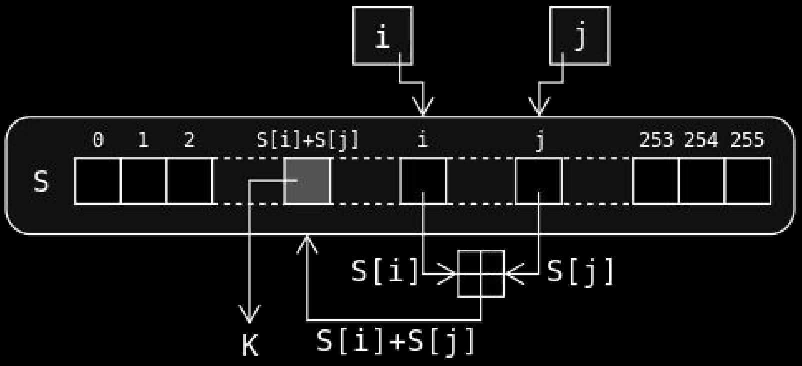
```
def RC4_init(self,k):
    Sbox=[]*255
    T=[]*256
    for i in range(256):
        Sbox.append(i)    #initialize sbox
        T.append(ord(k[i%len(k)]))    #make 256 bits key
    j=0
    for i in range(256):
        j = (j + Sbox[i] + T[i]) % 256
        Sbox[i],Sbox[j]=Sbox[j],Sbox[i]    # swap Sbox
    return Sbox
```

# Pseudo-random generation algorithm (PRGA)

- After the array S is initialized, the input key is no longer used
- Operate by byte Locate an element in the S box through a certain algorithm and XOR with the input byte to get the ciphertext
- The S box is also changed in the loop

# Pseudo-random generation algorithm (PRGA)

```
def RC4_crypt(self,m):
    c=''
    i=j=0
    S=self.Sbox
    for n in range(len(m)):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i],S[j]=S[j],S[i]    #swap Sbox
        t = (S[i] + S[j]) % 256    #generate random index
        c+='%02x'%(ord(m[n])^S[t])  #plain text Xor random index
    return c
```



# LAB

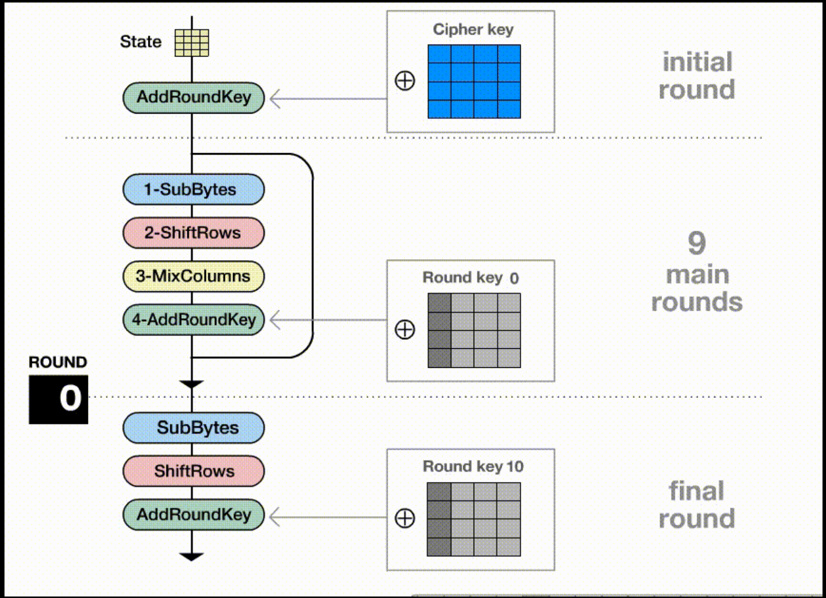
- dbgprint: Xor +RC4
- Biforst: modified RC4
- malware link(password:infected)
- Tool : findcrypt

# Openssl

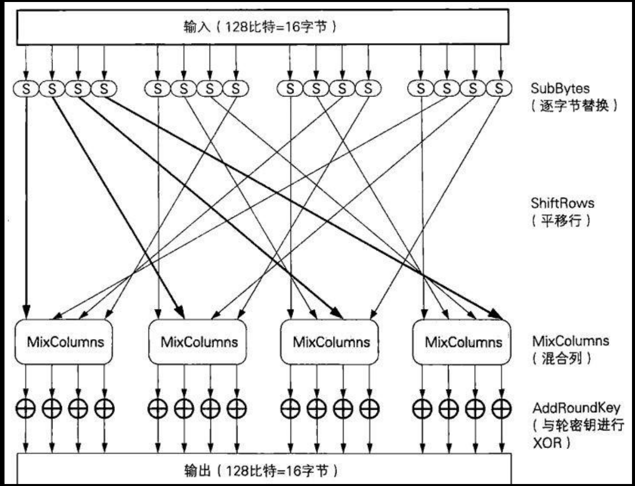
- openssl des-ecb –e –in xxx.txt –out yyy.out –k password (*DES encrypt*)
- openssl des-ecb –d –in yyy.out –out xxx.txt –k password (*DES decrypt*)
- openssl des-ede3 –d –in yyy.out –out xxx.txt –k password (*TDES encrypt*)
- openssl aes-128-ecb –d –in yyy.out –out xxx.txt –k password (*AES decrypt*)

# AES

- Symmetric block cipher
- AES128 =128-bit key, 10 rounds
- AES192 =192-bit key, 12 rounds
- AES256 =256-bit key, 14 rounds



