**WRITEUP**

**Team name:** bi0s

**Country:** India

**Username:** bi0s

**Challenge name:** conv

**Proof of flag:**



Elit cybernetica fusce stratagemata enigma penetratio exsertus. CTF{89c5cce663fce1500d22c2ef5112dc2885c491d37d3503118251bdd516b4dcc0} Combinatio complexus networkus quantum facilis vectura obfuscatus. Latitudo cripto diversus et preditus, securitas hexadecimale detectus phantasma scriptum. Insidiae infiltratio breviaria kernel status, protus obscura administratio.  
  
  
  
**Vulnerabilities identified(Summary)**  
  
The way the indices worked in the loop seems confusing but, since a series of addition and multiplication are being perfomed, and since we have the key provided, we can reverse this by mimicing the way the loop works and by simply undoing the addition under modulo via subtraction under modulo and multiplying the csum value with the modular inverse of the key stream to obtain the original plaintext back. It is easily reversible due to the key being available. After Understanding how the encryption function worked, I wrote a script to reverse the obfuscation.  
  
  
**Proof of Solving**  
  
  
Challenge premise  
  
The flag has been put through a convolution function and encrypted with a key.  
We have been given the key and the ciphertext and the goal is to reverse this and obtain the Original plaintext back.

Time for convolutions.

def conv(array1:bytes, array2:bytes) -> bytes:

len1, len2 = len(array1), len(array2)

res = [0]\*(len1 + len2 – 1)

for i in range(len1 + len2 - 1):

csum = 0

for j in range(max(0, i - len2 + 1), min(len1, i + 1)):

csum += array1[j] \* array2[i - j]

res[i] = csum % 256

return res

key = b'\xab\xec\xe9<\xaaC\x7fr\xeb\x8dgQ\xc0\x94\x01\x1d\xc03\x14\x97\xe2\x91\x97\xcf\x8b\x13?\x1d24w|'

cip = conv(plain1,key)

print(bytes(cip).hex())

Output:

cip = ''

cip = conv(plain1,key)

print(bytes(cip).hex())

How it works  
  
**So the function is taking in the plaintext array and key array and performing a sequence of multiplications and additions while sliding across the arrays.**  
**At the start there will be less number of operations performed while at the middle it will overlap completely as max and min of the loop indices allow it to choose the biggest bound, and towards the end the number of operations lessen.**  
  
**Steps**  
  
**We first initialize an empty array, Loop over this array for a length of ciphertext and assigning our csum as the ciphertext.**  
**Assign key and ciphertext/plaintext lengths, The inner loop needs to iterate through the length of key and we sequentially reverse the addition operation in the same way it was originally performed to obtain the original csum value. Once we have this value we now need to find the array1 value, i.e the plaintext, we can do this by trying to see if we can reverse multiplication under modulo by multiplying csum value we obtained as it was in the original conv function with the inverses of keystream and make sure it stays within the bounds of the modulus.**  
**Finally the plaintext including the flag can be printed.**

ct = bytes.fromhex('')

key = b'\xab\xec\xe9<\xaaC\x7fr\xeb\x8dgQ\xc0\x94\x01\x1d\xc03\x14\x97\xe2\x91\x97\xcf\x8b\x13?\x1d24w|'

def decipher(ct, key, plain\_len):

res = [0] \* plain\_len

len1, len2 = plain\_len, len(key)

for i in range(len1):

csum = ct[i]

for j in range(1, min(i + 1, len2)):

csum -= res[i - j] \* key[j] % 256

res[i] = csum \* pow(key[0], -1, 256) % 256

return bytes(res)

plain\_len = len(ct)

flag = decipher(ct, key, plain\_len)

print(flag.decode())