FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

CARRERA DE COMPUTACIÓN

MATERIA:

CRIPTOGRAFÍA Y SEGURIDAD DE LA INFORMACIÓN

DOCENTE:

ING. GIOVANNY MOCAYO

INTEGRANTES:

- Andino John
- Borja Diego
- Cajamarca Anthony
- Cruz Kevin
- Jami Mateo

TEMA:

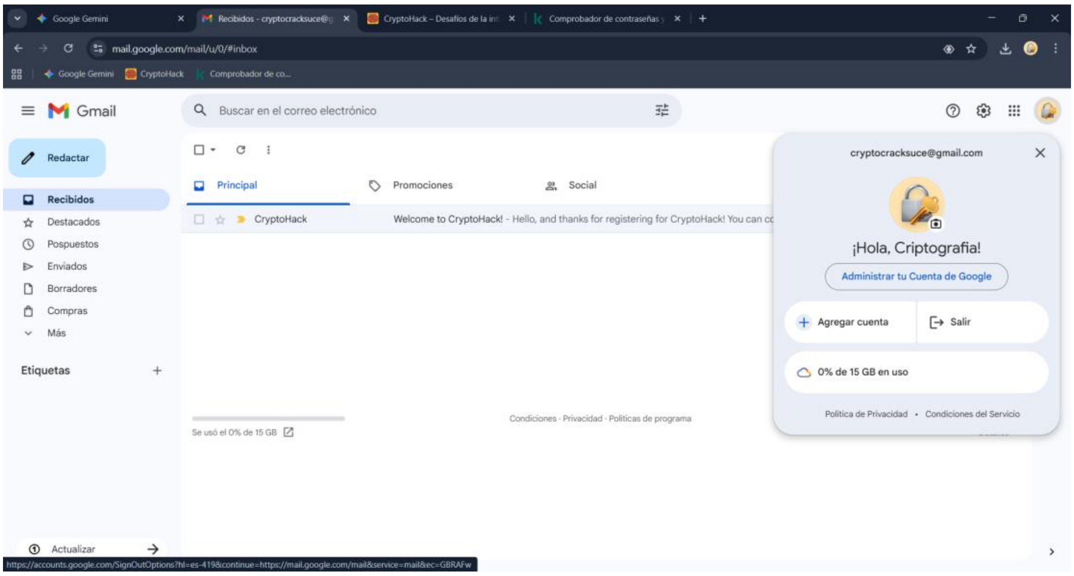
Plataforma CryptoHack

FECHA DE ENTREGA:

5 de enero del 2026

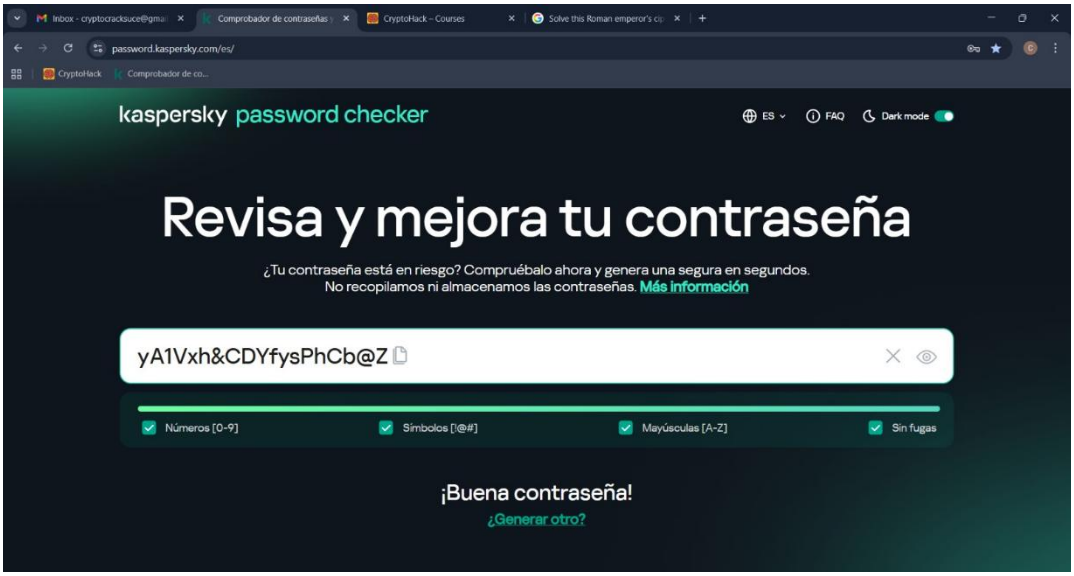
UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS
CARRERA DE COMPUTACIÓN

