6

```
# syml @ SYMLArch in ~/Sources/Crypto/Home2 [17:45:11]
$ python byte_at_a_time_ECB_decrypt.py
Rollin' in my 5.0
With my rag-top down so my hair can blow
The girlies on standby waving just to say hi
Did you stop? No, I just drove by
```

```python
import random
from typing import Union, Tuple

from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes, getRandomInteger
import base64


addition_m = base64.b64decode("Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd" \
             "24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbi" \
             "BzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW9" \
             "1IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK".encode())
global_key = long_to_bytes(getRandomInteger(128))
random_text = long_to_bytes(getRandomInteger(random.randint(256, 1024)))


def add_key_to_128bit(key: bytes) -> bytes:
    assert len(key) <= 16, "key is too long!"
    while len(key) < 16:
        key += b"\0"
    return key


def list_to_bytes(l):
    b = b""
    for i in l:
        b += chr(i).encode()
    return b


def bytes_to_list(b):
    l = []
    for i in b:
        l.append(i)
    return l


def add_iv_to_64bit(iv: bytes) -> bytes:
    assert len(iv) <= 16, "iv is too long!"
    while len(iv) < 16:
        iv += b"\0"
    return iv

```

```python

def AES_encrypt_ECB(plain_text: bytes, key: bytes, b64enc: bool = False) -> Union[str, bytes]:
    block_size = AES.block_size
    # PKCS7 padding
    if len(plain_text) % block_size != 0:
        padding = chr(block_size - len(plain_text) % block_size).encode()
        plain_text += padding * (block_size - len(plain_text) % block_size)
    key = add_key_to_128bit(key)
    cipher = AES.new(key, mode=AES.MODE_ECB)
    ct = cipher.encrypt(plain_text)
    if b64enc:
        return base64.b64encode(ct).decode()
    else:
        return ct


def AES_decrypt_ECB(cipher_text: bytes, key: bytes) -> Union[bytes, str]:
    key = add_key_to_128bit(key)
    cipher = AES.new(key, mode=AES.MODE_ECB)
    plaintext = cipher.decrypt(cipher_text)
    if plaintext[-1] < 20:
        plaintext = plaintext[:len(plaintext) - plaintext[-1]]
    return plaintext


def bytes_xor(a: bytes, b: bytes) -> bytes:
    result = b""
    for b1, b2 in zip(a, b):
        result += bytes([b1 ^ b2])
    return result


def target_ECB_hard(plain_text: bytes) -> bytes:
    """
    available to attacker
    """
    key = global_key
    plain_text = random_text + plain_text + addition_m
    return AES_encrypt_ECB(plain_text, key)


def detect_random_text_len(block_size: int) -> int:
    c1 = target_ECB_hard(b"")
    c2 = target_ECB_hard(b"A")
    index = 0
    for i in range(len(c1) // block_size):
        c1_t = c1[block_size * i : block_size * i + block_size]
        c2_t = c2[block_size * i: block_size * i + block_size]
        if c1_t != c2_t:
            index = i
            break
    size = 1
    while size < 16:
        c1 = target_ECB_hard(b"A" * size)
        c2 = target_ECB_hard(b"A" * (size - 1))
        c1_t = c1[block_size * index: block_size * index + block_size]
        c2_t = c2[block_size * index: block_size * index + block_size]
```

```
        if c1_t == c2_t:
            break
        size += 1
    return index * 16 + (16 - size + 1)



def detect_len() -> Tuple[int, int, int]:
    ulen = len(target_ECB_hard(b""))
    p1 = p2 = ''
    l1 = l2 = ulen
    while l1 == l2:
        p1 += 'A'
        l2 = len(target_ECB_hard(p1.encode()))
    l1 = l2
    while l1 == l2:
        p2 += 'A'
        l2 = len(target_ECB_hard((p1 + p2).encode()))
    # 返回：unknown-string长度，填充长度，加密块大小
    return (ulen - (len(p1) - 1), len(p1) - 1, len(p2))



def byte_at_a_time_ECB_decrypt() -> str:
    m_len, _, block_size = detect_len()
    random_len= detect_random_text_len(block_size)
    random_index = random_len // block_size
    padding_size =block_size - random_len % block_size
    recovered_list = [0 for i in range(padding_size - 1 + block_size)]
    try:
        for i in range(m_len):
            d = {}
            for j in range(256):
                m = recovered_list[i: i + block_size - 1 + padding_size]
                m.append(j)
                m = list_to_bytes(m)
                c = target_ECB_hard(m)[(random_index + 1) * block_size:
(random_index + 2) * block_size]
                d[c] = m
            m = [0 for j in range(block_size - i % block_size - 1 +
padding_size)]
            c = target_ECB_hard(list_to_bytes(m))[block_size * (i //
block_size + random_index + 1): block_size * (i // block_size + 2 +
random_index)]
            recovered_list.append(d[c][-1])
    except:
        pass
    ans = list_to_bytes(recovered_list[padding_size - 1 + block_size:])
    if ans[-1] < 20:
        plaintext = ans[:len(ans) - ans[-1]]
    return ans.decode()



print(byte_at_a_time_ECB_decrypt())


