

BÁO CÁO THỰC HÀNH

Môn học: Mật mã học

Tên chủ đề: Cryptohack

GVHD: Tô Trọng Nghĩa

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT219.N21.ANTT

STT	Họ và tên	MSSV	Email
1	Trần Minh Duy	21522010	21522010@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Nội dung	Tình trạng	Trang
1	Giải 10 challenges bất kỳ trên trang cryptohack	>100%	2 – 20
Điểm tự đánh giá			10/10

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

¹ Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1.

Finding Flags

2 pts • 39994 Solves

Each challenge is designed to help introduce you to a new piece of cryptography. Solving a challenge will require you to find a "flag".

These flags will usually be in the format `crypto{y0ur_f1rst_f14g}`. The flag format helps you verify that you found the correct solution.

Try submitting this flag into the form below to solve your first challenge.

Submit flag following the instruction `crypto{y0ur_f1rst_f14g}` and pass.

2.

Great Snakes

3 pts • 32821 Solves

Modern cryptography involves code, and code involves coding. CryptoHack provides a good opportunity to sharpen your skills.

Of all modern programming languages, Python 3 stands out as ideal for quickly writing cryptographic scripts and attacks. For more information about why we think Python is so great for this, please see the [FAQ](#).

Run the attached Python script and it will output your flag.

Challenge files:

- `great_snakes.py`

Resources:

- [Downloading Python](#)

Use <https://www.programiz.com/python-programming/online-compiler> to run `great_snakes.py`:

main.py		Shell
<pre>1 2 3 ords = [81, 64, 75, 66, 70, 93, 73, 72, 1, 92, 109, 2, 84, 109, 66, 75, 70, 90, 2, 92, 79] 4 5 print("Here is your flag:") 6 print("".join(chr(o ^ 0x32) for o in ords)) 7</pre>	<pre>Here is your flag: crypto{z3n_0f_pyth0n} ></pre>	

⇒ Flag: `crypto{z3n_0f_pyth0n}`

3.

☆ ASCII

5 pts • 29218 Solves

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).

Enter flag here: `crypto(FLAG)`

`crypto(ASCII_pr1nt4bl3)` SUBMIT

Coincidentally, using the similar code as the previous challenge, we can pass this challenge, just change the ords array:

main.py

Run

Shell

```

1
2
3 ords = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49,
4         110, 116, 52, 98, 108, 51, 125]
5 print("Here is your flag:")
6 print("".join(chr(o) for o in ords))
7

```

Here is your flag:

`crypto(ASCII_pr1nt4bl3)`

>

⇒ So flag is: `crypto(ASCII_pr1nt4bl3)`

4.

☆ Hex

5 pts • 27916 Solves

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

Enter flag here: `crypto(FLAG)`

`crypto(You_will_be_working_with_hex_strings_a_lot)` SUBMIT

Use cyberchef tool:

Recipe

From Hex

Delimiter: Auto

Input

63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f615f6c6f747d

Output

crypto{You_will_be_working_with_hex_strings_a_lot}

Or python code:

main.py

```
1 str = bytes.fromhex(
    ('63727970746f7b596f755f77696c6c5f62655f776f726b
    696e675f776974685f6865785f737472696e67735f615f6c
    6f747d').decode('ascii'))
2 print(str)
```

Run

Shell

```
crypto{You_will_be_working_with_hex_strings_a_lot}
>
```

⇒ The flag: `crypto{You_will_be_working_with_hex_strings_a_lot}`

5.

☆ Base64 10 pts · 24917 Solves

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 of a 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

Hint: 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.

Enter flag here: crypto/FLAG/

crypto/Base+64+Encoding+is+Web+Safe/ **SUBMIT**

Use code:

```
import base64

data = bytes.fromhex('72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf')
str = base64.b64encode(data).decode('utf-8')
print(str)
```

crypto/Base+64+Encoding+is+Web+Safe/

>

Use cyberchef tool:

Recipe	Input
From Hex Delimiter: Auto	72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf
To Base64 Alphabet: A-Za-z0-9+/=	
	Output crypto/Base+64+Encoding+is+Web+Safe/

⇒ The flag: crypto/Base+64+Encoding+is+Web+Safe/

6.