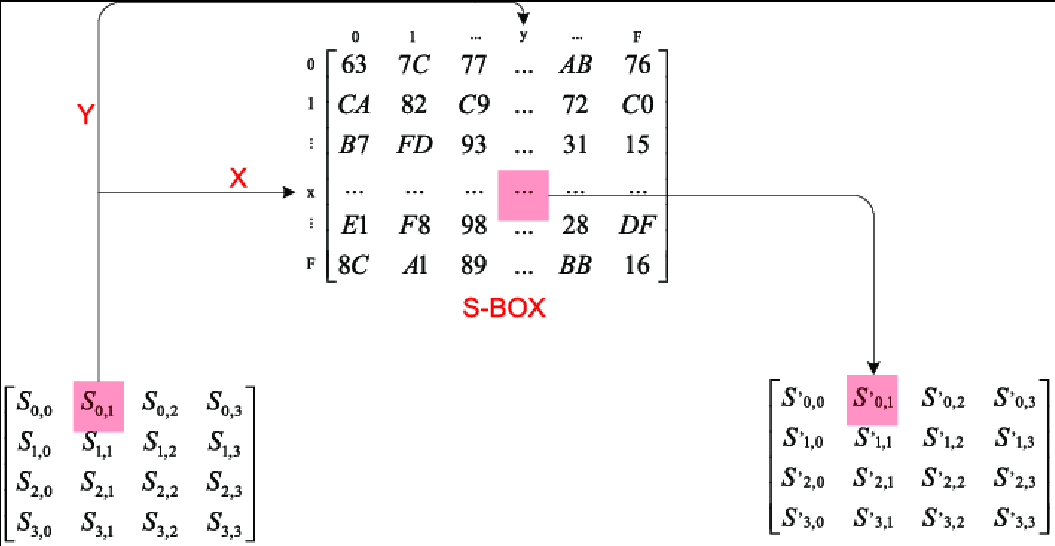
# AES : Steps to Encryption




- Subbytes :
  - Arbitrary Substitution (output similar to Scytale cipher)
- Shiftrows :
  - Rotational substitution(similar to ROT13)
- Mixcolumns :
  - Permutation (similar to the Column Shift example)
- Addroundkey :
  - An XOR function : Resaults in a new key for each round
- Repeat up to 14 time total

# AES : Steps to Encryption

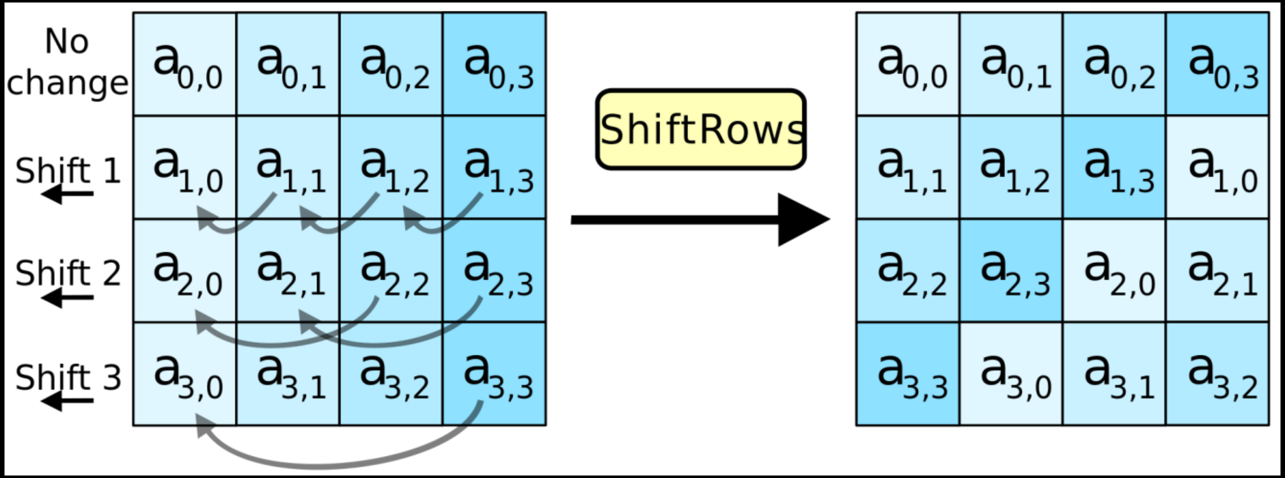
- Subbytes :
  - Arbitrary Substitution (output similar to Scytale cipher)




Ref.

# AES : Steps to Encryption

- Shiftrows :
  - Rotational substitution(similar to ROT13)




Ref.

# AES : Steps to Encryption

- Mixcolumns :
  - Permutation (similar to the Column Shift example)

(a) State (Input) × MixColumns Matrix = State (Output)

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \text{x} \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

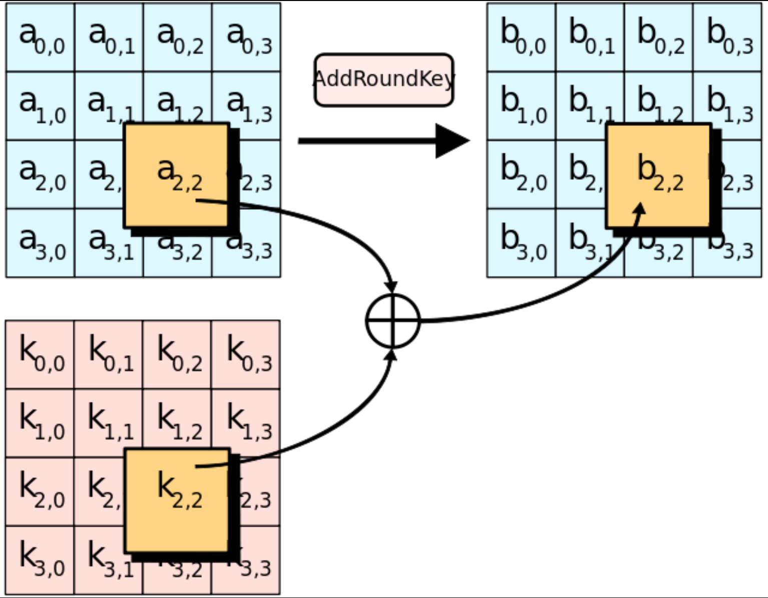
(b) State (Input) × InvMixColumns Matrix = State (Output)

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \text{x} \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Ref.