1. **Crear una cuenta de correo electrónico para el grupo.**



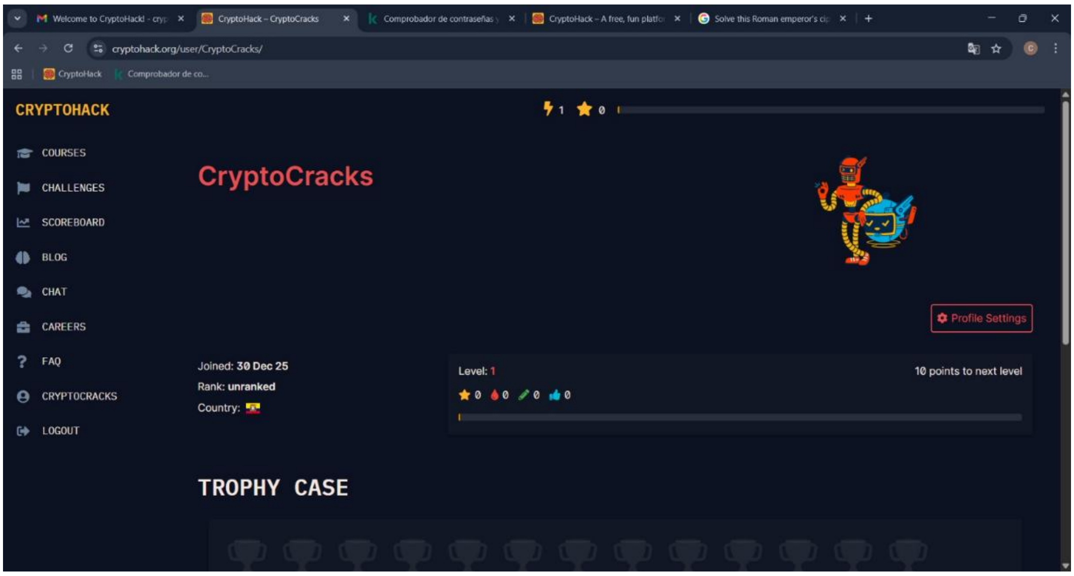
cryptocracksuce@gmail.com

Validar en https://password.kaspersky.com/es/



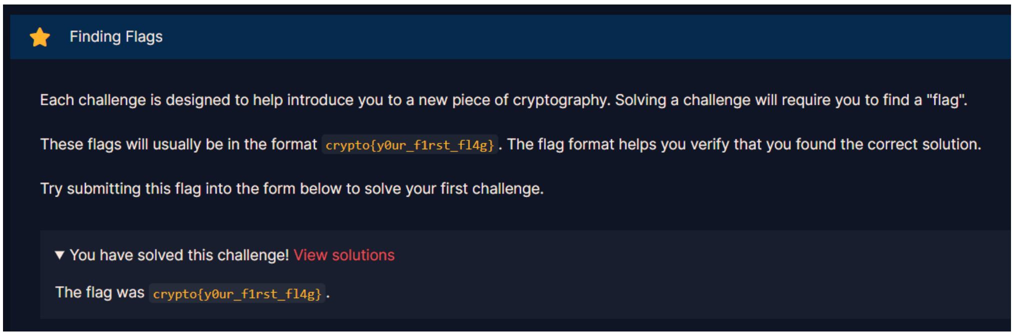
yA1Vxh&CDYfysPhCb@Z

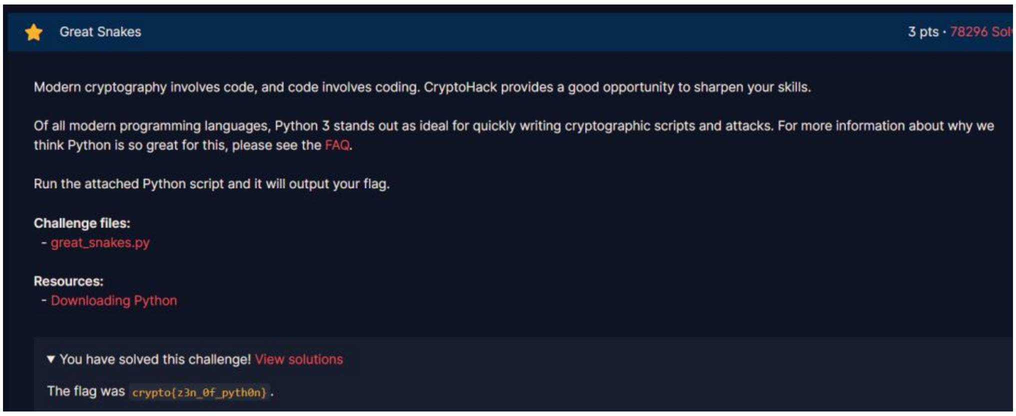
2. **Registrarse en el sitio https://cryptohack.org**



3. Resolver 7 retos

1. Introduction





Great Snakes

```python
import sys

if sys.version_info.major == 2:
    print("You are running Python 2, which is no longer supported. Please update to Python 3.")

ords = [81, 64, 75, 66, 70, 93, 73, 72, 1, 92, 109, 2, 84, 109, 66, 75, 70, 90, 2, 92, 79]

print("Here is your flag:")
print("".join(chr(o ^ 0x32) for o in ords))
```

```
Here is your flag:
crypto{z3n_0f_pyth0n}
```

Network Attacks 5 pts · 30808 Solv

Several of the challenges are dynamic and require you to talk to our challenge servers over the network. This allows you to perform man-in-the-middle attacks on people trying to communicate, or directly attack a vulnerable service. To keep things consistent, our interactive servers always send and receive JSON objects.

Such network communication can be made easy in Python with the `pwntools` module. This is not part of the Python standard library, so needs to be installed with pip using the command line `pip install pwntools`.

For this challenge, connect to `socket.cryptohack.org` on port `11112`. Send a JSON object with the key `buy` and value `flag`.

> The example script below contains the beginnings of a solution for you to modify, and you can reuse it for later challenges.

Connect at `socket.cryptohack.org 11112`

Challenge files:
- pwntools_example.py

▼ You have solved this challenge! View solutions

The flag was `crypto{sh0pp1ng_f0r_fl4g5}`.

```python
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))

        # 2. Recibir el banner inicial (opcional, para ver qué dice el server)
        # Leemos hasta encontrar un salto de línea o el final del buffer
        initial_data = s.recv(2048).decode()
        print("Servidor dice:", initial_data)

        # 3. Preparar el JSON que pide el desafío
        # El reto dice: enviar clave "buy" con valor "flag"
        request = {"buy": "flag"}

        # 4. Enviar los datos (convertir dict a string JSON y luego a bytes)
        # Es importante añadir el '\n' al final para que el servidor sepa que terminaste
        s.sendall((json.dumps(request) + "\n").encode())

        # 5. Recibir la respuesta con la flag
        response = s.recv(2048).decode()
        print("Respuesta recibida:", response)

solve()
```

```
Servidor dice: Welcome to netcat's flag shop!
What would you like to buy?
I only speak JSON, I hope that's ok.


Respuesta recibida: {"flag": "crypto{sh0pp1ng_f0r_fl4g5}"}
```

2. General – Encoding

ASCII 5 pts · 7

ASCII is a 7-bit encoding standard which allows the representation of text using the integers `0`-`127`.

Using the below integer array, convert the numbers to their corresponding ASCII characters to obtain a flag.

```
[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
```

In Python, the `chr()` function can be used to convert an ASCII ordinal number to a character (the `ord()` function does the opposite).

▼ You have solved this challenge! View solutions

The flag was `crypto{ASCII_pr1nt4bl3}`.

ASCII

```python
# La matriz de enteros proporcionada
numeros = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]

# Convertimos cada número a su carácter ASCII y los unimos en una sola cadena
flag = "".join(chr(c) for c in numeros)

print(f"Tu bandera es: {flag}")
```

```
Tu bandera es: crypto{ASCII_pr1nt4bl3}
```

When we encrypt something the resulting ciphertext commonly has bytes which are not printable ASCII characters. If we want to share our encrypted data, it's common to encode it into something more user-friendly and portable across different systems.

Hexadecimal can be used in such a way to represent ASCII strings. First each letter is converted to an ordinal number according to the ASCII table (as in the previous challenge). Then the decimal numbers are converted to base-16 numbers, otherwise known as hexadecimal. The numbers can be combined together, into one long hex string.

Included below is a flag encoded as a hex string. Decode this back into bytes to get the flag.

```
63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f615f6c6f747d
```

In Python, the `bytes.fromhex()` function can be used to convert hex to bytes. The `.hex()` instance method can be called on byte strings to get the hex representation.

Resources:
- ASCII table
- Wikipedia: Hexadecimal

▼ You have solved this challenge! View solutions

The flag was `crypto{You_will_be_working_with_hex_strings_a_lot}`.

Hex

```python
# La cadena hexadecimal proporcionada
hex_string = "63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f615f6c6f747d"

# 1. Convertimos la cadena hex a un objeto de bytes
flag_bytes = bytes.fromhex(hex_string)

# 2. Decodificamos los bytes a una cadena de texto (UTF-8/ASCII) para leerla
flag = flag_bytes.decode('utf-8')

print(f"La bandera es: {flag}")
```

```
La bandera es: crypto{You_will_be_working_with_hex_strings_a_lot}
```

Base64 10 pts · 61394 S

Another common encoding scheme is Base64, which allows us to represent binary data as an ASCII string using an alphabet of 64 characters. One character o Base64 string encodes 6 binary digits (bits), and so 4 characters of Base64 encode three 8-bit bytes.

Base64 is most commonly used online, so binary data such as images can be easily included into HTML or CSS files.

Take the below hex string, *decode* it into bytes and then *encode* it into Base64.

```
72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf
```

In Python, after importing the base64 module with `import base64`, you can use the `base64.b64encode()` function. Remember to decode the hex first as the challenge description states.

