Erel Regev

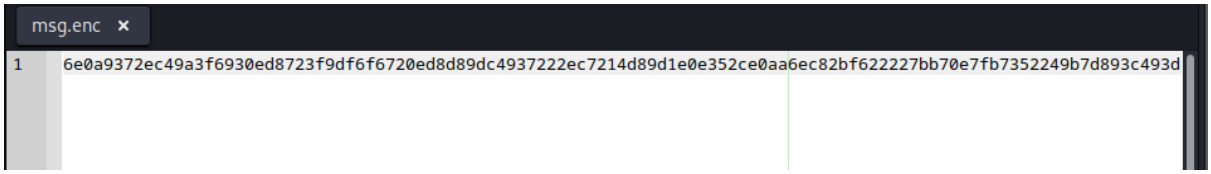
# Table of Contents



# Intro


Received files:


**Viewing msg.enc:**




**Viewing chall.py:**

Erel Regev

msg.enc × chall.py ×

```python
import string
from secret import MSG

def encryption(msg):
    ct = []
    for char in msg:
        ct.append((123 * char + 18) % 256)
    return bytes(ct)

ct = encryption(MSG)
f = open('./msg.enc','w')
f.write(ct.hex())
f.close()
```

## Chall.py

This code performs a simple encryption operation on a message using a basic mathematical formula and then writes the encrypted message to a file named msg.enc in hexadecimal format.

Obviously, the msg.enc message is encrypted, and we should probably reverse the operation the code performs.

Let's break it down before that:

**import string**

This imports the string module, which provides a collection of string constants, like ASCII letters, digits, punctuation, etc. However, in this code, the string module is not used, so this import seems unnecessary.

**from secret import MSG**

This line imports a variable MSG from a module named secret. The contents of the MSG variable are not provided in the code, so it's assumed to be defined elsewhere.

**def encryption(msg)**

This defines a function named encryption that takes a single argument msg. Inside this function, a loop iterates through each character in the input message.

**ct = []**

This initializes an empty list named ct that will hold the encrypted values of the characters.

**for char in msg**

This begins a loop that iterates through each character in the input msg.

HTB Challenge: - Subject: Crypto - Difficulty: Easy

Erel Regev

**ct.append((123 * char + 18) % 256)**

For each character, the code calculates a value using the formula (123 * char + 18) % 256, which involves multiplying the ASCII value of the character by 123, adding 18, and then taking the result modulo 256. The result is added to the ct list.

In modular arithmetic, taking the modulo of a number means finding the remainder when that number is divided by another number (the modulus). In the case of "modulo 256," it means finding the remainder when a number is divided by 256.

For example:

257 % 256 = 1, because 257 divided by 256 leaves a remainder of 1.

512 % 256 = 0, because 512 divided by 256 leaves a remainder of 0.

769 % 256 = 1, because 769 divided by 256 leaves a remainder of 1.

**return bytes(ct)**

Once all characters have been processed, the encrypted values are returned as bytes.

**ct = encryption(MSG)**

This line calls the encryption function with the input MSG and stores the encrypted bytes in the variable ct.

**f = open('./msg.enc','w')**

This opens a file named msg.enc in write mode, creating it if it doesn't exist.

**f.write(ct.hex())**

Here, the encrypted bytes are converted to hexadecimal format using the hex() method and written to the opened file.

After understanding the provided code, I wrote my own to decode this message:

HTB Challenge: - Subject: Crypto - Difficulty: Easy

Erel Regev

# Decrypt

msg.enc × chall.py × decrypt.py ×

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


def decryption(msg):
    pt = []
    for char in msg:
        char = char - 18
        char = 179 * char % 256
        pt.append(char)
    return bytes(pt)

with open('msg.enc') as f:
    ct = bytes.fromhex(f.read())

pt = decryption(ct)
print(pt)

```

And received the following output:

```
┌──(kali㉿kali)-[~/…/HTB/Challenges/Baby_Encryption/BabyEncryption]
└─$ python3 decrypt.py
b'Th3 nucl34r w1ll 4rr1v3 0n fr1d4y.\nHTB{l                                475}'
```