[2023 JBUCTF] crypto

same_nonce_2

Write-Up

문제 개요

제공 파일 : same_nonce_2.py

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from random import randint
from base64 import b64encode, b64decode

flag = open('/flag', 'rb').read()

secret = get_random_bytes(randint(33, 48))
key, nonce = get_random_bytes(16), get_random_bytes(12)
crypto = AES.new(key, AES.MODE_GCM, nonce=nonce)
enc_secret, secret_tag = crypto.encrypt_and_digest(secret)
enc_secret = b64encode(enc_secret).decode()
p1, p2 = get_random_bytes(randint(17, 32)), get_random_bytes(randint(17, 32))

ciphers = []
for p in [p1, p2]:
    crypto = AES.new(key, AES.MODE_GCM, nonce=nonce)
    c, t = crypto.encrypt_and_digest(p)
    c = b64encode(c+t).decode()
    ciphers.append(c)


for i in range(10):
    print('[1] show_ciphers')
    print('[2] show_secret')
    print('[3] verify')
    print('[4] exit')

    try:
        n = int(input('>> '))
        if n < 1 or n > 4:
            raise ValueError
    except:
        print('Retry')
        continue
    if n == 1:
        print(f'c1 : {ciphers[0]}')
        print(f'c2 : {ciphers[1]}')
        continue
    if n == 2:
        print(f'enc_secret : {enc_secret}')
        continue
    if n == 3:
        try:
            edata = input('edata : ')
            edata = b64decode(edata)
            cipher = edata[:-16]
            tag = edata[-16:]
            crypto = AES.new(key, AES.MODE_GCM, nonce=nonce)
            result = crypto.decrypt_and_verify(cipher, tag)
        except:
            print('Failed')
            continue
        if result == secret:
            print(f'flag : {flag.decode()}')
            exit()
    if n == 4:
        exit()
```

33 ~ 48 byte 크기의 secret과, 16byte key, 12byte nonce 생성

secret을 AES-GCM으로 암호화해서 암호화 값인 enc_secret과 인증 값인 secret_tag를 생성

enc_secret 값은 base64 인코딩하여 저장한다.

17 ~ 32 byte 크기의 p1, p2를 생성한 후, p1과 p2를 AES-GCM으로 암호화해서 암호화 값과 인증 값을 합치고, base64인코딩을 한 값을 ciphers 리스트에 저장한다.

1번 입력

위에서 p1, p2를 암호화 한 값과 인증 값을 합친 값을 base64인코딩해서 저장하고 있는 ciphers 리스트의 값들을 출력한다.

2번 입력

enc_secret의 값을 출력한다.

3번 입력

Base64 인코딩 된 값을 입력 받아 디코딩한 후, 그 값의 마지막 16byte를 인증 값으로, 나머지는 암호문으로 해서 복호화를 한다. 이때 인증 값이 암호문에 맞는 값이 아니면 예외가 발생해서 Failed를 출력하며, 맞는 값이면 그 암호문의 복호화 값이 secret 값인지 확인 후 flag를 출력한다.

4번 입력

exit() 함수를 호출해서 프로그램을 종료한다.

문제 풀이

GCM 상세 참고 자료

Galois/Counter Mode : https://url.kr/hs62co
NIST SP 800-38D,: https://csrc.nist.gov/pubs/sp/800/38/d/final 꼭 보세요~

GHASH 함수 구조

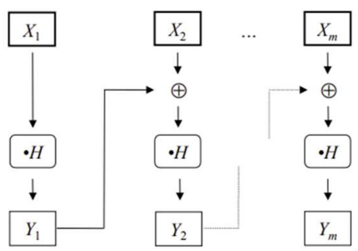

Figure 1: $\mathrm{GHASH}_H(X_1 \| X_2 \| \ldots \| X_m) = Y_m$.

GCTR 함수 구조

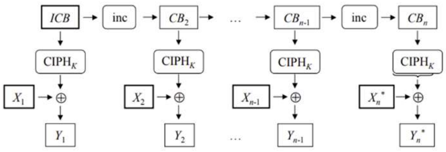

Figure 2: $GCTR_K(ICB, X_1 \| X_2 \| ... \| X_n^*) = Y_1 \| Y_2 \| ... \| Y_n^*$.

GCM 인증 값 생성 구조

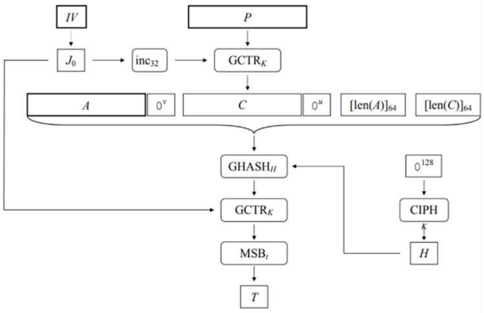

Figure 3: $GCM\text{-}AE_K(IV, P, A) = (C, T)$.

P : The plaintext.
C : The ciphertext.
T : The authentication tag.
A : The additional authenticated data
H : The hash subkey.
t : The bit length of the authentication tag.
0^s : The bit string that consists of s '0' bits.
$len(X)$: The bit length of the bit string X.

$[x]_s$: The binary representation of the non − negative integer x as a string of s bits, where $x < 2^s$.
$\ulcorner x \urcorner$: The least integer that is not less than the real number x.