▼ You have solved this challenge! View solutions

The flag was `crypto/Base+64+Encoding+is+Web+Safe/`.

Base64

```python
import base64

# 1. La cadena hexadecimal proporcionada
hex_string = "72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf"

# 2. Decodificar de hexadecimal a bytes (crudos)
binary_data = bytes.fromhex(hex_string)

# 3. Codificar esos bytes a Base64
base64_bytes = base64.b64encode(binary_data)

# 4. Convertir el resultado a string para que sea legible
base64_flag = base64_bytes.decode('utf-8')

print(f"Tu resultado en Base64 es: {base64_flag}")
```

```
Tu resultado en Base64 es: crypto/Base+64+Encoding+is+Web+Safe/
```

To illustrate:

```
message: HELLO
ascii bytes: [72, 69, 76, 76, 79]
hex bytes: [0x48, 0x45, 0x4c, 0x4c, 0x4f]
base-16: 0x48454c4c4f
base-10: 310400273487
```

Python's PyCryptodome library implements this with the methods `bytes_to_long()` and `long_to_bytes()`. You will first have to install PyCryptodome and import it with `from Crypto.Util.number import *`. For more details check the FAQ.

Convert the following integer back into a message:

```
11515195063862318899931685488813747395775516287289682636499965282714637259206269
```

▼ You have solved this challenge! View solutions

The flag was `crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}`.

Bytes and Big Integers

```
!pip install pycryptodome
```

Mostrar salida oculta

```
from Crypto.Util.number import long_to_bytes

# El entero proporcionado por el desafío
numero_grande = 11515195063862318899931685488813747395775516287289682636499965282714637259206269

# 1. Convertir el número (base 10) a bytes
mensaje_bytes = long_to_bytes(numero_grande)

# 2. Decodificar los bytes a una cadena de texto
flag = mensaje_bytes.decode('utf-8')

print(f"Tu bandera es: {flag}")
```

```
Tu bandera es: crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}
```

+ Código + Texto

Encoding Challenge 40 pts · 14509 Solves · 107 Soluti

Now you've got the hang of the various encodings you'll be encountering, let's have a look at automating it.

Can you pass all 100 levels to get the flag?

The `13377.py` file attached below is the source code for what's running on the server. The `pwntools_example.py` file provides the start of a solution.

For more information about connecting to interactive challenges, see the FAQ. Feel free to skip ahead to the cryptography if you aren't in the mood for a coding challenge!

> If you want to run and test the challenge locally, then check the FAQ to download the `utils.listener` module.

Connect at `socket.cryptohack.org 13377`

Challenge files:
- 13377.py
- pwntools_example.py

▼ You have solved this challenge! View solutions

The flag was `crypto{3nc0d3_d3c0d3_3nc0d3}`. Please do not share this flag or solutions outside of the CryptoHack platform.

Encoding Challenge

```python
import socket
import json
import base64
import codecs
from Crypto.Util.number import long_to_bytes

# Configuración de conexión
HOST = "socket.cryptohack.org"
PORT = 13377

def solve():
    # Establecer conexión
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))

        # El servidor enviará 100 niveles + el mensaje final con la bandera
        for i in range(101):
            # 1. Recibir datos del servidor
            data = s.recv(4096).decode()
            if not data: break

            received = json.loads(data)

            # Si el servidor nos manda la bandera, terminamos
            if "flag" in received:
```

3. General-Mathematics

Greatest Common Divisor — 15 pts · 30412 Solves · 122 Solutions

The Greatest Common Divisor (GCD), sometimes known as the highest common factor, is the largest number which divides two positive integers (a, b).

For $a = 12, b = 8$ we can calculate the divisors of a: $\{1, 2, 3, 4, 6, 12\}$ and the divisors of b: $\{1, 2, 4, 8\}$. Comparing these two, we see that $\gcd(a, b) = 4$.

Now imagine we take $a = 11, b = 17$. Both a and b are prime numbers. As a prime number has only itself and 1 as divisors, $\gcd(a, b) = 1$.

We say that for any two integers a, b, if $\gcd(a, b) = 1$ then a and b are coprime integers.

If a and b are prime, they are also coprime. If a is prime and $b < a$ then a and b are coprime.

> Think about the case for a prime and $b > a$, why are these not necessarily coprime?

There are many tools to calculate the GCD of two integers, but for this task we recommend looking up Euclid's Algorithm.

Try coding it up; it's only a couple of lines. Use $a = 12, b = 8$ to test it.

Now calculate $\gcd(a, b)$ for $a = 66528, b = 52920$ and enter it below.

Algoritmo

[1] 0 s

```python
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Resultado

[3] 0 s

```python
a = 66528
b = 52920

resultado = gcd(a, b)
print(resultado)
```

```
1512
```

UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS
CARRERA DE COMPUTACIÓN

Extended GCD — 20 pts · 24944 Solves · 84 Solutions

Let a and b be positive integers.

The extended Euclidean algorithm is an efficient way to find integers u, v such that

$$a \cdot u + b \cdot v = \gcd(a, b)$$

Later, when we learn to decrypt RSA ciphertexts, we will need this algorithm to calculate the modular inverse of the public exponent.

Using the two primes $p = 26513, q = 32321$, find the integers u, v such that

$$p \cdot u + q \cdot v = \gcd(p, q)$$

Enter whichever of u and v is the lower number as the flag.

Knowing that p, q are prime, what would you expect $\gcd(p, q)$ to be? For more details on the extended Euclidean algorithm, check out this page.

[4] 0s

```python
def extended_gcd(a, b):
    if b == 0:
        return a, 1, 0
    gcd, x1, y1 = extended_gcd(b, a % b)
    x = y1
    y = x1 - (a // b) * y1
    return gcd, x, y

p = 26513
q = 32321

gcd, u, v = extended_gcd(p, q)
print(gcd, u, v)
```

```
1 10245 -8404
```

Modular Arithmetic 1 — 20 pts · 24642 Solves · 29 Solutions

Imagine you lean over and look at a cryptographer's notebook. You see some notes in the margin:

```
4 + 9 = 1
5 - 7 = 10
2 + 3 = 5
```

At first you might think they've gone mad. Maybe this is why there are so many data leaks nowadays you'd think, but this is nothing more than modular arithmetic modulo 12 (albeit with some sloppy notation).

You may not have been calling it modular arithmetic, but you've been doing these kinds of calculations since you learnt to tell the time (look again at those equations and think about adding hours).

Formally, "calculating time" is described by the theory of congruences. We say that two integers are congruent modulo m if $a \equiv b \mod m$.

Another way of saying this, is that when we divide the integer a by m, the remainder is b. This tells you that if m divides a (this can be written as $m|a$) then $a \equiv 0 \mod m$.

Calculate the following integers:

$$11 \equiv x \mod 6$$
$$8146798528947 \equiv y \mod 17$$

The solution is the smaller of the two integers, (x, y), you obtained after reducing by the modulus.

[5] 0s

