HW 1 - Cybersecurity

PicoCTF

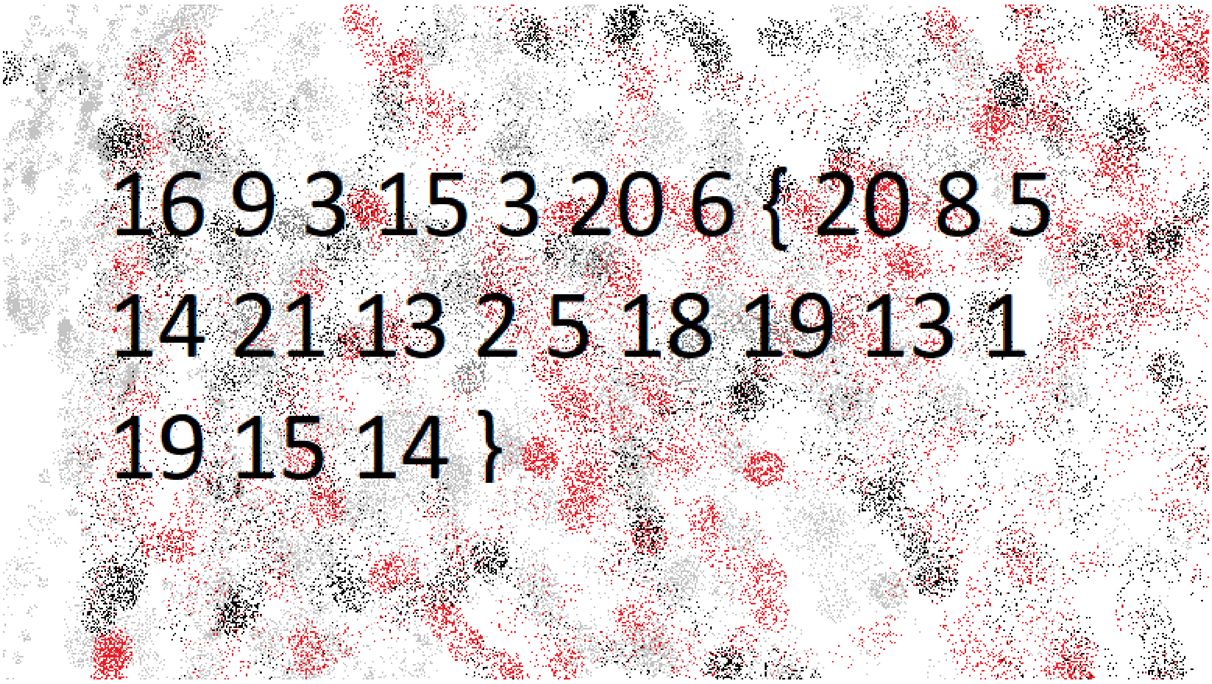
Mind your Ps and Qs (Kris Mendoza)

```
c= 1 #el c asignado por la platadorma

n= 2 #el n asignado por la platadorma

e= 65537

#%pip install factordb-pycli
from factordb.factordb import FactorDB
def dechypherRSA(c, n, e):
    f = FactorDB(n)
    f.connect()
    coprimes=f.get_factor_list()
    p=coprimes[0]
    q=coprimes[1]
    #Now we can calculate the totient
    phi = (p-1)*(q-1)
    print("totient: ",phi)
    # now we can calculate the private key
    # if e*d = 1 mod phi
    # then d = e^-1 mod phi
    d= pow(e,-1,phi)
    print("private key: ",d)
    #Now we can decrypt the message
    m = pow(c,d,n)
    print("message decrypted:",m)
    #Convert the message to hex
    m_hex = hex(m)
    print("hex message:" ,m_hex)
    #Convert the message to ascii
    m_ascii = bytearray.fromhex(m_hex[2:]).decode()
```

```
    print("message to ascci: ",m_ascii)
dechypherRSA(c,n,e)
```

The Numbers (Todos)

The challenge gave an image with numbers, all numbers maped to letters on the alphabeth. no code needed



The answer was PICOCTF{THENUMBERSMASON} (a reference to COD Black Ops)