Cryptosystems like RSA works on numbers, but messages are made up of characters. How should we convert our messages into numbers so that mathematical operations can be applied?

The most common way is to take the ordinal bytes of the message, convert them into hexadecimal, and concatenate. This can be interpreted as a base-16/hexadecimal number, and also represented in base-10/decimal.

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
```

Decimal to Hexadecimal converter

From

To

Decimal

Hexadecimal

Enter decimal number

11515195063862318899931685488813747

10

= Convert

✕ Reset

↕ Swap

Hex number (66 digits)

63727970746F7B336E633064316E365F346C6C5F3768335F7734795F6430776E7D

16

After num is converted to base64, we continue to hex decode to message:

Recipe

Input

From Hex

63727970746F7B336E633064316E365F346C6C5F3768335F7734795F6430776E7D

Delimiter

Auto


ABC 66 1

Output

crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}

⇒ Flag: crypto{3nc0d1n6_4ll_7t3_w4y_d0wn}

7.


XOR Starter

10 pts · 17196 Solves

XOR is a bitwise operator which returns 0 if the bits are the same, and 1 otherwise. In textbooks the XOR operator is denoted by \oplus , but in most challenges and programming languages you will see the caret `^` used instead.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

For longer binary numbers we XOR bit by bit: `0110 ^ 1010 = 1100`. We can XOR integers by first converting the integer from decimal to binary. We can XOR strings by first converting each character to the integer representing the Unicode character.

Given the string `label`, XOR each character with the integer `13`. Convert these integers back to a string and submit the flag as `crypto{new_string}`.

My python code and result:

main.py

Run

Shell

```

1 def xor_with_13(label):
2     result = ""
3     for char in label:
4         result += chr(ord(char) ^ 13)
5     return result
6
7 label = "label"
8 encrypted_label = xor_with_13(label)
9 print(encrypted_label)
    
```

aloha

>

⇒ The flag: crypto{aloha}

8.

☆ XOR Properties

15 pts · 14136 Solves

In the last challenge, you saw how XOR worked at the level of bits. In this one, we're going to cover the properties of the XOR operation and then use them to undo a chain of operations that have encrypted a flag. Gaining an intuition for how this works will help greatly when you come to attacking real cryptosystems later, especially in the block ciphers category.

There are four main properties we should consider when we solve challenges using the XOR operator

```
Commutative:  $A \oplus B = B \oplus A$ 
Associative:  $A \oplus (B \oplus C) = (A \oplus B) \oplus C$ 
Identity:  $A \oplus 0 = A$ 
Self-Inverse:  $A \oplus A = 0$ 
```

Let's break this down. Commutative means that the order of the XOR operations is not important. Associative means that a chain of operations can be carried out without order (we do not need to worry about brackets). The identity is 0, so XOR with 0 "does nothing", and lastly something XOR'd with itself returns zero.

Let's put this into practice! Below is a series of outputs where three random keys have been XOR'd together and with the flag. Use the above properties to undo the encryption in the final line to obtain the flag.

```
KEY1 = a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313
KEY2 ^ KEY1 = 37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e
KEY2 ^ KEY3 = c1545756687e7573db23aa1c3452a098b71a7fbf0fddddd5fc1
FLAG ^ KEY1 ^ KEY3 ^ KEY2 = 04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf
```

My python code and result:

main.py	Shell
<pre>1 KEY1 = 0xa6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313 2 KEY2_KEY1 = 0x37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e 3 KEY2_KEY3 = 0xc1545756687e7573db23aa1c3452a098b71a7fbf0fddddd5fc1 4 FLAG_KEY1_KEY3_KEY2 = 0x04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf 5 6 key2 = KEY1 ^ KEY2_KEY1 7 key3 = key2 ^ KEY2_KEY3 8 flag = FLAG_KEY1_KEY3_KEY2 ^ KEY1 ^ key2 ^ key3 9 10 print(bytes.fromhex(hex(flag)[2:]))</pre>	<pre>b'crypto{x0r_i5_ass0c1at1v3}' ></pre>

⇒ Flag: crypto{x0r_i5_ass0c1at1v3}

9.

☆ Favourite byte

20 pts · 13868 Solves

For the next few challenges, you'll use what you've just learned to solve some more XOR puzzles.

I've hidden some data using XOR with a single byte, but that byte is a secret. Don't forget to decode from hex first.