```python
x = 11 % 6
y = 8146798528947 % 17

print(x, y)
```

```
5 4
```

Modular Arithmetic 2 — 20 pts · 23386 Solves · 32 Solutions

We'll pick up from the last challenge and imagine we've picked a modulus p, and we will restrict ourselves to the case when p is prime.

The integers modulo p define a field, denoted $\mathbb{F}_p$.

> If the modulus is not prime, the set of integers modulo n define a ring.

A finite field $\mathbb{F}_p$ is the set of integers $0, 1, \ldots, p-1$, and under both addition and multiplication there are inverse elements b_+ and b_* for every element a in the set, such that $a + b_+ = 0$ and $a \cdot b_* = 1$.

> Note that the identity element for addition and multiplication is different! This is because the identity when acted with the operator should do nothing: $a + 0 = a$ and $a \cdot 1 = a$.

Lets say we pick $p = 17$. Calculate $3^{17} \mod 17$. Now do the same but with $5^{17} \mod 17$.

What would you expect to get for $7^{16} \mod 17$? Try calculating that.

This interesting fact is known as Fermat's little theorem. We'll be needing this (and its generalisations) when we look at RSA cryptography.

Now take the prime $p = 65537$. Calculate $273246787654^{65536} \mod 65537$.

Did you need a calculator?

[7] 0s

```python
result = pow(27324678765, 65536, 65537)
print(result)
```

```
1
```

Modular Inverting — 25 pts · 22412 Solves · 49 Solutions

As we've seen, we can work within a finite field $\mathbb{F}_p$, adding and multiplying elements, and always obtain another element of the field.

For all elements g in the field, there exists a unique integer d such that $g \cdot d \equiv 1 \mod p$.

This is the multiplicative inverse of g.

Example: $7 \cdot 8 = 56 \equiv 1 \mod 11$

What is the inverse element: $d = 3^{-1}$ such that $3 \cdot d \equiv 1 \mod 13$?

Think about the little theorem we just worked with. How does this help you find the inverse of an element?

```
[8]
inverse = pow(3, 11, 13)
print(inverse)
```

```
9
```

Comprobación

```
[9]
print((3 * 9) % 13)
```

```
1
```

4. Diffie-Hellman

Trabajando con campos — 10 puntos · 8655 Solves

El conjunto de números enteros módulo N, junto con las operaciones de suma y multiplicación forma un anillo $\mathbb{Z}/N\mathbb{Z}$. Básicamente, esto significa que sumar o multiplicar dos elementos cualesquiera del conjunto devuelve otro elemento del conjunto.

Cuando el módulo es primo: $N = p$, además se nos garantiza una inversa multiplicativa de cada elemento del conjunto, por lo que el anillo se promueve a un campo. En particular, nos referimos a este campo como un campo finito denotado $\mathbb{F}_p$.

El protocolo Diffie-Hellman funciona con elementos de algún campo finito $\mathbb{F}_p$, donde el módulo primo suele ser muy grande (miles de bits), pero para los siguientes desafíos mantendremos los números más pequeños para mayor compacidad.

Dado el primo $p = 991$, y el elemento $g = 209$, encuentra el elemento inverso $d = g^{-1}$ tal que $g \cdot d \mod 991 = 1$.

Retos.py > ...

```python
p = 991
g = 209

# Usando pow para inversa modular
d = pow(g, -1, p)
print(d)
```

```
PS C:\Users\Mateo Jami\Documents\Retos cripto> & C:/Python314/python.exe
ts/Retos cripto/Retos.py"
569
```

UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS
CARRERA DE COMPUTACIÓN

Generadores de grupos — 20 puntos · 7455 Solves · 24 soluciones

Cada elemento de un campo finito $\mathbb{F}_p$ se puede utilizar para crear un subgrupo H bajo acción repetida de multiplicación. En otras palabras, para un elemento g El subgrupo $H = \langle g \rangle = \{g, g^2, g^3, \ldots\}$

Un elemento primitivo de $\mathbb{F}_p$ es un elemento cuyo subgrupo $H = \mathbb{F}_p^*$, *es decir*, cada elemento distinto de cero de $\mathbb{F}_p$, se puede escribir como $g^n \mod p$ para algún número entero n. Debido a esto, a los elementos primitivos a veces se les llama generadores del campo finito.

Para el campo finito con $p = 28151$ encuentra el elemento más pequeño g que es un elemento primitivo de $\mathbb{F}_p$.

> Este problema se puede resolver mediante fuerza bruta, pero también existen formas inteligentes de acelerar el cálculo.

```python
p = 28151
# Factores primos de p-1 = 28150 = 2 * 5^2 * 563
# Solo necesitamos los primos únicos: [2, 5, 563]
factors = [2, 5, 563]

# Probar g desde 2 hacia arriba
for g in range(2, p):
    es_primitivo = True

    # Verificar criterio para cada factor primo
    for q in factors:
        if pow(g, (p-1)//q, p) == 1:
            es_primitivo = False
            break

    # Si pasó todas las pruebas, encontramos el generador
    if es_primitivo:
        print(f"Elemento primitivo más pequeño: {g}")
        break
```

```
Elemento primitivo más pequeño: 7
```

Computación de valores públicos — 25 puntos · 7540 Resuelve· 5 soluciones

Se utiliza el protocolo Diffie-Hellman porque se supone que el logaritmo discreto es un cálculo "difícil" para grupos cuidadosamente elegidos.

El primer paso del protocolo es establecer un primo p y algún generador del campo finito g. Estos deben elegirse cuidadosamente para evitar casos especiales en los que el registro discreto pueda resolverse con algoritmos eficientes. Por ejemplo, una prima segura $p = 2 \cdot q + 1$ Generalmente se elige de tal manera que los únicos factores de $p - 1$ son $\{2, q\}$ donde q es algún otro primo grande. Esto protege a DH de la Pohlig–algoritmo Hellman.

Luego, el usuario elige un número entero secreto $a < p - 1$ y calcula $g^a \mod p$. Esto puede transmitirse a través de una red insegura y, debido a la supuesta dificultad del logaritmo discreto, el número entero secreto no debería ser factible de calcular. El valor a se conoce como valor secreto, mientras que $A = g^a \mod p$ es el valor público.

Dados los parámetros del NIST:

```
g: 2
p:
2410312426921032588552076022197566074856950548502459942654116941958108831682612228890093858261341614673227141477904012196503648957050582631942730706805009223062734745341073406696246014589361659774041027169249453200378729434170325843778659198143763193776859869524088940195577346119843545301547043747207749969763750084308926339295559968882457872412993810129130294592999947926365264059284647209730384947211681434464714438488520940127459844288859336526896320919633919
```

Calcular el valor de $g^a \mod p$ para

```
a:
9721074438370337962458643162004582468469045984889816058567658904788530882468973454873284910377102192220389309433658486261941098303091793930182167633275721201247601400180386739998376433775904344138666111324039795471506590538973555933944925869784000443754656572960275929483495892164153637226683613286895889965413700975590903351376764115959493358573417971489261516942995759702928098053144314470434694474859576699499890902023202343378903232934018623049865998847322815
```

```python
# Parámetros dados
g = 2
p = 2410312426921032588552076022197566074856950548502459942654116941958108831682612228889009385

a = 972107443837033796245864316200458246846904598488981605856765890478853088246897345487328491

