401 – Find an illegally filmed image

Team Information

Team Name: kimbabasaksaksak

Team Member: Jaeheon Kim, Donghyun Kim, Soyoung Yoo, Minhee Lee

Email Address : uaaoong@gmail.com

Instructions

Description The police obtained a Pixel 3, Android 10 version smartphone with illegally filmed images of the suspect in the illegal filming incident. Just before confiscation, the suspect locked the screen of the smartphone he was using, and the suspect does not give out the password to unlock the smartphone. For the smartphone, kernel memory dump and encrypted user data partition image were obtained through a mobile forensic tool. Decrypt illegally shot images to collect evidence.

Target	Hash (MD5)
ramdump	C1C33D2781B3A65BB8B0E3FAF674C5BE
userdata.img	3FDA4C3B2E2242AAA5D7BDD02E6141F5

Questions

1) Obtain all FBE Master Keys of the acquired smartphone. (60 points)
    - Find the top 2 FBE Master Keys based on the frequency of occurrence for each FBE Key Descriptor.
    - Submission format: FBE Key Decriptor:MasterKey1:MasterKey2 in order, separated by colons (:), and described in one line without spaces.
2) Find all the file names of image files in the DCIM folder. (60 points)
3) Find all of the following 4 types of Extend Attributes related to FBE

for each image file obtained in Problem 2. (80 points)

- File contents encryption mode value (1byte)
- File Title Encryption Mode Value (1byte)
- FBE Key Descriptor(8bytes)
- FBE Nonce(16bytes)
- Submission format: FILENAME:FBE_CONTENT_MODE_VALUE:FBE_NAME_MODE_VALUE:FBE_KEY_DESCRIPTOR:FBE_NONCE
  (Separate with a colon (:) and describe in one line without spaces for each image file.)

4) Obtain all the Derivation Encryption Keys for each image file obtained in Problem 2. (100 points)

   - Submission format: FILENAME:FBE_NONCE:FBE_CONTENT_DEK
     (Separate with a colon (:) and describe each image file in one line without spaces.)

5) Decrypt the data of the image file with largest file size among the image files in the DCIM folder, and submit a list of all tweak values used in the decryption process and the SHA256 hash value of the decrypted image file. (100 points)

   - Submission format:
     - ■ tweak values list:
       Write a list of all tweak values on one line with no spaces, separated by commas.
     - ■ FILE_CONTENT_SHA256_HASH
       Describe one line without spaces.

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	The Sleuth Kit	Publisher:	Tobias Groß
Version:	-		
URL:	https://faui1-gitlab.cs.fau.de/tobias.gross/sleuthkit-ext4-fbe		

Name:	fbekeyfind	Publisher:	Tobias Groß
Version:	-		
URL:	https://faui1-files.cs.fau.de/public/one_key_to_rule/fbekeyfind.tar.gz		

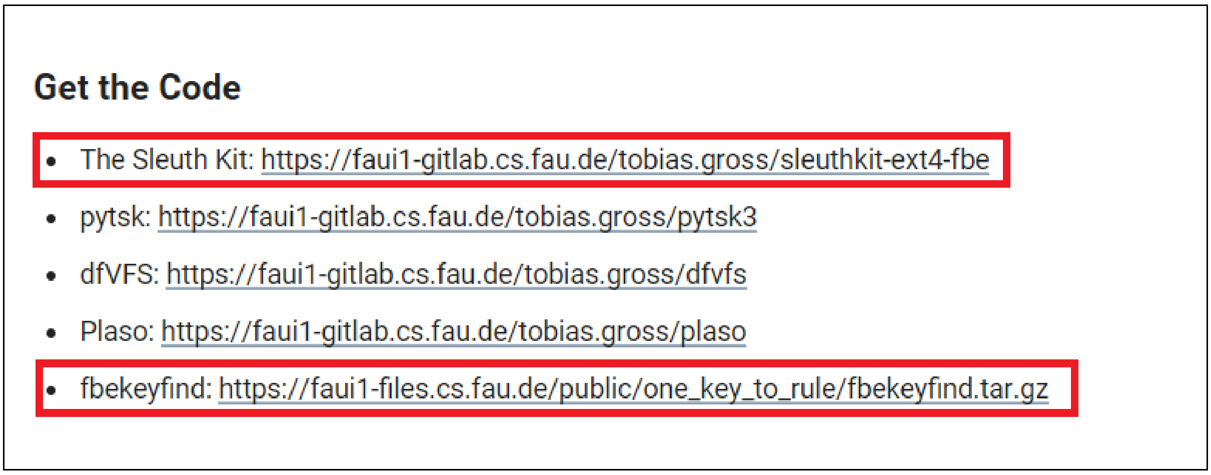
Name:	Python 3	Publisher:	Python
Version:	3.10.6		
URL:	https://www.python.org/		

Name:	aeskeyfind	Publisher:	J. Alex Halderman
Version:	1.0		
URL:	https://citp.princeton.edu/our-work/memory/code/		

Step-by-step methodology:

1) Obtain all FBE Master Keys of the acquired smartphone.

해당 문제에서 파일 기반 암호화 통칭 FBE(File Based Encryption)로 불리는 기술이 적용된 안드로이드 사용자 데이터 userdata.img 파일과 RAM 덤프 파일이 주어졌음.



Get the Code

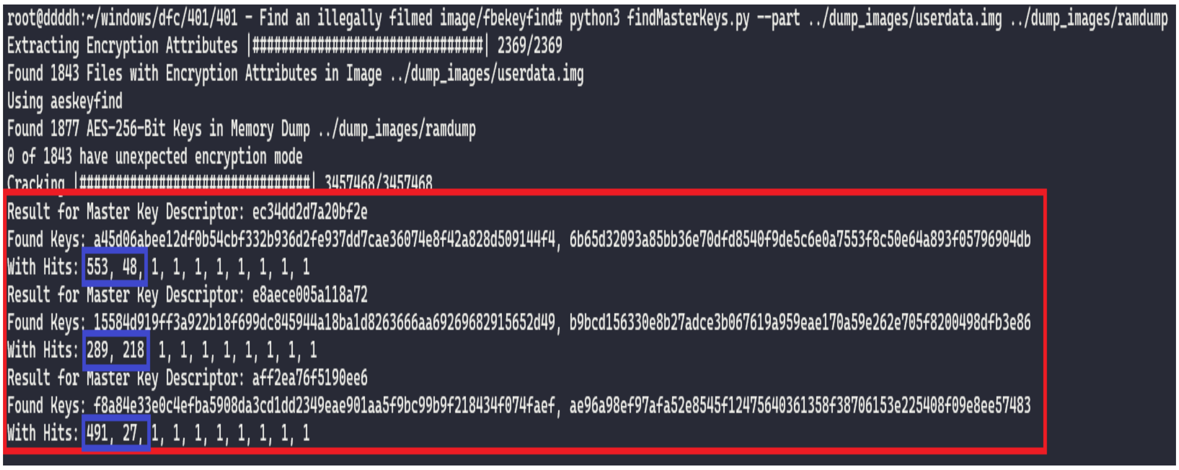
- The Sleuth Kit: https://faui1-gitlab.cs.fau.de/tobias.gross/sleuthkit-ext4-fbe
- pytsk: https://faui1-gitlab.cs.fau.de/tobias.gross/pytsk3
- dfVFS: https://faui1-gitlab.cs.fau.de/tobias.gross/dfvfs
- Plaso: https://faui1-gitlab.cs.fau.de/tobias.gross/plaso
- fbekeyfind: https://faui1-files.cs.fau.de/public/one_key_to_rule/fbekeyfind.tar.gz

[그림] 논문 저자의 공개도구

메모리 덤프에서 FBE 마스터 키를 도출해내는 방법을 검색하면 하나의 [1]논문만 나오며 해당 논문 저자가 만든 공개된 도구(fbekeyfind)를 사용하여 마스터키를 얻을 수 있음. 이때 TSK(The Sleuth Kit)은 ext4-fbe 버전으로 빌드하여 사용해야 하며 aeskeyfind를 사전에 설치하여야 함.