```
73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d
```

Enter flag here: crypto(FLAG)

crypto{0x10_15_my_f4v0ur173_by7e}

SUBMIT

Use cyberchef tool to brute-force:

Recipe

From Hex

Delimiter
Auto

XOR Brute Force

Key length
1

Sample length
100

Sample offset
0

Scheme
Standard

☐ Null preserving
☒ Print key

☐ Output as hex

Crib (known plaintext string)

Input

```
73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d
```

Output

crypto
next
previous
all
☐ match case
☐ regexp
☐ by word

```

Key = 09: zk`imvb)a()F(,Ft`F-o)lk(. *F{`.|d
Key = 0a: yhcjnua*b+*E+/EwcE|.l*oh+-)Exc-•g
Key = 0b: xibkot`+c*+D*.DvbD}/m+ni*,(Dyb,~f
Key = 0c: •nelhsg,d-,C-)CqCz(j,in-+/C~e+ya
Key = 0d: ~odmirf-e,-B,(BpdB{)k-ho,*•B•d*x`
Key = 0e: }lgnjqe.f/.A/+AsgAx*h.kl/-)A|g){c
Key = 0f: |mfokpd/g./@.*@rf@y+i/jm.(,}@)f(zb
Key = 10: crypto{0x10_15_my_f4v0ur173_by7e}
Key = 11: bsxqunz1y01^04^lx^g5w1ts062^cx6d|
Key = 12: ap{rvmy2z32]37]o{]d6t2wp351]`{5g•

```

Here we get a meaning result with xor key is 0x10, it's also in flag format too, so there is flag: crypto{0x10_15_my_f4v0ur173_by7e}

10.

☆ You either know, XOR you don't

I've encrypted the flag with my secret key, you'll never be able to guess it.

Remember the flag format and how it might help you in this challenge!

0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e077e263451150104

Enter flag here: crypto{FLAG}

Encode the string "crypto{" and xor with data string of the challenge, we will have first 14 hex digit is: 6d 79 58 4f 52 6b 65

Recipe

From Hex

Delimiter: Auto

XOR Brute Force

Key length: 1, Sample length: 100, Sample offset: 0

Scheme: Standard, ☐ Null preserving, ☒ Print key

☐ Output as hex, Crib (known plaintext string): {

XOR

Key: 6d79584f526b65, HEX, Scheme: Standard

☐ Null preserving

Input

0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e077e263451150104

Output

Find: [next] [previous] [all] ☐ match case ☐ regexp ☐ by word

crypto{%r~n-LQCneUAuY6ifjtJsu8JMVXszeboc1lrsGja

Input

6d79584f526b65

Output

Find

myXORke

=> Guess the key: "myXORkey"

Input

myXORkey

Output

Find

6d79584f526b6579

Yeah, it seems my guess is true, sometimes simple is best. By the way, the flag is:

`crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}`

11.

Greatest Common Divisor

15 pts • 12396 Solves • 69 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 $\text{gcd}(a,b) = 4$.

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, $\text{gcd}(a,b) = 1$.

We say that for any two integers a, b , if $\text{gcd}(a,b) = 1$ 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 $\text{gcd}(a,b)$ for $a = 66528$, $b = 52920$ and enter it below.

Belows are my code and result(*the top right corner of the picture*):

```

1 def gcd(a, b):
2     while a != b:
3         if a > b:
4             a = a - b
5         else:
6             b = b - a
7     return a
8
9 print(gcd(52920, 66528))

```

```

1512
>

```

⇒ Flag is '1512'.

12.

★ Extended GCD 20 pts • 10015 Solves • 48 Solutions

Let a and b be positive integers.

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

$$a * u + b * v = \text{gcd}(a, b)$$

💡 Later, when we learn to decrypt RSA, 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

$$p * u + q * v = \text{gcd}(p, q)$$

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

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

My code and result:

```

def extended_gcd(a, b):
    if b == 0:
        return (1, 0)
    else:
        x, y = extended_gcd(b, a % b)
        return (y, x - (a // b) * y)

```