$X \| Y$: The concatenation of two bit strings X and Y.
$X \oplus Y$: The bitwise exclusive − OR of two bit strings X and Y of the same length.
$X \bullet Y$: The product of two blocks, X and Y, regarded as elements of a certain binary Galois field. $(GF(2^{128}))$
ICB : Initial Counter Block

$MSB_s(X)$: The bit string consisting of the s left − most bits of the bit string X.
$GHASH_H(X)$: The output of the $GHASH$ function under the hash subkey H applied to the bit string X
$GCTR_K(ICB, X)$: The output of the $GCTR$ function for a given block cipher with key K applied to the bit string X with an initial counter block ICB.
$CIPH_K(X)$: The output of the forward cipher function of the block cipher under the key K applied to the block X

아래의 수식에서 C1, C2는 문제 코드에서 p1, p2의 암호화 값이고, T1 T2는 C1, C2의 인증 값이다.

C3는 enc_secret이고, T3는 enc_secret의 인증 값을 나타낸다.

$$136 <= len(C_1), len(C_2) <= 256$$
$$264 <= len(C_3) <= 384$$

$$C_n' = C_n \mid\mid 0^{128\lceil len(C_n)/128 \rceil - len(C_n)}$$
$$C_n' = C_{n,1} \mid\mid C_{n,2} \mid\mid C_{n,3} \mid\mid \ldots \mid\mid C_{n,m}$$
$$len(C_{n,m}) = 128$$

$$C_1' = C_{1,1} \mid\mid C_{1,2}$$
$$C_2' = C_{2,1} \mid\mid C_{2,2}$$
$$C_3' = C_{3,1} \mid\mid C_{3,2} \mid\mid C_{3,3}$$
$$len(T_1) = len(T_2) = len(T_3) = 128$$

$$A = 0$$
$$len(A) = 0$$
$$J_0 = nonce \mid\mid [1]_{32}$$
$$H = CIPH_K(0^{128})$$
$$L_n = ([len(A)]_{64} \mid\mid [len(C_n)]_{64})$$

$$T_1 = MSB_t(GCTR_K(GHASH_H(A \mid\mid C_{1,1} \mid\mid C_{1,2} \mid\mid L_1)))$$

A 는 0 이므로,

$$T_1 = MSB_t(GCTR_K(GHASH_H(C_{1,1} \mid\mid C_{1,2} \mid\mid L_1)))$$

$t = len(T_1) = 128$ 이므로,

$$T_1 = GHASH(C_{1,1} \mid\mid C_{1,2} \mid\mid L_1) \oplus CIPH_k(J_0)$$
$$T_1 = (L_1 \oplus (C_{1,2} \oplus (C_{1,1} \bullet H)) \bullet H) \bullet H \oplus CIPH_k(J_0)$$
$T_1 = C_{1,1} \bullet H^3 \oplus C_{1,2} \bullet H^2 \oplus L_1 \bullet H \oplus CIPH_k(J_0)$ 이다.

위와 같은 방법으로,
$T_2 = C_{2,1} \bullet H^3 \oplus C_{2,2} \bullet H^2 \oplus L_2 \bullet H \oplus CIPH_k(J_0)$ 이다.

$$T_1 = C_{1,1} \bullet H^3 \oplus C_{1,2} \bullet H^2 \oplus L_1 \bullet H \oplus CIPH_k(J_0)$$
$$T_2 = C_{2,1} \bullet H^3 \oplus C_{2,2} \bullet H^2 \oplus L_2 \bullet H \oplus CIPH_k(J_0)$$

T_1 과 T_2 를 xor 하면,
$$T_1 \oplus T_2 = (C_{1,1} \oplus C_{2,1}) \bullet H^3 \oplus (C_{1,2} \oplus C_{2,2}) \bullet H^2 \oplus (L_1 \oplus L_2) \bullet H$$
$$0 = (C_{1,1} \oplus C_{2,1}) \bullet H^3 \oplus (C_{1,2} \oplus C_{2,2}) \bullet H^2 \oplus (L_1 \oplus L_2) \bullet H \oplus (T_1 \oplus T_2)$$
이므로 위 3차 방정식을 풀면 H의 값을 알 수 있으며,
T_1 에 알아낸 H의 값을 대입하여 $CIPH_K(J_0)$의 값도 알아낼 수 있다.
또한, 알아낸 H, $CIPH_K(J_0)$ 을 T_3 에 대입하여 C_3 의 인증 값인 T_3 를 구할 수 있다.

$$T_3 = MSB_t(GCTR_K(GHASH_H(C_{3,1} \mid\mid C_{3,2} \mid\mid C_{3,3} \mid\mid L_3)))$$
$$T_3 = GCTR_K(GHASH_H(C_{3,1} \mid\mid C_{3,2} \mid\mid C_{3,3} \mid\mid L_3))$$
$$T_3 = GHASH_H(C_{3,1} \mid\mid C_{3,2} \mid\mid C_{3,3} \mid\mid L_3) \oplus CIPH_k(J_0)$$
$$T_3 = (L_3 \oplus (C_{3,3} \oplus (C_{3,2} \oplus (C_{3,1} \bullet H)) \bullet H) \bullet H) \bullet H) \oplus CIPH_k(J_0)$$
$$T_3 = C_{3,1} \bullet H^4 \oplus C_{3,2} \bullet H^3 \oplus C_{3,3} \bullet H^2 \oplus L_3 \bullet H \oplus CIPH_k(J_0)$$

위 수식들을 통하여 p1, p2의 암호문 C1, C2의 인증 값인 T1, T2를 이용해서 enc_secret의 인증 값인 T3를 구할 수 있다는 것을 알 수 있다.

이제 이 수식을 파이썬과 sagemath를 이용해서 구현하겠다.

exploit.py