[1] https://www.cs1.tf.fau.de/research/system-security-group/one-key-to-rule/

fbekeyfind 도구를 사용하여 각 Descriptor 별 발생 빈도가 높은 상위 2개의 키를 마스터 키로 가정할 수 있음. 여기서 도출된 마스터 키는 추후 암호화된 파일 타이틀과 콘텐츠를 복호화 할 때 쓰이며 정상 복호화 여부를 통해 올바른 마스터 키인지 검증이 가능함.



```
root@ddddh:~/windows/dfc/401/401 - Find an illegally filmed image/fbekeyfind# python3 findMasterKeys.py --part ../dump_images/userdata.img ../dump_images/ramdump
Extracting Encryption Attributes |################################| 2369/2369
Found 1843 Files with Encryption Attributes in Image ../dump_images/userdata.img
Using aeskeyfind
Found 1877 AES-256-Bit Keys in Memory Dump ../dump_images/ramdump
0 of 1843 have unexpected encryption mode
Cracking |################################| 3457468/3457468
Result for Master Key Descriptor: ec34dd2d7a20bf2e
Found Keys: a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4, 6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db
With Hits: 553, 48, 1, 1, 1, 1, 1, 1, 1, 1
Result for Master Key Descriptor: e8aece005a118a72
Found Keys: 15584d919ff3a922b18f699dc845944a18ba1d8263666aa69269682915652d49, b9bcd156330e8b27adce3b067619a959eae170a59e262e705f8200498dfb3e86
With Hits: 289, 218 1, 1, 1, 1, 1, 1, 1, 1
Result for Master Key Descriptor: aff2ea76f5190ee6
Found Keys: f8a84e33e0c4efba5908da3cd1dd2349eae901aa5f9bc99b9f218434f074faef, ae96a98ef97afa52e8545f12475640361358f38706153e225408f09e8ee57483
With Hits: 491, 27, 1, 1, 1, 1, 1, 1, 1, 1
```

[그림] fbekeyfind 도구 사용 결과

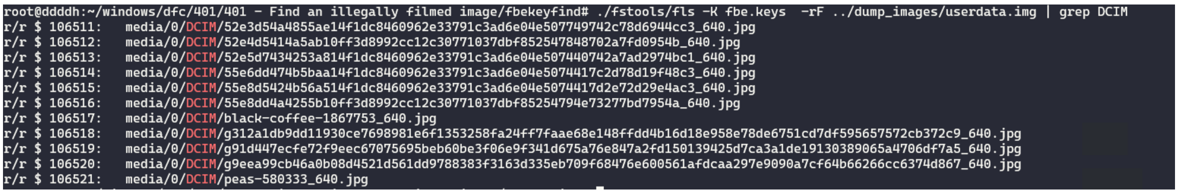
아래는 도구의 결과를 표로 나타낸 결과임.

Descriptor	Master Key1	Master Key2
ec34dd2d7a20bf2e	a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4	6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db
e8aece005a118a72	15584d919ff3a922b18f699dc845944a18ba1d8263666aa69269682915652d49	b9bcd156330e8b27adce3b067619a959eae170a59e262e705f8200498dfb3e86
aff2ea76f5190ee6	f8a84e33e0c4efba5908da3cd1dd2349eae901aa5f9bc99b9f218434f074faef	ae96a98ef97afa52e8545f12475640361358f38706153e225408f09e8ee57483

```
ec34dd2d7a20bf2e:a45d06abee12df0b54cbf332b936d2fe937dd7cae36074e8f42a828d509144f4:6b65d32093a85bb36e70dfd8540f9de5c6e0a7553f8c50e64a893f05796904db
e8aece005a118a72:15584d919ff3a922b18f699dc845944a18ba1d8263666aa69269682915652d49:b9bcd156330e8b27adce3b067619a959eae170a59e262e705f8200498dfb3e86
aff2ea76f5190ee6:f8a84e33e0c4efba5908da3cd1dd2349eae901aa5f9bc99b9f218434f074faef:ae96a98ef97afa52e8545f12475640361358f38706153e225408f09e8ee57483
```

2) Find all the file names of image files in the DCIM folder.

사전에 구한 마스터 키와 TSK(The Sleuth Kit)의 fls를 사용하면 DCIM 폴더 내 이미지 파일들을 식별할 수 있음.



```
root@ddddh:~/windows/dfc/401/401 - Find an illegally filmed image/fbekeyfind# ./fstools/fls -K fbe.keys  -rF ../dump_images/userdata.img | grep DCIM
r/r $ 106511:   media/0/DCIM/52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
r/r $ 106512:   media/0/DCIM/52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
r/r $ 106513:   media/0/DCIM/52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
r/r $ 106514:   media/0/DCIM/55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
r/r $ 106515:   media/0/DCIM/55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
r/r $ 106516:   media/0/DCIM/55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
r/r $ 106517:   media/0/DCIM/black-coffee-1867753_640.jpg
r/r $ 106518:   media/0/DCIM/g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg
r/r $ 106519:   media/0/DCIM/g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg
r/r $ 106520:   media/0/DCIM/g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg
r/r $ 106521:   media/0/DCIM/peas-580333_640.jpg
```

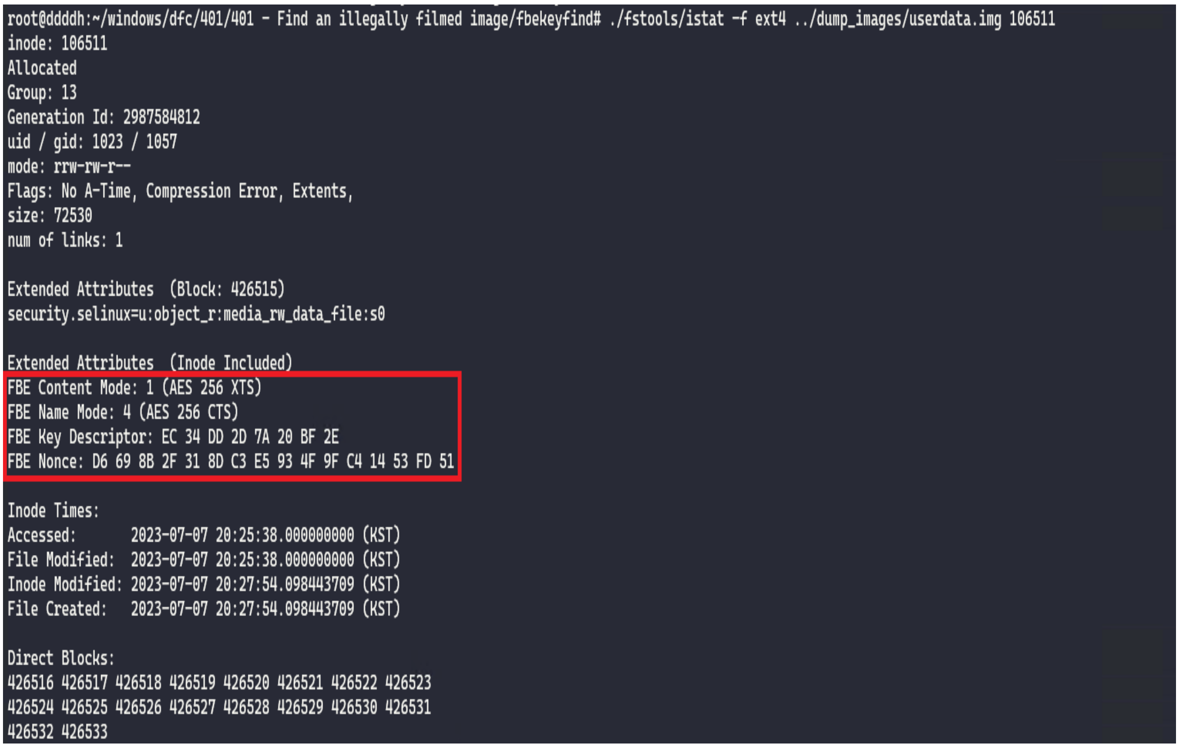
[그림] fls 수행 결과(DCIM 내 이미지 목록)

fls 도구 수행 결과 DCIM 폴더 내에 존재하는 이미지 확장자를 가진 파일은 총 11개로 다음과 같음.

media/0/DCIM/52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg
media/0/DCIM/52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
media/0/DCIM/52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg
media/0/DCIM/55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg
media/0/DCIM/55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg
media/0/DCIM/55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg
media/0/DCIM/black-coffee-1867753_640.jpg
media/0/DCIM/g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg
media/0/DCIM/g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg
media/0/DCIM/g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg
media/0/DCIM/peas-580333_640.jpg

3) Find all of the following 4 types of Extend Attributes related to FBE for each image file obtained in Problem 2

지문에서 요구하는 Attributes Value는 File Title/Contents의 Encryption Mode, Descriptor, Nonce 값이며 TSK(The Sleuth Kit)의 istat을 사용하여 구할 수 있음.