No Padding, No Problem (Stefano Ulloa)

“Oracles can be your best friend, they will decrypt anything, except the flag's ciphertext. How will you break it?” (Profe dijo que el deber era para el viernes y el viernes ya estaba cerrado todos los del picoCTF, por suerte solo me faltaba este pero lo hice despues del cierre)

```
n = #nof the problem
e = 65537
c = #c of the problem
m2 = 2 # any integer
x = pow(m2, e, n) # encrypt(m2) = m2^e % n
```

```
# encrypt(m1) * encrypt(m2) = encrypt(m1 * m2)
C = c * x

print(f"Decript in the oracle: {C} ")
# decrypt C using oracle since we dont know d to do it manual.
P = int(input('Decrypted C: '))

# P / m2 = m
m = P // m2
m = bytearray.fromhex(format(m, 'x')).decode() #hex to ascii
print('*'*60)
print(f'm = {m}')
```

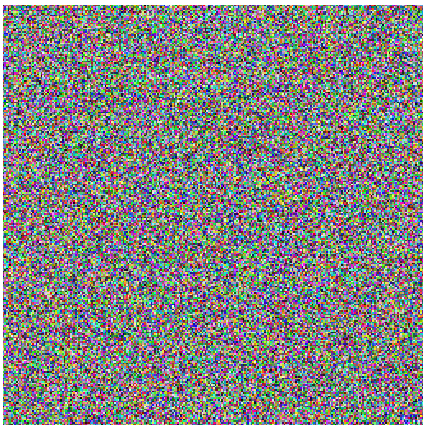
Easy 1 (Emilia Saenz)

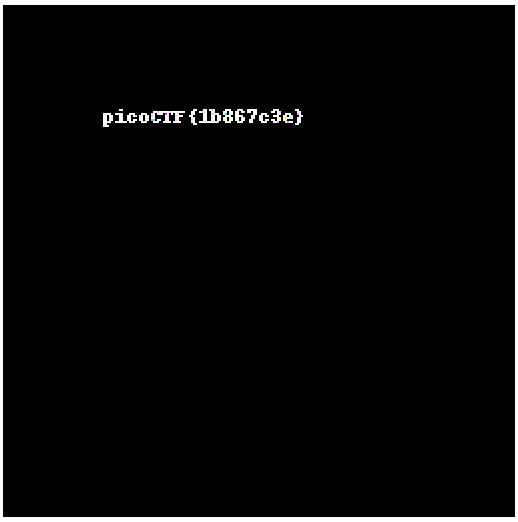
“The one time pad can be cryptographically secure, but not when you know the key. Can you solve this? We've given you the encrypted flag, key, and a table to help
`UFJKXQZQUNB` with the key of `SOLVECRYPTO` "

It was a Vigenere cipher. the key was the same for everyone - `picoCTF{CRYPTOISFUN}`

Pixelated

The problem gave you two images that look like noise, if you took the element-wise XOR operation (and change white to black) you get a second image with a code






```
# -*- coding: utf-8 -*-
"""PicoCTF Pixelated

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10B8kOPwD2x_ux0z-
"""

from PIL import Image
import numpy as np

image1_path = '/content/scrambled1.png'
image2_path = '/content/scrambled2.png'
image1 = Image.open(image1_path)
image2 = Image.open(image2_path)

image1_array = np.array(image1)
image2_array = np.array(image2)

# Perform an element-wise XOR operation between the two image
result_array_xor = np.bitwise_xor(image1_array, image2_array)

# Change pure white (255, 255, 255) to black (0, 0, 0)
# Pure white in an 8-bit image would be where all color chann
```

```
white_pixels = np.all(result_array_xor == [255, 255, 255], ax
result_array_xor[white_pixels] = [0, 0, 0]

result_image_xor = Image.fromarray(result_array_xor)
result_image_xor_path = '/content/result_image_xor.png'
result_image_xor.save(result_image_xor_path)

result_image_xor_path
```

Information security HW

Exercise 1

Let n be a positive integer. A Latin square of order n is an $n \times n$ array L of the integers $1, \ldots, n$ such that every one of the n integers occurs exactly once in each row and each column of L. An example of a Latin square of order 3 is as follows:

	C1	C2	C3
R1	1	2	3
R2	3	1	2
R3	2	3	1

Given any Latin square L of order n, we can define a related Latin Square Cryptosystem. Let the sets $P = C = K = 1, \ldots, n$, be the sets representing the space for the plaintext, ciphertext and keys. For $1 \leq i \leq n$, the encryption rule e_i is defined to be $e_i(j) = L(i, j)$. Here, i would be the key, j the plaintext, and $e_i(j)$ the ciphertext.