```

1 extended_gcd(26513, 32321)
(10245, -8404)

```

⇒ Flag is '-8404' (because it is the lower number).

13.

Modular Arithmetic 1

20 pts • 10048 Solves • 13 Solutions

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

$$\begin{aligned} 4 + 9 &= 1 \\ 5 - 7 &= 10 \\ 2 + 3 &= 5 \end{aligned}$$

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 $a \equiv b \pmod{m}$.

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 $m \mid a$) then $a \equiv 0 \pmod{m}$.

Calculate the following integers:

$$\begin{aligned} 11 &\equiv x \pmod{6} \\ 8146798528947 &\equiv y \pmod{17} \end{aligned}$$

The solution is the smaller of the two integers.

My code and result:

```
1 min(11%6, 8146798528947%17)
4
```

⇒ Flag is '4' (Is this challenge too simple? I don't know :>)

14.

Modular Arithmetic 2

20 pts • 9552 Solves

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 $\{0, 1, \dots, p-1\}$, and under both addition and multiplication there is an inverse element b for every element a in the set, such that $a + b = 0$ and $a * b = 1$.

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 $a * 1 = a$.

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

What would you expect to get for $7^{16} \pmod{17}$? 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 $273246787654^{65536} \pmod{65537}$.

Did you need a calculator?

Again, my code and result:

```
1 pow(273246787654, 65536, 65537)
1
```

⇒ Flag is '1'.

But there is a question here: *"Did you need a calculator?"*

Looking at Fermat's little theorem...

if p is prime, for every integer a :

$$\text{pow}(a, p) = a \bmod p$$

and, if p is prime and $\text{abs}(a)$ is an integer less than p :

$$\text{pow}(a, p-1) = 1 \bmod p$$

So let's check

$$\text{pow}(273246787654, 65536) \bmod 65537$$

Notice that 65536 is exactly 65537-1,

If 273246787654 and 65537 are coprime,

then the result or flag is '1':

main.py	Run	Shell
<pre>1 def gcd(a, b): 2 while a != b: 3 if a > b: 4 a = a - b 5 else: 6 b = b - a 7 return a 8 9 print(gcd(273246787654, 65537))</pre>		<pre>1 > </pre>

Look at the result, those 2 numbers are coprime ⇒ the flag is 1.

So we just need to check if 2 numbers, 273246787654 and 65537, are coprime or not, this can be calculated by hand, don't need a calculator...

15.

★ Modular Inverting

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 $g * d \equiv 1 \pmod{p}$.

This is the multiplicative inverse of g .

Example: $7 * 8 = 56 \equiv 1 \pmod{11}$

What is the inverse element: $3 * d \equiv 1 \pmod{13}$?



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

My code and result:

```
def mod_inverse(a, m):
    # Calculate the modular inverse of a mod m using the extended Euclidean algorithm
    if gcd(a, m) != 1:
        raise ValueError("a and m are not coprime")
    x, y = extended_gcd(a, m)
    return x % m
```

```
1 mod_inverse(3, 13)
9
```

This code use some math like Extended Eucidean algorithm, you can find the explain in the internet... Any way, the flag is '9'.

16.

★ Network Attacks

5 pts · 19344 Solves

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.

Python makes such network communication easy with the `telnetlib` module. Conveniently, it's part of Python's standard library, so let's use it for now.

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 `nc socket.cryptohack.org 11112`

Challenge files:

- `telnetlib_example.py`

Challenge file code:

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

import telnetlib
import json

HOST = "socket.cryptohack.org"
PORT = 11112

tn = telnetlib.Telnet(HOST, PORT)

def readline():
    return tn.read_until(b"\n")

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

def json_send(hsh):
    request = json.dumps(hsh).encode()
    tn.write(request)

print(readline())
print(readline())
print(readline())
print(readline())

request = {
    "buy": "clothes"
}
json_send(request)

response = json_recv()

print(response)
```

When run:

```
b"Welcome to netcat's flag shop!\n"
b'what would you like to buy?\n'
b"I only speak JSON, I hope that's ok.\n"
b'\n'
{'error': 'Sorry! All we have to sell are flags.'}
```