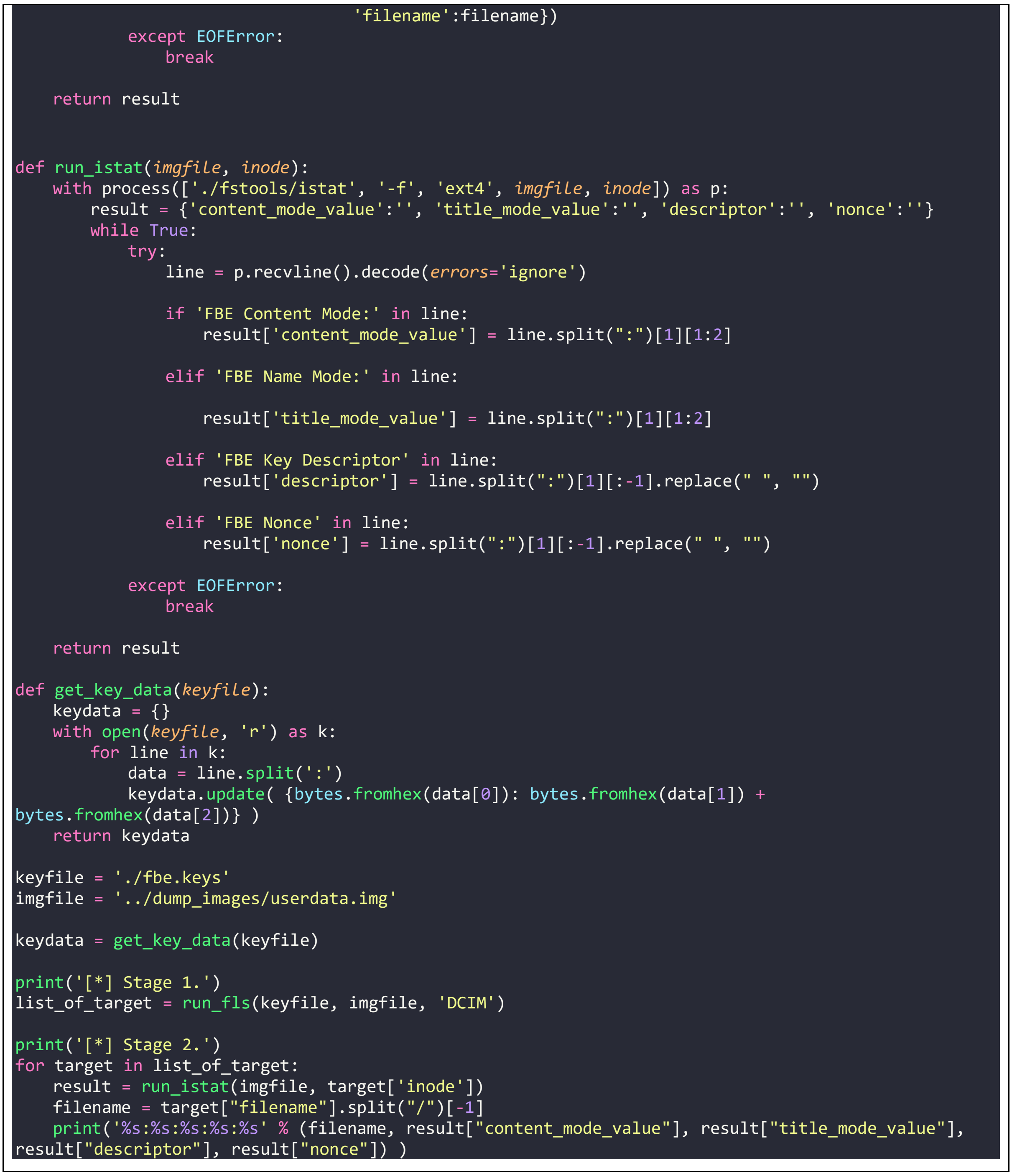


```
root@ddddh:~/windows/dfc/401/401 - Find an illegally filmed image/fbekeyfind# ./fstools/istat -f ext4 ../dump_images/userdata.img 106511
inode: 106511
Allocated
Group: 13
Generation Id: 2987584812
uid / gid: 1023 / 1057
mode: rrw-rw-r--
Flags: No A-Time, Compression Error, Extents,
size: 72530
num of links: 1

Extended Attributes  (Block: 426515)
security.selinux=u:object_r:media_rw_data_file:s0

Extended Attributes  (Inode Included)
FBE Content Mode: 1 (AES 256 XTS)
FBE Name Mode: 4 (AES 256 CTS)
FBE Key Descriptor: EC 34 DD 2D 7A 20 BF 2E
FBE Nonce: D6 69 8B 2F 31 8D C3 E5 93 4F 9F C4 14 53 FD 51

Inode Times:
Accessed:       2023-07-07 20:25:38.000000000 (KST)
File Modified:  2023-07-07 20:25:38.000000000 (KST)
Inode Modified: 2023-07-07 20:27:54.098443709 (KST)
File Created:   2023-07-07 20:27:54.098443709 (KST)

Direct Blocks:
426516 426517 426518 426519 426520 426521 426522 426523
426524 426525 426526 426527 426528 426529 426530 426531
426532 426533
```

[그림] istat 수행 결과

fls 수행 결과 중 DCIM 폴더 내에 있는 파일들의 inode 값이 필요하기 때문에 python pwntools 모듈을 사용하여 다음과 같은 스크립트를 작성할 수 있음.

```python
from pwn import *

context.log_level = 'error'

def run_fls(keyfile, imgfile, filter=''):
    with process(['./fstools/fls', '-K', keyfile, '-rF', imgfile]) as p:

        result = []
        while True:
            try:
                line = p.recvline().decode(errors='ignore')
                if filter in line:
                    #print(line[:-1])

                    inode    = line.split(':')[0].split(' ')[-1]
                    filename = line.split(':')[1][:-1]

                    result.append( {'inode':inode,
```



```
                                   'filename':filename})
            except EOFError:
                break

    return result


def run_istat(imgfile, inode):
    with process(['./fstools/istat', '-f', 'ext4', imgfile, inode]) as p:
        result = {'content_mode_value':'', 'title_mode_value':'', 'descriptor':'', 'nonce':''}
        while True:
            try:
                line = p.recvline().decode(errors='ignore')

                if 'FBE Content Mode:' in line:
                    result['content_mode_value'] = line.split(":")[1][1:2]

                elif 'FBE Name Mode:' in line:

                    result['title_mode_value'] = line.split(":")[1][1:2]

                elif 'FBE Key Descriptor' in line:
                    result['descriptor'] = line.split(":")[1][:-1].replace(" ", "")

                elif 'FBE Nonce' in line:
                    result['nonce'] = line.split(":")[1][:-1].replace(" ", "")

            except EOFError:
                break

    return result

def get_key_data(keyfile):
    keydata = {}
    with open(keyfile, 'r') as k:
        for line in k:
            data = line.split(':')
            keydata.update( {bytes.fromhex(data[0]): bytes.fromhex(data[1]) +
bytes.fromhex(data[2])} )
    return keydata

keyfile = './fbe.keys'
imgfile = '../dump_images/userdata.img'

keydata = get_key_data(keyfile)

print('[*] Stage 1.')
list_of_target = run_fls(keyfile, imgfile, 'DCIM')

print('[*] Stage 2.')
for target in list_of_target:
    result = run_istat(imgfile, target['inode'])
    filename = target["filename"].split("/")[-1]
    print('%s:%s:%s:%s:%s' % (filename, result["content_mode_value"], result["title_mode_value"],
result["descriptor"], result["nonce"]) )
```

다음은 스크립트의 수행 결과이며 해당 지문에 대한 답변임.