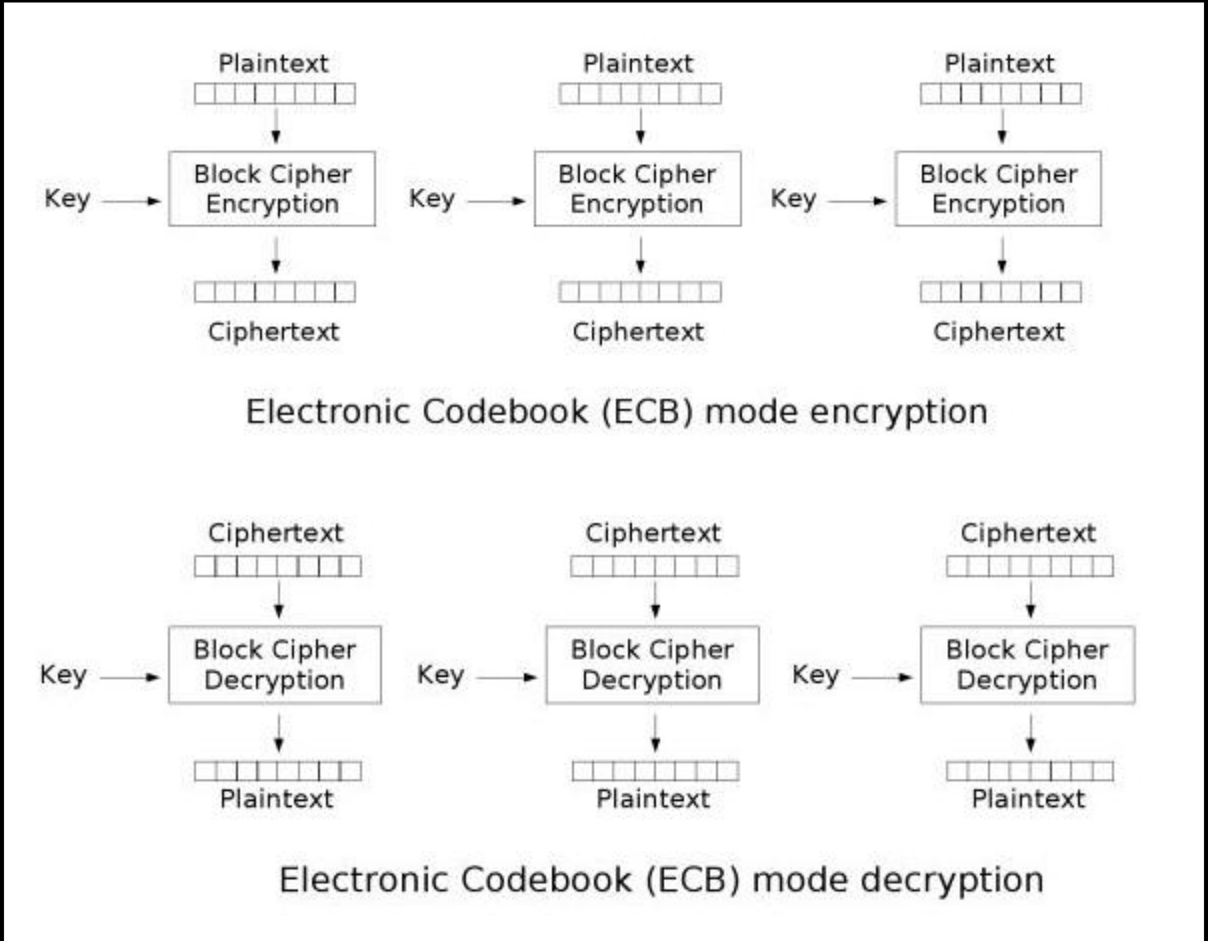
# AES : Steps to Encryption

- Addroundkey :
  - An XOR function : Resaults in a new key for each round



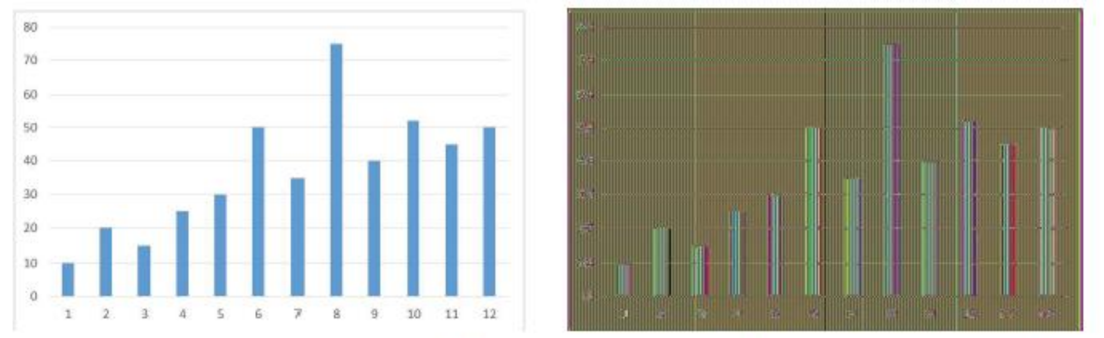

Ref.

# ECB


Plaintext
Plaintext
Plaintext
Key
Block Cipher Encryption
Key
Block Cipher Encryption
Key
Block Cipher Encryption
Ciphertext
Ciphertext
Ciphertext
Electronic Codebook (ECB) mode encryption
Ciphertext
Ciphertext
Ciphertext
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Plaintext
Plaintext
Plaintext
Electronic Codebook (ECB) mode decryption