```python
from pwn import *
from base64 import b64encode, b64decode

p = remote('172.17.0.2', 10003)

p.sendafter(b'>> ', b'1\n')
c1 = b64decode(p.recvline()[5:-1])
c2 = b64decode(p.recvline()[5:-1])
p.sendafter(b'>> ', b'2\n')
c3 = b64decode(p.recvline()[13:-1])

t1 = c1[-16:]
t2 = c2[-16:]
c1 = c1[:-16]
c2 = c2[:-16]

print(f'c1 = \'{c1.hex()}\'')
print(f't1 = \'{t1.hex()}\'')
print(f'c2 = \'{c2.hex()}\'')
print(f't2 = \'{t2.hex()}\'')
print(f'c3 = \'{c3.hex()}\'')

tags = input('tags : ')
tags = list(tags.split(' '))

for tag in tags:
    tag = bytes.fromhex(tag.zfill(16*2))
    edata = b64encode(c3 + tag)
    p.sendafter(b'>> ', b'3\n')
    p.sendafter(b'edata : ', edata + b'\n')
    result = p.recvline()[:-1]
    if result != b'Failed':
        print(result.decode())
        exit()
```

먼저 알아낼 수 있는 값인 C1, C2, T1, T2, C3의 값을 알아낸다.

그리고 이 값들을 이용해서 T3 값을 계산하기 위해서는 GF(2^128)에서의 연산과 3차 방정식의 해를 구해야 하므로 sagemath라는 툴을 사용하겠다.

calc_tag.sage