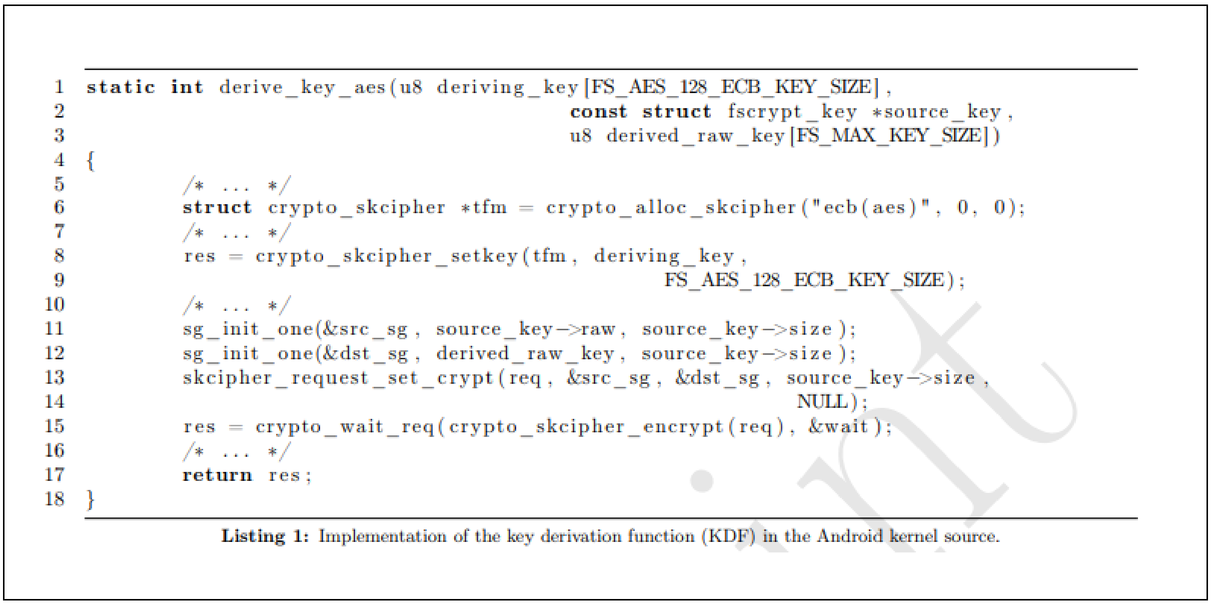
```
52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg:1:4:EC34DD2D7A20BF2E:D6698B2F318DC3E5934F9FC41453FD51
52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg:1:4:EC34DD2D7A20BF2E:F74D2D27FB6DE66BEB50431CBEDB2F49
52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg:1:4:EC34DD2D7A20BF2E:CF92299A65DB6027C56EE9DA77B8C3B8
```

```
55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg:1:4:EC34DD2D7A20BF2E:DFAD5017231C4B2D61E181D41D0939BB
55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg:1:4:EC34DD2D7A20BF2E:949CEC68E90588FD715096E1B1AA5D3D
55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg:1:4:EC34DD2D7A20BF2E:5FCBBF00942D371E1B73ACE806786AFA
black-coffee-1867753_640.jpg:1:4:EC34DD2D7A20BF2E:51C34B6909EA3598DAAD02214F2AF0F6
g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg:1:4:EC34DD2D7A20BF2E:4BB0AB4E5C3AC8F00B14260418402A06
g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg:1:4:EC34DD2D7A20BF2E:904D446A5F5A7710BF87A4DCCB59851E
g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg:1:4:EC34DD2D7A20BF2E:AC4589492E37397F1E26343E2D8C57D7
peas-580333_640.jpg:1:4:EC34DD2D7A20BF2E:2674FC48B1E313579AFB5C25C80AADC6
```

4) Obtain all the Derivation Encryption Keys for each image file obtained in Problem 2.

암호화된 파일 콘텐츠를 복호화하기 위한 키는 저자의 논문에도 구하는 방법이 기재되어 있음. AES-128-ECB 모드 Nonce 값을 키로 MasterKey를 암호화한 값을 파일 콘텐츠 암호화 모드1(AES-256-XTS) 복호화 키로 사용하게 됨.

1) One Key to Rule Them All: Recovering the Master Key from RAM to break Android's File-Based Encryption – 5p (https://faui1-files.cs.fau.de/public/one_key_to_rule/one_key.pdf)

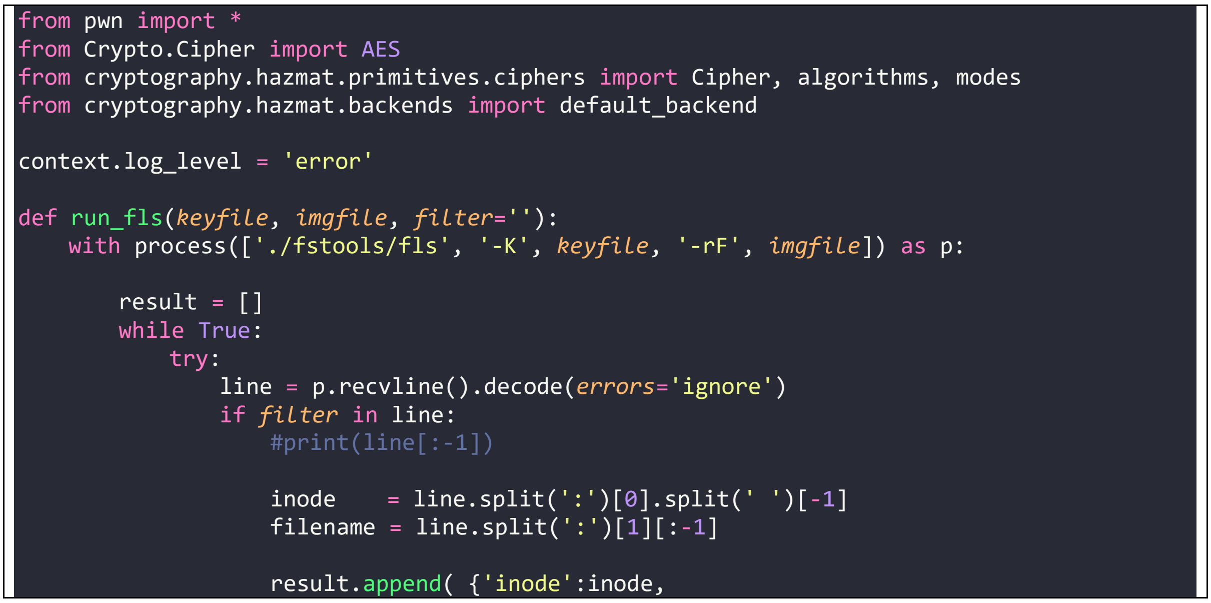


```
static int derive_key_aes(u8 deriving_key[FS_AES_128_ECB_KEY_SIZE],
                          const struct fscrypt_key *source_key,
                          u8 derived_raw_key[FS_MAX_KEY_SIZE])
{
        /* ... */
        struct crypto_skcipher *tfm = crypto_alloc_skcipher("ecb(aes)", 0, 0);
        /* ... */
        res = crypto_skcipher_setkey(tfm, deriving_key,
                                     FS_AES_128_ECB_KEY_SIZE);
        /* ... */
        sg_init_one(&src_sg, source_key->raw, source_key->size);
        sg_init_one(&dst_sg, derived_raw_key, source_key->size);
        skcipher_request_set_crypt(req, &src_sg, &dst_sg, source_key->size,
                                   NULL);
        res = crypto_wait_req(crypto_skcipher_encrypt(req), &wait);
        /* ... */
        return res;
}
```