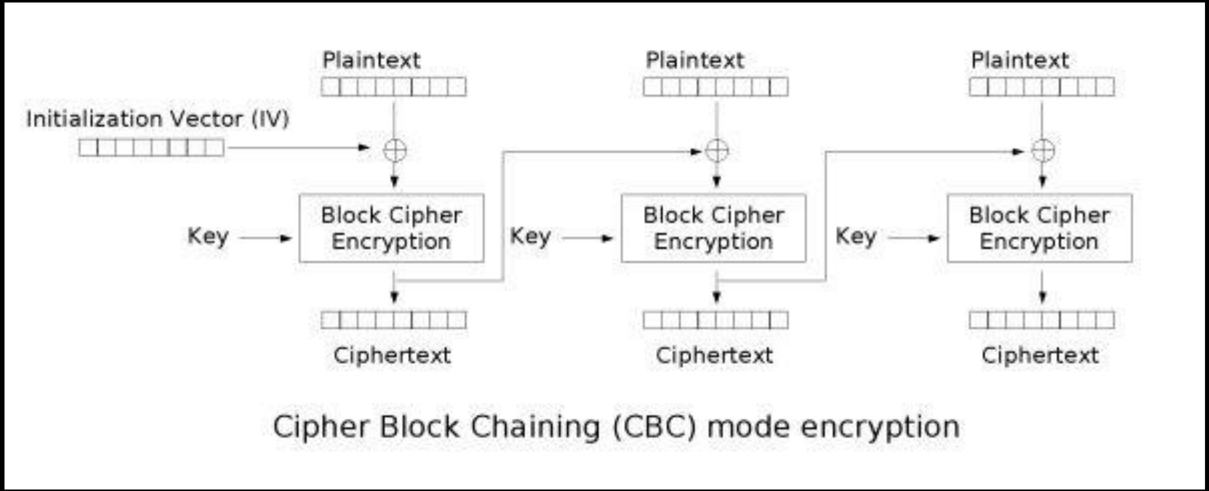
# ECB

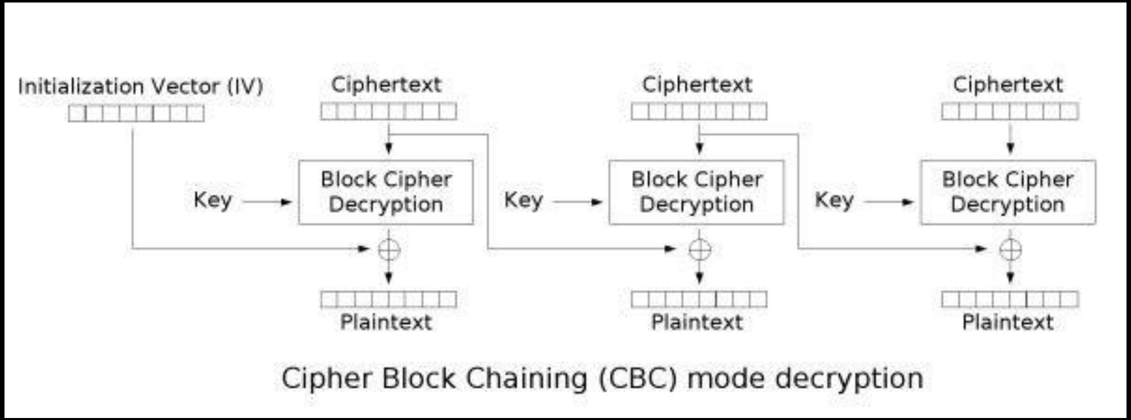
- 較大的圖案，如果以 block 作單位，單獨經過 128 bit block 加密後，仍然可以可以看出原圖的樣貌。




(a) The original image (pic_original.bmp)　(b) The encrypted image (pic_encrypted.bmp

# CBC


Plaintext
Plaintext
Plaintext
Initialization Vector (IV)
Key
Block Cipher Encryption
Key
Block Cipher Encryption
Key
Block Cipher Encryption
Ciphertext
Ciphertext
Ciphertext
Cipher Block Chaining (CBC) mode encryption


Initialization Vector (IV)
Ciphertext
Ciphertext
Ciphertext
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Key
Block Cipher Decryption
Plaintext
Plaintext
Plaintext
Cipher Block Chaining (CBC) mode decryption

# Exercise - AES

Encrypted with AES in ECB mode. All values base64 encoded

ciphertext =
I300ryGVTXJVT803Sdt/KcOGlyPStZkeIHKapRjzwWf9+p7fIWkBnCWu/IWls+5S

key =iyq1bFDkirtGqiFz7OVi4A==

# Solution

- First decode the key from base64 to hex.
  - Key in base64 : iyq1bFDkirtGqiFz7OVi4A==
  - Key in Hex : 8B2AB56C50E48ABB46AA2173ECE562E0
- Then copy the ciphertext to a file and run the following openssl command.
  - openssl enc -d -aes-128-ecb -in aesecbcipher.txt -out aesecbplain.txt -nopad -nosalt -K 8B2AB56C50E48ABB46AA2173ECE562E0 -iv 0 -base64
- cat the aesecbplain.txt file.
  - cat aesecbplain.txt
  - flag{do_not_let_machines_win_2d4975bc}__________

# Solution

```
import base64
from Crypto.Cipher import AES

key =base64.b64decode("6v3TyEgjUcQRnWuIhjdTBA==")
ciphertext =
base64.b64decode("rW4q3swEuIOEy8RTIp/DCMdNPtdYopSRXKSLYnX9NQe8z+LMsZ6Mx/x8pwGwofdZ")
crypter =AES.new(key, AES.MODE_ECB)
plaintext =crypter.decrypt(ciphertext).decode("utf-8")

print(plaintext)
```

# Exercise - AES

We encrypted a flag with AES-ECB encryption using a secret key, and got the hash:

`e220eb994c8fc16388dbd60a969d4953f042fc0bce25dbef573cf522636a1ba3fafa1a7c21ff824a5824c5dc4a376e75`

However, we lost our plaintext flag and also lost our key and we can't seem to decrypt the hash back :(.

Luckily we encrypted a bunch of other flags with the same key. Can you recover the lost flag using this?

```
e220eb994c8fc16388dbd60a969d4953f042fc0bce25dbef573cf522636a1ba3fafa1a7c21ff824a5824c5dc4a376e75e220e
b994c8fc16388dbd60a969d4953f042fc0bce25dbef573cf522636a1ba3fafa1a7c21ff824a5824c5dc4a376e75

block1: e220eb994c8fc16388dbd60a969d4953
block2: f042fc0bce25dbef573cf522636a1ba3
block3: fafa1a7c21ff824a5824c5dc4a376e75


abctf{looks_like_gospel_feebly}:
e220eb994c8fc16388dbd60a969d4953 <-- // abctf{looks_like
6d896bd7d6da9c4ce3eac5e4832c2f64     //_gospel_feebly}

abctf{verism_evg_you_can_break_ajugas}:
528c30c67c57968fa131684d07c1fa9c     // abctf{verism_evg
f042fc0bce25dbef573cf522636a1ba3 <-- // _you_can_break_a
c0bd6ceeec8e817f1be7b09a9a8b0fb8     // jugas}

abctf{amidin_ogees}:
5f0ec66748ad4e9c512616572dd9197b     // abctf{amidin_oge
fafa1a7c21ff824a5824c5dc4a376e75 <-- // es}
```

so now we just put the three plaintext blocks together to obtain the key:
abctf{looks_like_you_can_break_aes}

```
block1: e220eb994c8fc16388dbd60a969d4953 - abctf{looks_like
block2: f042fc0bce25dbef573cf522636a1ba3 - _you_can_break_a
block3: fafa1a7c21ff824a5824c5dc4a376e75 - es}
```