# Calcular A = g^a mod p usando pow() de Python (exponenciación modular eficiente)
A = pow(g, a, p)

print("Valor público A = g^a mod p:")
print(A)
```

```
Valor público A = g^a mod p:
1806857697840726523322586721820911358489420128129248078673933653533930681676181753849411715714173604352323556558783759252661061186320274214883104886050164368129191719707402291577330485499513522368289395359523901406138025022522412429238971591272160519144672389532393673832265070057319485399793101182682177465364396277424717543434017666343807276970864475830391776403957550678362368319776566025118492062196941451265638054400177248572271342548616103967411990437357924
```

5. **Secretos compartidos informáticos.**

Secretos compartidos informáticos — 30 puntos · 7186 Solves · 11 soluciones

Ahora es el momento de calcular un secreto compartido utilizando datos recibidos de tu amiga Alice. Como antes, utilizaremos los parámetros del NIST:

```
g: 2
p:
2410312426921032588552076022197566074856950548502459942654116941958108831682612228890093858261341614673227141477904012196503648957050582631942730706805009223062734745341073406696246014589361659774041027169249453200378729434170325843778659198143763193776859869524088940195577346119843545301547043747207749969763750084308926339295559968882457872412993810129130294592999947926365264059284647209730384947211681434464714438488520940127459844288859336526896320919633919
```

Has recibido el siguiente número entero de Alice:

```
A:
70249943217595468278554541264975482909289174351516133994495821400710625291840101960595720462672604202133493023241393916394629829526272643847352371534839862030410331485087487331809285533195024369287293217083414424096866925845838641840923193480821332056735592483730921055532222505605661664236182285229504265881752580410194731633895345823963910901731715743835775619780738974844840425579683385344491015955892106904647602049559477279345982530488299847663103078045601
```

Generas tu entero secreto b y calcula tu valor público $B = g^b \mod p$, que le envías a Alice.

```
b:
12019233252903990344598522535774963020395770409445296724034378433497976840167805970589960962221948290951873387728102115996831454482299243226839490999713763440412177965861508773420532266484619126710566414914227560103715336696193210379850575047730388378348266180934946139100479831339835896583443691529372703954589071507717917136906770122077739814262298488662138085608736103418601750861698417340264213867753834679359191427098195887112064503104510489610448294420720
B:
518386956790041579928056815914221837599234551655144585133414727838977145777213383018096662516814302583841858901021822273505120728451788412967971809038854090670743265187138208169355155411883063541881209288967735684152473260687799664130956969450297407027926009182761627800181901721840557870828019840218548188487260441829333603432714023447029942863076979487889569452186257333512355724725941390498966546682790608125613166744820307691068563387354936732643569654017172
```

```python
# Parámetros públicos
g = 2
p = 2410312426921032588552076022197566074856950548502459942654116941958108831682612228889009385

# Valor público de Alice
A = 702499432175954682785545412649754829092891743515161339944958214007106252918401019605957204

# Tu secreto b
b = 120192332529039903445985225357749630203957704094452967240343784334979768401678059705899609

# Tu valor público B (para verificar)
B_verificar = 51838695679004157992805681591422183759923455165514458513341472783897714577721338

# Calcular secreto compartido: s = A^b mod p
secreto_compartido = pow(A, b, p)

print("Secreto compartido (A^b mod p):")
print(secreto_compartido)

# Verificación: calcular B = g^b mod p y comparar
B_calculado = pow(g, b, p)
print(f"\nVerificación:")
print(f"B calculado: {B_calculado}")
print(f"B dado:      {B_verificar}")
print(f"¿Coinciden?: {B_calculado == B_verificar}")
```

```
Secreto compartido (A^b mod p):
1174130740413820656533832746034841985877302086316388380165984436672307692443711310285014138545204369
4954787251028826734278921045391209523937889610519929016496940631798535983114738203412158799653431363
5143641052285071740844580204300316465834800657740855869350222028570089340467459256762629757122202790
2631157072143330043118418467094237965591198440803970726604537807146703763571606861448354607502654664
700390453794493176794678917352634029713320615865940720837909466

Verificación:
B calculado: 518386956790004157992805681591422183759923455165514458513341472783897714577721338301809
6662516814302583841858901021822273505120728451788412967971809038854090670743265187138208169355155411
8830635418812092889677356841524732606877996641309569694502974070279260091827616278001819017218405578
7082801984021854818848726044182933360343271402344702994286307697948788956945218625733351235572472594
1390498966546682790608125613166744820307691068563387354936732643569654017172
B dado:      518386956790004157992805681591422183759923455165514458513341472783897714577721338301809
6662516814302583841858901021822273505120728451788412967971809038854090670743265187138208169355155411
8830635418812092889677356841524732606877996641309569694502974070279260091827616278001819017218405578
7082801984021854818848726044182933360343271402344702994286307697948788956945218625733351235572472594
1390498966546682790608125613166744820307691068563387354936732643569654017172
¿Coinciden?: True
```

Derivación de claves simétricas — 40 puntos · 6426 Solves · 8 Soluciones

Alice quiere enviarte su bandera secreta y te pide que generes un secreto compartido con ella. También te dice que utilizará el estándar NIST:

```
g: 2
p:
2410312426921032588552076022197566074856950548502459942654116941958108831682612228890093858261341614673227141477904012196503648957050582631942730706805009223062734745341073406696246014589361659774041027169249453200378729434170325843778659198143763193776859869524088940195577346119843545301547043747207749969763750084308926339295559968882457872412993810129130294592999947926365264059284647209730384947211681434464714438488520940127459844288859336526896320919633919
```

Recibes el siguiente número entero de Alice:

```
A:
1122187391395429088880564359534373424013016249772931962692237907571990334483528877513809272625610512061159061737608547288558662879685086684299624481742865016924065000555267977830144740364467977206555914781236397216033805882207640219686011643468275165718132888489024688846101943642459655423609111976363316080620471928236879737944217503462265615774774318986375878440978819238346077908864116156831874695817477772477121232820827728424890845769152726027520772901423784
```

Luego generas tu número entero secreto y calculas el público, que le envías a Alice.

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import hashlib

# Parámetros públicos
p = 241031242692103258855207602219756607485695054850245994265411694195810883168261222889009385
g = 2

# Datos del problema
A = 112218739139542908880564359534373424013016249772931962692237907571990334483528877513809272

b = 197395083814907028991785772714920885908249341925650951555219049411298436217190605190824934

iv_hex = '737561146ff8194f45290f5766ed6aba'
ciphertext_hex = '39c99bf2f0c14678d6a5416faef954b5893c316fc3c48622ba1fd6a9fe85f3dc72a29c394cf4

# 1. Calcular secreto compartido s = A^b mod p
shared_secret = pow(A, b, p)

# 2. Derivar clave AES (SHA1, primeros 16 bytes)
sha1 = hashlib.sha1()
sha1.update(str(shared_secret).encode('ascii'))
key = sha1.digest()[:16]

# 3. Descifrar
ciphertext = bytes.fromhex(ciphertext_hex)
iv = bytes.fromhex(iv_hex)

cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(ciphertext)

# 4. Quitar padding PKCS7
def is_pkcs7_padded(message):
    padding = message[-message[-1]:]
    return all(padding[i] == len(padding) for i in range(0, len(padding)))
```

```python
if is_pkcs7_padded(plaintext):
    flag = unpad(plaintext, 16).decode('ascii')
else:
    flag = plaintext.decode('ascii')

print("Flag:", flag)
```

```
PS C:\Users\Mateo Jami\Documents\Retos cripto>    pip install pycryptodome
```

```
PS C:\Users\Mateo Jami\Documents\Retos cripto> & C:/Python314/python.exe "c
ts/Retos cripto/Retos.py"
Flag: crypto{sh4r1ng_s3cret5_w1th_fr13nd5}
```

Inyección de parámetros 60 puntos · 5057 Resuelve

Estás en condiciones no solo de interceptar el intercambio de claves DH de Alice y Bob, sino también de reescribir sus mensajes. Piense en cómo puede jugar con la ecuación DH que calculan y, por lo tanto, evitar la necesidad de resolver cualquier problema de logaritmo discreto.