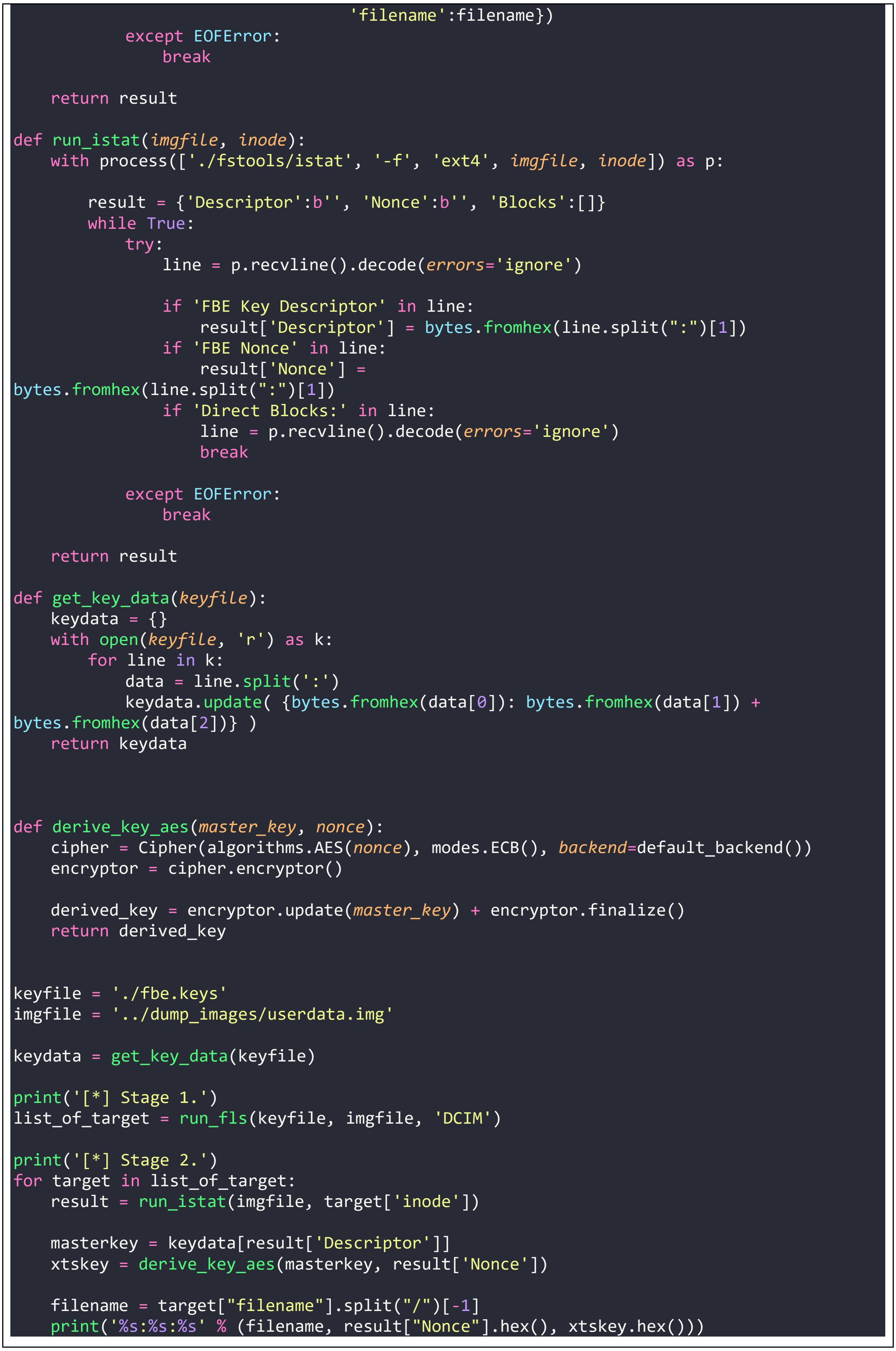
Listing 1: Implementation of the key derivation function (KDF) in the Android kernel source.

[그림] 1)저자의 논문 내 derive_key_aes 관련 발췌

다음과 같은 코드를 만들어 DEK(Derivation Encryption Key)를 구할 수 있음.



```
from pwn import *
from Crypto.Cipher import AES
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

context.log_level = 'error'

def run_fls(keyfile, imgfile, filter=''):
    with process(['./fstools/fls', '-K', keyfile, '-rF', imgfile]) as p:

        result = []
        while True:
            try:
                line = p.recvline().decode(errors='ignore')
                if filter in line:
                    #print(line[:-1])

                    inode    = line.split(':')[0].split(' ')[-1]
                    filename = line.split(':')[1][:-1]

                    result.append( {'inode':inode,
```



```
                                    'filename':filename})
            except EOFError:
                break

    return result

def run_istat(imgfile, inode):
    with process(['./fstools/istat', '-f', 'ext4', imgfile, inode]) as p:

        result = {'Descriptor':b'', 'Nonce':b'', 'Blocks':[]}
        while True:
            try:
                line = p.recvline().decode(errors='ignore')

                if 'FBE Key Descriptor' in line:
                    result['Descriptor'] = bytes.fromhex(line.split(":")[1])
                if 'FBE Nonce' in line:
                    result['Nonce'] =
bytes.fromhex(line.split(":")[1])
                if 'Direct Blocks:' in line:
                    line = p.recvline().decode(errors='ignore')
                    break

            except EOFError:
                break

    return result

def get_key_data(keyfile):
    keydata = {}
    with open(keyfile, 'r') as k:
        for line in k:
            data = line.split(':')
            keydata.update( {bytes.fromhex(data[0]): bytes.fromhex(data[1]) +
bytes.fromhex(data[2])} )
    return keydata



def derive_key_aes(master_key, nonce):
    cipher = Cipher(algorithms.AES(nonce), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()

    derived_key = encryptor.update(master_key) + encryptor.finalize()
    return derived_key


keyfile = './fbe.keys'
imgfile = '../dump_images/userdata.img'

keydata = get_key_data(keyfile)

print('[*] Stage 1.')
list_of_target = run_fls(keyfile, imgfile, 'DCIM')

print('[*] Stage 2.')
for target in list_of_target:
    result = run_istat(imgfile, target['inode'])

    masterkey = keydata[result['Descriptor']]
    xtskey = derive_key_aes(masterkey, result['Nonce'])

    filename = target["filename"].split("/")[-1]
    print('%s:%s:%s' % (filename, result["Nonce"].hex(), xtskey.hex()))
```