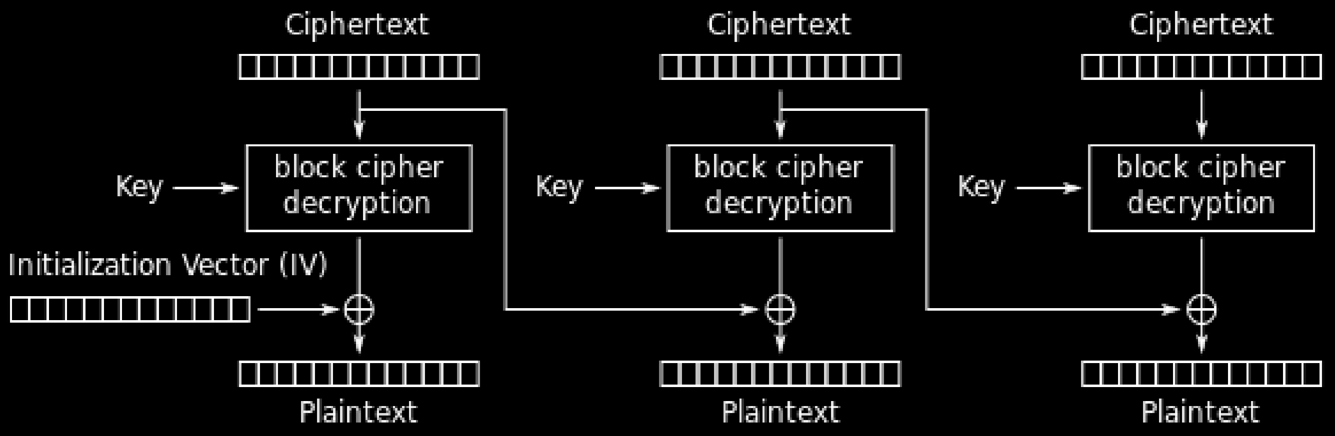
# Padding Oracle Attack

- Decrypt any given ciphertext without knowing the key and IV.
- Padding Oracle Attack attacks generally need to meet the following conditions
  - Encryption algorithm using PKCS5 Padding. Of course, the way OAEP is filled in asymmetric encryption may also be affected.
  - The grouping mode is CBC mode.
- Attacker ability
  - An attacker can intercept messages encrypted by the above encryption algorithm.
  - The attacker can interact with the padding oracle (the server): the client sends the ciphertext to the server, and the server will use some kind of return information to inform the client whether the padding is normal.

# Padding

| | BLOCK #1 | | | | | | | | BLOCK #2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Ex 1 | F | I | G | | | | | | | | | | | | | |
| Ex 1 (Padded) | F | I | G | 0x05 | 0x05 | 0x05 | 0x05 | 0x05 | | | | | | | | |
| Ex 2 | B | A | N | A | N | A | | | | | | | | | | |
| Ex 2 (Padded) | B | A | N | A | N | A | 0x02 | 0x02 | | | | | | | | |
| Ex 3 | A | V | O | C | A | D | O | | | | | | | | | |
| Ex 3 (Padded) | A | V | O | C | A | D | O | 0x01 | | | | | | | | |
| Ex 4 | P | L | A | N | T | A | I | N | | | | | | | | |
| Ex 4 (Padded) | P | L | A | N | T | A | I | N | 0x08 | 0x08 | 0x08 | 0x08 | 0x08 | 0x08 | 0x08 | 0x08 |
| Ex 5 | P | A | S | S | I | O | N | F | R | U | I | T | | | | |
| Ex 5 (Padded) | P | A | S | S | I | O | N | F | R | U | I | T | 0x04 | 0x04 | 0x04 | 0x04 |

# CBC Decryption




Cipher Block Chaining (CBC) mode decryption

```
0x39 0x73 0x23 0x22 0x07 0x6a 0x26 0x3d   (middle)

0x6d 0x36 0x70 0x76 0x03 0x6e 0x22 0x39   (correct IV)

T    E    S    T    0x04 0x04 0x04 0x04   (Plaintext)


0x39 0x73 0x23 0x22 0x07 0x6a 0x26 0x3d   (middle)

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00   (wrong IV)

0x39 0x73 0x23 0x22 0x07 0x6a 0x26  0x3d   (Plaintext)


0x39 0x73 0x23 0x22 0x07 0x6a 0x26 0x3d   (middle)

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x3c   (guess IV)

0x39 0x73 0x23 0x22 0x07 0x6a 0x26         (Plaintext)
```

Middle[8] ^ orginal iv[8] = plain[8]
Middle[8] ^ guess iv[8] = 0x01

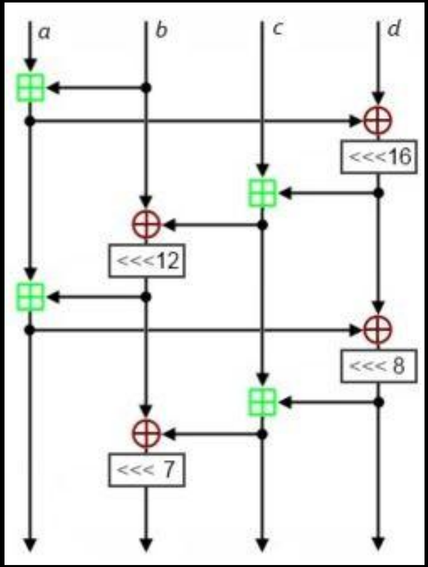
middle[8] = 0x01 ^ guess iv[8]
plain[8]= 0x01 ^ 0x3c ^ 0x39=0x04

https://skysec.top/2017/12/13/padding-oracle%E5%92%8Ccbc%E7%BF%BB%E8%BD%AC%E6%94%BB%E5%87%BB/

# DEMO

https://web.engr.oregonstate.edu/~rosulekm/crypto/padding.html

# Chacha20




- modified from salsa
- stream cipher
- 256-bit key
- 64-bit nonce
- 64-bit block counter
- Outputs a 64-byte block of key stream and increments block counter in each invocation
- Plaintext is XOR'ed with the key stream

https://zh.wikipedia.org/wiki/Salsa20
https://www.ietf.org/proceedings/88/slides/slides-88-tls-1.pdf