Correct the code: change “clothes” to “flag” because:

Send a JSON object with the key `buy` and value `flag`.


```

1  import telnetlib
2  import json
3
4  HOST = "socket.cryptohack.org"
5  PORT = 11112
6
7  tn = telnetlib.Telnet(HOST, PORT)
8
9
10
11 def readline():
12     return tn.read_until(b"\n")
13
14 def json_rcv():
15     line = readline()
16     return json.loads(line.decode())
17
18 def json_send(hsh):
19     request = json.dumps(hsh).encode()
20     tn.write(request)
21
22
23 print(readline())
24 print(readline())
25 print(readline())
26 print(readline())
27
28
29 request = {
30     "buy": "flag"
31 }
32 json_send(request)
33
34 response = json_rcv()
35
36 print(response)

```

b"Welcome to netcat's flag shop!\n"
b'What would you like to buy?\n'
b'I only speak JSON, I hope that's ok.\n'
b'\n'
{'flag': 'crypto{sh0pp1ng_f0r_fl4g5}'}

⇒ The flag: crypto{sh0pp1ng_f0r_fl4g5}.

17.

☆ Quadratic Residues
25 pts · 7027 Solves

We've looked at multiplication and division in modular arithmetic, but what does it mean to take the square root modulo an integer?

For the following discussion, let's work modulo $p = 29$. We can take the integer $a = 11$ and calculate $a^2 = 5 \bmod 29$.

As $a = 11$, $a^2 = 5$, we say the square root of 5 is 11.

This feels good, but now let's think about the square root of 18. From the above, we know we need to find some integer a such that $a^2 = 18$.

Your first idea might be to start with $a = 1$ and loop to $a = p-1$. In this discussion p isn't too large and we can quickly look.

Have a go, try coding this and see what you find. If you've coded it right, you'll find that for all $a \in \mathbb{F}_p^*$ you never find an a such that $a^2 = 18$.

What we are seeing, is that for the elements of \mathbb{F}_p^* , not every element has a square root. In fact, what we find is that for roughly one half of the elements of \mathbb{F}_p^* , there is no square root.

💡 We say that an integer x is a *Quadratic Residue* if there exists an a such that $a^2 = x \bmod p$. If there is no such solution, then the integer is a *Quadratic Non-Residue*.

In other words, x is a quadratic residue when it is possible to take the square root of x modulo an integer p .

In the below list there are two non-quadratic residues and one quadratic residue.

Find the quadratic residue and then calculate its square root. Of the two possible roots, submit the smaller one as the flag.

💡 If $a^2 = x$ then $(-a)^2 = x$. So if x is a quadratic residue in some finite field, then there are always two solutions for a .

```

p = 29
ints = [14, 6, 11]

```

Use a simple loop to find the quadratic residue x and all roots a respectively:

Look at the result, we can see the flag is '8', because it's the lower root of quadratic residue 6

⇒ Flag: '8'

```

1  p = 29
2  ints = [14, 6, 11]
3
4  for i in range(p):
5      q = (i*i) % p
6      if q in ints:
7          print(i, q)

```

8 6
21 6

18.

☆ Encoding Challenge
40 pts · 8396 Solves

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 using the incredibly convenient pwntools library, which we recommend. If you'd prefer to use Python's in-built telnetlib, `telnetlib_example.py` is also provided.

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!

Connect at `nc socket.cryptohack.org 13377`

Challenge files:

- `13377.py`
- `pwntools_example.py`
- `telnetlib_example.py`

Enter flag here: crypto{FLAG}
SUBMIT

13377.py code: there is what's running on the server:

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

from Crypto.Util.number import bytes_to_long, long_to_bytes
from utils import listener # this is cryptohack's server-side module and not
                             part of python
import base64
import codecs
import random