Utilice el script de "Derivando claves simétricas" para descifrar la bandera una vez que haya recuperado el secreto compartido.

Conéctate en `socket.cryptohack.org 13371`

```python
def is_pkcs7_padded(data):
    padding = data[-data[-1]:]
    return all(padding[i] == len(padding) for i in range(len(padding)))

def decrypt_flag(shared_secret, iv_hex, ciphertext_hex):
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode())
    key = sha1.digest()[:16]
    iv = bytes.fromhex(iv_hex)
    ciphertext = bytes.fromhex(ciphertext_hex)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    if is_pkcs7_padded(plaintext):
        return unpad(plaintext, 16).decode()
    return plaintext.decode()

# Conectar
r = remote('socket.cryptohack.org', 13371)

# 1. Recibir datos de Alice
line1 = r.recvline().decode().strip()  # Intercepted from Alice: {...}
print("From Alice:", line1)
# Extraer JSON
json_str = line1.split("Intercepted from Alice: ")[1]
data = json.loads(json_str)
p = int(data['p'], 16)
g = int(data['g'], 16)
A = int(data['A'], 16)

# 2. Enviar a Bob con A = 1
to_bob = {"p": data['p'], "g": data['g'], "A": hex(1)}
r.sendline(json.dumps(to_bob).encode())
print("Sent to Bob:", to_bob)
```

```python
# 3. Recibir de Bob
line2 = r.recvline().decode().strip()  # Send to Bob: Intercepted from Bob: {...}
print("From Bob:", line2)
# Manejar el prefijo "Send to Bob: " si existe
if "Send to Bob: " in line2:
    line2 = line2.split("Send to Bob: ")[1]
json_str = line2.split("Intercepted from Bob: ")[1]
data_bob = json.loads(json_str)
B = int(data_bob['B'], 16)

# 4. Enviar a Alice con B = 1
to_alice = {"B": hex(1)}
r.sendline(json.dumps(to_alice).encode())
print("Sent to Alice:", to_alice)

# 5. Recibir flag cifrado
line3 = r.recvline().decode().strip()  # Send to Alice: Intercepted from Alice: {...}
print("Encrypted flag:", line3)
# Manejar el prefijo "Send to Alice: " si existe
if "Send to Alice: " in line3:
    line3 = line3.split("Send to Alice: ")[1]
json_str = line3.split("Intercepted from Alice: ")[1]
data_enc = json.loads(json_str)
iv = data_enc['iv']
encrypted_flag = data_enc['encrypted_flag']

# 6. Calcular secreto compartido = 1
shared_secret = 1

# 7. Descifrar
flag = decrypt_flag(shared_secret, iv, encrypted_flag)
print("\nFlag:", flag)

r.close()
```

```
Flag: crypto{n1c3_0n3_m4ll0ry!!!!!!!!}
Encrypted flag: Send to Alice: Intercepted from Alice: {"iv": "c3d895dc60df432ead245a5885587ddb", "encrypted_flag": "
76186f598dfe7d3010dcbafb2a1b54af2be41531677e7b7b5ee3f974bc6936b"}
```

Grado de exportación — 100 puntos · 4390 Resuelve

Alice y Bob utilizan bases de código heredadas y necesitan negociar parámetros que ambos admiten. Usted ha sido el hombre en el medio de este paso de negociación y puede observar pasivamente a partir de entonces. ¿Cómo vas a arruinarles el día esta vez?

Conéctate en `socket.cryptohack.org 13379`

```python
r = remote('socket.cryptohack.org', 13379)

# 1. Fuerza DH64 en la negociación
line = r.recvline().decode()
data = json.loads(line.split("Intercepted from Alice: ")[1])
data["supported"] = ["DH64"]
r.sendline(json.dumps(data).encode())

# 2. Reenvía elección de Bob
line = r.recvline().decode()
data = json.loads(line.split("Intercepted from Bob: ")[1])
r.sendline(json.dumps(data).encode())

# 3. Obtén p, g, A
line = r.recvline().decode()
data = json.loads(line.split("Intercepted from Alice: ")[1])
p = int(data["p"], 16)
g = int(data["g"], 16)
A = int(data["A"], 16)
r.sendline(json.dumps(data).encode())  # Reenvía sin modificar

# 4. Obtén B
line = r.recvline().decode()
data = json.loads(line.split("Intercepted from Bob: ")[1])
B = int(data["B"], 16)
r.sendline(json.dumps(data).encode())  # Reenvía sin modificar

# 5. Obtén flag cifrada
line = r.recvline().decode()
data = json.loads(line.split("Intercepted from Alice: ")[1])
iv = data["iv"]
enc_flag = data["encrypted_flag"]
```

```python
print("p =", p)
print("g =", g)
print("A =", A)
print("B =", B)
print("iv =", iv)
print("enc_flag =", enc_flag)

from sympy.ntheory.residue_ntheory import discrete_log

print("\nCalculando logaritmo discreto... (esto puede tomar ~1 minuto)")
a = discrete_log(p, A, g)
print(f"a = {a}")

# Secreto compartido
shared_secret = pow(B, a, p)
print(f"Secreto compartido = {shared_secret}")

# Descifrar
def decrypt_flag(shared_secret, iv_hex, ciphertext_hex):
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode())
    key = sha1.digest()[:16]
    iv = bytes.fromhex(iv_hex)
    ciphertext = bytes.fromhex(ciphertext_hex)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    # PKCS7 unpad
    pad_len = plaintext[-1]
    if 1 <= pad_len <= 16 and all(p == pad_len for p in plaintext[-pad_len:]):
        plaintext = plaintext[:-pad_len]
    return plaintext.decode()

flag = decrypt_flag(shared_secret, iv, enc_flag)
print(f"\nFlag: {flag}")

r.close()
```

```
Calculando logaritmo discreto... (esto puede tomar ~1 minuto)
a = 7453845955321759563
Secreto compartido = 6719096046534603348

Flag: crypto{d0wn6r4d35_4r3_d4n63r0u5}
[*] Closed connection to socket.cryptohack.org port 13379
```

☆ Cliente estático — 100 puntos · 2171 Resuelve

Acabas de terminar de escuchar a escondidas una conversación entre Alice y Bob. Ahora tienes la oportunidad de hablar con Bob. ¿Qué vas a decir?