# Exercise - chacha20

- Betaflash let's go in Cuba and dance amigo !!
- Flag format: CTF{sha256}
- {"nonce": "dcfu+qXOX30=",
"ciphertext":
"nT0/C209haz3XQs6JcvrEhkbRXnzZiyR87vI82VDvfaQh9eajLNIz
kG51TnZg81g7IEPd3UJElZz8xhCMlVb/cXHJO9h",
"key": "Fidel_Alejandro_Castro_Ruz_Cuba!"}

# Solution

```
data = {"nonce": "dcfu+qXOX30=", "ciphertext":
"nT0/C209haz3XQs6JcvrEhkbRXnzZiyR87vI82VDvfaQh9eajLNIzkG51TnZg81g7IEPd3UJElZz8xhCMlVb/cXHJO9h",
           "key": "Fidel_Alejandro_Castro_Ruz_Cuba!"}
nonce = base64.b64decode(data['nonce'])
ct = base64.b64decode(data['ciphertext'])
key = data['key']
cipher = ChaCha20.new(key=key, nonce=nonce)
plaintext = cipher.decrypt(ct)
print(plaintext)
```

# RSA Algorithm : Components

- Trapdoor Secrets
  - $n = p \cdot q$ where p,q are large primes
- Public Key
  - $(n, e)$
  - e is relative prime to $\phi(n) = (p-1)(q-1)$
  - $1 < e < \phi(n)$
- Private Key
  - $(n, d)$
  - $d = e^{-1} \bmod \phi(n)$ is the multiplicative inverse of e
- Encryption / Digital Signature Verification
  - $c = m^e \bmod n$ where m is plaintext and c is ciphertext
- Decryption / Digital Signature Production
  - $m = c^d \bmod n$

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

# RSA Algorithm : Rationale

Why it is secure ?

Knowing p, q and e,one can easily compute : $n = p \cdot q$

$\Phi(n) = (p-1)(q-1)$

$d = e^{-1} \bmod \phi(n)$

Without knowing p and q, but **knowing n and e**,one cannot compute : $\phi(n)$ or d

# RSAtool

- install
    - git clone https://github.com/ius/rsatool.git
    - cd rsatool
    - python rsatool.py -h
- create the private key
    - python rsatool.py -f PEM -o private.pem -p 1234567 -q 7654321

# Openssl

- openssl genrsa -out rsa_privatekey.pem -passout pass:password -des3 1024

  (*generate RSA private key*)
- openssl rsa -in rsa_privatekey.pem -passin pass:password -pubout -out rsa_publickey.pem

  (*generate RSA piblic key*)
- openssl rsautl -encrypt -pubin -inkey rsa_publickey.pem -in xxx.txt -out yyy.txt

(*use public key to encrpt*)

- openssl rsautl -decrypt -inkey rsa_privatekey.pem -in yyy.txt -out xxx.txt

(*use private key to decrypt*)

# python libary

- primefac
- gmpy
- gmpy2
- pycrypto

# Exercise - RSA

- RSA encryption/decryption is based on a formula that anyone can find and use, as long as they know the values to plug in. Given the encrypted number 150815, d =1941, and N =435979, what is the decrypted number?
- Hint : decrypted =(encrypted) ^d mod N

# Solution

In [1]: (150815 ** 1941)%435979

Out[1]: 133337

# Exercise - RSA

N:
3741592354701721309889381965208805269479525216209323620503086632435957883085839921208813593652589497238199117581980132026446664892479873140251696709262732133672370201885877427160173143201913506667625410392382419849344731886566106159184746739633319924087500474512532051584364528143545642830036966669459509085491971754045805331321421113569313243306318436024125402954828419757838847668012665523371291054078690207302260415387505356286197177088380292863667614709860563352301711487340275368205445432518010932308091862229408067182216388458165217386018430837461033749741205755194187976428780122341637095182039465998369598 11

e: 3

ciphertext (c):
220531641393113403104644076762054198480109121635122278918096713058566904355486632590577086715037761182074675981524777851689940322904706670039678785238851138989304327971328099823574644032248343130535890157869293537843907779677706032141 0661

# Solution

When encrypting with low encryption exponents (e.g., $e = 3$) and small values of the m, (i.e., $m < n^{1/e}$) the result of $m^e$ is strictly less than the modulus n. In this case, ciphertexts can be easily decrypted by taking the e^th root of the ciphertext over the integers.

$$c \equiv m^e \mod n$$
$$m < \sqrt[e]{n}$$
$$m = \sqrt[e]{c}$$

RSA attack method : https://www.anquanke.com/post/id/84632

# Solution

```
import gmpy2,binascii
e=3
c=2205316413931134031046440767620541984801091216351222789180593875373829950
860542792110364325728088504479780803714561464250589795961097670884274813261
4961128825808920204872610581181576195861568155315614552152903612743349771377
261636930849125
m=gmpy2.iroot(c,e)[0]
print (binascii.unhexlify(hex(m).strip('lL')[2:]))
```

# Exercise - RSA

c:
607508702447641453265913870171643760321711341292492729986378754
8188620337158625

n:
16995743251555690273217604748218275023627813110906708466533535
245776011465591891

e: 65537

http://factordb.com/

# Solution

p =166402962209062256362900394698423820317

q =102136061918194068640310910627905419563823

```
from Crypto.Util.number import inverse
p = 166402962209062256362900394698423820317
q = 102136061918194068640310910627905419563823
c = 6075087024476414532659138701716437603217113412924927299863787548188620337158625
n = 16995743251555690273217604748218275023627813110906708466533535245776011465591891
e = 65537

phi = (q-1)*(p-1)
d = inverse(e, phi)
print (hex(pow(c,d,n))[2:-1]).decode('hex')
```

# Exercise - RSA (take home)

- Exercise

# Exercise - Decrypt_RSA (take home)

The following two files are given for you.

