CS 406: Cryptography and Network Security
Hands On - 2

Chaitanya Garg, 210050039

February 4, 2024

Challenge - 1

This challenge is pretty straight forward. We just read the iv and key given to us and use them in the openssl command to decrypt the ciphertext. We could also use the AES module from Crypto.Cipher library to decrypt the cipher within python instead of using the bash command.

The flag comes out to be: cs406{op3n_2_d3crypt10n_1_4m}

```python
import os

with open('iv.hex', 'r') as file:
    iv = file.readline().strip()

with open('key.hex', 'r') as file:
    key = file.readline().strip()

os.system(f"openssl enc -aes-128-cbc -d -in ciphertext.bin -out plaintext.txt -K {key} -iv {iv}")
os.system(f"cat plaintext.txt")
```

Challenge - 2

In this Challenge we are to use the fact that in ECB mode, same block of text gives the same ciphertext block. Also, on reading encryptor.py, it is very clear that while encrypting, 1 block has only 1 character of the message, copied to fill the entire block. We are also given much of the message, hence we can just create a mapping from a ciphertext block to the corresponding character in the plaintext. We can then use this mapping, for the remaining blocks of the ciphertext to get the flag. Here, it might be the case that a particular character of the flag was not used in the message substring given to us. In this case we just put a placeholder character (tilde) in the flag. On running the script, the flag happens to contain 2 placeholder characters, right in place for the curly brackets, hence we replace them to get the flag.

The flag comes out to be: cs406{r3dund4nt_l34k4g3}

```python
HEADER = "_Have you heard about the quick brown fox which jumps over the lazy dog?\n__The decimal number system uses the digits 0123456789!\n___The flag is: "

with open("ciphertext.bin", 'rb') as file:
    cipher = file.read().hex()
    cipher = [cipher[i:i+32] for i in range(0, len(cipher), 32)]

mapping = {}
for cip, char in zip(cipher, HEADER):
    mapping[cip] = char

FLAG = ""

for cip in cipher[len(HEADER):]:
    if cip in mapping.keys():
        FLAG += mapping[cip]
    else:
        FLAG += "~"

print(FLAG)
```

Challenge - 3

We first send to the server, in choice 1, a valid parameter $x1\_x2$ Where $x1$ and $x2$ are the blocks (hence $x1$ is 16 bytes). Since this is a valid parameter the server will return us its encryption = $y1\_y2$. Here, $y1 = E_k(x1 \oplus IV)$, we can ignore $y2$ here. Now we construct a random 16 byte cipher block = y'. Now we send to the server, in choice 2, a ciphertext y'_y1. With a very high probability, this will decrypt to a plaintext which does not have valid characters and padding (otherwise we can just run the attack again), in which case the server returns the invalid plaintext = $z1\_z2$ it got on decryption. Here $z2 = D_k(y1) \oplus y'$. We can substitute $y1$ here. Hence $z2 = D_k(E_k(x1 \oplus IV)) \oplus y'$. Hence $z2 = x1 \oplus IV \oplus y'$. Here we know $z2$, $x1$, y', so we can get IV from this relation.

Since here IV = Key, we just found out the key used by the server. Now we can just encrypt "admin=true" and send that in choice2 to the server, to which the server will respond with the encryption of the flag. We can decrypt that to get the flag, since we already know the key.

The flag comes out to be: cs406{fu11_k3y_r3c0v3ry_ftw_1mpl3m3nt_w1th_c4r3}

```
x1 = "a"*15 + "="
x2 = "random"

y1_y2 = choice1(x1 + x2)
y1 = y1_y2[:32]

y_dash = "ff"*16
_, z1_z2 = choice2(y_dash + y1)

z2 = z1_z2[32:]

key = strxor(x1.encode(), strxor(bytes.fromhex(y_dash), bytes.fromhex(z2)))

cipher = AES.new(key, AES.MODE_CBC, iv=key)
ciphertext = cipher.encrypt(pad("admin=true".encode(), AES.block_size))

_, encrypted_flag = choice2(ciphertext.hex())

cipher = AES.new(key, AES.MODE_CBC, iv=key)
flag_padded = cipher.decrypt(bytes.fromhex(encrypted_flag))
flag = unpad(flag_padded, AES.block_size).decode()

print(flag)
```

Challenge - 4

Here we use the fact that for consecutive encryptions, the same stream is used by the sever, just shifted. Hence, if we keep our first message long enough, we can just xor the ciphertext it gives with the message to get the stream cipher. Then for the second message, we send "!flag", in which case the flag's ciphertext will be given to us. We can then shift the stream we calculated appropriately and xor that with the flag's cipher to get the flag.

The flag comes out to be: cs406{y0u_kn0w_th3_gr34t35t_f1lm5_0f_4ll_t1m3_w3r3_n3v3r_m4d3}

```
dummy = "a"*16*40

cip1, cip2 = send_to_server(dummy)

stream = strxor(bytes.fromhex(cip2), dummy.encode())
stream = stream[16*20:]

_, flag_enc = send_to_server("!flag")

flag_enc_bytes = bytes.fromhex(flag_enc)
flag = strxor(flag_enc_bytes, stream[:len(flag_enc_bytes)]).decode()

print(flag)
```

Bonus

In this Challenge, the whole algorithm is already given to use, we just have to implement it in code. We already have the IV and the flag encryption (= C) given to us.

So here, for each block of the flag ciphertext, we iterate over all the bytes from last to first. For each byte, we try all of the 256 possible characters in C'. When we find the character which gives a valid padding, we update the flag plaintext in the corresponding block and byte. We then update C' in all places right of current byte as per the algorithm, to get correct character on next byte position.

The exact code used is also added.

The flag comes out to be: cs406{sid3_ch4nn3l_d4ng3r}

```
iv_hex = IV.hex()
iv_hex = [iv_hex[i:i+2] for i in range(0, len(iv_hex), 2)]

C = [flag_enc[i:i+16].hex() for i in range(0, len(flag_enc), 16)]
C = [[block[i:i+2] for i in range(0, len(block), 2)] for block in C]
P = [["00"] * 16 for _ in range(len(C))]

for k in range(len(P)):
    C_dash = ["00"] * 16
    for byte in range(15, -1, -1):
        for i in range(256):
            C_dash[byte] = str(hex(i)[2:]).zfill(2)
            attack_str = ''.join(C_dash) + ''.join(C[k])

            if validate_padding(IV.hex(), attack_str):
                P2_dash_byte = bytes.fromhex(str(hex(16 - byte)[2:]).zfill(2))
                C_dash_byte = bytes.fromhex(C_dash[byte])
                if k != 0:
                    C_k_minus_1_byte = bytes.fromhex(C[k - 1][byte])
                else:
                    C_k_minus_1_byte = bytes.fromhex(iv_hex[byte])
                P[k][byte] = strxor(P2_dash_byte, strxor(C_dash_byte,
    C_k_minus_1_byte)).hex()

                P2_dash_byte_2 = bytes.fromhex(str(hex(17 - byte)[2:]).zfill(2))
                for b in range(byte, 16):
                    if k != 0:
                        C_k_minus_1_b = bytes.fromhex(C[k - 1][b])
                    else:
                        C_k_minus_1_b = bytes.fromhex(iv_hex[b])
                    C_dash[b] = strxor(P2_dash_byte_2, strxor(bytes.fromhex(P[k][b]),
     C_k_minus_1_b)).hex()
                break

P = [''.join(block) for block in P]
FLAG = bytes.fromhex(''.join(P)).decode('ascii')
print(FLAG)
```