Give a complete proof that this Latin Square Cryptosystem achieves perfect secrecy provided that every key is used with equal probability.

Perfect secrecy is formally defined by the condition that the probability distribution of the plaintext is independent of the ciphertext, which can be written as:

$$P(P = p|C = c) = P(P = p)$$

Here, P represents plaintext, C represents ciphertext, p is a specific plaintext, and c is a specific ciphertext.

To have perfect secrecy, every possible ciphertext must be equally likely for a given plaintext, no matter what the plaintext is. In other words, for a given plaintext, the key should not make some ciphertexts more likely than others.

In a Latin Square, if we pick any plaintext (a column), and any ciphertext (a value in that column), there is exactly one key (a row) that will result in that ciphertext for that plaintext. No matter which plaintext you start with, there is exactly one key for each possible ciphertext. Since the key space is the same size as the plaintext space, and each key is used with equal probability, the ciphertext does not favor any particular plaintext.

Therefore, the probability that any plaintext pp will encrypt to any given ciphertext cc is 1/n, assuming a uniform distribution of keys. This is the same as the probability of choosing any plaintext without seeing the ciphertext, which is also 1/n (since there are n plaintexts). Thus, the ciphertext gives no additional information about the plaintext, and the system has perfect secrecy.

Exercise 2

Consider a cryptosystem in which the sets representing the plaintext, ciphertext and keys are: $P = a, b, c$, $K = K1, K2, K3$ and $C = 1, 2, 3, 4$. Suppose the encryption matrix is as follows:

	a	b	c
K1	1	2	3
K2	2	3	4
K3	3	4	1

Given that keys are chosen equiprobably, and the plaintext probability distribution is $Pr[a] = 1/2$, $Pr[b] = 1/3$, $Pr[c] = 1/6$, compute $H(P)$, $H(C)$, $H(K)$, $H(K|C)$, and $H(P|C)$.

$$H(X) = -\sum_{i}^{n} P(x_i) \log_2 P(x_i)$$

$$H(P) = -(\frac{1}{2}log_2\frac{1}{2} + \frac{1}{3}log_2\frac{1}{3} + \frac{1}{6}log_2\frac{1}{6})$$
$$H(K) = -(\frac{1}{3}log_2\frac{1}{3} + \frac{1}{3}log_2\frac{1}{3} + \frac{1}{3}log_2\frac{1}{3})$$

```
import math

# Given probabilities
P_a = 1/2
P_b = 1/3
P_c = 1/6

# Since keys are chosen equiprobably, probability of each key
P_k = 1/3

# Entropy of the plaintext P
```

```
H_P = -(P_a * math.log2(P_a) + P_b * math.log2(P_b) + P_c * ma

# Entropy of the keys K (since all are equiprobable, it's a s
H_K = -3 * (P_k * math.log2(P_k))

# For H(C), we need to compute the probability distribution o
# Given the encryption matrix, each plaintext encrypts to one
# We'll compute the probabilities based on the provided encry
# Since each key is equally likely, we can find the probabili
# keys for a given plaintext and summing the probabilities.

# For a 3x3 matrix, we have 3 ciphertext outcomes for each pl
# Each plaintext letter 'a', 'b', 'c' can result in one of 3 
# The probabilities of the ciphertexts will be the sum of the
P_ciphertexts = {
    # Ciphertext 1 can come from plaintext a with key K1, b w
    1: P_a * (1/3) + P_b * (1/3) + P_c * (1/3),
    # Ciphertext 2 can come from plaintext a with key K2, b w
    2: P_a * (1/3) + P_b * (1/3) + P_c * (1/3),
    3: P_a * (1/3) + P_b * (1/3) + P_c * (1/3),
    4: P_a * (1/3)
}

# Now we compute H(C) using the probabilities of the cipherte
H_C = -sum(P_ciphertexts[c] * math.log2(P_ciphertexts[c]) for

# For H(K|C) and H(P|C), we need to consider the probability 
# However, since the keys are chosen equiprobably and the enc
# knowing the ciphertext determines the key and the plaintext
# Therefore, H(K|C) and H(P|C) should be 0 because if you kno

H_K_given_C = 0
H_P_given_C = 0

H_P, H_K, H_C, H_K_given_C, H_P_given_C
```