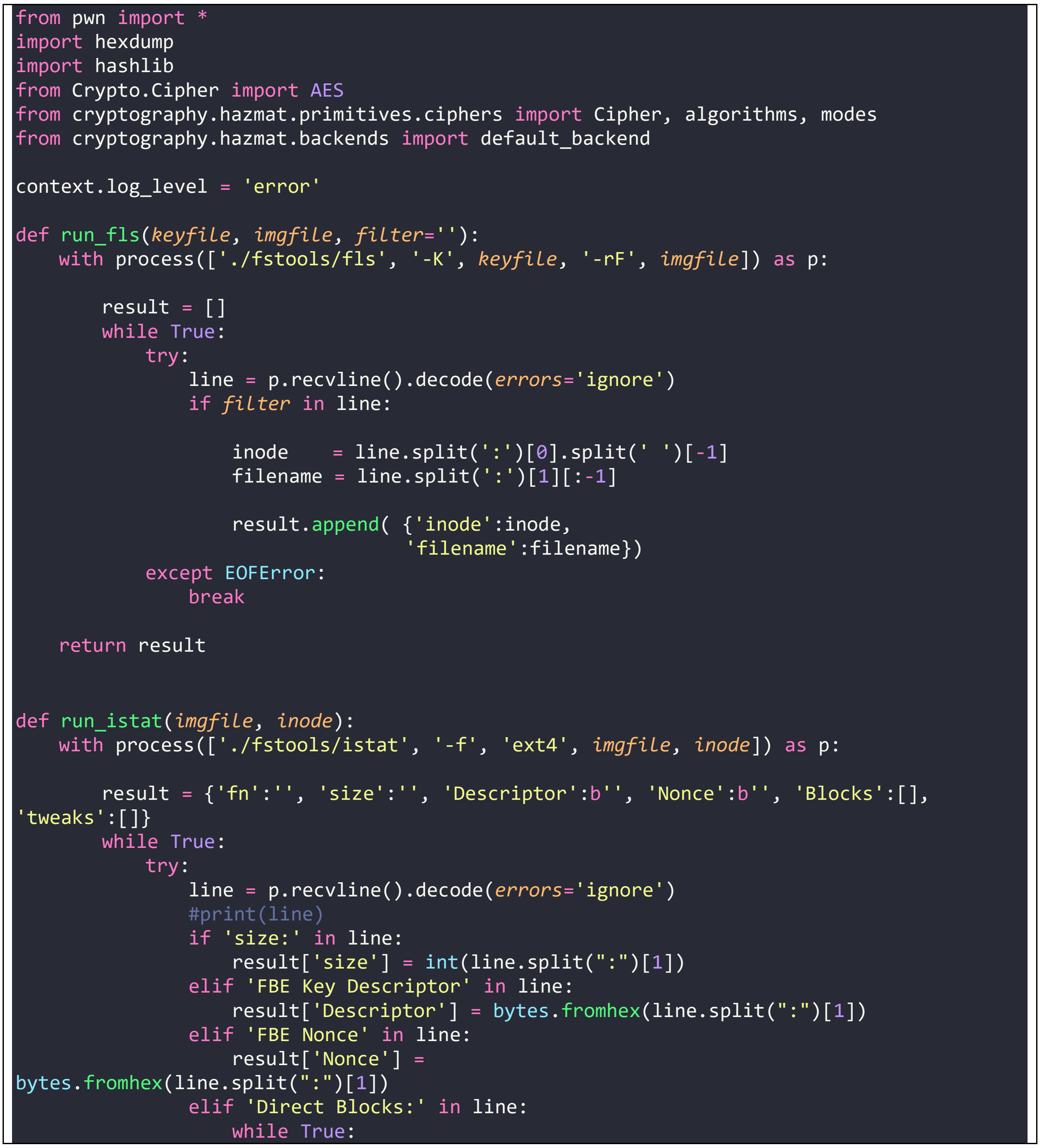
다음은 스크립트의 수행 결과이며 해당 지문에 대한 답변임.

```
52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg:d6698b2f318dc3e5934f9
fc41453fd51:399a4ca0c2af48e9d7f8448c1dcb6b32ef5a267852252ba38b2f7870a3b20444c81feb383c3e0d06
958390e929590e085e7a702b3e0de1ff6e7fefde3b5c5825
52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg:f74d2d27fb6de66beb50431
cbedb2f49:6c1c651e49eedf560dbdf6254144390241ffdc956eb72646b26dae5eac974c29974aaa1abd4a709da
c3f21093b25a1e170538a46b818fb55de9b9760c6f0d983
52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg:cf92299a65db6027c56e
e9da77b8c3b8:32a1fcaf50cd0c5fe65c77d4feab8ee499ca56f387660ff871ea1ed26e9a0f840a20e2c2392b4982
0685085c9318dcab2c51722c2d2c992dae884227956ccdb9
55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg:dfad5017231c4b2d61e1
81d41d0939bb:4c2633b0025cf260c06f161ffaf63eec0374f266a25b06d945b41791b705bf1fda2062e5e5276c9
15d6b4c139839487d08014c937323788060326c5fa9089be3
55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg:949cec68e90588fd7150
96e1b1aa5d3d:f0fcddc1e3acb4b1ab1af541d4080d1fd4e27e661f0a3d4bb808eccad569bed6971193b929ea4a
4f07f78ca017f1bc5182e4e72d956bb2215081abd1efc16f50
55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg:5fcbbf00942d371e1b73ace
806786afa:4289bb1d732afb0c4e13bddc01cbc9e13f993f059b3a4d03ed75553741f63244a79b8dc1b12637452
514d254f47c29c745b12cac2c9488f720c4c85aba05852d
black-coffee-
1867753_640.jpg:51c34b6909ea3598daad02214f2af0f6:ce65d1067b0e99ea24969f5d68fa37479e2e5c2938d7
ecd73e48e165b5255d80bd2ca92e93cd7bdbc6c67371264db5acc7b48ff91225f2252dfb51d3a997cad1
g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572c
b372c9_640.jpg:4bb0ab4e5c3ac8f00b14260418402a06:9723762d0cbaf02619ef0640e7679b679563f4c348fef2
86998587f37e157961b72b877f3d01bed88b3dfe5ec2ee4d2940d6472fc3c11641f9c8e81150d5e545
g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a470
6df7a5_640.jpg:904d446a5f5a7710bf87a4dccb59851e:1f1edae1d15e44e583f1408a63495e5fef299da257647f
1d7ad9e3aaabf91a63ed7d6400235f5f4789883cefd0987ac1552f30751fdb5fd23eb2e5189986974c
g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6
374d867_640.jpg:ac4589492e37397f1e26343e2d8c57d7:51b351438327ef7d04da0c5df809b41f21561af2fab8
7665453a86fe2b5a602c2daa2e71c00e77bbd199f2c8fa7636a736618d27fbb494d3b397db760ba2fba2
peas-
580333_640.jpg:2674fc48b1e313579afb5c25c80aadc6:b7c83d30c9db1be792930a0d3082782a4a34ee060c0c
5f4cf03533df3beed7f4c096de4d9c559b5130d799e3e04a60894cc9787b87a0942a98936b0426c52519
```

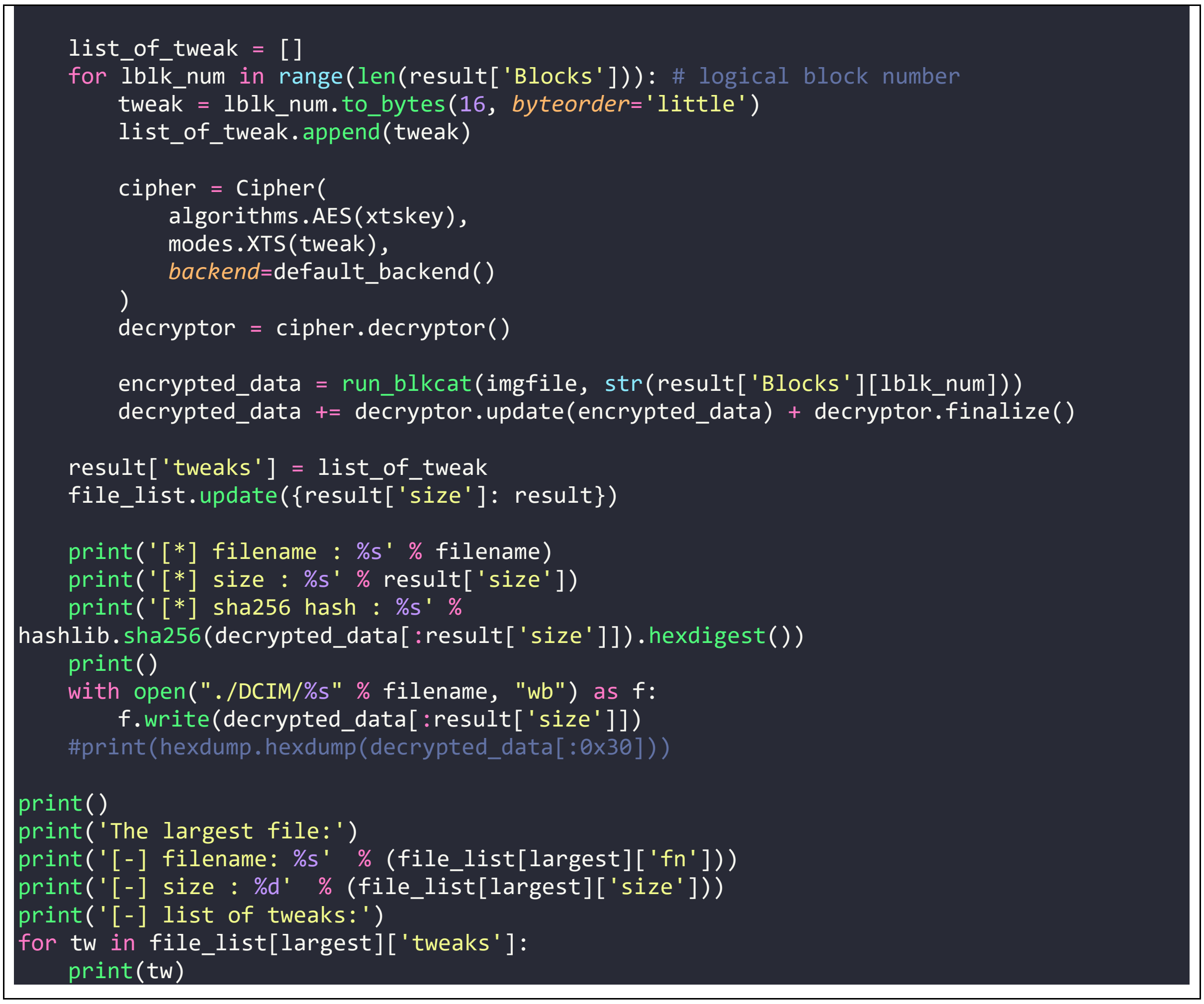
5) Decrypt the data of the image file with largest file size among the image files in the DCIM folder, and submit a list of all tweak values used in the decryption process and the SHA256 hash value of the decrypted image file.