1. flag.txt
2. public-key.pem

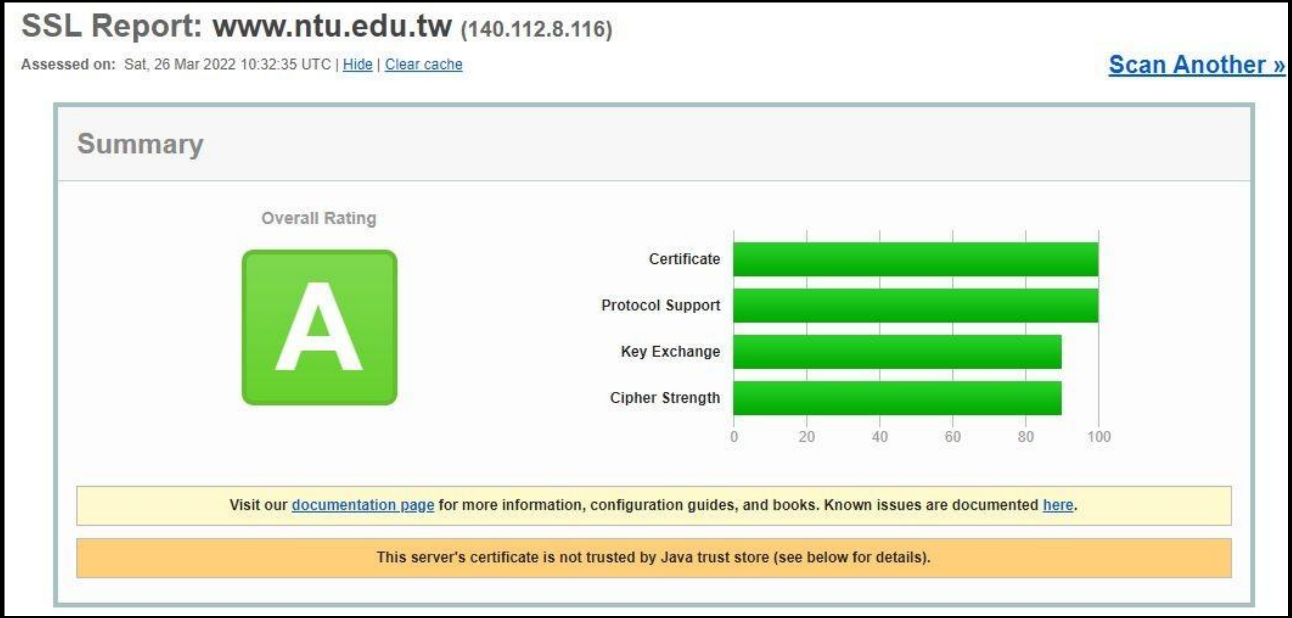
Enjoy decrypting !!!

File Link : link

# SSL / TLS overview

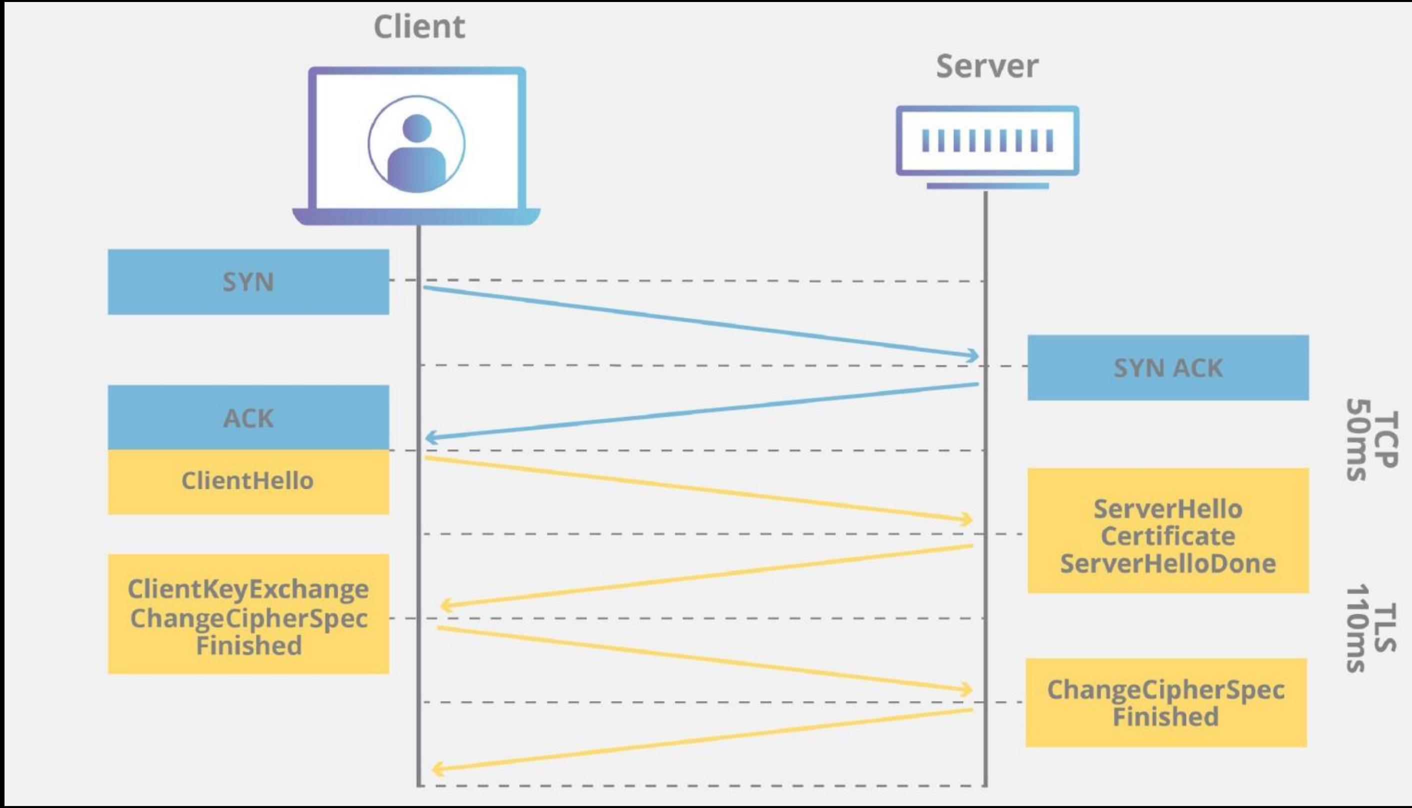
- Goal: building a secure end-to-end channel
  - SSL =Secure Sockets Layer (predecessor)
  - TLS =Transport Layer Security (standard)
  - HTTPS =HTTP over SSL/TLS
- Security requirements
  - Confidentiality
  - Integrity
  - Authentication (mostly server authentication, client authentication in TLS is rare)