This gives us the following output:

Entropy Measure	Value (bits)

Entropy of the plaintext (H(P))	1.459
Entropy of the keys (H(K))	1.585
Entropy of the ciphertext (H(C))	2.016
Conditional entropy of key given ciphertext (H(K|C))	0 bits (since knowing the ciphertext, the plaintext is determined)
Conditional entropy of plaintext given ciphertext (H(P|C))	0 bits (since knowing the ciphertext, the plaintext is determined)

Exercise 3

Compute $H(K|C)$ and $H(K|P,C)$ for the Affine Cipher, assuming that keys are used equiprobably and the plaintexts are equiprobable.

$$E(x) = (ax + b) \mod m$$
$$H(K \mid C) = H(K, C) - H(C)$$
$$H(K \mid P, C) = H(K, P, C) - H(P, C)$$

```
m = 26
phi_m = 12
num_keys = phi_m * m

# Probability of each key is 1/num_keys (since keys are equip
P_key = 1 / num_keys

# Entropy of the key space H(K)
H_K = -num_keys * (P_key * math.log2(P_key))

# Since the Affine Cipher is a bijective mapping for a given
# as the plaintext space. Hence, the entropy of the ciphertex
# Since we're assuming plaintexts are equiprobable and there
P_plaintext = 1 / m
H_P = -m * (P_plaintext * math.log2(P_plaintext))

# Given H(K), we can now calculate H(K|C) using the assumptio
# Since the ciphertext is a deterministic function of the pla
# Therefore, H(K|C) = H(K) - H(C)
```

```
# However, since plaintexts are equiprobable and the Affine C
# the ciphertext space will have the same number of possibili
# As a result, H(K|C) is the same as H(K) since knowing the c

H_K_given_C = H_K
H_K_given_P_C = 0  # As previously explained

H_K, H_K_given_C, H_K_given_P_C
```

- Entropy of the key space H(K): approximately 8.285 bits
- Conditional entropy of the key given the ciphertext H(K|C): approximately 8.285 bits
- Conditional entropy of the key given both plaintext and ciphertext H(K|P,C): 0 bits

The conditional entropy H(K|C)is equivalent to the entropy of the key space because knowing the ciphertext does not reduce the uncertainty of the key without additional information. The conditional entropy H(K|P,C) is 0 because if both the plaintext and ciphertext are known, the key can be determined deterministically.

Exercise 4

Show that the unicity distance of the Hill Cipher (with an $m \times m$ encryption matrix) is less than $\frac{m}{R_L}$. (Note that the number of alphabetic characters ina plaintext of this length is $\frac{m^2}{R_L}$.)

The unicity distance of a cipher is the amount of plaintext required to ensure that the ciphertext can be decrypted uniquely, regardless of the key that is used. It is based on the redundancy of the language used in the plaintext.

The unicity distance U can be calculated as the point where the amount of plaintext P makes the number of possible keys that can produce that plaintext equal to the number of possible plaintexts of the same length. Mathematically, this is when:

$$
\begin{gathered}
\mathbb{R}^{m^2} = \mathbb{R}^{UP} \\
\mathbb{R}^{m^2} = \mathbb{R}^{\frac{m^2}{R_L}U} \\
m^2 = \frac{m^2}{R_L} \cdot U \\
U = R_L
\end{gathered}
$$

QED. the unicity distance of the Hill Cipher is equal to the redundancy of the language, RL, which is less than the total number of characters in the alphabet R