파일 데이터 암호화 모드는 1(AES-256-XTS)로 Key 이외 tweak 값이 필요함. 해당 FBE 버전에서는 tweak 값을 0부터 순차적으로 증가되는 lblk_num(Logical Block Number)으로 사용하며 사용된 블록 수만큼 lblk_num이 증가되는 구조임.

지문 4번을 해결하는 과정에서 만들어 두었던 AES-256-XTS Key를 사용하여 파일 데이터를 복호화 하는 스크립트를 만들면 다음과 같음.

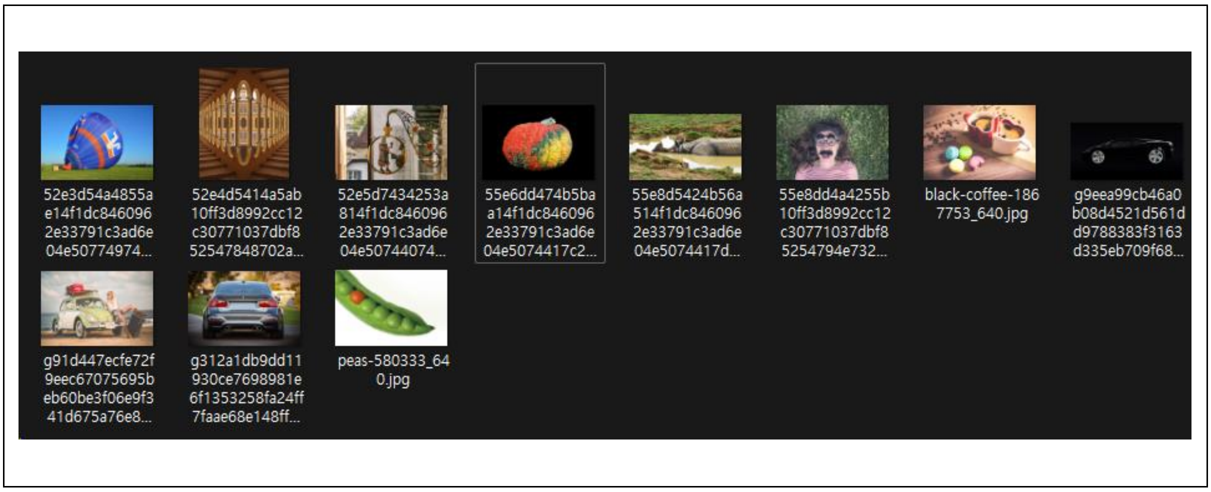


```python
from pwn import *
import hexdump
import hashlib
from Crypto.Cipher import AES
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

context.log_level = 'error'

def run_fls(keyfile, imgfile, filter=''):
    with process(['./fstools/fls', '-K', keyfile, '-rF', imgfile]) as p:

        result = []
        while True:
            try:
                line = p.recvline().decode(errors='ignore')
                if filter in line:

                    inode    = line.split(':')[0].split(' ')[-1]
                    filename = line.split(':')[1][:-1]

                    result.append( {'inode':inode,
                                    'filename':filename})
            except EOFError:
                break

    return result


def run_istat(imgfile, inode):
    with process(['./fstools/istat', '-f', 'ext4', imgfile, inode]) as p:

        result = {'fn':'', 'size':'', 'Descriptor':b'', 'Nonce':b'', 'Blocks':[],
'tweaks':[]}
        while True:
            try:
                line = p.recvline().decode(errors='ignore')
                #print(line)
                if 'size:' in line:
                    result['size'] = int(line.split(":")[1])
                elif 'FBE Key Descriptor' in line:
                    result['Descriptor'] = bytes.fromhex(line.split(":")[1])
                elif 'FBE Nonce' in line:
                    result['Nonce'] =
bytes.fromhex(line.split(":")[1])
                elif 'Direct Blocks:' in line:
                    while True:
                        line = p.recvline().decode(errors='ignore')
                        for blockid in line[:-2].split(' '):
                            try:
                                result['Blocks'] += [int(blockid)]
                            except:
                                break


            except EOFError:
                break

    return result


# ./sleuthkit-ext4-fbe-master/tools/fstools/blkcat -i raw -f
ext4 ./fbekeyfind/userdata.img 426516 > test1.block
def run_blkcat(imgfile, blockid):
    with process(['./fstools/blkcat', '-i', 'raw', '-f', 'ext4', imgfile, blockid,]) as
p:
        return p.recvall()


def get_key_data(keyfile):
    keydata = {}
    with open(keyfile, 'r') as k:
        for line in k:
            data = line.split(':')
            keydata.update( {bytes.fromhex(data[0]): bytes.fromhex(data[1]) +
bytes.fromhex(data[2])} )
    return keydata

def derive_key_aes(master_key, nonce):
    cipher = Cipher(algorithms.AES(nonce), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()

    derived_key = encryptor.update(master_key) + encryptor.finalize()
    return derived_key


keyfile = './fbe.keys'
imgfile = '../dump_images/userdata.img'

keydata = get_key_data(keyfile)

largest = 0
file_list = {}
list_of_target = run_fls(keyfile, imgfile, filter='DCIM')
for target in list_of_target:
    #print(f'target filename : {target["filename"]}, inode : {target["inode"]}')
    filename = target["filename"].split("/")[-1]

    result = run_istat(imgfile, target['inode'])
    result['fn'] = filename

    if result['size'] > largest:
        largest = result['size']

    #if len(result['Descriptor']) == 0 or len(result['Blocks']) == 0 :
    #    continue

    masterkey = keydata[result['Descriptor']]

    xtskey = derive_key_aes(masterkey, result['Nonce'])
    decrypted_data = b""
```