# SSL Server Test




https://www.ssllabs.com/ssltest

# TLS Handshake






https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

# TLS Handshake: Client Hello & Server Hello

- Client announces the highest supported protocol version and supported cryptographic algorithms in decreasing order of preference
- Server responses with strongest protocol version and algorithm supported by both client and server
- The exchange is in plaintext

# Cipher Suite

- Cipher suite =key exchange, cipher spec
- Key exchange methods
  - RSA, encrypt key with receiver's public key
  - Fixed Diffie-Hellman, public key certificate contains public DH key
  - Ephemeral Diffie-Hellman, public key is used to sign temporary DH key
  - Anonymous Diffie-Hellman, DH without authentication
- Cipher spec
  - Cipher Algorithm (RC4, RC2, DES, 3DES, DES40, IDEA, AES)
  - MAC Algorithm (HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384)
  - Hash size (0 or 16 for MD5, 20 for SHA-1)

## Cipher Suites

### # TLS 1.2 (suites in server-preferred order)

| | | | | |
|---|---|---|---|---|
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) | ECDH secp256r1 (eq. 3072 bits RSA) | FS | | 256 |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) | ECDH secp256r1 (eq. 3072 bits RSA) | FS | | 128 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) | ECDH secp256r1 (eq. 3072 bits RSA) | FS | WEAK | 256 |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) | ECDH secp256r1 (eq. 3072 bits RSA) | FS | WEAK | 128 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) | ECDH secp256r1 (eq. 3072 bits RSA) | FS | WEAK | 256 |
| TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d) | | | WEAK | 256 |
| TLS_RSA_WITH_AES_256_CBC_SHA (0x35) | | | WEAK | 256 |
| TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa) | | | WEAK | 112 |

https://www.ssllabs.com/ssltest/analyze.html?d=www.ntu.edu.tw

# Password Attacks

- Dictionary Files
  - kali linux
    - /usr/share/wordlists/
- Key-space Brute Force
  - crunch

```
root@kali:~# crunch 6 6 0123456789ABCDEF -o crunch1.txt
Crunch will now generate the following amount of data: 117440512 bytes
112 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 16777216
100%
root@kali:~# wc -l crunch1.txt
16777216 crunch1.txt
```

# John-the-ripper

- Wordlist
    - `john -wordfile:<dict> <crackfile>`
- Single Crack
    - `john -single <crackfile>`
- Incremental Mode
    - `john -incremental:Alpha <crackfile>`

https://countuponsecurity.files.wordpress.com/2016/09/jtr-cheat-sheet.pdf

# Hashcat

- Wordlist
    - `hashcat -m 1800 <shadow_file> <dict>`
- Brute Force
    - `hashcat -m 1800 -a 3 <shadow_file> ?l?l?l?l --force`

https://hashcat.net/wiki/doku.php?id=hashcat
https://tinyapps.org/docs/hashcat.html

# Steganography

1st 5 bits = color, contrast, etc.

Last 3 bits = Insignificant subtleties the human eye cannot discern

JPEG / PNG / etc.

Byte: 10101|010

1 byte = 1 pixel

Replace these bits…

File Hidden

The right picture has a 60kb text file hidden in it with a tool called steghide.

# Exercise - Stego

- Exercise1
- Exercise2
- Exercise3
- Exercise4(take home) hint : png filter
- Exercise5 (take home) hint: png file, change palette
- Exercise6(take home) hint : LSB and brute force

binwalk example.jpg
dd if=example.jpg bs=1 skip=1972141 of=foo.zip

Foremost –v X.png

# fcrackzip

- Example7

```
fcrackzip -b -c 'aA1!' -u -l 1-6 00000192.zip
```

PASSWORD FOUND!!!!: pw == RH4

# Stego - Online tools

- https://aperisolve.fr/
- https://tools.miku.ac/

# Solution

- Exercise1 : supa_secret_flagzor
- Exercise2 : ABCTF{PNG_S0_C00l}
- Exercise3 : 6307834008eb8edbe18c7a20ee4a909d
- Exercise4 : pctf{keep_doge_alive_2014}
- Exercise5 : DrgnS{WhenYouGazeIntoThePNGThePNGAlsoGazezIntoYou}
- Exercise6 : LSB_is_ubiquitous

# Malware Decryption LAB

- dbgprint: Xor +RC4
- Biforst: modified RC4
- yty:Xor+Base64
- Loadinfo:AES+Base64
- malware link(pass:infected)
- Tool : findcrypt

# Crypto - Useful Tools

- SageMath - Mathematics software system
- Pycrypto - The Python Cryptography Toolkit
- Sympy - Python library for symbolic mathematics
- Yafu - Automated integer factorization
- FeatherDuster - An automated, modular cryptanalysis tool
- Hash Extender - A utility tool for performing hash length extension attacks
- PkCrack - A tool for Breaking PkZip-encryption
- RSACTFTool - A tool for recovering RSA private key with various attack
- RSATool - Generate private key with knowledge of p and q
- XORTool - A tool to analyze multi-byte xor cipher

# Steganography - Useful Tools

- Convert - Convert images b/w formats and apply filters
- Exif - Shows EXIF information in JPEG files
- Exiftool - Read and write meta information in files
- Exiv2 - Image metadata manipulation tool
- binwalk
- foremost
- fcrackzip
- ImageMagick - Tool for manipulating images
- Outguess - Universal steganographic tool
- Pngtools - For various analysis related to PNGs
- Steganabara - Tool for stegano analysis written in Java
- Stegbreak - Launches brute-force dictionary attacks on JPG image
- stegextract - Detect hidden files and text in images
- Steghide - Hide data in various kind of images
- Stegsolve - Apply various steganography techniques to images
- Zsteg - PNG/BMP analysis

# Resouces

- 可汗学院公开课
- 深入浅出密码学——常用加密技术原理与应用
- https://cryptopals.com/
- 加密與解密