Conéctate en `socket.cryptohack.org 13373`

```python
def is_pkcs7_padded(message):
  padding = message[-message[-1]:]
  return all(padding[i] == len(padding) for i in range(0, len(padding)))


con=remote('socket.cryptohack.org',13373,level='debug')
con.recvuntil(b'Intercepted from Alice: ')
alice=json.loads(con.recvline())
p_hex,g_hex,A_hex=alice['p'],alice['g'],alice['A']
con.recvuntil(b'Intercepted from Bob: ')
B_hex=json.loads(con.recvline())['B']
con.recvuntil(b'Intercepted from Alice: ')
content=json.loads(con.recvline())
iv_hex,enc_flag_hex=content['iv'],content['encrypted']
to_bob={"p":p_hex,"g":A_hex,"A":'0x1'}
con.sendline(json.dumps(to_bob))
con.recvuntil(b'Bob says to you: ')
shared_secret=int(json.loads(con.recvline())['B'],16)
con.recvline()

def decrypt_flag(shared_secret: int, iv: str, ciphertext: str):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    # Decrypt flag
    ciphertext = bytes.fromhex(ciphertext)
    iv = bytes.fromhex(iv)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)

    if is_pkcs7_padded(plaintext):
        return unpad(plaintext, 16).decode('ascii')
    else:
        return plaintext.decode('ascii')

print(decrypt_flag(shared_secret,iv_hex, enc_flag_hex))
```

```
[DEBUG] Received 0x11 bytes:
    b'Bob says to you: '
[DEBUG] Received 0x279 bytes:
    b'{"B": "0x58afd7f3bdc9cdf2eedac4606562282d4fac33c9421c362e055e32481842bab5d9c8a5a7d62efce0d605d5cd11d03c1746a1c90
11efea56ca1ec4ada9092e586cdc433b408afdb1c1af42c3a46abfd8bee3440361fcbb384be18857f0e621137b39ca98765b920344c910cc0c08d9
655be3dbd90d01d9f6c3cde3ca4a8351c75a86a4bc5a2c9ee078dea40a4192e8d424e50f8485614191606cbe4a9dc7210c4380619bf2120352e6f5
38067fd56dd65f30d7d78d932dd01efbfe7488b2d0022"}\n'
    b'Bob says to you: {"iv": "ab9cbe5d7093ae1ca623e4c6af7ae3ac", "encrypted": "05163a87f3a458d45399347d3977172268160e
a767327b98654d8abe2aaeb916a6b587192a080e3576a67350a2973226f009fce18796e62d4ff1b487d3b76685685e573664f4048401c53293d7bf
3cb1"}\n'
crypto{n07_3ph3m3r4l_3n0u6h}
[*] Closed connection to socket.cryptohack.org port 13373
```

6. Aditivo

★ Aditivo — 70 puntos · 2347 Solves · 12 soluciones

Alice y Bob decidieron hacer su DHKE en un grupo aditivo en lugar de un grupo multiplicativo. ¿Qué podría salir mal?

Utilice el script de "Derivando claves simétricas" para descifrar la bandera una vez que haya recuperado el secreto compartido.

Conéctate en `socket.cryptohack.org 13380`

```python
def decrypt_flag(shared_secret, iv_hex, ciphertext_hex):
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode())
    key = sha1.digest()[:16]
    cipher = AES.new(key, AES.MODE_CBC, bytes.fromhex(iv_hex))
    plain = cipher.decrypt(bytes.fromhex(ciphertext_hex))
    pad_len = plain[-1]
    if 1 <= pad_len <= 16 and all(p == pad_len for p in plain[-pad_len:]):
        plain = plain[:-pad_len]
    return plain.decode()

r = remote('socket.cryptohack.org', 13380)

# Recibir línea con parámetros de Alice
line = r.recvline().decode().strip()
print(line)
# Extraer JSON
if "Intercepted from Alice: " in line:
    json_str = line.split("Intercepted from Alice: ")[1]
else:
    json_str = line
data = json.loads(json_str)
p = int(data["p"], 16)
g = int(data["g"], 16)
A = int(data["A"], 16)

# Recibir línea con B de Bob
line = r.recvline().decode().strip()
print(line)
if "Intercepted from Bob: " in line:
    json_str = line.split("Intercepted from Bob: ")[1]
else:
    json_str = line
data = json.loads(json_str)
B = int(data["B"], 16)
```

```python
# Recibir ciphertext
line = r.recvline().decode().strip()
print(line)
if "Intercepted from Alice: " in line:
    json_str = line.split("Intercepted from Alice: ")[1]
else:
    json_str = line
data = json.loads(json_str)
iv = data["iv"]
enc = data["encrypted"]

# Calcular a = A * g^{-1} mod p (grupo aditivo)
g_inv = pow(g, -1, p)
a = (A * g_inv) % p
# Secreto compartido s = a * B mod p ( = a*b*g )
shared_secret = (a * B) % p

# Descifrar
flag = decrypt_flag(shared_secret, iv, enc)
print("Flag:", flag)

r.close()
```

```
234993ef06348fba19b870b80378d7d44204d7cd45ffdb53a9a76334ce43a0ab6e5d1b15919151830881e410cb27990f873991d899b073ab6ee129
21db7f7c72448c8030e7c34898c3e76c0ab5cbb3f56826d9d0a2488f7245f"}
Intercepted from Alice: {"iv": "784263ce3ba38d3521eb2f117c884164", "encrypted": "bea07212333e6d13b95d26bec216fae4a3d83
026eeae520f133c57762c2e0f64509a387e417fbd5fdbed001369474660"}
Intercepted from Alice: {"iv": "784263ce3ba38d3521eb2f117c884164", "encrypted": "bea07212333e6d13b95d26bec216fae4a3d83
026eeae520f133c57762c2e0f64509a387e417fbd5fdbed001369474660"}
026eeae520f133c57762c2e0f64509a387e417fbd5fdbed001369474660"}
Flag: crypto{cycl1c_6r0up_und3r_4dd1710n?}
[*] Closed connection to socket.cryptohack.org port 13380
```

Cliente estático 2 — 120 puntos · 1523 Solves · 15 soluciones

Bob se volvió un poco más cuidadoso con la forma en que verifica los parámetros. Sigue insistiendo en utilizar el *p*y *g* valores proporcionados por su pareja. ¿Me pregunto si se perdió algo?

Conéctate en `socket.cryptohack.org 13378`

```python
r = remote('socket.cryptohack.org', 13378)

def json_recv():
    line = r.recvline()
    return json.loads(line.decode())

def json_send(hsh):
    request = json.dumps(hsh).encode()
    r.sendline(request)

def smooth_p():
    mul = 1
    i = 1
    while 1:
        mul *= i
        if (mul + 1).bit_length() >= p.bit_length() and isPrime(mul + 1):
            return mul + 1
        i += 1

r.recvuntil("Intercepted from Alice: ")
res = json_recv()
p = int(res["p"], 16)
g = int(res["g"], 16)
A = int(res["A"], 16)

r.recvuntil("Intercepted from Bob: ")
res = json_recv()
B = int(res["B"], 16)

r.recvuntil("Intercepted from Alice: ")
res = json_recv()
iv = res["iv"]
ciphertext = res["encrypted"]

s_p = smooth_p()
print(s_p.bit_length())
```

```python
r.recvuntil("send him some parameters: ")
json_send({
    "p": hex(s_p),
    "g": hex(2),
    "A": hex(A)
    })


r.recvuntil("Bob says to you: ")
res = json_recv()
B = int(res["B"], 16)
b = discrete_log(s_p, B, 2)

shared_secret = pow(A, b, p)


def is_pkcs7_padded(message):
    padding = message[-message[-1]:]
    return all(padding[i] == len(padding) for i in range(0, len(padding)))

def decrypt_flag(shared_secret: int, iv: str, ciphertext: str):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    # Decrypt flag
    ciphertext = bytes.fromhex(ciphertext)
    iv = bytes.fromhex(iv)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)

    if is_pkcs7_padded(plaintext):
        return unpad(plaintext, 16).decode('ascii')
    else:
        return plaintext.decode('ascii')

print(decrypt_flag(shared_secret, iv, ciphertext))
```

```
c:\Users\Mateo Jami\Documents\Retos cripto\Retos.py:49: BytesWarning: Text is not bytes; assu
. See https://docs.pwntools.com/#bytes
  r.recvuntil("send him some parameters: ")
c:\Users\Mateo Jami\Documents\Retos cripto\Retos.py:57: BytesWarning: Text is not bytes; assu
. See https://docs.pwntools.com/#bytes
  r.recvuntil("Bob says to you: ")
crypto{uns4f3_pr1m3_sm4ll_oRd3r}
[*] Switching to interactive mode
Bob says to you: {"iv": "17a699a05af2ff8b9cf738b21ef9e23a", "encrypted": "f354f1c5f4ed3b2010b
674c67837f9f7738ae11c883864b6ec8bc33c694fb1f869305cdb1b359d9a8d2ebf92842539994d0598c48ce5a6fa
```