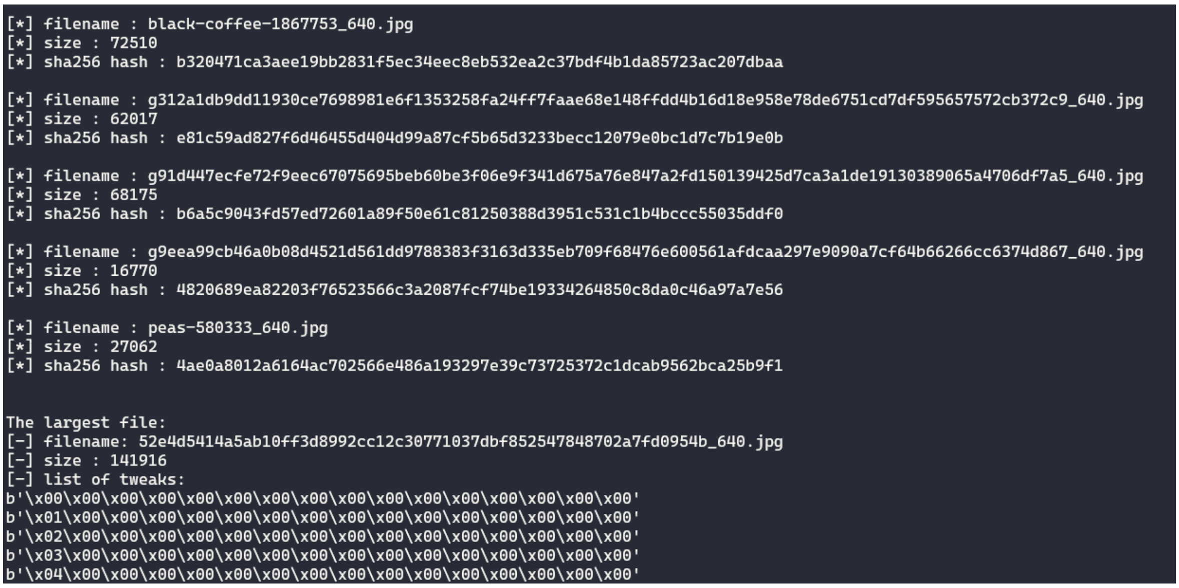
```
    list_of_tweak = []
    for lblk_num in range(len(result['Blocks'])): # logical block number
        tweak = lblk_num.to_bytes(16, byteorder='little')
        list_of_tweak.append(tweak)

        cipher = Cipher(
            algorithms.AES(xtskey),
            modes.XTS(tweak),
            backend=default_backend()
        )
        decryptor = cipher.decryptor()

        encrypted_data = run_blkcat(imgfile, str(result['Blocks'][lblk_num]))
        decrypted_data += decryptor.update(encrypted_data) + decryptor.finalize()

    result['tweaks'] = list_of_tweak
    file_list.update({result['size']: result})

    print('[*] filename : %s' % filename)
    print('[*] size : %s' % result['size'])
    print('[*] sha256 hash : %s' %
hashlib.sha256(decrypted_data[:result['size']]).hexdigest())
    print()
    with open("./DCIM/%s" % filename, "wb") as f:
        f.write(decrypted_data[:result['size']])
    #print(hexdump.hexdump(decrypted_data[:0x30]))

print()
print('The largest file:')
print('[-] filename: %s'  % (file_list[largest]['fn']))
print('[-] size : %d'  % (file_list[largest]['size']))
print('[-] list of tweaks:')
for tw in file_list[largest]['tweaks']:
    print(tw)
```

위 스크립트의 결과물로 실행 경로의 DCIM 폴더에 복호화된 이미지들을 확인할 수 있음.


52e3d54a4855ae14f1dc8460962e33791c3ad6e04e50774974...
52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a...
52e5d7434253a814f1dc8460962e33791c3ad6e04e50744074...
55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2...
55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d...
55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e732...
black-coffee-1867753_640.jpg
g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68...
g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e8...
g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ff...
peas-580333_640.jpg

[그림] 복호화된 DCIM 이미지

또한, 스크립트 실행 중 출력되는 문자열을 확인하여 각 파일의 사이즈와 해쉬 그리고 가장 큰 이미지 복호화에 사용된 tweak 리스트를 확인할 수 있음.



```
[*] filename : black-coffee-1867753_640.jpg
[*] size : 72510
[*] sha256 hash : b320471ca3aee19bb2831f5ec34eec8eb532ea2c37bdf4b1da85723ac207dbaa

[*] filename : g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg
[*] size : 62017
[*] sha256 hash : e81c59ad827f6d46455d404d99a87cf5b65d3233becc12079e0bc1d7c7b19e0b

[*] filename : g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg
[*] size : 68175
[*] sha256 hash : b6a5c9043fd57ed72601a89f50e61c81250388d3951c531c1b4bccc55035ddf0

[*] filename : g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg
[*] size : 16770
[*] sha256 hash : 4820689ea82203f76523566c3a2087fcf74be19334264850c8da0c46a97a7e56

[*] filename : peas-580333_640.jpg
[*] size : 27062
[*] sha256 hash : 4ae0a8012a6164ac702566e486a193297e39c73725372c1dcab9562bca25b9f1


The largest file:
[-] filename: 52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg
[-] size : 141916
[-] list of tweaks:
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x04\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

[그림] 스크립트 출력 문자열

다음은 스크립트 출력 문자열을 정리한 결과임.

filename	sha256 hash	size
g9eea99cb46a0b08d4521d561dd9788383f3163d335eb709f68476e600561afdcaa297e9090a7cf64b66266cc6374d867_640.jpg	4820689ea82203f76523566c3a2087fcf74be19334264850c8da0c46a97a7e56	16770
peas-580333_640.jpg	4ae0a8012a6164ac702566e486a193297e39c73725372c1dcab9562bca25b9f1	27062
g312a1db9dd11930ce7698981e6f1353258fa24ff7faae68e148ffdd4b16d18e958e78de6751cd7df595657572cb372c9_640.jpg	e81c59ad827f6d46455d404d99a87cf5b65d3233becc12079e0bc1d7c7b19e0b	62017
g91d447ecfe72f9eec67075695beb60be3f06e9f341d675a76e847a2fd150139425d7ca3a1de19130389065a4706df7a5_640.jpg	b6a5c9043fd57ed72601a89f50e61c81250388d3951c531c1b4bccc55035ddf0	68175
55e6dd474b5baa14f1dc8460962e33791c3ad6e04e5074417c2d78d19f48c3_640.jpg	2cb1519114fb99754f0eadd35626531471a3038b87ef08a5e54b946eb45ea271	71341
black-coffee-1867753_640.jpg	b320471ca3aee19bb2831f5ec34eec8eb532ea2c37bdf4b1da85723ac207dbaa	72510
52e3d54a4855ae14f1dc8460962e33791c3ad6e04e507749742c78d6944cc3_640.jpg	f4ae338500ffb56066cb7f7233365184714cf82eb19dc24b8e59375c2b46ef8f	72530
52e5d7434253a814f1dc8460962e33791c3ad6e04e507440742a7ad2974bc1_640.jpg	55c8b9e2d4764fb4e127b10f9c18138c16e08020c7f23cba884468f26f2c63eb	80218
55e8d5424b56a514f1dc8460962e33791c3ad6e04e5074417d2e72d29e4ac3_640.jpg	da05d7ac7a5e6d3d6fc1eab5e2232edbee42e004eac60aa595309ced88545ee5	92537

55e8dd4a4255b10ff3d8992cc12c30771037dbf85254794e73277bd7954a_640.jpg	ad8de40199468f5052cde65e1ff8e0b00db51ba61abe7c5542ceb5963a76a663	113000
52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg	33a51636684e932a956d53a686e63bfd2528bf2e87ad55fdede78261e5c68ef3	141916

가장 크기가 큰 이미지는 “52e4d5414a5ab10ff3d8992cc12c30771037dbf852547848702a7fd0954b_640.jpg” 이미지 파일이며 해당 이미지의 복호화 당시 tweak values list와 sha256 정보는 다음과 같음.

■ tweak values list:

Tweak values에 사용된 logical block number는 0, 1, 2, 3… 이지만 실제 AES-256-XTS에서 Tweak으로 사용되는 값을 16바이트이며 복호화에 사용된 값은 다음과 같음.

```
₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x01₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x02₩x00₩x00₩x00₩x00₩x
00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x03₩x00₩x00₩x00₩x00₩x00₩x00₩x00
₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x04₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00,₩x05₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x
00₩x00₩x00,₩x06₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,
₩x07₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x08₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x09₩x00₩x00₩x00₩x00₩x
00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x0a₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩
x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x0b₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00
₩x00₩x00₩x00₩x00₩x00,₩x0c₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00,₩x0d₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩
x0e₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x0f₩x00₩x00
₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x10₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x11₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x
00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x12₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00
₩x00₩x00₩x00₩x00₩x00,₩x13₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00,₩x14₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩
x15₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x16₩x00₩x00
₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x17₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x18₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x
00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x19₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00
₩x00₩x00₩x00₩x00₩x00,₩x1a₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00,₩x1b₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩
x1c₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x1d₩x00₩x00
₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x1e₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x1f₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x
```

```
00₩x00₩x00₩x00₩x00₩x00₩x00₩x00,₩x20₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00
₩x00₩x00₩x00₩x00₩x00,₩x21₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x0
0₩x00₩x00,₩x22₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00
```

■ FILE_CONTENT_SHA256_HASH

33a51636684e932a956d53a686e63bfd2528bf2e87ad55fdede78261e5c68ef3