FLAG = "crypto{????????????????}"
ENCODINGS = [
    "base64",
    "hex",
    "rot13",
    "bigint",
    "utf-8",
]

with open('/usr/share/dict/words') as f:
    WORDS = [line.strip().replace("'", "") for line in f.readlines()]

class Challenge():
    def __init__(self):
        self.challenge_words = ""
        self.stage = 0

    def create_level(self):
        self.stage += 1
        self.challenge_words = "_".join(random.choices(WORDS, k=3))
        encoding = random.choice(ENCODINGS)

        if encoding == "base64":
            encoded = base64.b64encode(self.challenge_words.encode()).decode()
            # wow so encode
        elif encoding == "hex":
            encoded = self.challenge_words.encode().hex()
        elif encoding == "rot13":
            encoded = codecs.encode(self.challenge_words, 'rot_13')
```

```

def create_level(self):
    self.stage += 1
    self.challenge_words = "_".join(random.choices(WORDS, k=3))
    encoding = random.choice(ENCODINGS)

    if encoding == "base64":
        encoded = base64.b64encode(self.challenge_words.encode()).decode()
        # wow so encode
    elif encoding == "hex":
        encoded = self.challenge_words.encode().hex()
    elif encoding == "rot13":
        encoded = codecs.encode(self.challenge_words, 'rot_13')
    elif encoding == "bigint":
        encoded = hex(bytes_to_long(self.challenge_words.encode()))
    elif encoding == "utf-8":
        encoded = [ord(b) for b in self.challenge_words]

    return {"type": encoding, "encoded": encoded}

#
# This challenge function is called on your input, which must be JSON
# encoded
#
def challenge(self, your_input):
    if self.stage == 0:
        return self.create_level()
    elif self.stage == 100:
        self.exit = True
        return {"flag": FLAG}

    if self.challenge_words == your_input["decoded"]:
        return self.create_level()

    return {"error": "Decoding fail"}

listener.start_server(port=13377)

```

We can see at line 9 there is a flag: `crypto{????????????????????}`

I have submitted that flag but it's not the flag need to find, the flag may be (or should be) hidden in this code.

Continue, because I want to avoid installing many libraries on my computer which I may not use later, so I use `tebnet` code to start:

```

1  import telnetlib
2  import json
3
4  HOST = "socket.cryptohack.org"
5  PORT = 13377
6
7  tn = telnetlib.Telnet(HOST, PORT)
8
9  def readline():
10     return tn.read_until(b"\n")
11
12  def json_recv():
13     line = readline()
14     return json.loads(line.decode())
15
16  def json_send(hsh):
17     request = json.dumps(hsh).encode()
18     tn.write(request)

```

```

19
20     received = json_recv()
21
22     print("Received type: ")
23     print(received["type"])
24     print("Received encoded value: ")
25     print(received["encoded"])
26
27     to_send = {
28         "decoded": "changeme"
29     }
30     json_send(to_send)
31
32     json_recv()
33

```

Here is my code after addition and correction:

```

1  import telnetlib
2  import json
3  import base64
4  import binascii
5  import codecs
6  import sys
7
8  HOST = "socket.cryptohack.org"
9  PORT = 13377
10
11 tn = telnetlib.Telnet(HOST, PORT)
12
13 def readline():
14     return tn.read_until(b"\n")
15
16 def json_rcv():
17     line = readline()
18     return json.loads(line.decode())
19
20 def json_send(hsh):
21     request = json.dumps(hsh).encode()
22     tn.write(request)
23
24 def decode(t, data):
25     if t == 'base64':
26         return base64.b64decode(data).decode('utf-8')
27     elif t == 'hex':
28         return binascii.unhexlify(data).decode('utf-8')
29
30     elif t == 'bigint':
31         return binascii.unhexlify(data.replace('0x', '')).decode('utf-8')
32     elif t == 'rot13':
33         return codecs.encode(data, 'rot_13')
34     elif t == 'utf-8':
35         s = ""
36         for c in data:
37             s += chr(c)
38         return s
39
40 while True:
41     received = json_rcv()
42
43     if "flag" in received:
44         print("FLAG: %s" % received["flag"])
45         sys.exit(0)
46
47     to_send = {
48         "decoded": decode(received["type"], received["encoded"])
49     }
50     json_send(to_send)

```

FLAG: `crypto{3nc0d3_d3c0d3_3nc0d3}`
An exception has occurred, use %tb to see the full traceback.
SystemExit: 0

⇒ The flag: `crypto{3nc0d3_d3c0d3_3nc0d3}`

HẾT