Guión para niños

¡Encontré este genial script en Github y lo he estado usando para guardar mis secretos de cualquiera que escuche el wifi de la escuela!

Archivos de desafío:
- script.py
- salida.txt

```python
from Crypto.Cipher import AES
import hashlib

p = 241031242692103258855207602219756607485695054850245994265411694195810883168261222889009385
g = 2
A = 539556019868756019035615487062583764545019803793635712947528463889304486869497162061335997
B = 652888676809466256406904653886313023288609075262748718135045355786028783611182379919130347
iv = 'c044059ae57b61821a9090fbdefc63c5'
encrypted_flag = 'f60522a95bde87a9ff00dc2c3d99177019f625f3364188c1058183004506bf96541cf241dad1

# El shared_secret real es:
shared_secret = A ^ B ^ 2  # porque g = 2
print("Shared secret:", shared_secret)

def decrypt_flag(shared_secret: int, iv: str, ciphertext: str):
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    ciphertext = bytes.fromhex(ciphertext)
    iv = bytes.fromhex(iv)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    # PKCS7 unpad
    padding_len = plaintext[-1]
    return plaintext[:-padding_len].decode('ascii')

shared_secret = A ^ B ^ 2
flag = decrypt_flag(shared_secret, iv, encrypted_flag)
print("Flag:", flag)
```

```
Shared secret: 1168437420478847128567287495143285894399296237433016158919001493779411967894206164077206908954446484725
3895997645677279526225946106849344851418859267034362066422001424942855463762558757747007799173679733657415510917670483
3608571415223186588256752371083348181694910653505091124800726138587931371746070290662874593191815902807354809922001290
2186566302657796028622790643541788969391922680522075556816558276927136104225021342933713272949133441034264101082531947
970191
Flag: crypto{b3_c4r3ful_w1th_y0ur_n0tati0n}
```

The Matrix 75 pts · 1099 Solves · 6 Solutions

I must get out of here. I must get free, and in this mind is the key, my key!

Challenge files:
- the_matrix.sage
- flag.enc

About SageMathCell

SageMathCell

Type some Sage code below and press Evaluate.

```
# En SageCell, pega el contenido de flag.enc como string
enc_data = '''00000001111101100001101010010001001011000110001001
10111010010100110001011011111110011000001001000001
01011101000110110101010100100100111110110110011111
11101100011011010001111011011000100010110001001010
11111111101001010101001011111101010010011010101001
10010011101000000001000011010010111101111001110110
00011100010011010110000000001011000111010101001011
01001111000110100111110011000101011110010111111110
01111001110011000100000110010101011111010000000011
11001101010111111011110110101010001101001001111101
```

Evaluate

Language: Sage

Share

```
Calculando orden de C...
Orden de C = 351843720888315
Exponente d = 300252250821563

Bytes recuperados (primeros 100):
b'crypto{there_is_no_spoon_66eff188}\xbbIQ\xc0\xf6,w\x82\xbd\xf6\xd7\xdc^3\x94\xa9\xed\xbc\x19\x14\xb1u[c\x1a\xe8/\x92r+4\x85\xe7\x82\xc6Gi\xa7\xfbR\x9cR8+\\\xa2\xb3\x11!+\x8a\xcb6\xc4x[\x92\x07\xa0y\r\x02_

Texto ASCII recuperado (primeros 200 caracteres):
crypto{there_is_no_spoon_66eff188}IQ,w^3u[c/r+4GiRRB+\!+Gx[y_zU$`SNEu\O_wxr~e,7G$(Tloo X|[^aC7%C \Yy
dV #iMu %*RI1P*w)xDY        WJ7w#)gV~a ,wM
[P&;.Qr 6@ck6RiCPL >

¡Flag encontrada!
crypto{there_is_no_spoon_66eff188}
```

Help | Powered by SageMath

7. Hash-Function

Jack's Birthday Hash — 20 pts · 3862 Solves · 13 Solutions

Today is Jack's birthday, so he has designed his own cryptographic hash as a way to celebrate.

Reading up on the key components of hash functions, he's a little worried about the security of the `JACK11` hash.

Given any input data, `JACK11` has been designed to produce a deterministic bit array of length 11, which is sensitive to small changes using the avalanche effect.

Using `JACK11`, his secret has the hash value: `JACK(secret) = 01011001101`.

Given no other data of the `JACK11` hash algorithm, how many unique secrets would you expect to hash to have (on average) a 50% chance of a collision with Jack's secret?

```python

import math

bits = 11
N = 2**bits  # N = 2048


k = N * math.log(2)


print(f"Respuesta redondeada (entero): {round(k)}")
```

```
Respuesta redondeada (entero): 1420
```

Jack's Birthday Confusion — 30 pts · 3328 Solves · 12 Solutions

The last computation has made Jack a little worried about the safety of his hash, and after doing some more research it seems there's a bigger problem.

Given no other data of the `JACK11` hash algorithm, how many unique secrets would you expect to hash to have (on average) a 75% chance of a collision between two distinct secrets?

> Remember, given any input data, `JACK11` has been designed to produce a deterministic bit array of length 11, which is sensitive to small changes using the avalanche effect.

```python

import math
import matplotlib.pyplot as plt

bits = 11
N = 2**bits
probabilidad_objetivo = 0.75
print(f"1. Espacio del Hash (N): {N}")
print(f"2. Probabilidad de colisión buscada: {probabilidad_objetivo * 100}%\n")


probabilidad_sin_colision = 1.0
k = 0
umbral_sin_colision = 1 - probabilidad_objetivo

while probabilidad_sin_colision > umbral_sin_colision:
    k += 1

    probabilidad_sin_colision *= (N - (k - 1)) / N

print(f"--- RESULTADOS ---")
print(f"Secretos necesarios (Cálculo exacto): {k}")
```

```
1. Espacio del Hash (N): 2048
2. Probabilidad de colisión buscada: 75.0%

--- RESULTADOS ---
Secretos necesarios (Cálculo exacto): 76
```