```

7

```
# syml @ SYMLArch in ~/Sources/Crypto/Home2 [17:46:22]
$ python padding_valid.py
b'ICE ICE BABY\x04\x04\x04\x04'
OK
b'ICE ICE BABY\x05\x05\x05\x05'
Invalid Padding
```

```python
def padding_validation(text: bytes):
    padding_size = text[-1]
    assert padding_size <= 16, "Invalid Padding"
    for i in range(padding_size):
        assert text[len(text) - i - 1] == padding_size, "Invalid Padding"


m1 = b"ICE ICE BABY\x04\x04\x04\x04"
try:
    print(m1)
    padding_validation(m1)
    print("OK")
except Exception as e:
    print(e)
m2 = b"ICE ICE BABY\x05\x05\x05\x05"
try:
    print(m2)
    padding_validation(m2)
    print("OK")
except Exception as e:
    print(e)

```

8

```
# syml @ SYMLArch in ~/Sources/Crypto/Home2 [17:47:16]
$ python CBC_bitflipping.py
user input:  AAAAAAAAAAAAAAAA
AES CBC encrypt:  b'\xba\xb8\x08\x14\x86\x86\xfd\\,H\xads:\xa9LwR\xd0\x18\x9
\xd2\x9f\xda\xc3\td\x86\xe6\xd4\xfa\xa77\xb6e\x11\x90\xe7xTp\xb6?\xf43\xac\x
\xf4'
AES_PAYLOAD: b'Z\xf4+@R\x94\xe5\x8d\xd0\x8a>\xdb\xd7\xce\xd2_'
IS ADMIN
```

```python
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes, getRandomInteger


key = long_to_bytes(getRandomInteger(128))
iv = long_to_bytes(getRandomInteger(64))
pre = b"comment1=cooking%20MCs;userdata="
suf = b";comment2=%20like%20a%20pound%20of%20bacon"


def pkcs7(plain_text: bytes) -> bytes:
    block_size = AES.block_size
```

```
    # PKCS7 padding
    if len(plain_text) % block_size != 0:
        padding = chr(block_size - len(plain_text) % block_size).encode()
        plain_text += padding * (block_size - len(plain_text) % block_size)
    return plain_text


def add_iv_to_64bit(iv: bytes) -> bytes:
    assert len(iv) <= 16, "iv is too long!"
    while len(iv) < 16:
        iv += b"\0"
    return iv


def add_key_to_128bit(key: bytes) -> bytes:
    assert len(key) <= 16, "key is too long!"
    while len(key) < 16:
        key += b"\0"
    return key


def bytes_xor(a: bytes, b: bytes) -> bytes:
    result = b""
    for b1, b2 in zip(a, b):
        result += bytes([b1 ^ b2])
    return result


def AES_encrypt_CBC(plain_text: bytes, key: bytes, iv: bytes) -> bytes:
    plain_text = pkcs7(plain_text)
    key = add_key_to_128bit(key)
    iv = add_iv_to_64bit(iv)
    cipher = AES.new(key, mode=AES.MODE_CBC, iv=iv)
    ct = cipher.encrypt(plain_text)
    return ct


def AES_decrypt_CBC(cipher_text: bytes, key: bytes, iv: bytes) -> bytes:
    key = add_key_to_128bit(key)
    iv = add_iv_to_64bit(iv)
    cipher = AES.new(key, mode=AES.MODE_CBC, iv=iv)
    plaintext = cipher.decrypt(cipher_text)
    # removing padding
    if plaintext[-1] < 20:
        plaintext = plaintext[:len(plaintext) - plaintext[-1]]
    return plaintext


def encrypt(plain_text: bytes) -> bytes:
    plain_text = plain_text.replace(b";", b"%3B").replace(b"=", b"%3D")
    plain_text = pre + plain_text + suf
    return AES_encrypt_CBC(plain_text, key, iv)


def is_admin(plain_text: bytes) -> bool:
    l = plain_text.split(b";")
    for i in l:
        if i == b"admin=true":
```

```
            return True
    return False


def decrypt(ct: bytes) -> bool:
    plain_text = AES_decrypt_CBC(ct, key, iv)
    return is_admin(plain_text)


padding = b"A" * 16
data = encrypt(padding)
print("user input: ", padding.decode())
print("AES CBC encrypt: ", data)
AES_prefix = data[0 : len(pre)]
AES_payload = data[len(pre) : len(pre) + len(padding)]
AES_suffix = data[len(pre) + len(padding):]
AES_payload = bytes_xor(bytes_xor(AES_payload, b';comment2=%20lik'), b';admin=true;xxx=')
print("AES_PAYLOAD:", AES_payload)
if decrypt(AES_prefix + AES_payload + AES_suffix):
    print("IS ADMIN")
else:
    print("NOT ADMIN")
```