Challenge Name: Feedback

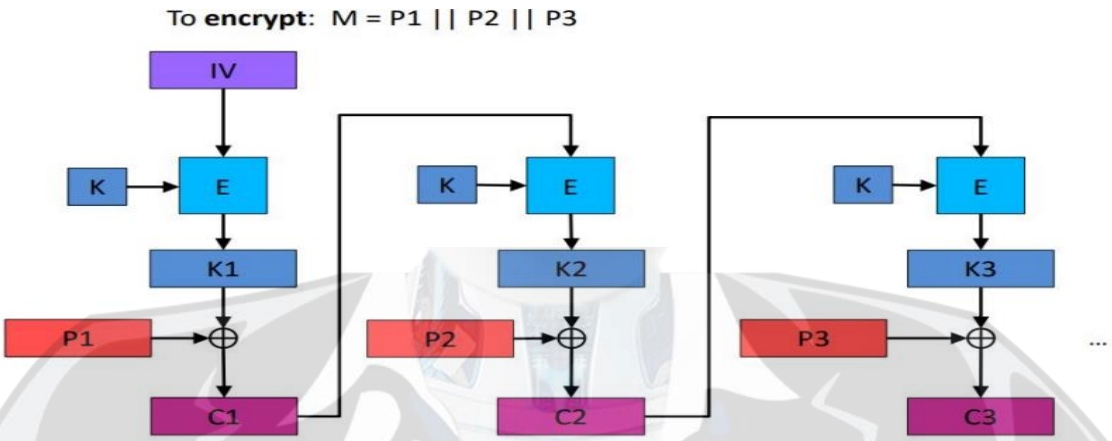
Category: Cypto

This challenge provides its own source code

```
from Crypto import Random
from Crypto.Cipher import AES
import base64
import binascii


class AESCipher:
    def __init__(self):
        self.IV=b"IVhere"
        self.bs = 16
        self.key = b"Keyhere"


    def encrypt(self, string):
        raw = self._pad(string)
        cipher = AES.new(self.key, AES.MODE_CFB, self.IV)
        return binascii.hexlify(cipher.encrypt(raw.encode())).decode()

    def _pad(self, s):
        return s + (16 - len(s) % 16) * chr(16 - len(s) % 16)


a = AESCipher()

while(1):
    print(a.encrypt(input(">>> ")))
```

Whatever user provides is encrypted with AES_CFB and echoed back to the user.

This is how AES CFB works:



IV is AES encrypted with our key, and xor'ed with the plaintext to give ciphertext.

Since this encryptor is reusing IV and key, if we were to encrypt "0x00", we would compromise K1, and xor it with the flag to get first block of the flag.

For next step, we can encrypt "P1\x00". This way we will get the same C1, and compromise K2 used in flag. Xor'ing K2 with flags C2 will give us second block of the flag.

Since no segment size is provided, AES_CFB will not use 16 byte blocks, but will run like a stream crypto, with each block being 8 bits. We can automate this process to get the flag one byte at a time. (solution script is included)

Flag: STMCTF{kn0wn_pl41nTexT_1n_A3S_15_Pur3_M4g1c}