```python
# SageMath version 10.0
#
# Installation Guide : https://doc.sagemath.org/html/en/installation/index.html

def pad(x):
    return x + '00' * ((16-(len(x)//2)%16)%16)

def length(A, C):
    return ( int((len(A)//2)*8).to_bytes(8, byteorder='big') + int(len(C)//2 * 8).to_bytes(8, byteorder='big') ).hex().zfill(32)

def hex2poly(hexx, x):
    poly = 0
    binary = bin(int(hexx, 16))[2:].zfill(128)
    for i in range(len(binary)):
        poly += int(binary[i]) * x^i
    return poly

F, a = GF(2^128, name="a", modulus=x^128 + x^7 + x^2 + x + 1).objgen()
H = PolynomialRing(F, name="H").gen()

# exploit.py output
#
c1 = 'ad163689d664241d4b33e82a060f38347117004c3ddb9f3adc6a27ec'
t1 = '407fae2ac3b9cc87f77ee29393622097'
c2 = 'a523ae1ebc1cbdf2b4c8e3b8bed3dc7eb9c170c7296c65'
t2 = 'd1b4367b30af865a2333b232866362ef'
c3 = '302e84eae1c92b98546a4c28450cc9485067339b7cb7b00d4bb4c19d000e215efdd3394c5223c9057d'
#
# exploit.py output

A = ''# aad
l1= length(A, c1)
#l1 : [len(A)]_64 || [len(C1)]_64
l2= length(A, c2)
#l2 : [len(A)]_64 || [len(C2)]_64
l3= length(A, c3)
#l3 : [len(A)]_64 || [len(C3)]_64

l1 = hex2poly(l1, a)
l2 = hex2poly(l2, a)
l3 = hex2poly(l3, a)

c1 = pad(c1)
c2 = pad(c2)
c3 = pad(c3)

c1 = [c1[i:i+32] for i in range(0, len(c1), 32)]
c2 = [c2[i:i+32] for i in range(0, len(c2), 32)]
c3 = [c3[i:i+32] for i in range(0, len(c3), 32)]

c1 = [hex2poly(c1[i], a) for i in range(0, len(c1))]
c2 = [hex2poly(c2[i], a) for i in range(0, len(c2))]
c3 = [hex2poly(c3[i], a) for i in range(0, len(c3))]

t1 = hex2poly(t1, a)
t2 = hex2poly(t2, a)

ciph_j = (c1[0]*H^3) + (c1[1]*H^2) + (l1*H) + t1    # CIPH(j0)
p = (c1[0] + c2[0])*H^3 + (c1[1] + c2[1])*H^2 + (l1 + l2)*H +(t1 + t2)
t3 = (c3[0]*H^4) + (c3[1]*H^3) + (c3[2]*H^2) + (l3*H) + ciph_j

tag_list = []
for H, m in p.roots():
    tag = t3(H)
    tag = str(int(bin(tag.to_integer())[2:].zfill(128)[::-1], 2).to_bytes(16, byteorder='big').hex())
    tag_list.append(tag)

for tag in tag_list:
    print(f'{tag}', end=' ') # exploit.py input
```

exploit.py를 실행 후, 출력 결과를 exploit.sage 코드에 넣어서 실행하면 3차 방정식이므로 최대 3개의 enc_secret의 인증 값 후보들이 나온다. 이 인증 값들을 입력을 기다리고 있는 exploit.py에 입력해서 flag를 얻을 수 있다.

FLAG

scpCTF{635835b528cdcc44c8eb000321368e83f66c2f3d9feb}